5-2015

Ultra-Low-Power Configurable Analog Signal Processor for Wireless Sensors

James Kelly Griffin
University of Tennessee - Knoxville, jgriff48@vols.utk.edu

To the Graduate Council:

I am submitting herewith a thesis written by James Kelly Griffin entitled "Ultra-Low-Power Configurable Analog Signal Processor for Wireless Sensors." I have examined the final electronic copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Jeremy Holleman, Major Professor

We have read this thesis and recommend its acceptance:

Benjamin Blalock, Garret Rose

Accepted for the Council:
Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Ultra-Low-Power Configurable Analog Signal Processor for Wireless Sensors

A Thesis Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

James Kelly Griffin

May 2015

In dedication to my parents for providing me with their love and support throughout my education.

Acknowledgements

I would like to thank my advisor, Professor Jeremy Holleman, for all his help with my graduate project as well as the priceless education he has provided me with at the University of Tennessee. His mentoring has provided me with an education that will allow me to join the work force and create new innovative technology. Professor Ben Blalock has taught me a great deal about analog electronics and helped me obtain real life experience. For this I will always be thankful. I'm very thankful to the entire Integrated Silicon Systems group who has assisted me in every step of my graduate project. A special thanks to Tan Yang and Junjie Lu for helping build components in my signal processor as well as the overall design.

I would like to thank the research consortium CDADIC for funding my research assistantship that allowed me to attend graduate school.

I would like to thank the administrative assistants, Dana Bryson, Julia Elkins, and Melanie Kelley for their help throughout the years.

Lastly I would like to thank my co-workers and friends among the EECS department for their advice and company in the lab. Thank you Jake Shelton, Jeff Dix, Alex McHale, Shahriar "Pollob" Jahan, Nicholas Poore, Terence Randall, Logan Taylor, Ifana Mahbub, Pranshu Bansal, and Jeremy Langford.

Abstract

The demand for on-chip low-power Complementary Metal Oxide Semiconductor (CMOS) analog signal processing has significantly increased in recent years. Digital signal processors continue to shrink in size as transistors half in size every two years. However, digital signal processors (DSP's) notoriously use more power than analog signal processors (APS's). This thesis presents a configurable analog signal processor (CASP) used for wireless sensors. This CASP contains a multitude of processing blocks include the following: low pass filter (LPF), high pass filter (HPF) integrator, differentiator, operational transconductance amplifier (OTA), rectifier with absolute value functionality, and multiplier. Each block uses current-mode processing and operates in the sub-threshold region of operation. Current-mode processing allows for noise reduction, lower power consumption, and better dynamic range. Each block contains configurable current sources and capacitor banks for maximum adaptability. The blocks were designed, simulated, and fabricated in Cadence using IBM's 130nm CMOS process. The processing blocks were combined into a four by three array and connected using specially designed interconnect fabric. A test structure including the LPF, HPF, and multiplier was also constructed for characterization purposes. The main goals for this project are frequency compression and creating a non-linear energy operator for neural spike detection.

The test results for the low-pass filter, integrator, and frequency divider reflected the simulated values. The other blocks didn't perform as well as in simulation. The interconnect fabric ties all the blocks together and achieved maximum configurability

with negligible attenuation. In simulation, frequency compression was achieved with 30μ[micro]W of power from a 1V supply rail.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Recent advances in science and semiconductor technology have created exciting possibilities for new-age micro electronic systems. The project at hand deals with the acoustic signals emitted and observed by a bat. Another use could be to examine neural recordings from a bat brain. Advancements in science allow humans to discover new technology to better the world. For example, understanding the echolocation of bats could lead to new radar systems with higher precision. In order to sense brain activity, one must be able to sense minute spikes in the brain. This is achieved by placing micro-electrodes on brain tissue and amplifying the signals sensed. These signals will be either converted to digital form by an analog to digital converter (ADC), or processed for easier ADC conversion. Finally a transmitter will send the signals from the sensor chip to a base station.

The combination of a sensor, ADC, and transmitter can be described as a microprocessor. With transistors reducing size according to Moore's Law, microprocessors show dramatic signs of improvement in the areas of power and price. This trend has created a large market for miniature wireless sensors used for studying neurological signals. A major area of research is how to process the data acquired from these sensors in a power efficient manner. Designers face trade-offs between processing data before transmission or after transmission.

The typical fashion for micro sensors is to send the raw signal directly into an ADC. Then this data can be transmitted back to a base station where the data can be processed by a digital signal processor (DSP). The problem with this data transfer method is achieving maximum resolution with limited power. When detecting neuro-spikes, designers usually acquire signals from an array of micro-electrodes. Each channel must be amplified and digitized, creating the need for multi channel ADCs. In order to digitize full waveforms, systems are typically limited to a small number of channels by their power constraints. However, by reducing the information in each channel, higher channel counts can be obtained. For example, in [4], 100 channels of neural spike threshold detectors are digitized through an ADC and data is transmitted back to another board. Such rudimentary processing discards potentially useful information. If data is lost between the sensor and ADC, then the final results may not accurately portray the information desired. Designers could use higher resolution ADCs; however, these ADCs would burn significantly more power and exceed available power produced by small batteries. Therefore, the data in general must be reduced prior to transmission.

Analog signal processing offers a potential solution to the problem. Analog circuits can perform many operations on raw data from the sensors with a space and power savings of up to three orders of magnitude compared to a digital solution [2]. An ultra-low power analog signal processor (ASP) can be used on each channel to reduce strain on the ADC and transmitter, while taking full advantage of the data received from the sensor. This processor can solve the problem of data transfer by processing data prior to digital conversion.

Presented in this thesis is a configurable analog signal processor (CASP), operating at ultra-low power levels. The analog processing blocks include a low-pass filter (LPF), high-pass filter (HPF) integrator, differentiator, operational transconductance amplifier (OTA), rectifier with absolute value functionality, and multiplier. The blocks use current-mode signal processing techniques in the weak inversion region of

operation. Each element was placed in an array and connects with multi-directional interconnect fabric for maximum flexibility.

This thesis is organized to give the reader a full scope of the project goals, research, and final results. Chapter 2 describes previous research on ASPs including the challenges and low power capabilities. Chapter 3 will work through the design procedures and provide simulations captured by Cadence. Chapter 4 will depict and compare the final experimental measurements of CASP with the simulated results. Finally Chapter 5 will summarize results, draw conclusions on the findings, and describe future work.

Chapter 2

Background

This chapter addresses the background of field programmable analog arrays (FPAAs) and other information concerning this thesis. Section 2.1 talks about the differences in analog and digital integrated circuits in the areas of power, size, and design techniques. Section 2.2 will discuss previous field programmable analog array (FPAA) designs and analog transistor techniques. Section 2.3 covers the inversion mode of operation used in this thesis. Finally, section 2.4 discusses some figures of merit for analog signal blocks.

2.1 Integrated Circuits

Integrated circuits (ICs) have had a profound effect on human interaction. Before the integrated circuit, people relied on vacuum tubes and solid-discrete state transistors to control electricity. These devices use large amounts of area and power. Since the 1950's, ICs have taken over and are now part of everyday life. An IC can be defined as an electronic circuit containing a transistor and a combination of the following: resistor, capacitors, diodes, or inductors. The transistor acts as a switch controlling the flow of electricity depending on the voltages applied to its terminals. The other components can be used along with a transistor to create amplifiers, flashing lights, and numerous other electronic circuits. In modern days ICs can be implemented

using analog or digital techniques. Analog circuits refers to electronics dealing with continuous, variable signals. On the contrary, digital circuits deal with only two levels of signals usually called ones and zeros. There are many pros and cons for each IC.

2.1.1 Analog Circuits

Life can be considered analog in nature, so one would expect analog electronics to be superior over digital electronics. This is not the case at all. The main advantage of analog circuits is using direct input and output signals that don't need to be converted to the digital domain. No conversion means no data can be lost while being converted. Another advantage can be the power used for analog computation is far less than for a digital computation. Consequently, analog circuits are ideal for low power ICs that interface directly with the outside world.

The difficult attributes of analog designs are design, flexibility, and susceptibility to noise interference. Automation tools for analog designs have been created, but they do not consider all effects. Analog designers must evaluate all parasitics and noise contributions from surrounding circuitry. Many transistor matching techniques, like common-centroid or ABBA, are used in layout to guarantee accurate functionality [5]. When proper techniques are followed, analog designs can be powerful tools for signal processing. Unfortunately analog designs can not be easily scaled down to new technology; designers must consider short-channel effects and parasitics and redesign circuits to fit new technologies.

2.1.2 Digital Circuits

Digital circuits dominate the electronics industry. Their popularity comes from many factors. For one, digital designs are easily scaled to new technology. Also digital designs can be synthesized from code. These attributes open up endless possibilities to create complicated logic structures in a timely manner. One of the most prominent digital applications is field programmable gate arrays (FPGAs).

FPGAs are widely used for signal processing, because they are easily programmed for different applications. Their large power dissipation is the only drawback to being used in micro sensors.

Other negative aspects of digital circuits are power consumption and conversion loss. When integrated into systems, digital circuits must use an ADC to digitize the outside signal. Once digitized, signals are more easily processed. Extensive work has been done on ADCs in order to achieve minimal information loss during conversion; however, there will always be some loss associated with ADCs. An example of their high power consumption can be seen when comparing an analog translinear computing circuit to a synthesized digital circuit. Figure 2.1 computes x^2/y for two input currents. The axis show I_X and I_{OUT} as the x-axis and y-axis, respectively; the different curves are from varying I_Y. Simulations indicate that a digitally synthesized counter circuit with the same dynamic range consumes about ten times more energy per operation in comparison to the analog circuit.

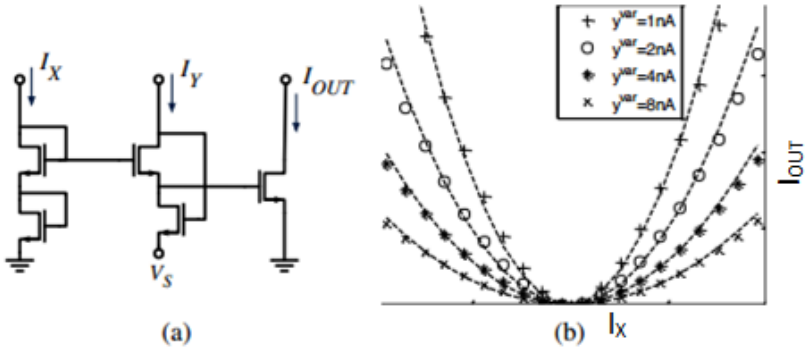


Figure 2.1: (a) Analog translinear circuit computing x^2/y (b) Correspondence between ideal and actual results of x^2/y computation

2.2 Field Programmable Analog Arrays

For years digital field programmable gate arrays (FPGAs) sufficed designers processing needs. However, new fields of study demand lower power design that cannot

handle bulky, power hungry digital circuits. This effect has led to the development of FPAAs. Designers have created FPAAs using several different techniques. The most popular techniques being floating gate arrays and continuous-time operational transconductance amplifiers (OTAs). In [7] and [1] the processing blocks are called configurable analog blocks (CABs). These CABs can be configured to complete different tasks based on processing needs.

The floating gate based FPAA presented in [1] contains 32 CABs and over 50,000 floating-gate elements. The use of floating gate switches allows their design to eliminate memory for configuring switches. The floating gate approach uses less area, but the programming becomes much more cumbersome. The routing techniques used in this design use nearest neighbor as well as global wires. This allowed them to pass a signal with 57 MHz bandwidth when using one near neighbor and one global line. The interconnect fabric is a very important part of any array design. If not designed properly, the bandwidth will limit the entire FPAA to low frequencies making it useless. The CABs in this design contain floating gate OTAs, translinear Gilbert multipliers, and folded Gilbert multipliers. Each cell processes signals in a different way creating a large scale programmable analog array. [1]

The second related FPAA is a continuous-time operational transconductance amplifier and capacitor (OTA-C) filter written by [7]. This analog array consists of 40 CABs, which are OTA-Cs. By altering the transconductance and capacitor setup, they created filters that operate from several kilohertz to several megahertz. This design utilizes a programmable current mirror to enable control over the OTA-C filter. Similar techniques will be explained in this thesis. The OTA-C FPAA was able to achieve a 6th order bandpass filter and a fourth order bi-quad cascaded filter. [7]

2.3 Subthreshold Analog Circuits

2.3.1 MOSFET Regions of Operation

MOSFETs can operate in the following three different modes of operation: weak-inversion, moderate-inversion, and strong-inversion mode. Transistors in weak and strong-inversion can be saturated or not saturated. These regions of operation for a minimum sized NFET in IBM's 8RF process are seen in figure 2.2. For the simulation, the threshold voltage was 137 mV. The operational modes are based on the voltage relationships between the source, gate, and drain of the transistor. Weak-inversion occurs when V_{GS} is less than the threshold voltage. Strong-inversion happens when V_{GS} is greater than the threshold voltage plus about 200 mV. The regions above and below the curve define the saturation and non-saturation regions, respectively. Most transistors are used in the moderate and strong-inversion region because this is when they are considered turned on. However, as designers looking for ways to save power, utilize the weak-inversion region.

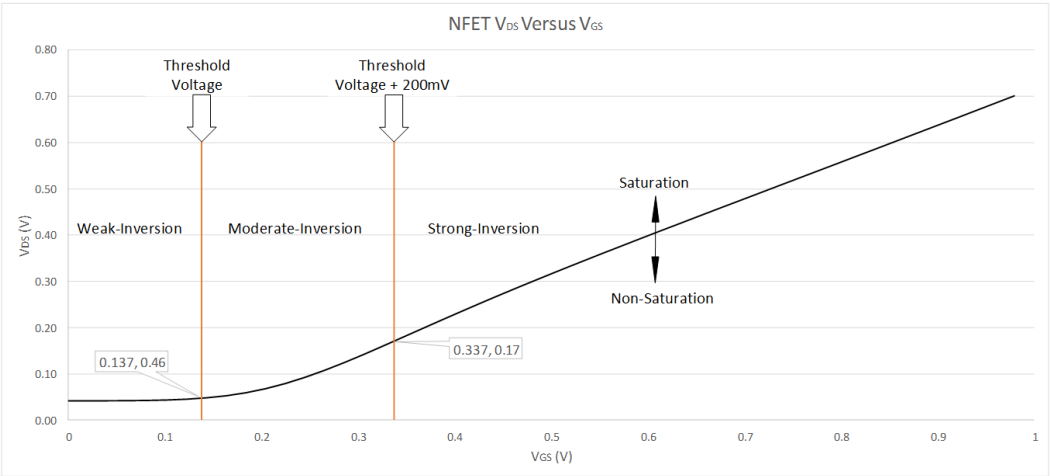


Figure 2.2: NFET Characterization for Regions of Operation

The weak-inversion region will be considered the subthreshold region throughout this thesis. This operation occurs when the gate to source voltage is less than the threshold voltage, $V_{GS} < V_{th}$. In the late 1970's, Vittoz was experimentally deriving

the characteristics of the subthreshold region [10]. With advanced control capabilities of today, designers can more accurately operate in this region. As V_{GS} approaches V_{th}, the transistor produces a minute current with an exponential relationship described by equation 2.1,

$$I_{DS} = I_o \times (W/L) \times e^{k(\frac{V_{GS}}{U_T})} \times e^{(1-k)(\frac{V_{BS}}{U_T})} \tag{2.1}$$

$$\text{if } V_{DS} >> V_{th}$$

where I_o is a positive constant current, k or kappa is a technology-dependent parameter assumed to be constant, V_{BS} is the bulk to source voltage, and U_T is the thermal voltage equal to KT/q [9]. Many advantages can be taken from the exponential involving V_{GS}.

2.3.2 Translinear Principle

The exponential relationships in MOSFETs are refereed to as translinear circuits, which were first discovered by Barrie Gilbert in 1975 for bipolar junction transistors (BJTs) [3]. The word "translinear" refers to the exponential I/V characteristics of bipolar transistors seen in equation 2.2.

$$I_C = I_S \times e^{k(\frac{V_{BE}}{U_T})} \tag{2.2}$$

The translinear principle applied to MOSFETs is further explored by Teresa Serrano-Gotarredona [9]. She explains how an equal number of oppositely connected translinear elements can create a loop relationship. The product of the currents in the clockwise direction equals the opposite product of the counterclockwise currents, creating a translinear loop. Using this relationship and capacitors, designers can create powerful analog processing blocks. Subthreshold circuits are very sensitive to matching errors and perform more slowly than transistors in other modes of operation. This attribute does not make subthreshold circuits ideal for high speed use;

however, most signals associated with neurological brain waves are at low frequencies. Since only the currents are under scrutiny, these circuits can be classified as current-mode circuits. The voltages just need to bias the transistors in the correct mode of operation.

2.4 FPAA Performance Metric

2.4.1 Dynamic Range

One figure of merit used for evaluating the processing blocks is dynamic range. The dynamic range is the ratio of the maximum level of a parameter that does not distort the signal to the minimum detectable level. For this thesis, dynamic range will be based on the maximum input amplitude and the output integrated noise level. For the maximum input range, a technique called total harmonic distortion, THD, will be used. THD is the ratio of the power in all the harmonics to the power in the fundamental frequency, described by equation 2.3.

$$\text{THD} = \frac{\sum \text{Harmonics}_{dB}}{\text{Fundamntal Frequency}_{dB}} \tag{2.3}$$

THD can easily be calculated on a network analyzer as well as in simulation. To find the maximum input current for the dynamic range, the input will be increased until the THD is 1.0%. The noise level will be determined for the minimal detectable level of input. A noise simulation or test will be performed. The output will be in $\text{A}_{RMS}/\sqrt{Hz}$. The noise will be squared, integrated over frequency, and the square root will be taken to find the final $\text{A}_{RMS,noise}$ level. Finally, the maximum input will be divided with the minimum input, and dynamic range will be calculated, as seen in equation 2.4.

$$DR = 20 \times \log(\frac{A_{RMS,input}}{A_{RMS,noise}}) \tag{2.4}$$

Chapter 3

Design and Simulations

This chapter works through the design and simulation for each block and the overall hierarchy of the Configurable Analog Signal Processor (CASP) presented. Throughout this work, the chip will be referred to as CASP. The first section, 3.1, presents a configurable current mirror used in every block of the CASP design. Section 3.2 presents the low-pass filter design, which is used to realize an integrator as well. A similarly designed high-pass filter and differentiator are demonstrated in section 3.3. The next sections, 3.4, 3.5, 3.6, and 3.7 will describe an OTA, multiplier, and rectifier, and frequency divider respectively. Then, the interconnect fabric for CASP will be presented in section 3.8. Finally, section 3.9 will pull all the blocks together and demonstrate the full analog signal processor.

3.1 Configurable Current Bias

3.1.1 Current Mirror Design

Designing a universal configurable current source presented a few of its own challenges. The first attempt was to use a self biasing current source and add a current mirror for reconfigurability. However, a safer design was implemented by providing an external voltage to a global biasing block. This block connects to several reconfigurable current

mirrors across the chip. Figure 3.1 portrays an entire configurable current source block. The part labeled "global bias" is only implemented one time on the chip, and the Vb1 and Vb2 pins are connected to every current mirror throughout the system. The current mirror is a cascode current mirror design that includes an extra row of transistors acting as switches. All of the transistors in the current mirror and global bias have the same gate length, 240 nm, and gate width, 360 nm. The scaling is done with gate fingers. Adding fingers to a transistor makes it have two gates separated by a pad. In figure 3.1, the transistors are set up so that each branch has twice as many fingers as the previous branch, starting with a single finger transistor. The global biasing transistors all have eight fingers, which allows the current in the first three branches of the current mirror to be scaled down while the other current branches are scale up. This current mirror design suffices for any current biases needed in CASP.

3.1.2 Current Sinking and Sourcing

The additional circuitry above the current mirror provide functionality to sink current instead of sourcing current. The pin labeled "R" runs through an inverter to create "R_{bar}." The switches S1 and S2 are transmission gate switches that operate based on the values of R and R_{bar}. When R is set high to VDD, R_{bar} turns off the NMOS transistor and turns on both the switches. This allows current to flow through the PMOS cascode current mirror providing current source. The PMOS current mirror was altered from a standard cascode current mirror in order to achieve higher output impedance. The PMOS current stage re-design was necessary for the current source to work properly with some of the processing blocks. When R is set low to ground, the current flows out through the NMOS transistor providing a current sink.

3.1.3 Configurable Current Bias Simulations

Extensive simulation were performed on the configurable current mirror to test its functionality. The current mirror was also used when simulating other blocks such

as the LPF, HPF and multiplier. The current sourcing capabilities range from 3 nA to 490 nA, as shown in figure 3.2. The current sinking capabilities range from 3 nA to 626 nA, as shown in figure 3.3. The simulations were run by setting each control switch, D0-D7, to a specific bit and the incrementing the eight bits from 1 to 255. The spikes in the simulations are due to switching activity and should be ignored because the current source won't be switching during operation. The large jumps in current are due to switching on of the last three current branches. These three branches produce significantly more current than the other branches, and when they turn on, the other branches are turned off. The current source will only provide a DC current to processing blocks.

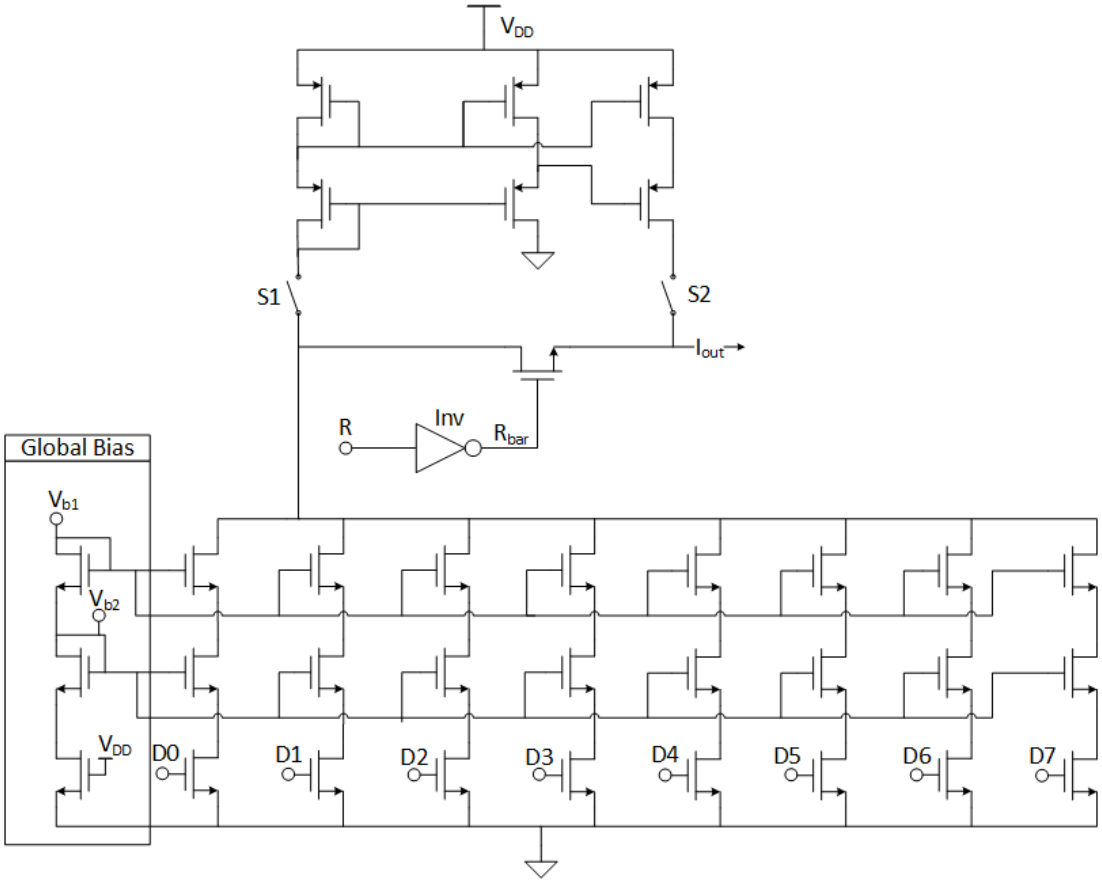


Figure 3.1: Configurable Current Mirror Structure Including Global Bias

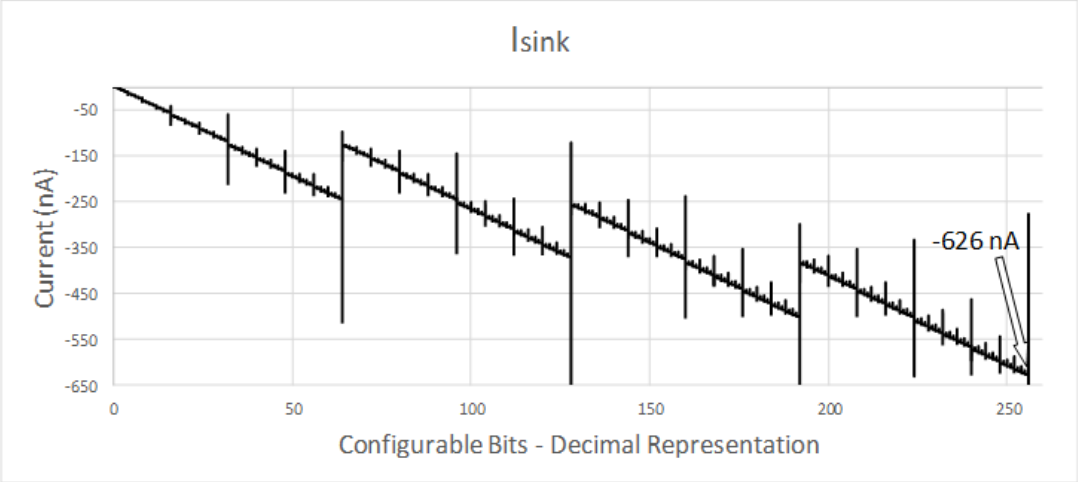


Figure 3.2: Configurable Current Mirror Sink Simulation

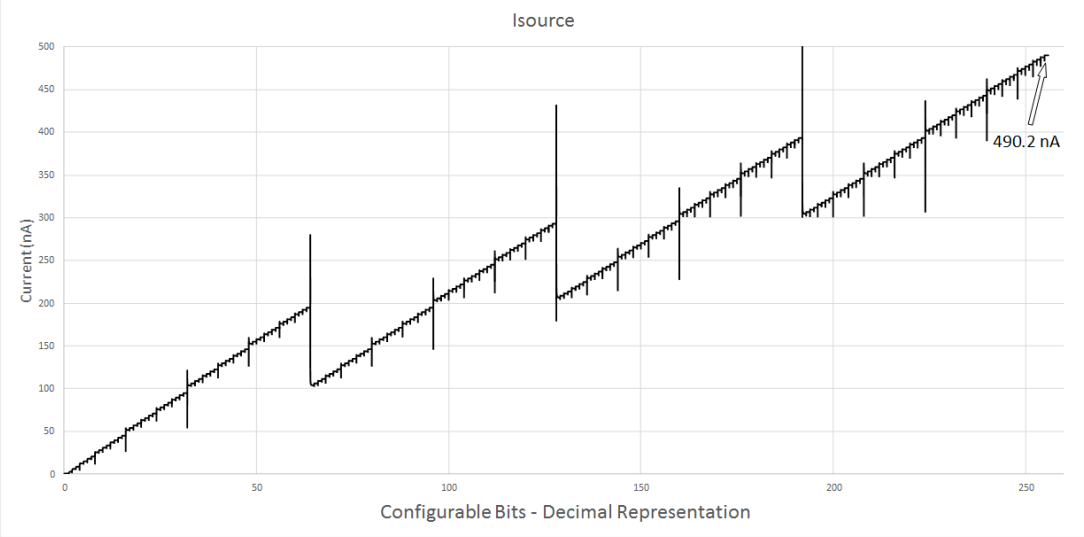


Figure 3.3: Configurable Current Mirror Source Simulation

3.1.4 Configurable Current Source Layout

The layout for the configurable current mirror was challenging. All of the transistors in a particular current branch need to be matched with each other. Also the PMOS transistors need to be matched among each other. An important layout variation that needs to be accounted for is called gradients [5]. A gradient runs any direction across a chip and produce small property changes such as: oxide thickness, carrier mobilities, and threshold voltages. Gradient-induced mismatch can be minimized by reducing the distance between the centroids of matched devices [5]. A centroid refers to the center point of a transistors overall area. In order to ensure matching, all the NMOS transistors are grouped together at the bottom of the layout in figure 3.4. Branches from ground are inserted in between each row of NMOS transistors to create ideal conditions for the substrates of these devices. The PMOS transistors are similarly matched in the top left corner, and their substrate is well connected to VDD. The switches and inverter are grouped to the right of the PMOS devices, and the entire design is enclosed in a guard ring for isolation from other blocks. The layout area was 2000 μm^2.

3.2 Low-Pass Filter and Integrator

The low-pass filter (LPF) is designed using the translinear principle, which was introduced in section 2.3. Translinear circuits are ideal for low-power, low-noise designs. The low-noise advantage is realized through current-mode circuits. The noise contribution at low frequencies are flicker or 1/f noise, and at higher frequencies, white noise takes over but is still very minimal. Overall, the thermal noise contributions to the translinear circuit are much lower than in voltage mode designs. Since the transistors operate in the subthreshold region, low-power is easily achieved by keeping the voltage rail low. In the area of neurological research, low-pass filters are important because high frequency noise can interfere with the intended signals from a brain.

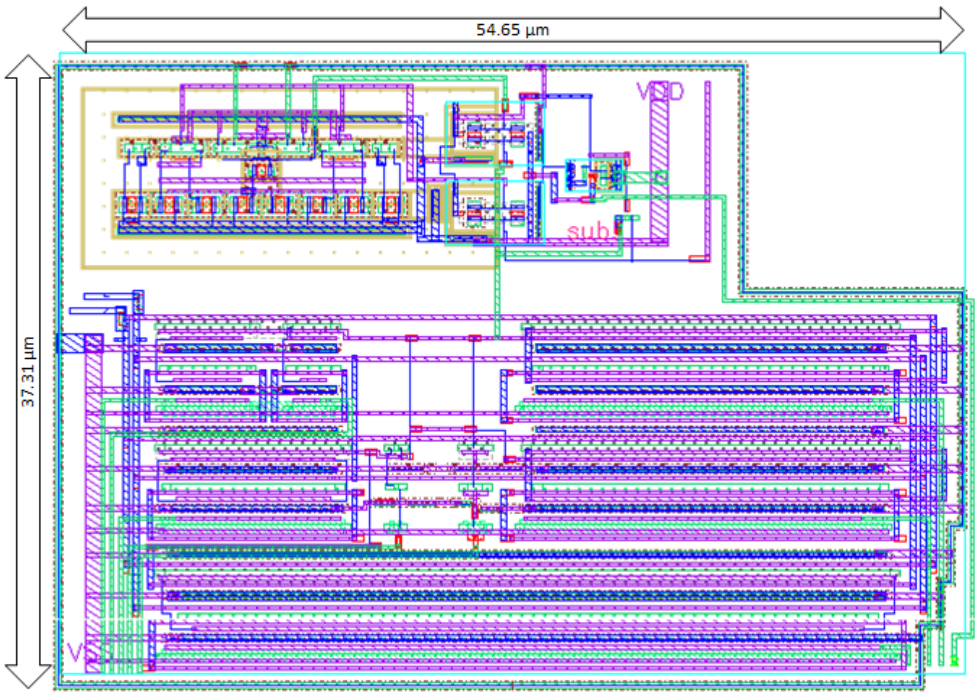


Figure 3.4: Configurable Current Source Layout

3.2.1 LPF Design and Mathematical Derivation

The math for the LPF is cumbersome, but presented below is the method that led to the final LPF circuit. First, an important inverting translinear structures is shown in figure 3.5, [6]. The structure relationships are described by the equation below it. These relationships will come into play when synthesizing a LPF using math. The following image and derivation is based off a presentation by Bradley Minch in 2010, [6].

From a mathematical standpoint, equation 3.1 represents a first order LPF. Using a ratio of signal current to unit current, an ordinary differential equation can be realized in equation 3.2. In this equation, "y" is the input, and "x" is the output.

$$x = y + \tau \frac{dy}{dt} \tag{3.1}$$

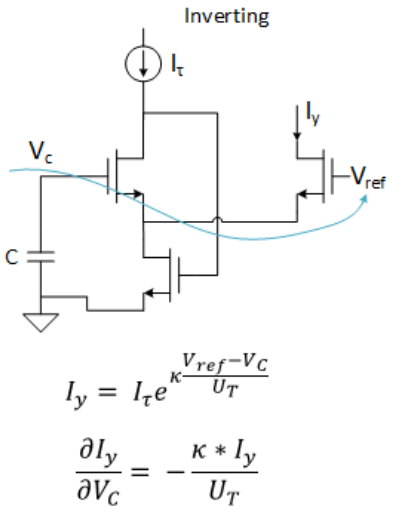


Figure 3.5: Output Translinear Structures

$$\frac{I_x}{I_1} = \frac{I_y}{I_1} + \tau \left(\frac{d}{dt}\right)\left(\frac{I_y}{I_1}\right) \quad ==> I_x = I_y + \tau \frac{dI_y}{dt} \tag{3.2}$$

Next a log-compressed voltage state variable, V_y, is introduced giving equation 3.3. From the inverting structure relationship and dividing by I_y, equation 3.3 will create equation 3.4.

$$I_x = I_y + \tau \left(\frac{dI_y}{dV_y}\right)\frac{dV_y}{dt} \tag{3.3}$$

$$\frac{I_x}{I_y} = 1 + \left(\frac{-\kappa \times \tau}{U_t}\right)\frac{dV_y}{dt} \tag{3.4}$$

By multiplying the last term in equation 3.4 by C/C to introduce capacitance into the equations, equation 3.5 is created. Two relationships lead to equation 3.6. The first is the relationship for a capacitor's current to voltage, $I_c = C\frac{dV}{dt}$. The second is the relationship is the time constant, τ, to the current, I_τ, described by $\tau = \frac{C \times U_T}{I_\tau \times \kappa}$. Multiplying through by I_τ gives equation 3.7.

$$\frac{I_x}{I_y} = 1 + \left(\frac{-k \times \tau}{C \times U_t}\right)\frac{C \times dV_y}{dt} \tag{3.5}$$

$$\frac{I_x}{I_y} = 1 - \left(\frac{1}{I_\tau}\right) I_c \tag{3.6}$$

$$I_\tau - I_c = \frac{I_x \times I_\tau}{I_y} = I_p \tag{3.7}$$

In section 2.3, translinear loops were introduced. Now those loops need to be further explained. According to the translinear principle (TLP), the products of clockwise and counterclockwise translinear element's currents inside a closed loop are equal, [3]. Instead of BJT's, the translinear elements are MOSFETs biased in the subthreshold region. From the TLP, the currents in figure 3.6 can be defined by I_x*I_τ = I_y*I_p.

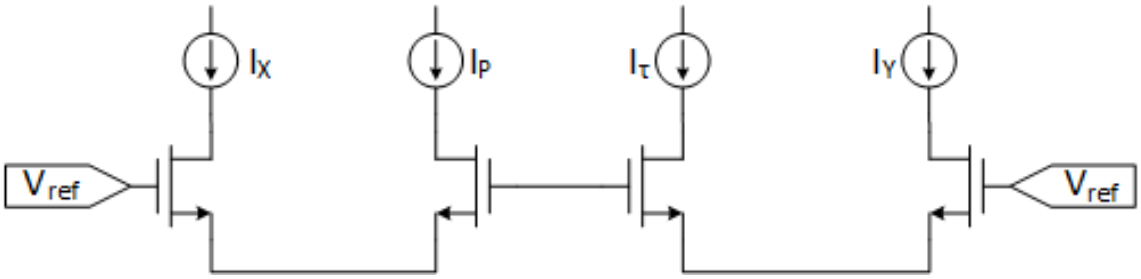


Figure 3.6: Translinear Loop

Next, the final equation, 3.7, needs to be accomplished by using the inverting structure in figure 3.5. With the addition of a current mirror to provide biasing I_τ, the final schematic for the LPF created in figure 3.7. Using Kirchhoff's current law, equation 3.7 can be seen as: $I_\tau = I_c + I_p$. Also, the output needed to source current, not sink current, in order to work with subsequent stages, so the addition of a PMOS current mirror was added to the output. The current provided by I_{bias} is the I_τ current from the mathematical derivation. This lengthy derivation creates a LPF from the original LPF transfer equation seen in equation 3.2. The next subsection will describe the simulations of the LPF.

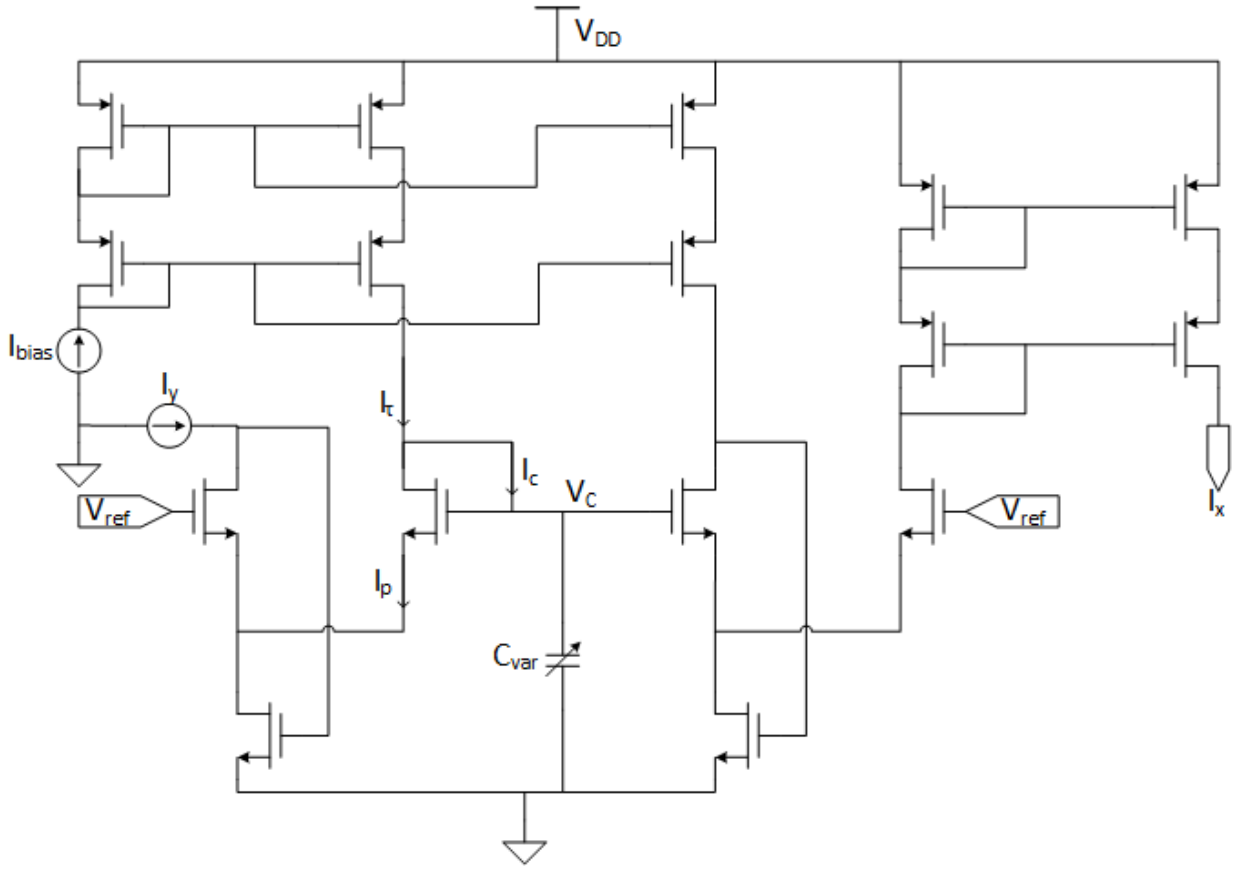


Figure 3.7: Translinear Low Pass Filter Circuit

3.2.2 LPF Simulation

All simulations were completed using cadence 6.1.5. The first simulation tests the different configurations of the LPF. This LPF is configurable by two different methods. The biasing current can be varied on a binary scale with eight bits of resolution from 3 nA to 490 nA using a configurable current source presented in section 3.1. C_{var} can also be changed between four different capacitors with values equal to: 750 fF, 5 pF, 20 pF and 90 pF. The capacitors add for a maximum capacitance of 115.75 pF. The current configuration allows the low-pass filter to have a multitude of corner frequencies for each capacitance value tabulated in table 3.1. The tabulated values are from simulations and show that the LPF has a wide tuning range reaching from 125 Hz to 1.5 MHz. For these simulations the output was sent through a diode connected NMOS transistor; this output matches the impedance seen if the output is sent to another processing block. All of the AC simulations for each high and low corner frequency are depicted together in figure 3.8. The simulated power consumption with the lowest and highest bias current is 110.8 nW and 1.96 μW, respectively. This power consumption weighs heavily on the input bias current provided to the LPF because the output signal is sent through a PMOS current mirror. The frequency range and power consumption are acceptable for low-power neurological signal processing.

Another important simulation for this LPF to test the dynamic range. For this test setup, the LPF is in the following configuration: the bias current is set to 366 nA and the all of the capacitors were on adding up to 115.75 pF of capacitance. This setup makes the corner frequency at 10.5 kHz. First, a noise measurement needs to be made. The noise simulation was run in Cadence with the input current source set as the input noise and a 0 V voltage source on I_{out} set as the output noise. The simulation was run from 1 Hz to 1 MHz with a logarithmic scale and 30 points per decade. The output noise waveform is in $A/\sqrt{Hz}$, shown in figure 3.9. This resulting output noise waveform was then squared and integrated from 100 Hz to 100 kHz with respect to frequency and the square root was taken to come up with an output

integrated noise number in $A_{RMS,noise}$ units. Next, the output integrated noise needs to be divided by the midband gain of the LPF, which is -0.515 mdB. Then a transient simulation is run and the total harmonic distortion, (THD), is calculated using the cadence calculator. The input offset current is set to 300 nA, frequency set to 1 kHz, and the amplitude is varied until the THD equals 1%. This amplitude, 206 nA_{PP}, is divided by $\sqrt{2}$ to give the amplitude $A_{RMS,input}$ units. Finally, as seen in equation 3.8, the dynamic range is calculated to be 58 dB.

$$DR = 20 \times \log(\frac{A_{RMS,input}}{A_{RMS,noise}}) = 20 \times \log(\frac{145\ nA_{RMS}}{180\ pA_{RMS}}) = 58 \tag{3.8}$$

3.2.3 LPF Layout

Analog processing requires many layout techniques to insure proper matching among transistors. For the LPF, the transistors in the translinear loop, Q1-Q4 in figure 3.7, need to be matched as accurately as possible. Also the current mirrors for biasing I_τ and the output stage need to be matched, respectively. Any mismatch can lead to gain errors due to channel-length modulation. Channel-length modulation, which is a shortening of transistor channel length, between two transistors that can create DC current offsets in the path among two matched transistors. In order to mitigate mismatch, techniques from [5] were implemented. For transistors Q1-Q4, the multiplicity was set to two, meaning two transistors hooked up in parallel equaled one transistor. Then the devices were interdigitated in an DCBAABCD pattern with dummy transistors added on the outsides. Q5-Q6, Q7-Q9, Q10-Q12, Q13 and Q15, and Q14 and Q16 were also interdigitated using similar patterns with dummy devices. The layout is presented in figure 3.10. The addition of the capacitors and configurable current source makes the layout sixteen times larger as seen in figure 3.11. The LPF core layout has and area of 1,600 μm^2 and the full layout has an area of 49,250 μm^2.

Table 3.1: LPF Corner Frequencies

Capacitance	Low Corner	High Corner
750 fF	11.83 kHz	1.52 MHz
5 pF	2.53 kHz	334.8 kHz
20 pF	671 Hz	87.81 kHz
90 pF	152 Hz	19.58 kHz
115.75 pF	125 Hz	15.6 kHz

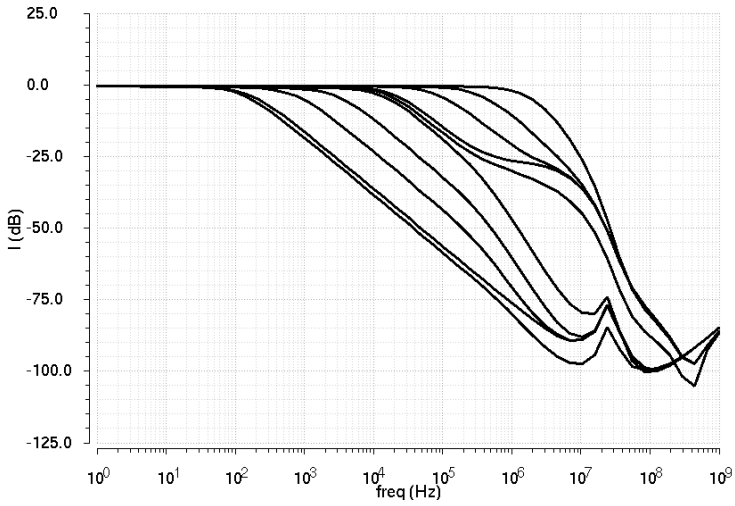
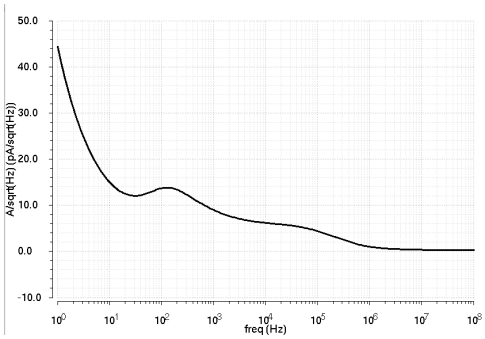
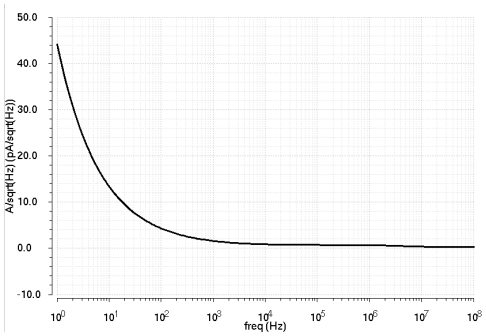


Figure 3.8: LPF Corner Frequency Simulation



(a) LPF Low Corner Noise



(b) LPF High Corner Noise

Figure 3.9: LPF Noise Simulations

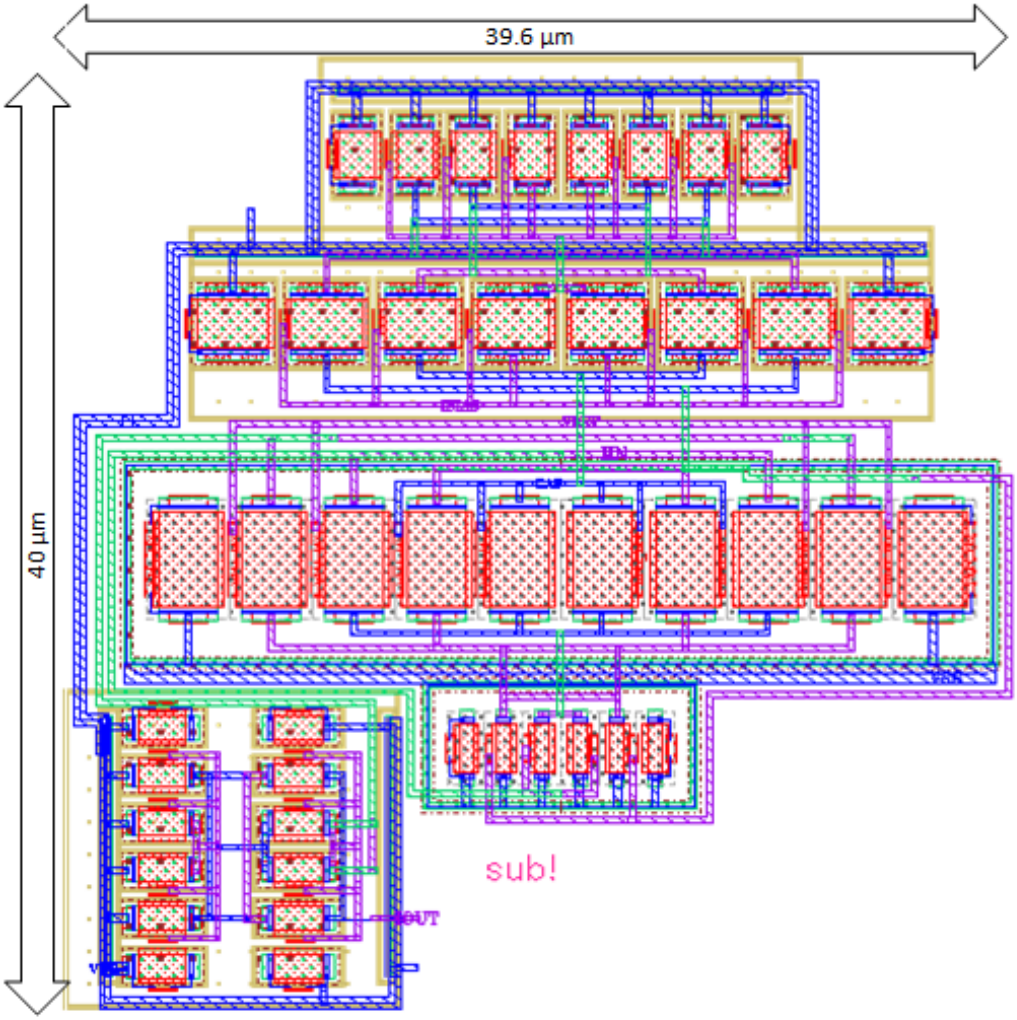


Figure 3.10: LPF Layout

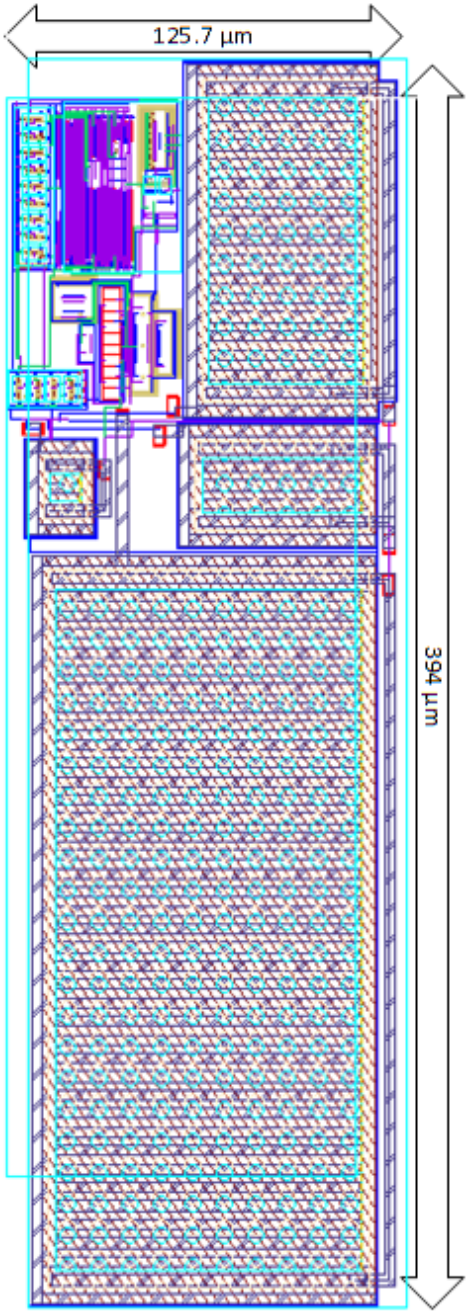


Figure 3.11: LPF Layout

3.2.4 Integrator Design

An integrator is a continuous analog counter accumulating the input into the output. A current-mode integrator performs time integration of an electric current. Therefore, the output is the total charge accumulated from the input. This function can be completed using the LPF presented in section 3.2.1 with an offset current. The one change from the LPF is that the capacitance for the integrator is set to 80 pF. The final integrator schematic is shown in figure 3.12. This large capacitance keeps the corner frequency low. In order to achieve integration, the signal frequency needs to be at least four times greater than the corner frequency. Frequencies below or near the corner frequency will pass straight through. Signals with frequencies greater than the corner frequency will experience a lag time in charging the capacitor that is reflected to the output.

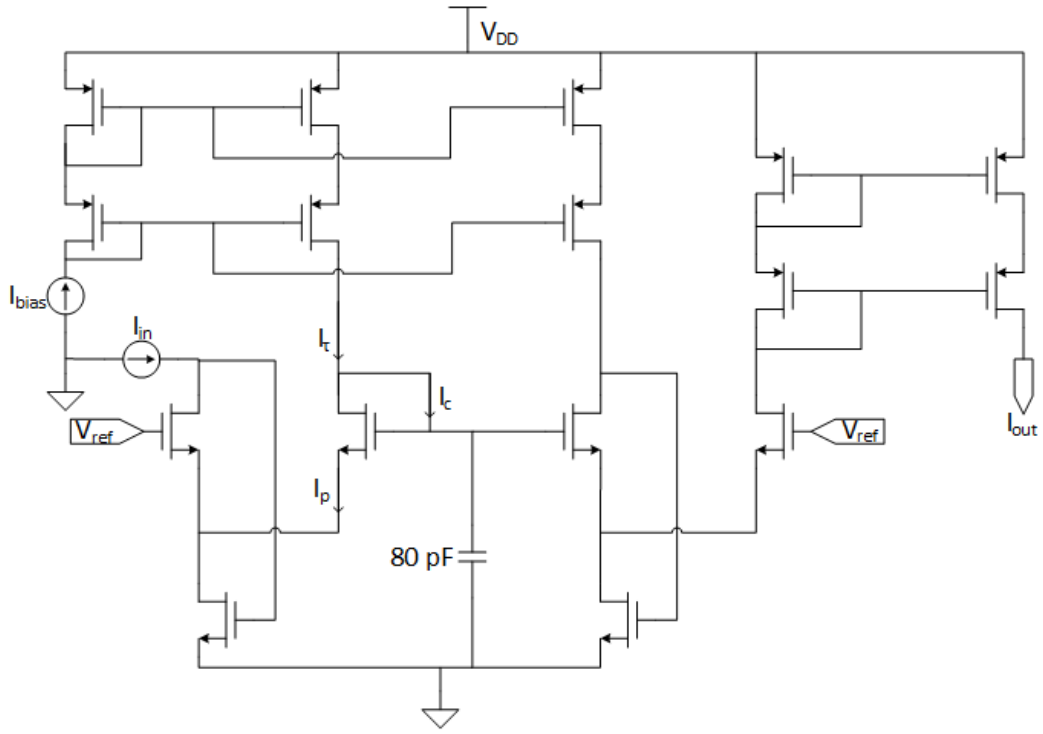


Figure 3.12: Integrator Schematic

3.2.5 Integrator Simulations

The integrator simulations include an AC and transient analysis. The noise analysis of this integrator closely follows the analysis of the LPF presented in section 3.2.2. The biasing current for the integrator is provided using the previously discussed configurable current source. This allows the corner frequency of the integrator to be varied from 49 Hz to 7.58 kHz. Figure 3.13 shows an AC simulation when a single branch of the configurable current source is on at a time.

Next two transient analyses were run to show the integration of a sine wave and a square wave. The integration of a sine wave is a cosine wave, and the integration of a square wave is a triangular wave shown in figures 3.14 and 3.15, respectively. For these transient simulation, the cutoff frequency was set to 2 kHz, the DC input was 200 nA, the input amplitude was 25 nA, and frequency was set to 8 kHz. Each figure correctly represent an integrating function. The simulated power consumption with the lowest and highest bias current is 56.27 nW and 987.5 nW, respectively. This power is also based on the input bias current to the integrator.

3.2.6 Integrator Layout

The full integrator layout is shown in figure 3.16 and consumes 30,390 μm^2. On-chip capacitors consume massive amounts of area. Different designs could create an integrator with smaller on chip capacitance.

3.3 High-Pass Filter and Differentiator

A high-pass filter (HPF) in combination with a LPF can create a bandpass filter. Both high-pass filtering and bandpass filtering can be very useful in signal processing. When trying to study brain waves at specific frequencies a bandpass filter would be ideal. Also a high pass filter can eliminate any low frequency interference.

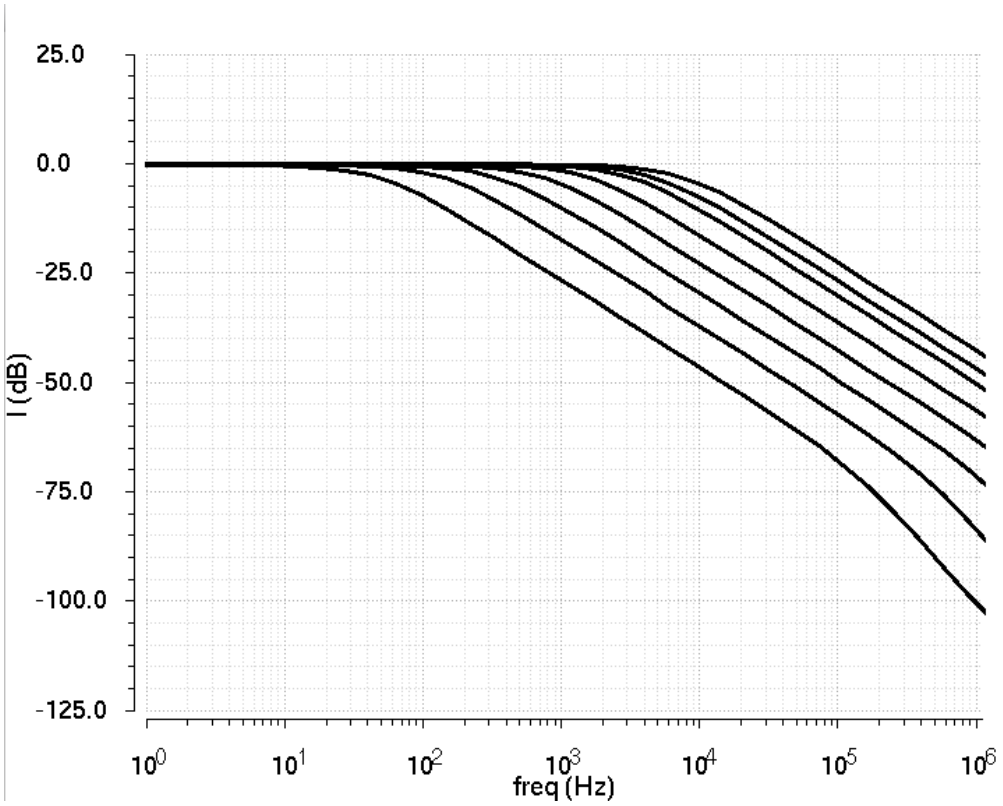


Figure 3.13: Integrator Corner Frequencies

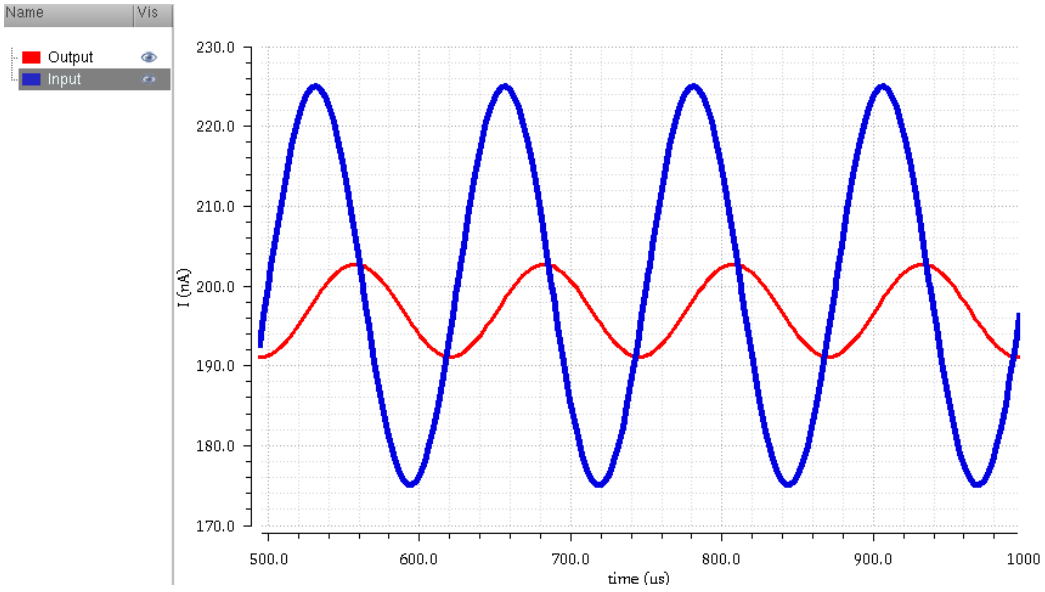


Figure 3.14: Integrator Transient Response to Sine Wave

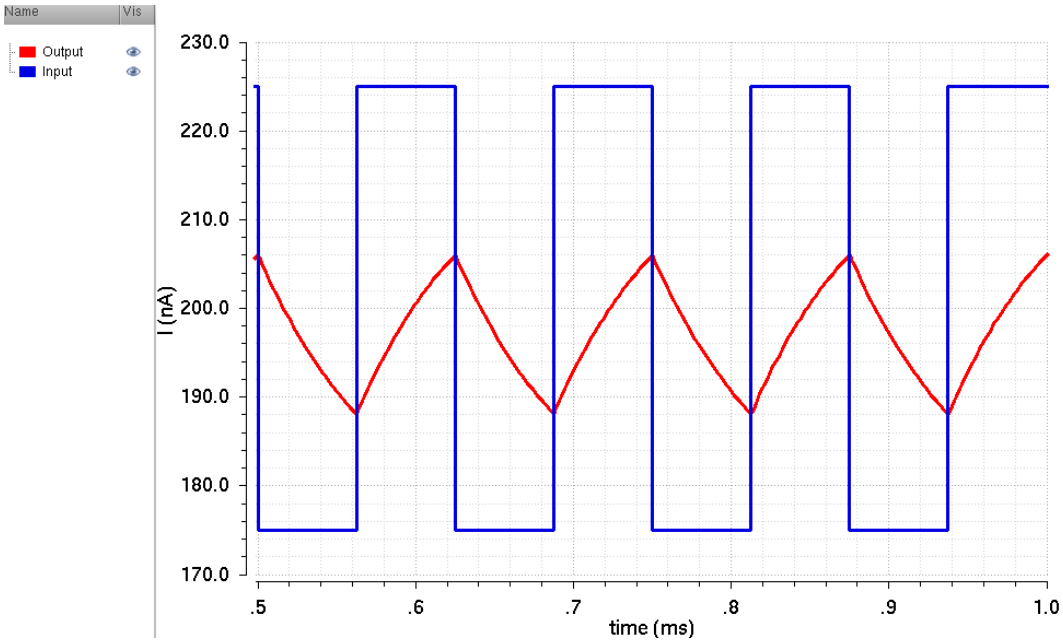


Figure 3.15: Integrator Transient Response to Square Wave

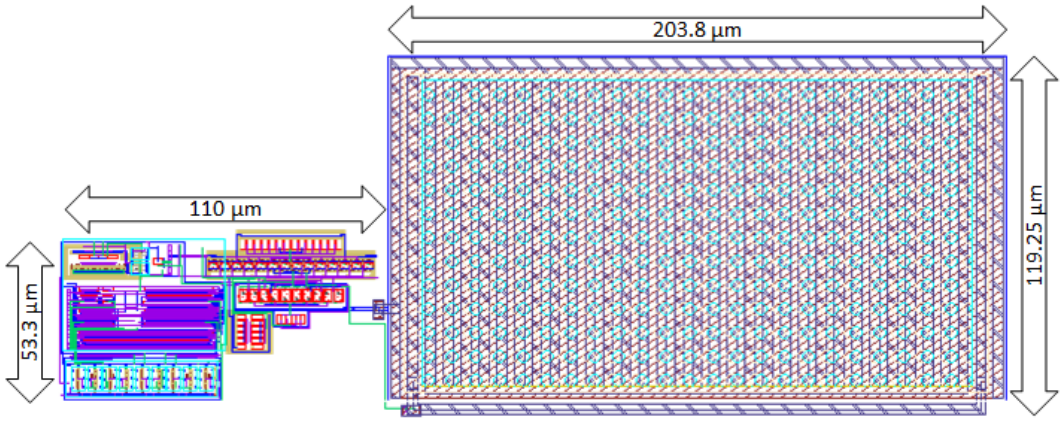


Figure 3.16: Integrator Layout

3.3.1 HPF Design

This HPF design is based off the LPF translinear circuit proposed in section 3.2. The technique used to create the HPF can be explained using transfer functions. The following set of equations will prove that a LPF signal subtracted from the original signal will produce a HPF. The transfer function for a LPF can be described by equation 3.9, where I_{x1} is the input and I_{y1} is the output. An all pass filter transfer function can be described by equation 3.10, where I_{x2} is the input and I_{y2} is the output.

$$\frac{I_{y1}}{I_{x1}} = \frac{1}{1+sC} \tag{3.9}$$

$$\frac{I_{y2}}{I_{x2}} = 1 \tag{3.10}$$

Then the output to the LPF transfer function is subtracted from the output of the all-pass filter transfer function resulting in ΔI_y shown in equation 3.11. Since the same input is being used for each transfer function, $I_{x1} = I_{x2} = I_x$, which leads to an end result shown in equation 3.12.

$$\Delta I_y = I_{y2} - I_{y1} = I_{x2} - \frac{I_{x1}}{1+sC} \tag{3.11}$$

$$\Delta I_y = I_x - \frac{I_x}{1+sC} = \frac{sCI_x}{1+sC} \tag{3.12}$$

The final result is the transfer equation for a HPF. As frequency approaches infinity, sC = 1 + sC and $\Delta I_y = I_x$, and as frequency approaches zero, $\Delta I_y = 0/1 = 0$. Following this logic, an additional current mirror was added to the LPF schematic in figure 3.7 to create the HPF in figure 3.17. This filter was design with assistance from Tan Yang, part of the Integrated Silicon Systems group at the University of Tennessee..

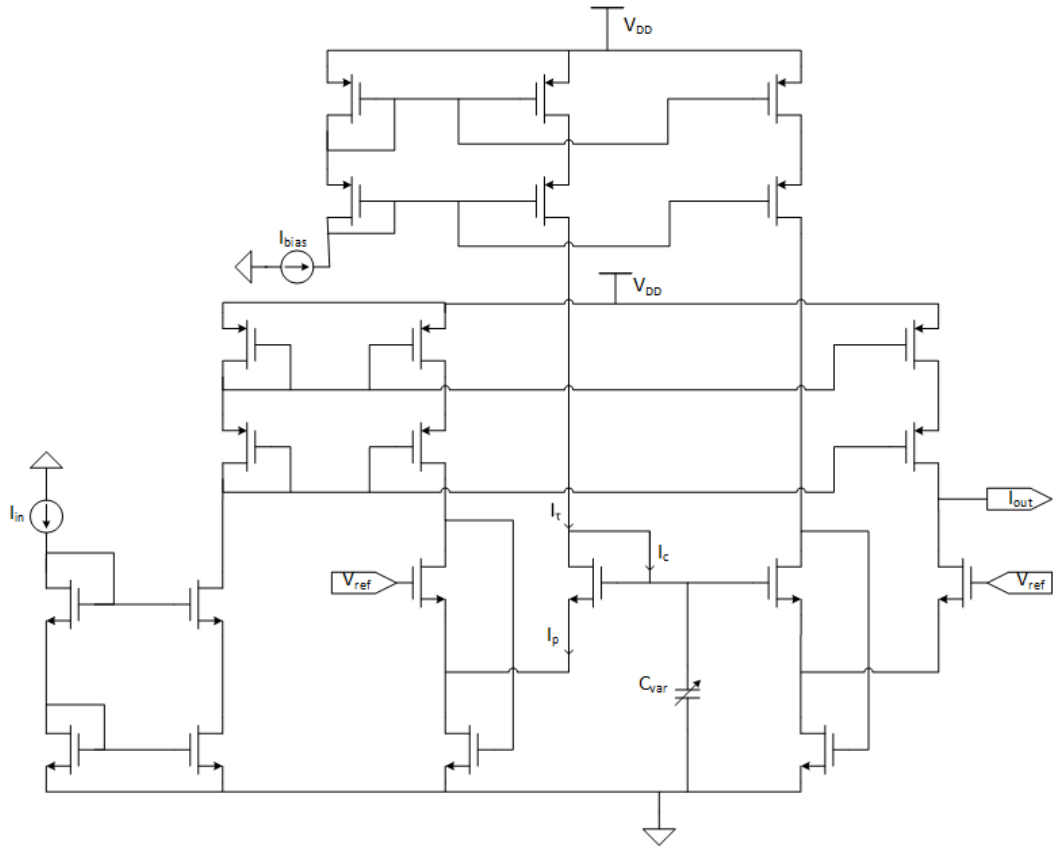


Figure 3.17: HPF Schematic

3.3.2 HPF Simulation

The HPF is configurable by changing the biasing current using the configurable current source and by changing the capacitance. The C_{var} in the HPF contains the following capacitances: 5.3 pF, 10.6 pF, 21.3 pF, and 42.6 pF. Any combination of capacitors can be turned on making the total possible capacitance nearly 80 pF. The output signal is the input signal minus the LPF output, and therefore the low-frequency current will be subtracted from the output. In order to compensate an additional current source was added to the output of the HPF to restore the DC bias level. For simulations, the output is linked to a diode connected NMOS transistor. In order to bias the output NMOS transistor for simulations, the output current source is configured to produce 44 nA of current. Table 3.2 lists the high and low corner frequencies for each capacitance in the HPF. The AC responses for these

corner frequencies can be seen in figure 3.18. The HPF frequencies above 2 MHz are attenuated, so the AC simulation is only run from 1 Hz to 1.5 MHz. This effect is due to the nature of the biasing of these transistors and their switching speeds in the subthreshold region of operation. The power consumption across the range of frequency presented spans from 761 nA to 2.61 μW.

Next a noise and transient simulations were carried out to find the dynamic range of the HPF. The test setup for the HPF has the following parameters: bias current is set to 53.9 nA, all the capacitors were on, and the output bias current is set to 284 nA. This setup creates a corner frequency of 2.85 kHz. The same noise measurement as the one in section 3.2.2 is used for the HPF noise simulation. The output noise was squared, integrated from 100 Hz to 500 kHz with respect to frequency, and the square root is taken to find the output integrated noise in A_{RMS}. Then the output integrated noise is divided by the midband gain of -0.201 dB. The final output integrated noise level is 1.192 nA $_{RMS}$. Now a transient response is observed in order to find an amplitude leading to a THD of 1%. The transient input is a sine wave with a 300 nA DC offset at 50 kHz. The amplitude that makes the THD equal to 1% is 280 nA. The THD scales highly with the input bias current as long as the output bias current is set to equal magnitudes. Using the numbers above a dynamic range equation, 3.13, is presented.

$$DR = 20 \times \log(\frac{A_{RMS,input}}{A_{RMS,noie}}) = 20 \times \log(\frac{396\ nA_{RMS}}{1.22\ nA_{RMS}}) = 50.2 \tag{3.13}$$

3.3.3 HPF Layout

The layout for the HPF consists of the LPF layout with an additional PMOS current mirror for the input signal. Again an interdigitated layout method is used. The final HPF core is shown in figure 3.19. The entire core is only 1,760 μm^2. Figure 3.20

Table 3.2: HPF Corner Frequencies

Capacitance	Low Corner	High Corner
5.3 pF	2.66 kHz	312.3 kHz
10.6 pF	1.36 kHz	163.4 kHz
21.3 pF	690.8 Hz	86.24 kHz
42.6 pF	352 Hz	44.22 kHz
79.8 pF	183.7 Hz	23.27 kHz

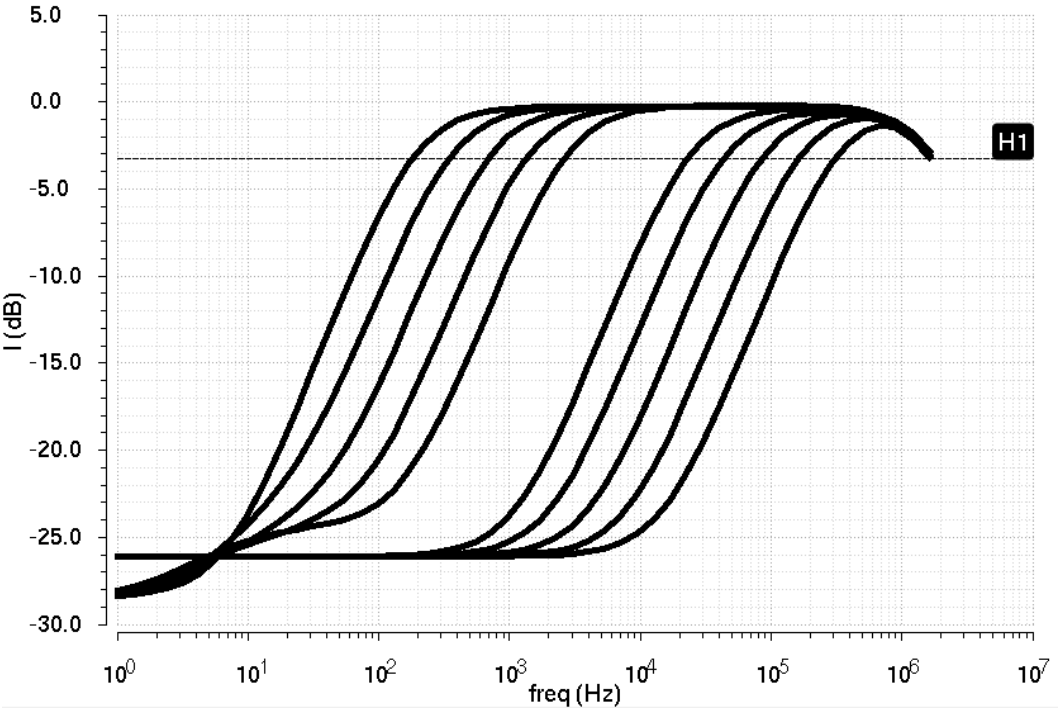


Figure 3.18: HPF Corner Frequency Simulation

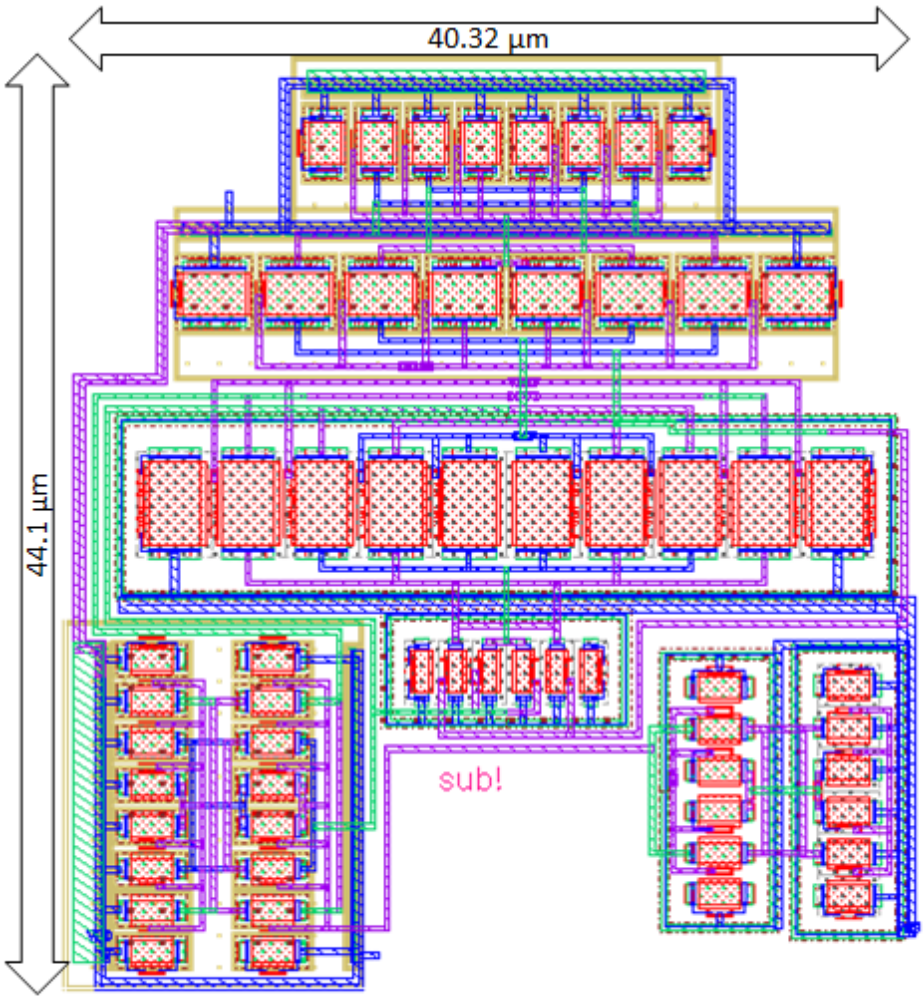


Figure 3.19: HPF Core Layout

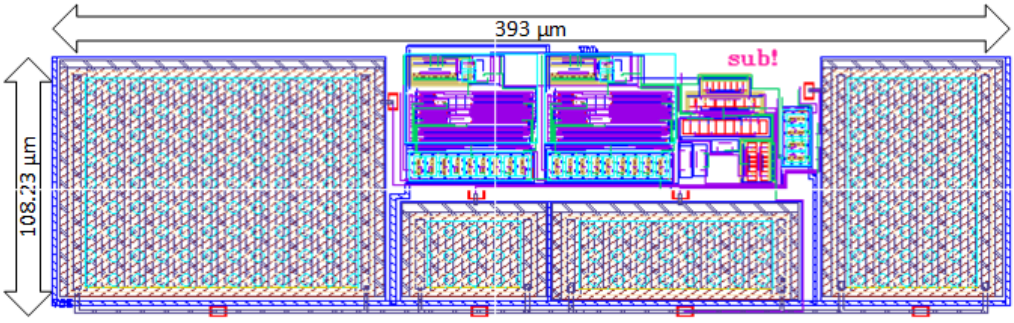


Figure 3.20: HPF Full Layout

depicts the entire HPF layout including the capacitors. This layout consumes an area of 42,728 μm^2.

3.3.4 Differentiator Design

The differentiator works very similarly to the integrator. The HPF is used with a set capacitor to create the differentiator. A differentiator will output the slope of the input signal. As with the integrator, the differentiator input needs to be much lower than the corner frequency. There will be attenuation in the output signal, but it will be the derivative of the input. The schematic is seen in 3.21.

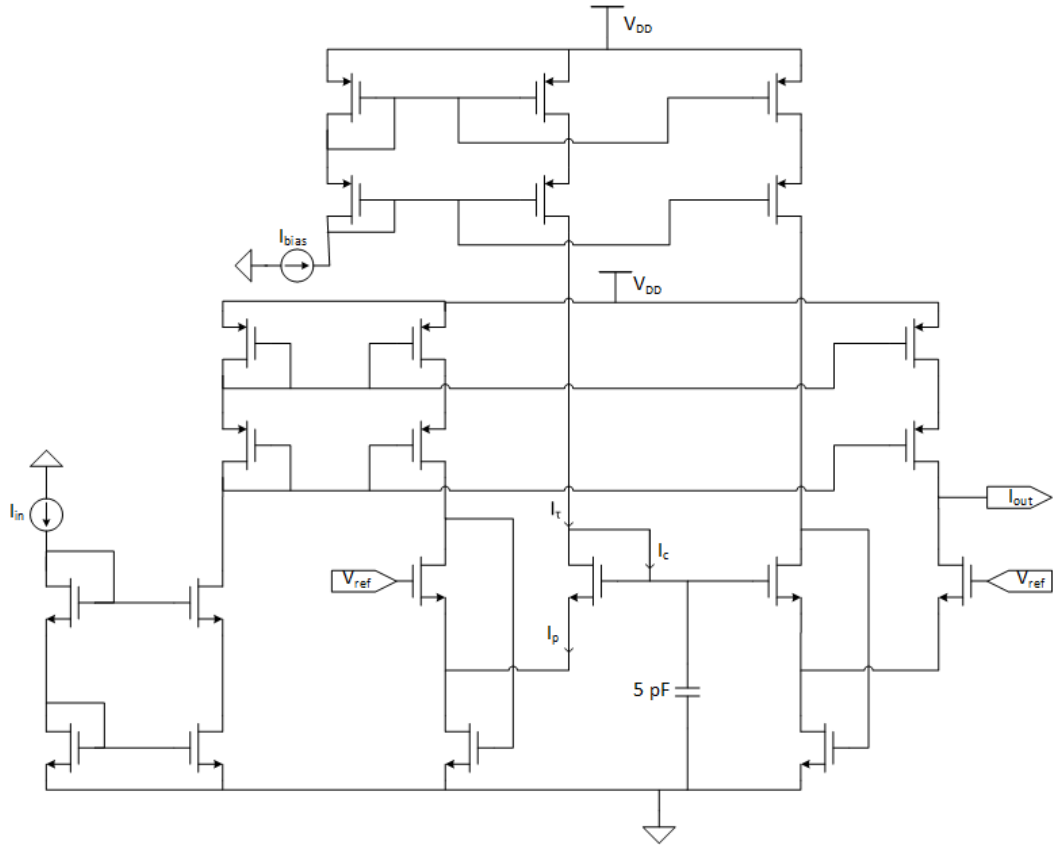


Figure 3.21: Differential AC Simulation

3.3.5 Differentiator Simulations

The differentiator is simulated for its corner frequency as well as the transient response. Configurable current sources are used for the bias current and on the output. The output current is set to 93 nA for this test. The AC response showing the extremes of the variable corner frequency is seen in figure 3.23. The max and min frequencies are 2.85 kHz and 250 kHz, respectively. A transient simulation will demonstrate the functionality of the differentiator. The input bias current is set to 24 nA, giving the differentiator a corner frequency of 20.3 kHz. The output bias current is set to 93 nA. Figure 3.24 shows the input sign wave and the output derivative of a sine wave, which is a cosine wave. The power consumption of the derivative ranges from 455.8 nW to 2.3 μW.

3.3.6 Differentiator Layout

The core layout is the same as the HPF core layout in figure 3.19. The full layout including the current sources is in figure 3.22 and spans over an area of 11,725 μm^2.

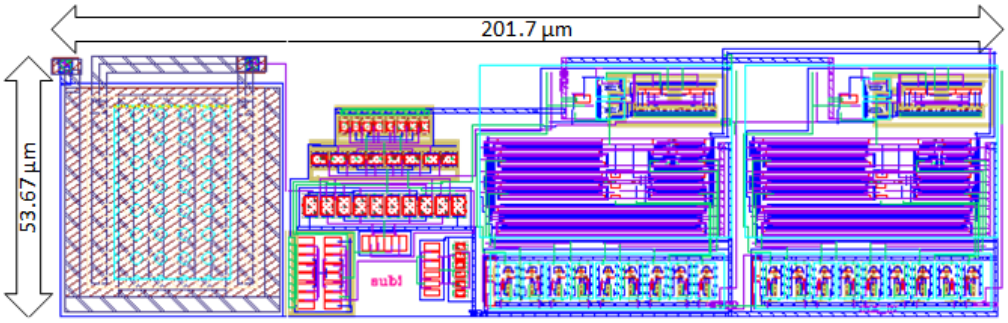


Figure 3.22: Differential Full Layout

3.4 Operational Transconductance Amplifier

An operational transconductance amplifier, (OTA), converts a differential voltage into a current. When processing brain signals, voltages signals are captured from

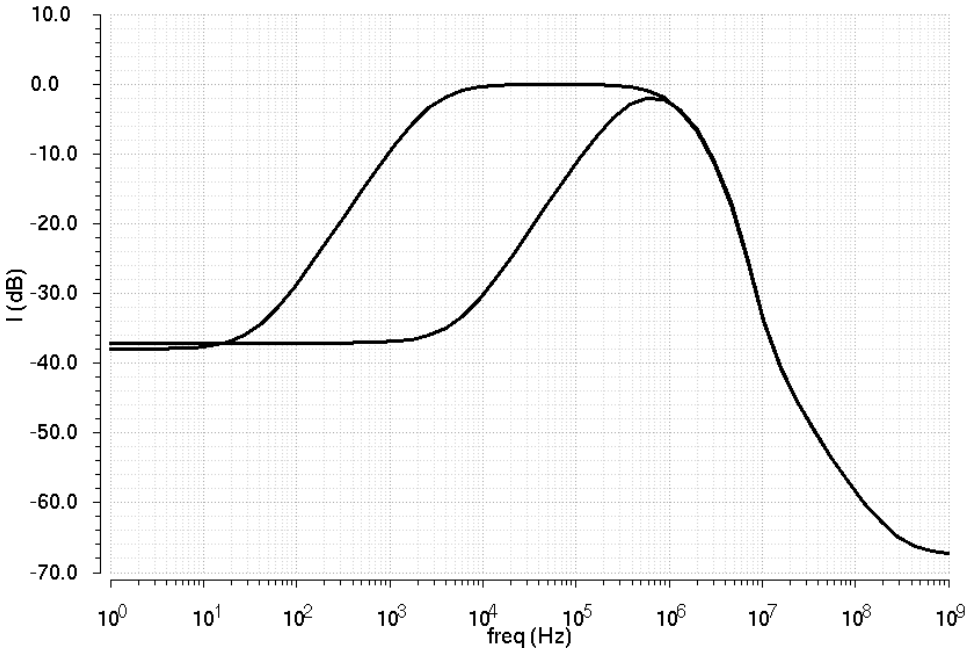


Figure 3.23: Differential AC Simulation

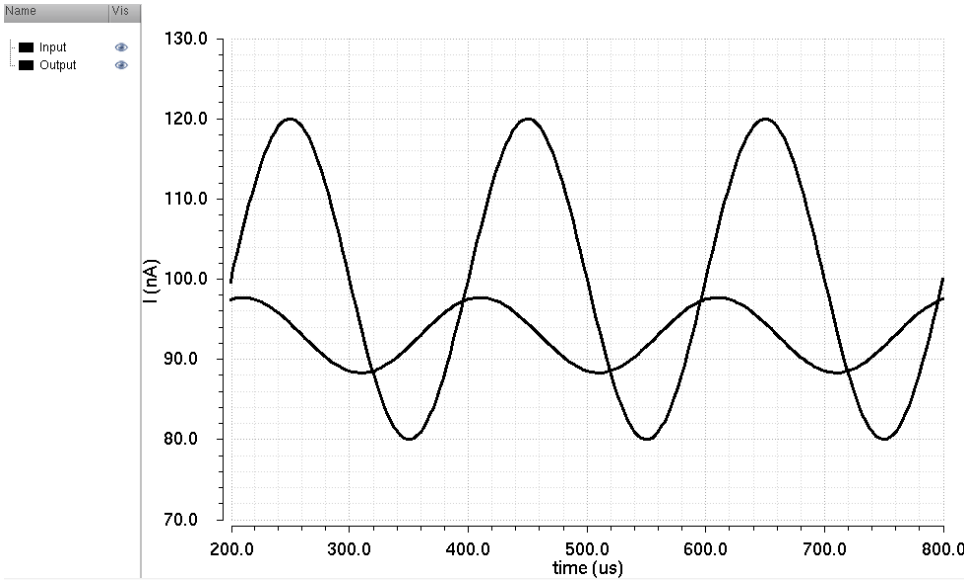


Figure 3.24: Differential Transient Simulation

brain waves. An OTA is crucial for conversion from voltage to current so that the signals can be processed with current-mode signal processors. An on chip OTA can also provide an easy way to test other processing blocks using a voltage waveform generator. The OTA selected for this analog signal process was designed by Tan Yang.

3.4.1 OTA Design

This OTA operates in the subthreshold region like the other processing blocks that are a part of this analog signal processor. The schematic is presented in figure 3.25. The differential input signal, ΔV_{in}, is connected to the gates of Q1 and Q2; each gate gets $\Delta V_{in}/2$. The small signal current through Q1 and Q2 are described by equation 3.14, where the currents are of equal magnitude and opposite direction. Once the currents are mirrored to the output, the small signal output voltage can be derived as equation 3.15, where r_{op} and r_{on} are the small signal resistances for transistors Q5 and Q7, respectively. Plugging equation 3.14 into equation 3.15 and dividing by the input voltage ΔV, the OTA gain is found and presented in equation 3.16. The bias current determines the transconductance, g_m, value and allows for variable gain.

$$i_d = g_m \times (\frac{\Delta V}{2}) \tag{3.14}$$

$$V_{out} = -2 \times i_d \times (r_{op}||r_{on}) \tag{3.15}$$

$$A_v = \frac{V_o}{V_i} = -g_m \times (r_{op}||r_{on}) \tag{3.16}$$

3.4.2 OTA Simulations

An AC simulation is run to find the open loop gain, corner frequency, and calculate the gain bandwidth product, (GBP) of the OTA. For the AC simulation the bias

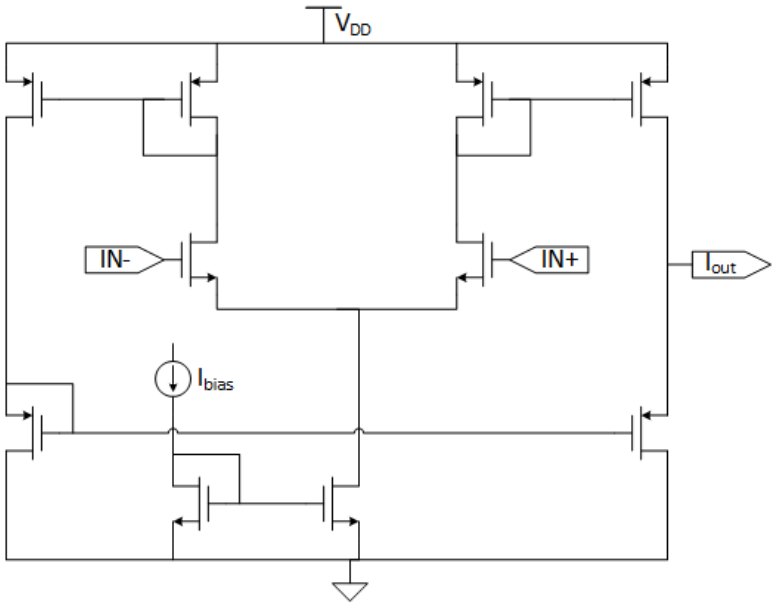


Figure 3.25: OTA schematic

current is set with a configurable current source block to both extremes, 2.82 nA and 490.7 nA. The negative terminal is connected to a voltage source with AC set to zero and 500 mV DC. The positive input is given an AC magnitude of 1 and a DC voltage of 500 mV. Figure 3.26 is obtained by sweeping the frequency at the highest and lowest bias currents. The highest and lowest corner frequencies, f_{-3dB}, are 636.9 kHz and 11.17 kHz, respectively. The gains for each corner are 24.3 dB and 21.1 dB, respectively. From these values, the GBP can be calculated using equation 3.17. The GBP spans from 10.44 MHz to 126.78 kHz. The power consumption for these simulations ranges from 19.56 nW to 2.72 μW.

A transient simulation was run to observe the response to a sine wave input voltage. The "IN-" node was set to 500 mV, and the "IN+" node was set to a sine wave with a DC bias of 500 mV and an AC magnitude of 40 mV. The bias current is set to 94.7 nA. The inputs and outputs are shown in figure 3.27. The resulting output is a current sine wave with DC offset of 76.5 nA and an AC magnitude of 78 nA_{pp}.

$$GBP = A_v \times f_{-3dB} \tag{3.17}$$

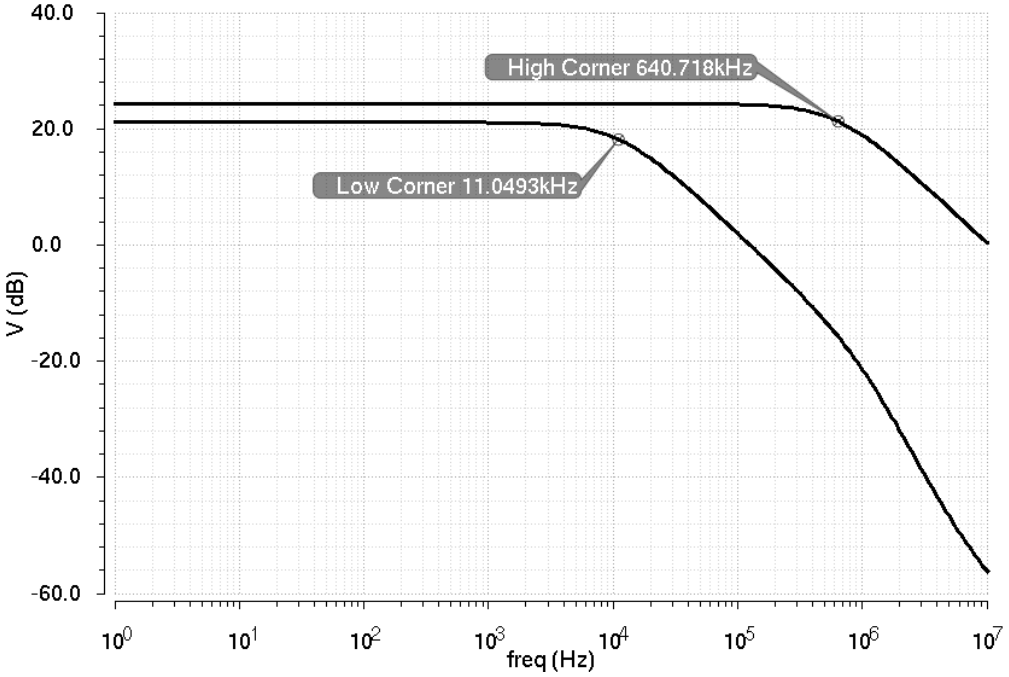


Figure 3.26: OTA AC Simulation

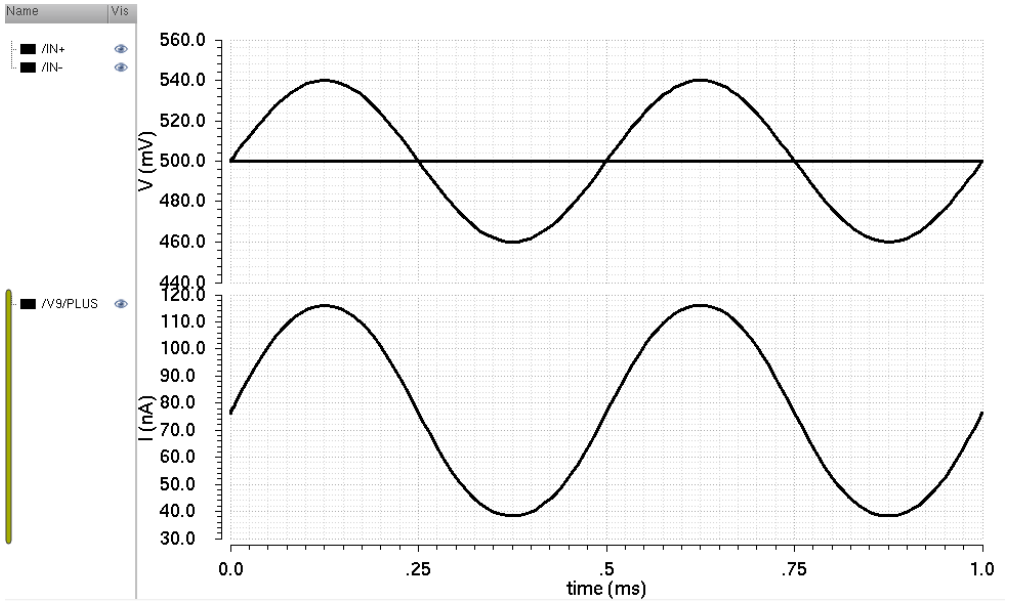


Figure 3.27: OTA Transient Simulation

3.4.3 OTA Layout

The OTA core layout was completed by Tan Yang. The layout is presented in figure 3.28, and area for this layout is 310 μm^2. The addition to the OTA core was a configurable current source resulting in figure 3.29 with an area of 3,437 μm^2. The OTA was implemented into the analog signal processor in two ways: one block allows control of V_{in+} and V_{in-}, and the other block allows control of V_{in+} and I_{bias}, where V_{in-} is set to the reference voltage, V_{ref}. This will be further explained in section 3.9.

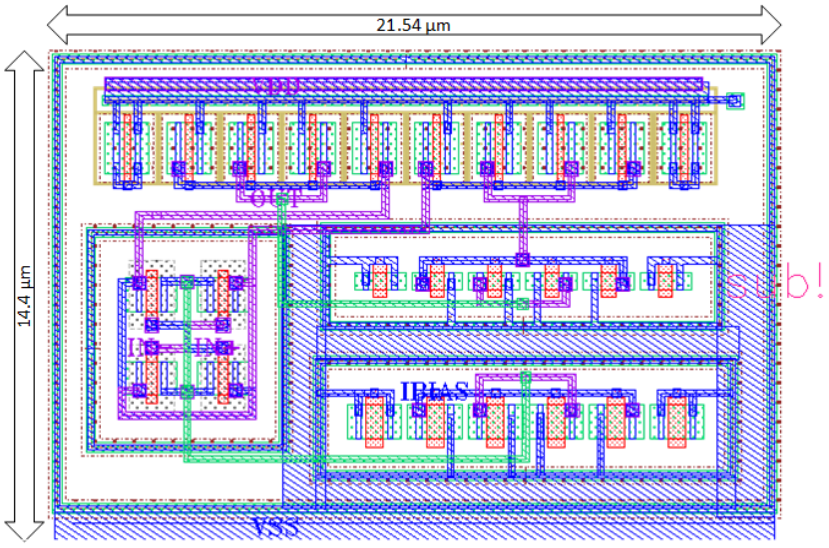


Figure 3.28: OTA Core Layout

3.5 Multiplier

Multipliers have many applications in the analog signal processing domain. They are useful in modulators, nonlinear filtering, programmable-gain amplifiers, root-mean-square converters, etc. Analog multipliers can achieve high resolution with minimal power compared to digital multipliers that can achieve greater resolution with significantly more power. The application this multiplier was intended for was a frequency compression circuit that will be discussed in section 3.9. The multiplier is

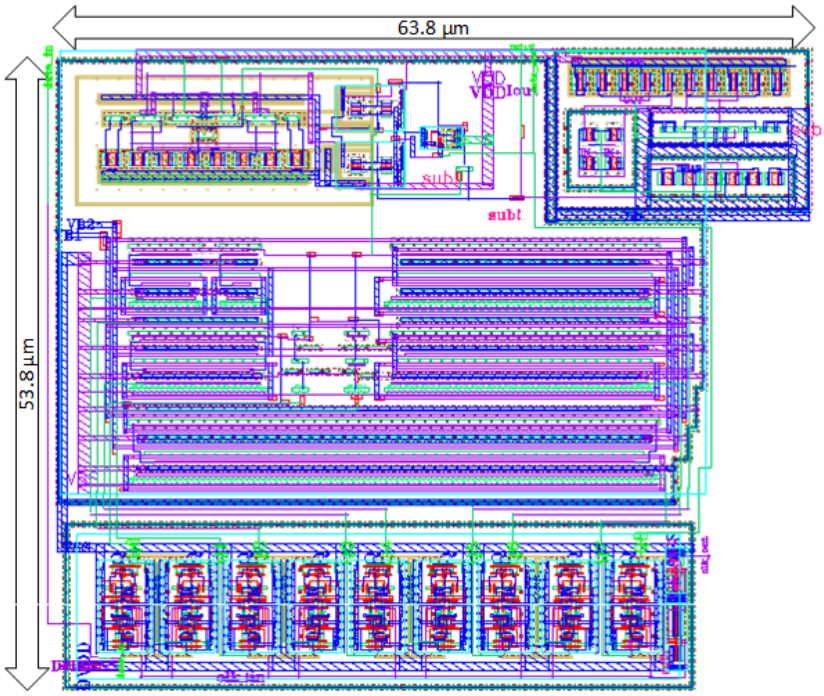


Figure 3.29: OTA Full Layout

designed using subthreshold transistors configured in a translinear loop. The design and optimization was completed by Tan Yang as well.

3.5.1 Multiplier Design

The multiplier schematic is presented in figure 3.30. There are two identical translinear loops in this design. The current mirror with an input labeled Ibias, provides the currents I_c. The translinear loop is described in equation 3.18. This loop equation can also describe the relationship between Q8-Q11. I_{y+} is then mirrored across to the output where I_{y-} is subtracted from it. The resulting current is sent to the output and described by equation 3.19. Because of the current subtraction, an additional configurable current source was placed on the output to provide sufficient DC current for the output. There are several inputs for this multiplier, so a simplification was made in the final design. The inputs I_{A-} and I_{B-} are tied to configurable current sources. The bias current is also provided by a current source.

This setup allows the subtracted signal to consist of purely DC levels. The inputs available for signals are I_{A+} and I_{B+}.

$$V_{gs,Q5} + V_{gs,Q6} = V_{gs,Q4} + V_{gs,Q7} \;\therefore\; I_{A+} \times I_{B+} = I_C \times I_{y+} \tag{3.18}$$

$$I_{y+} - I_{y-} = \frac{I_{A+} \times I_{B+}}{I_C} - \frac{I_{A-} \times I_{B-}}{I_C} \;\therefore\; I_{out} = \frac{I_{A+} \times I_{B+}}{I_C} - I_{DC} \tag{3.19}$$

3.5.2 Multiplier Simulations

Transient simulations were performed to show the functionality of the multiplier. The first simulation will multiply two sine waves at different frequencies. The inputs were set to the following configuration: I_{A-} is 9.6 nA_{DC}, I_{B-} is 8.1 nA_{DC}, I_{A+} is a sine wave equal to 80 nA_{DC} + 40 nA_{AC}, and I_{B+} is a sine wave equal to 40 nA_{DC} + 10 nA_{AC}. The result is shown in figure 3.31. For a second simulation the inputs were set to the following configuration: I_{A-} is 9.6 nA_{DC}, I_{B-} is 8.1 nA_{DC}, I_{A+} is a square wave going from 0 nA to 40 nA, and I_{B+} is a sine wave equal to 40 nA_{DC} + 10 nA_{AC}. These result is shown in figure 3.32.

3.5.3 Multiplier Layout

The layout for the multiplier requires matching between the transistors in the translinear loop as well as the current mirror providing the bias current. The interdigitated design techniques from [5] were used again with dummy transistors on the outsides of each DCBAABCD layout pattern. The core layout is shown in figure 3.33. The full layout with the configurable current sources is shown in figure 3.34. The sizes of the core and full layout are 1,207 μm^2 and 14,371 μm^2, respectively.

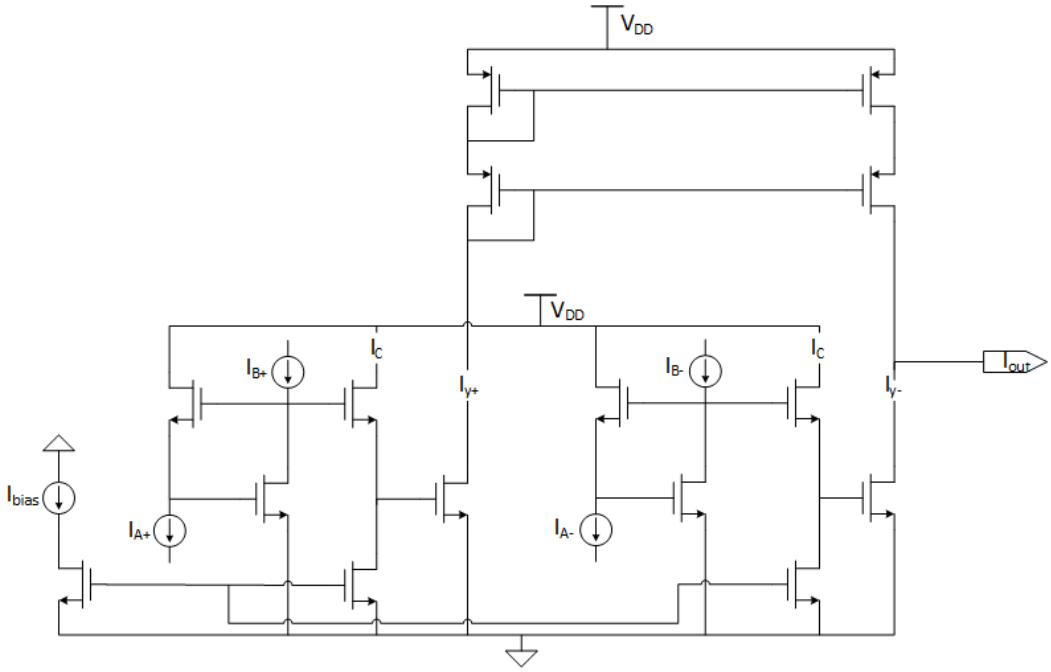


Figure 3.30: Multiplier Schematic

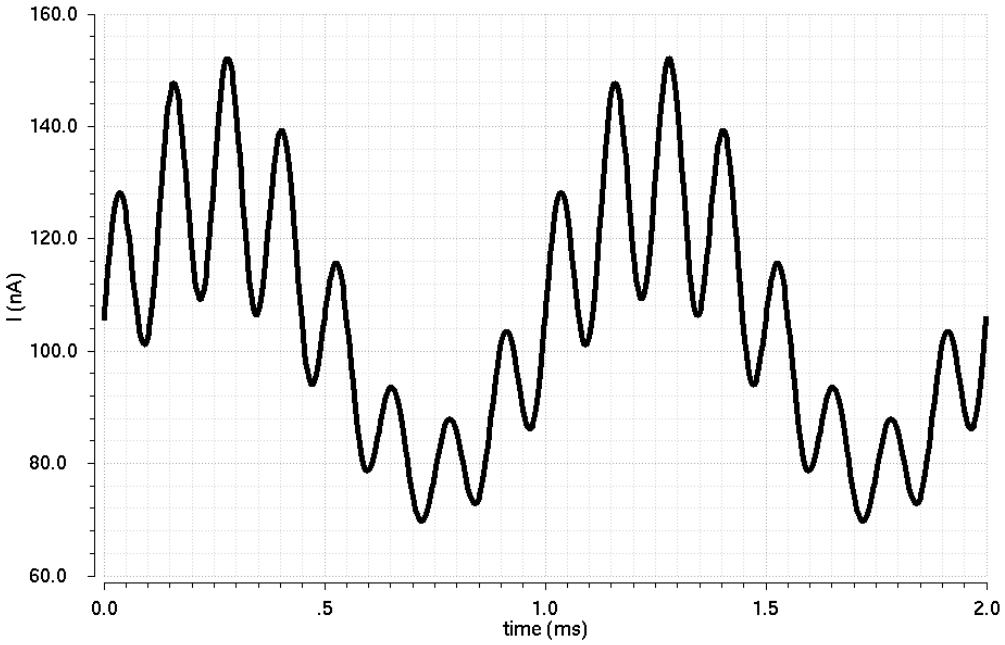


Figure 3.31: Multiplier Simulation: Sine Wave Multiplication

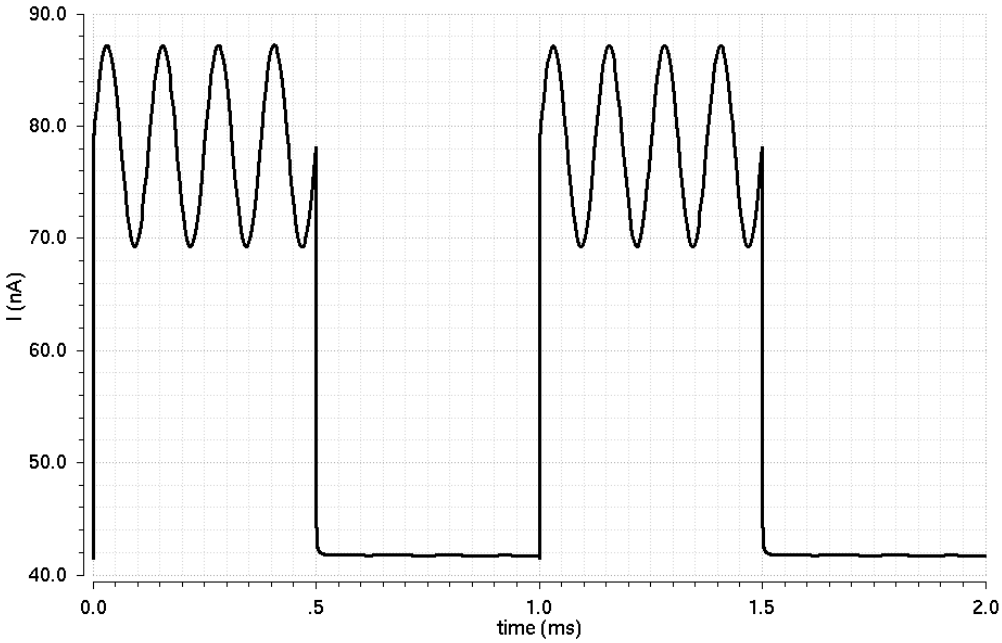


Figure 3.32: Multiplier Simulation: Square * Sine Wave

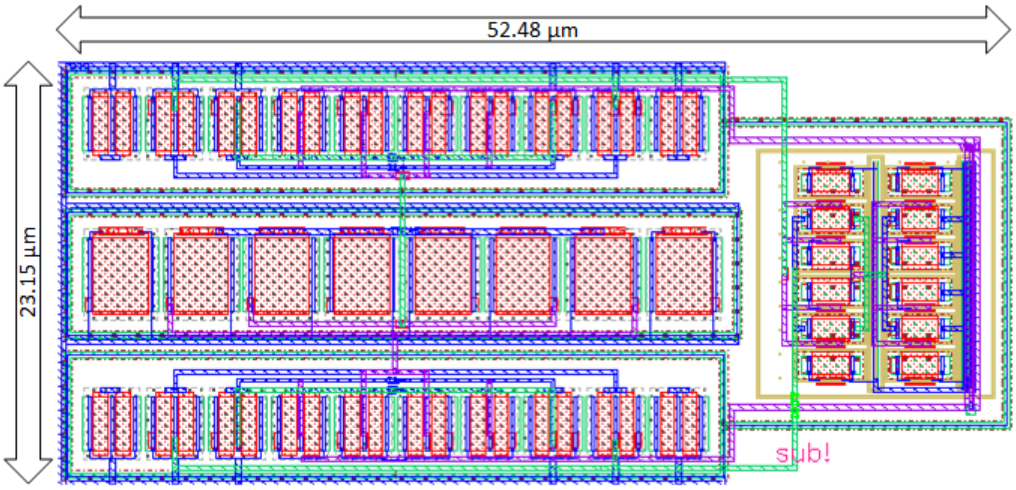


Figure 3.33: Multiplier Core Layout

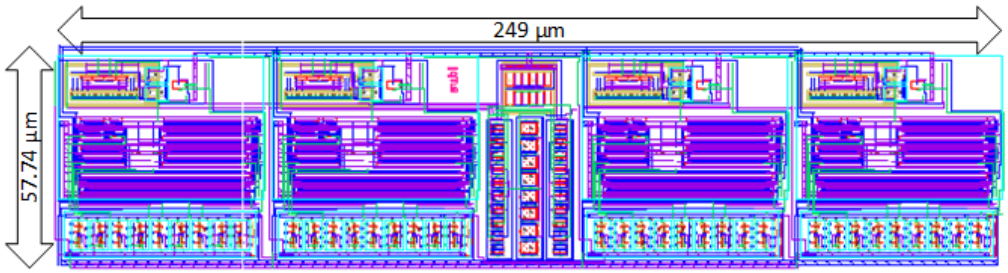


Figure 3.34: Multiplier Full Layout

3.6 Rectifier

The final processing block is an absolute value or rectifier block. The block performs an absolute value function on an input current and also outputs the sign in voltage as 0 V for negative and 1 V for positive. This function is very useful in analog signal processing. Specifically for the CASP presented, it will help perform envelope detection and frequency compression. The block was originally created by Junjie Lu, part of the Integrate Silicon Systems group at the University of Tennessee.

3.6.1 Rectifier Design

The schematic for the rectifier is shown in figure 3.35. The node V_{ref} is set to 500 mV. V_{bn} is set by the drain of a diode connected NMOS to ground that has a DC current flowing through it. V_{bn} sets the current mirror that biases twp differential pairs. V_{ref} biases one branch of each differential pair. The input sees a PMOS and NMOS transistor. If the current is sourced to the input, it will flow directly through the PMOS transistor, Q2, to the output. Conversely, if the current is being sinked to the input, the current is sent through the NMOS transistor, Q1, and reflected through the PMOS current mirror of Q3 and Q4. The differential pair, Q9 and Q10, provide negative feedback from one inversion through transistor Q10.

Node "A" is set by the V_{gs} relationship to either Q1 or Q2. The biasing for the second differential pair, Q12 and Q13, sets the "SIGN" output. The "SIGN" output is 0 V for negative currents and VDD for positive currents. This differential pair is

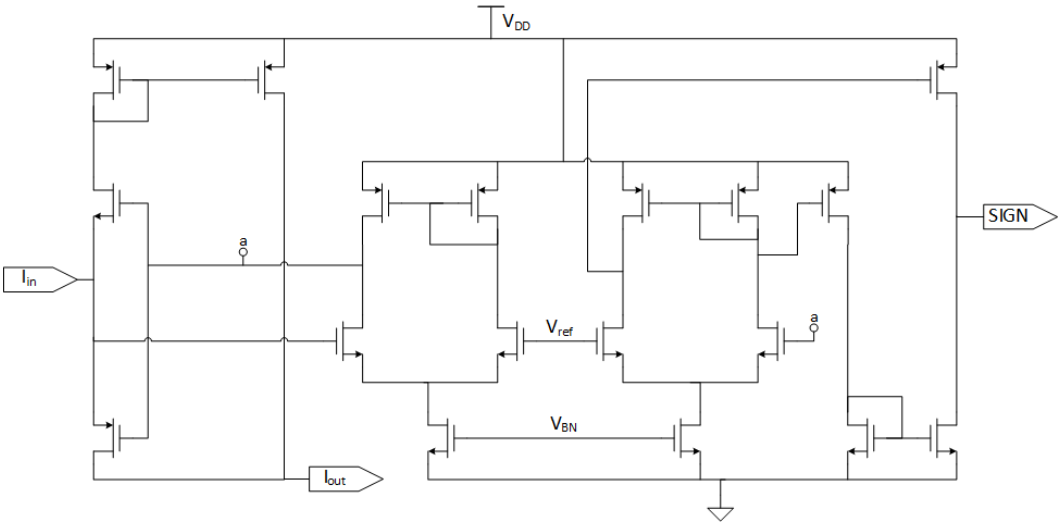


Figure 3.35: Rectifier Schematic

a comparator that sets the class AB output stage made of Q15-Q18. A few design changes had to be made from the original rectifier including: removal of the cascode transistors in the PMOS current mirror, Q3-Q4, for lower voltage capabilities, and the output stage was changed to a class AB structure for faster discharging. Junjie was generous enough to make theses changes in the schematic and layout.

3.6.2 Rectifier Simulations

A transient simulation was run on the rectifier to test its functionality. A simple simulation was set up with the V_{ref} set to 500 mV. The current setting V_{bn} was set using a configurable current source to 43.75 nA, which made V_{bn} equal 470 mV. The input signal is a sine wave at 15 kHz with a 50 nA amplitude based around 0 nA_{DC}. Figure 3.36 shows the input, output, and SIGN output signals from top to bottom. This performs as expected. The rectifier is a non-linear block and therefore an AC simulation is not necessary. In simulation, the rectifier works upwards to 500 kHz.

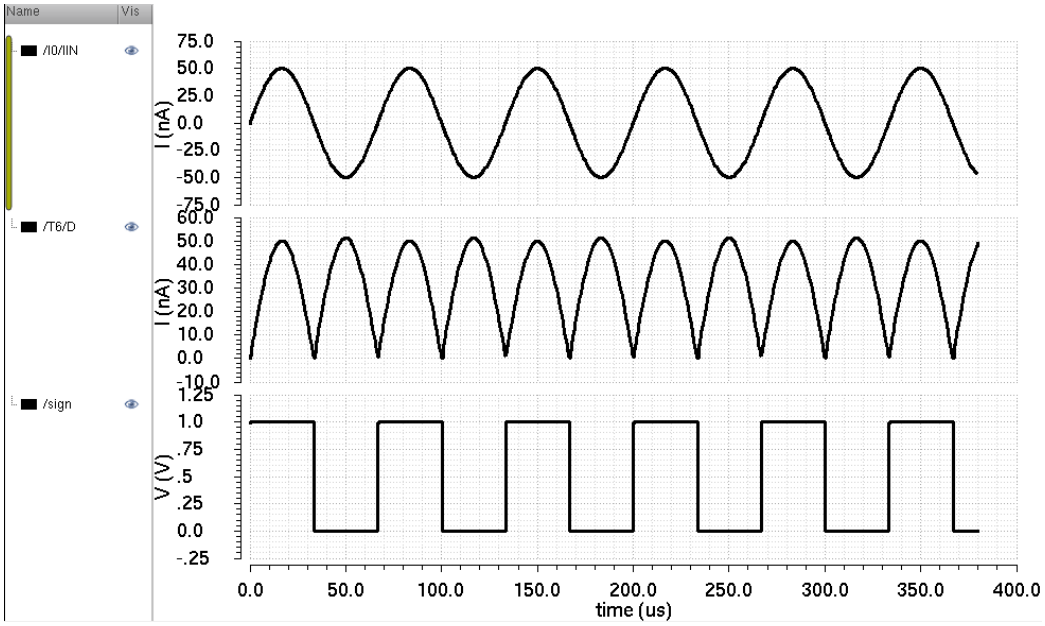


Figure 3.36: Rectifier Transient Simulation

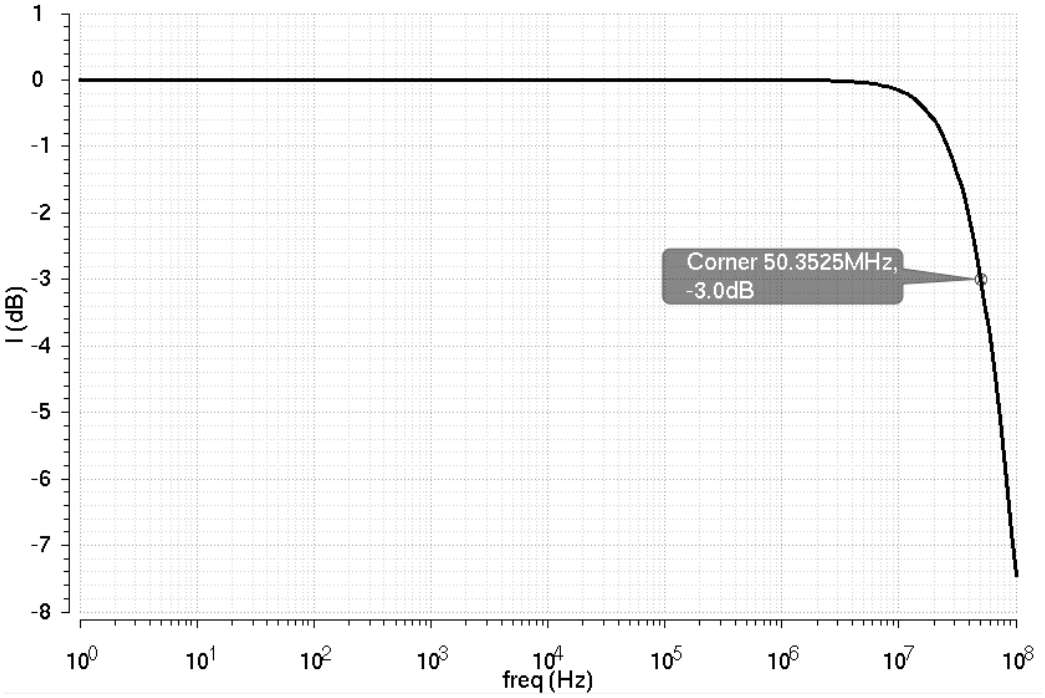


Figure 3.37: Rectifier AC Simulation

3.6.3 Rectifier Layout

The layout was completed by Junjie as well. Important note for the layout were common centroid matching and keeping node "Iin" isolated to reduce coupling capacitance. The core layout is shown in figure 3.40, and the final layout is shown in figure 3.41. The layout sizes are 221 μm^2 and 3,181 $\mu\mu$m^2, respectively.

3.7 Frequency Divider

The only digital block implemented on CASP is a frequency divider. A frequency divider can be very useful for signal processing. This block will be used with the SIGN output of the rectifier to reduce the output frequency. The design is simple and created using the default D flip-flops provided in the IBM-8RF digital library. A schematic is presented in figure 3.38. The switches are transmission gate (TG) switches controlled by shift registers. The output is configurable from divide by two to divide by sixteen on a binary scale. A simulation of the frequency being divided can be seen in figure 3.39. This block is purely digital and required no other verification.

3.8 Interconnect Block

The interconnect fabric for an analog signal processor is crucial to the end results. The connections between blocks need to be able to convey accurate signals with little to no attenuation from the interconnect. Large FPGAs have several different methods of interconnecting their blocks like programmable switches and buses. Many papers on FPAAs were reviewed in deciding how to interconnect CASP's processing blocks. The ideas of global and horizontal interconnects were experimented with but not implemented. The following subsection describes the interconnect blocks applied.

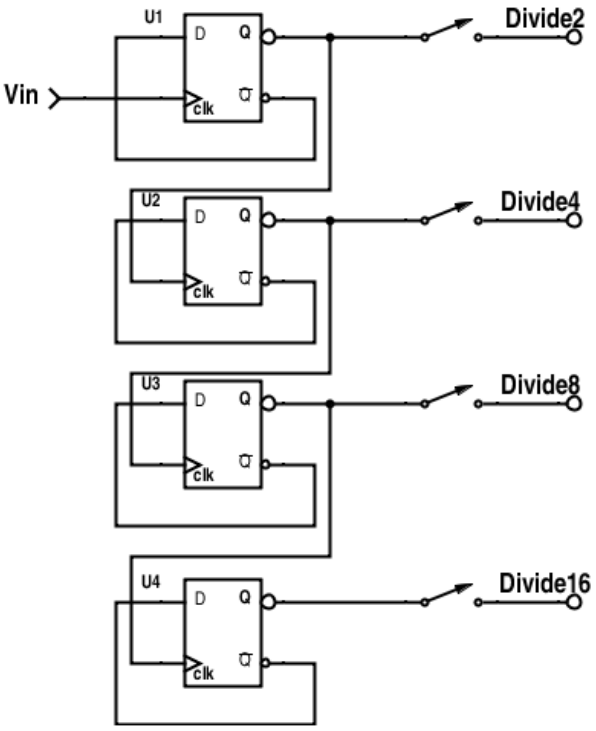


Figure 3.38: Frequency Divider Schematic

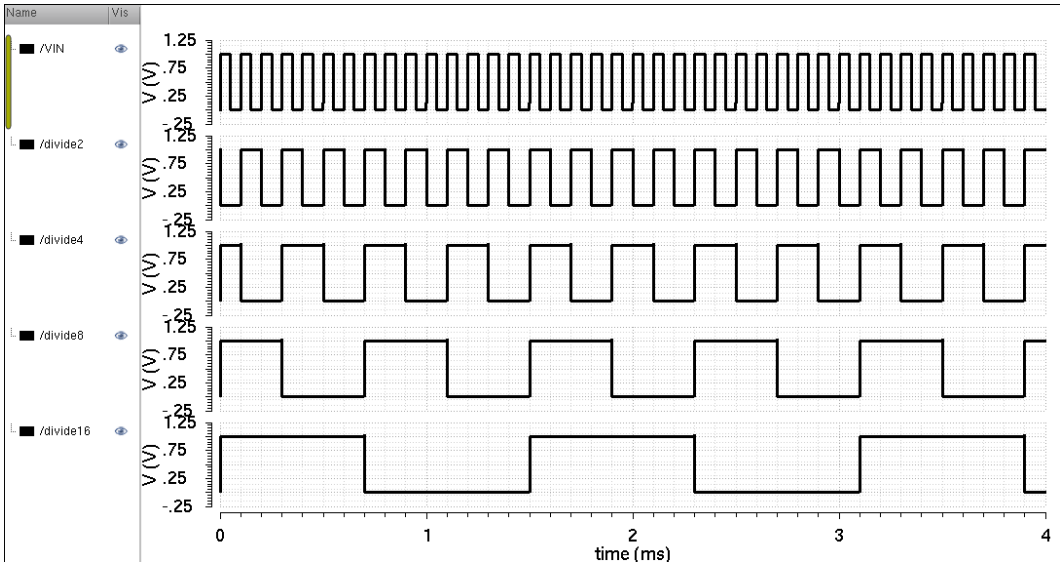


Figure 3.39: Frequency Divider Simulation

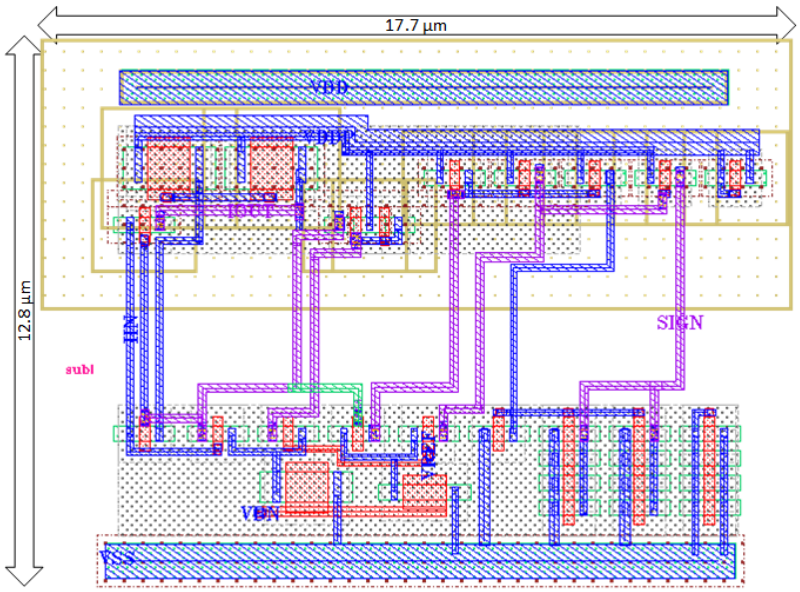


Figure 3.40: Rectifier Core Layout

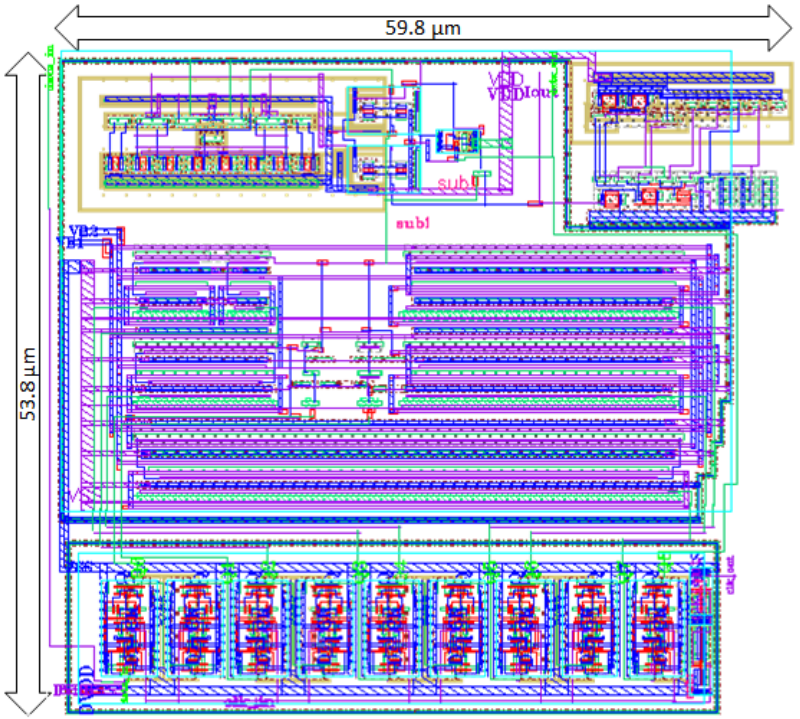


Figure 3.41: Rectifier Full Layout

3.8.1 Interconnect Design

For passing the signals, transmission gates were designed. The transmission gate (TG) consists of a NMOS-PMOS pair as seen in figure 3.42. The two control switches are connect by three inverters that scale up from 1X to 2X to 4X. These inverters make sure that CTRL1 is the opposite of CTRL2. Eight of these TG create the final interconnect block. The full schematic is shown in figure 3.43. Each one of the switches in this figure are TG switches controlled by a single switch pin. All of the switch pins are connected to a digital shift register. There are many shift registers throughout the board, which will be explained in section 3.9. The interconnections are labeled with respect to their directions such as: IN1L is input 1 left, OUT2B is output 2 bottom, etc. They pins are labeled input and output, but all of them are actually bi-directional. The labels with a "1" in them can only connect to other "1's" and the same with the "2's" labels. This creates a block that can pass two signals and connect in any direction necessary.

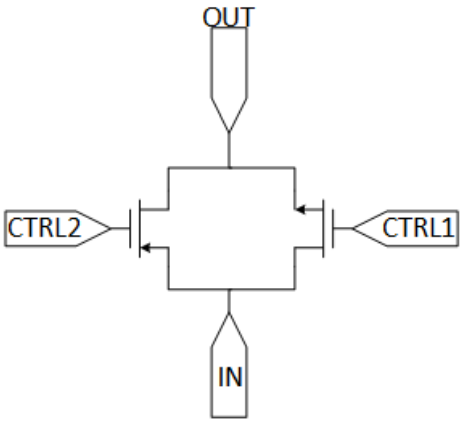


Figure 3.42: Transmission Gate Schematic

3.8.2 Interconnect Simulations

Transient simulations were run on the interconnect block to verify signals pass through with no attenuation. The simulations were setup with inputs connected to sign

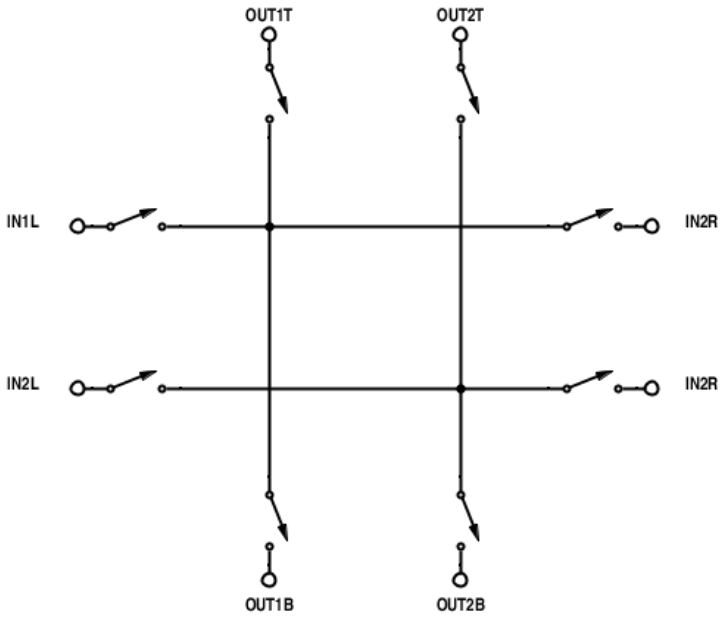


Figure 3.43: Interconnect Schematic

waves and outputs connected to resistors and capacitors that represent the loads of processing blocks. The simulation results showed negligible attenuation in signals regardless of the connection path. The results are not shown here because they are just overlapping sine waves with the same magnitude. An AC simulation shows that the interconnect block can pass signals up to 370 MHz. The AC simulation is shown in figure 3.44. The curve that rolls off more quickly is the response of two interconnect blocks tested in series. The simulations indicated that signals should be able to move from block to block uninterrupted.

3.8.3 Interconnect Layout

The interconnect block was laid out in a manner to make it easily integrated in between processing cells. The structure was designed to have the inputs and outputs run to the very outsides of the whole layout. Figure 3.45 shows the interconnect block fully laid out including its 8-bit shift register.

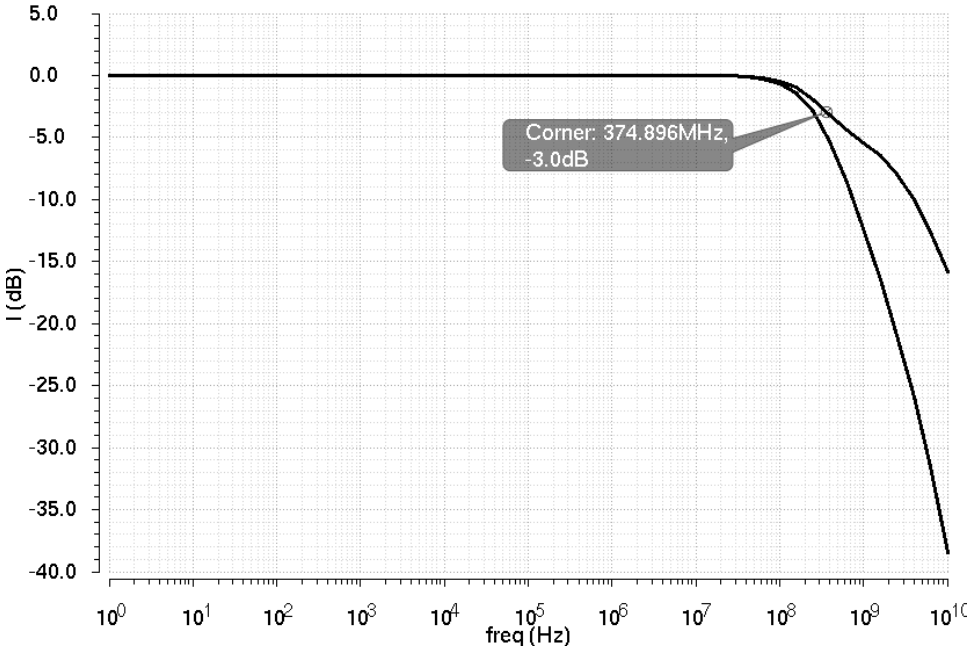


Figure 3.44: Interconnect AC Simulation

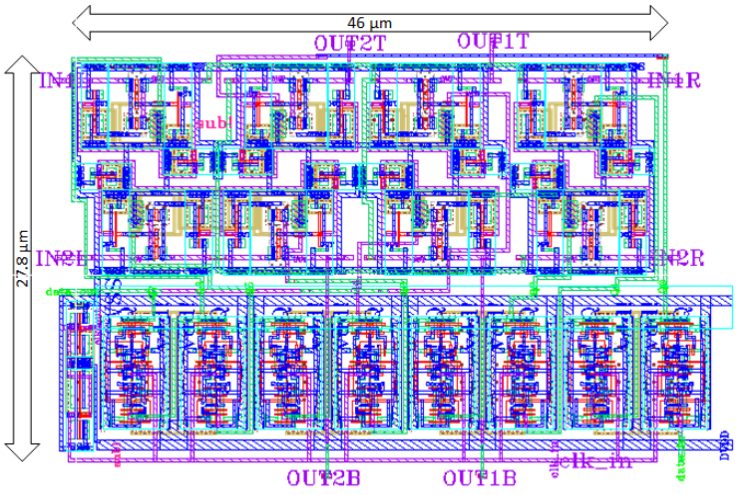


Figure 3.45: Interconnect Layout

3.9 Final Configurable Analog Signal Processor

All of the processing blocks come together to create this configurable analog signal processor, (CASP). The inspiration for CASP comes from low-power applications. It will be a part of an overall system and help save power by pre-processing information before it is converted by an ADC. CASP is a four by three array with twelve processing blocks. The next section describes the overall design of CASP. Then the top level simulations are presented. The layout is covered last.

3.9.1 CASP Design

The CASP schematic is shown in figure 3.46. The twelve blocks inside consist of the following: 2 OTAs, 2 LPFs, 2 rectifiers, 2 differentiators, 1 multiplier, 1 HPF, 1 integrator, and 1 frequency divider. The blocks are integrated into a web of interconnect blocks. Every block in the design has configurability; therefore, each block must have a way to set the bits that control them. This problem is solved by using shift registers. The shift registers are also made of D flip-flops from the IBM-8RF digital library. In total, there are 278 configurable control bits. The bits are programmed in using a MSP430ez. The bits are generated using C++ code. The bits are then converted to decimal numbers and implemented on the MSP430ez. The MSP430ez code converts the decimal numbers back to binary and loads them into CASP in the correct order. A C++ code example is show in appendix A.1. The Python script that converts the bit-stream out into decimal number is shown in appendix A.2.

3.9.2 CASP Simulation

Considering all of the configurable bits, the top level simulations were difficult to perform. The first simulation presented is an envelope detector circuit. The two processing blocks utilized are the rectifier and the LPF. The input to the rectifier is a

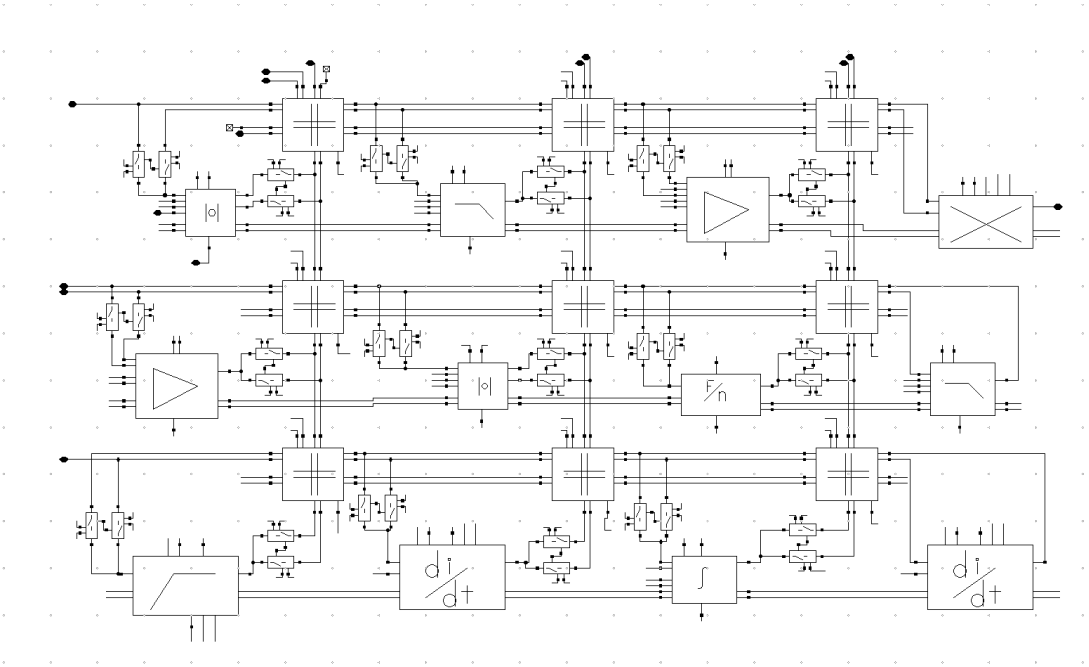


Figure 3.46: CASP Schematic

modulated sign wave produced from a 500 kHz sine wave and a 4 kHz sign wave. The modulation circuit uses an ideal multiplier based on Verilog-A code. The signal is rectified and then sent to the LPF. The high frequency component is averaged, which results in an output sine wave at 4 kHz with a DC offset. Figure 3.47 shows the input, rectified signal, and output. This simulation uses 355 nW of power. Another top level simulation is frequency compression. This function is completed using the envelope detector along with the frequency divider and OTA. The input is the same modulated wave in the previous simulation. The "SIGN" output of the rectifier is sent to the frequency divider. Once the frequency is divided it is hooked up to the positive input of the OTA with the negative input at 500 mV. The bias current for the OTA is the output of the envelope detector circuit. This arrangement changes the gain of the OTA with respect to the lower frequency of the modulated wave. Unfortunately, there was feedback from the high frequency signal in the rectifier that affected the output of the envelope detector; therefore for simulations purposes an ideal 4 kHz sine wave is used to show how this function could be completed. The results are shown in

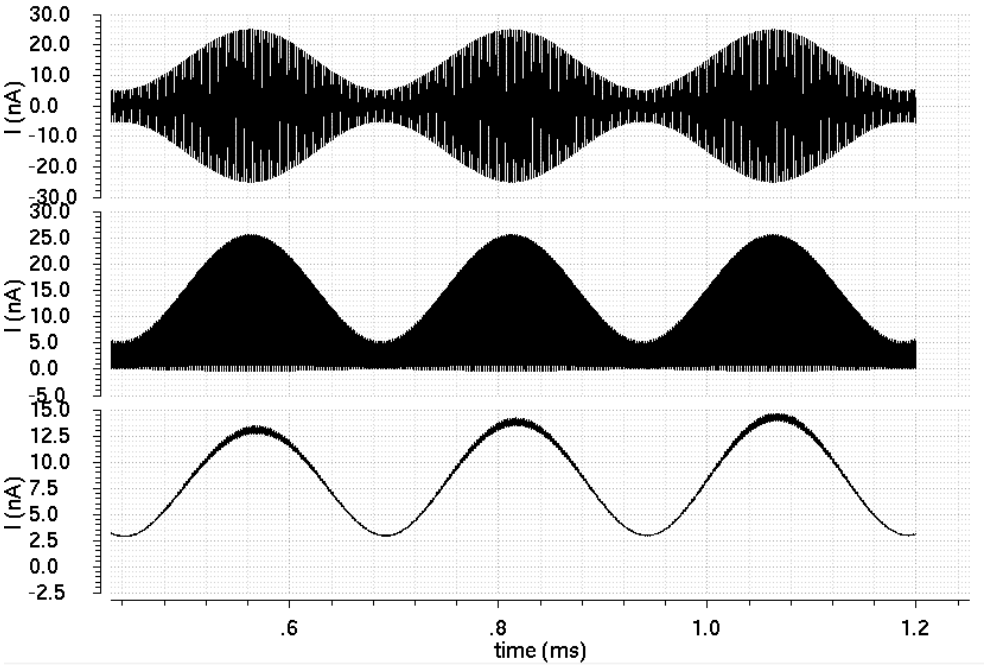


Figure 3.47: Envelope Detection Simulation

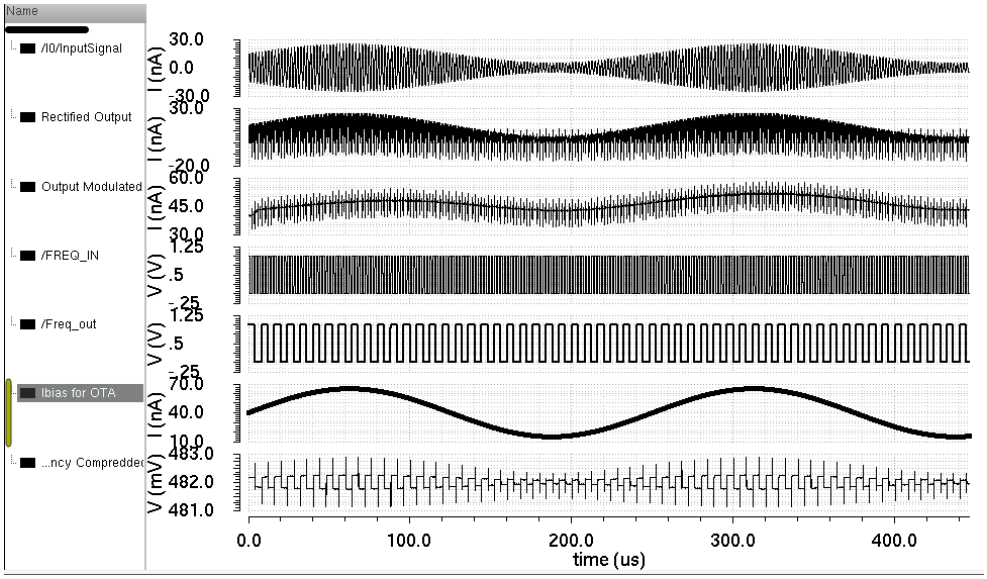


Figure 3.48: Frequency Compression Simulation

figure 3.48. The power for this operation is 30 μW. More operations will be explored for the next version of CASP.

3.9.3 CASP Layout

The layout for CASP was an extremely large task. Each block was laid out in a manner that was suppose to simplify the final layout. This was true for the most part, but more structured design could have been utilized. The blocks were placed based on the schematic. The block were various sizes, so the placement was a little more difficult than intended. Then the routing was completed, and the final design was put into a pad frame. The final layout is shown in figure 3.49. The total size is 340,820 μm^2.

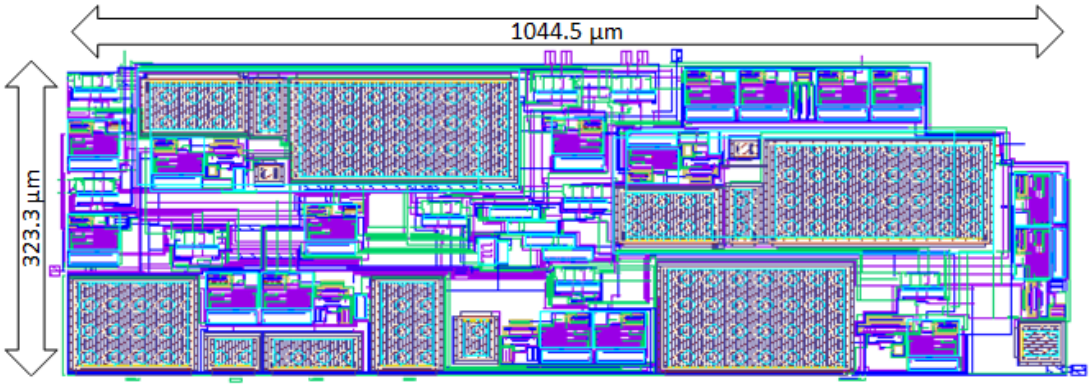


Figure 3.49: Final CASP Layout

Chapter 4

Experimental Results

This chapter walks through the results obtained from lab testing. Most of the blocks worked correctly on the test bench, but some blocks didn't perform as they did in simulation. The results for the LPF and integrator are in section 4.1. The HPF and integrator didn't work as expected, but some ideas on why they failed are presented in section 4.2. The OTA test results are in section 4.3. The multiplier block also faced challenges on the test bench, and some solutions are presented in section 4.4. The rectifier results are presented in figure 4.5. Lastly the frequency divider results are presented in section 4.6. Not all of the blocks worked well enough to create any of the top level simulations like the envelope detector. The generation of a zero DC offset sine wave, made the final simulations very difficult. Overall, the results presented below show that CASP has useful processing blocks. All of the experiments used a MSP430ez for programming of bits. The test setup required a printed circuit board (PCB) to build the input stage circuitry and extend the pins of the chip through a project. The input stage converts voltage to current using the circuit in figure 4.1; the output current equals $\frac{5V-V_{in}}{R1}$. All of the pins for CASP are sent to pin headers where they can be connected to respective stages. There was an additional board needed for a bias T network that was omitted from the initial PCB design. Also the voltage regulators on the original board didn't work correctly, so another board

supplied voltage regulators to bias VDD, DVDD, V_{ref}, and V_{b1}. The full test setup is shown in figure 4.2. The CASP chip goes in the green socket located on the bottom right, the bias T is on the top right, and the voltage regulators are on the left. Also a voltage divider for the clock and data pins is on the top left of figure 4.2.

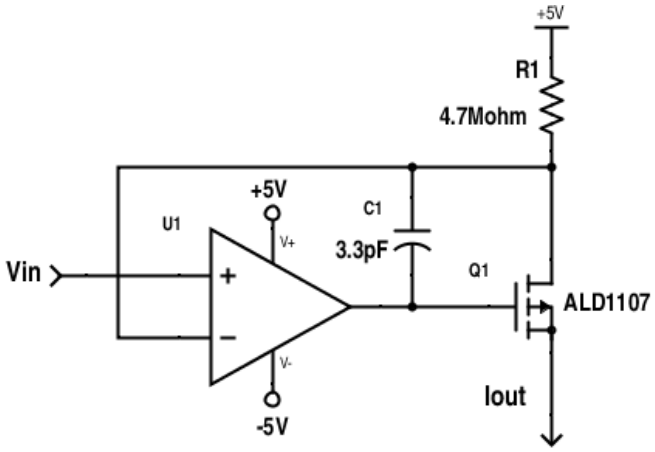


Figure 4.1: Input Stage for Testing

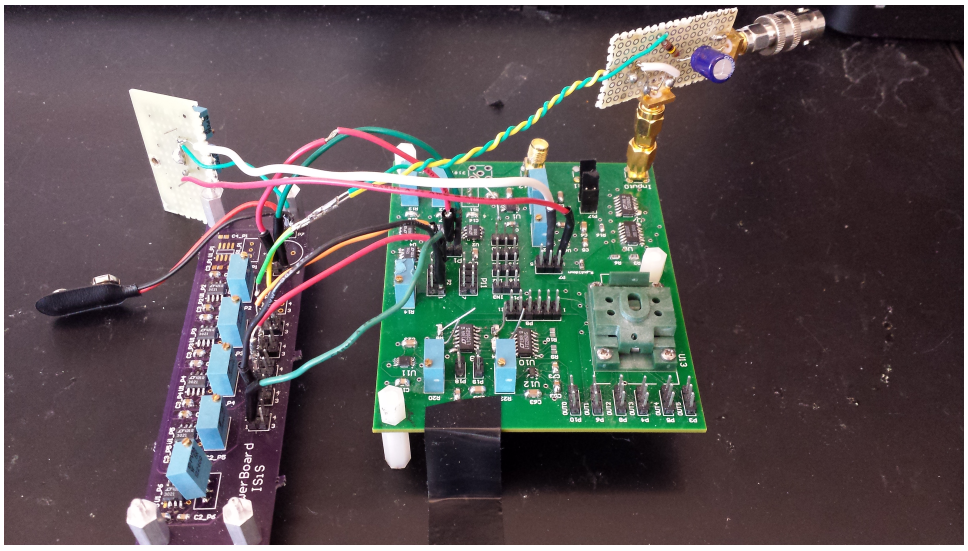


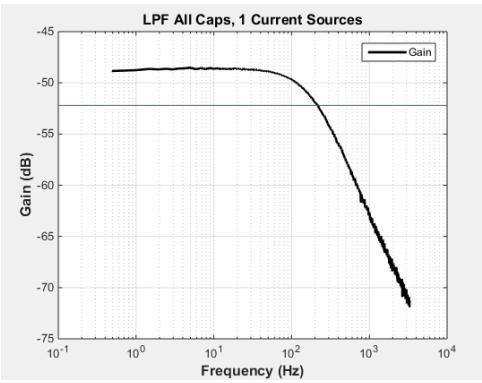
Figure 4.2: PCB Board for Test Setup

4.1 LPF and Integrator Results

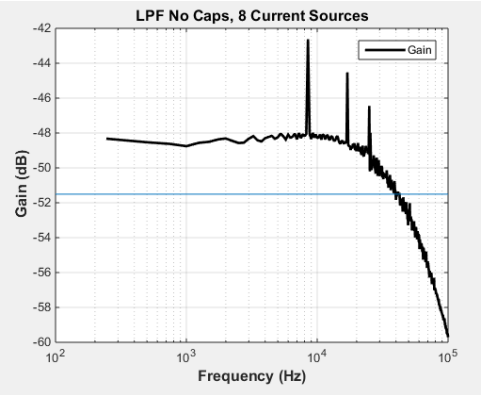
4.1.1 LPF Results

The equipment used for test was a network analyzer, SR770, and a current preamplifier, SR570. A bias T circuit is used to combine a DC voltage offset with the network analyzer's AC output. This signal is fed to the input stage where it is converted to AC current with DC offset current. The signal is low-pass filtered on CASP and then sent to the SR570. The SR570 converts current to voltage with programmable capabilities for gain; these tests use the 100 nA/V gain setting. The SR570 output is then sent back into the network analyzer where the results can be extracted using the GPIO cable and MATLAB. The network analyzer has capabilities to find the gain, THD, and noise response in $V/\sqrt{Hz}$.

First, a gain test is run to find the corner frequency. The results for the maximum and minimum corner frequency are shown in figure 4.3. The span reaches from 404 Hz to 42.5 kHz. The high frequency corner is expected to be less than simulated due to parasitic capacitance and limitation of the IBM-8RF process. Also noise induced from the surrounding environment caused the large spikes seen in figure 4.3.



(a) LPF Low Corner Gain

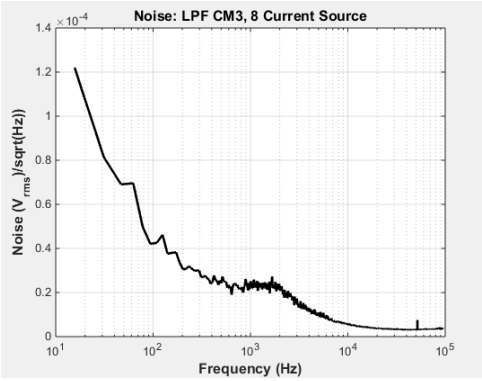


(b) LPF High Corner Gain

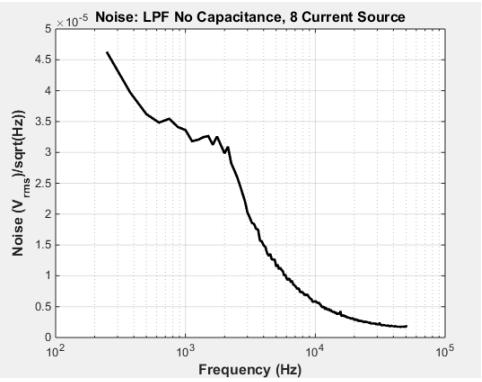
Figure 4.3: LPF Gain Results

Next a noise test is performed. For the noise test, the input is set to the voltage seen in normal operation in order to correctly bias the circuit. All of the voltages

are pulled from a 9V battery, and the circuit is placed in a grounded metal box. The output is sent to the network analyzer, which is measuring the power spectral density (PSD) in $V_{RMS}/\sqrt{Hz}$. Figure 4.4 show the noise result. After the data is obtained, it needs to be converted from voltage to current by dividing by 100 nA/V. Then the data is squared, integrated over the entire frequency range available, and the square root is taken to find the noise in nA_{RMS}. The data needs to be divided by the midband gain obtained in the AC test. Then the THD is found using the network analyzer. The THD test show an input amplitude of 2.55 dBV creates a THD of 1%. Then using equation 4.1, the dynamic range is found to be 46. Compared to a dynamic range of 58 in simulation. The power dissipation ranges from 645 nW_{VDD} + 85 nW_{DVDD} to 4.2 μW_{VDD} + 85 nW_{DVDD} from these tests.



(a) LPF Low Corner Noise



(b) LPF High Corner Noise

Figure 4.4: LPF Noise Results

$$DR \;=\; 20 \times \log(\frac{A_{RMS,input}}{A_{RMS,noie}}) \;= 20 \times \log(\frac{94.83\ nA_{RMS}}{287.3\ pA_{RMS}}) \;= 46 \qquad (4.1)$$

4.1.2 Integrator Results

The integrator was tested using the same instruments as the LPF. The AC results for the integrator are presented in figure 4.5. These measurements were taken with 1, 3, 5, and all 8 current source branches activated. The high and low corner frequencies

are 141 Hz and 13.4 kHz, respectively, compared to simulated values of 49 Hz and 7.58 kHz. The power consumption ranged from 730 nW to 4.3 μW. The noise analysis was identical to the LPF measurements, which makes sense when considering the similarities in design. The dynamic range for the integrator is compared to a similar integrator in table 4.1.

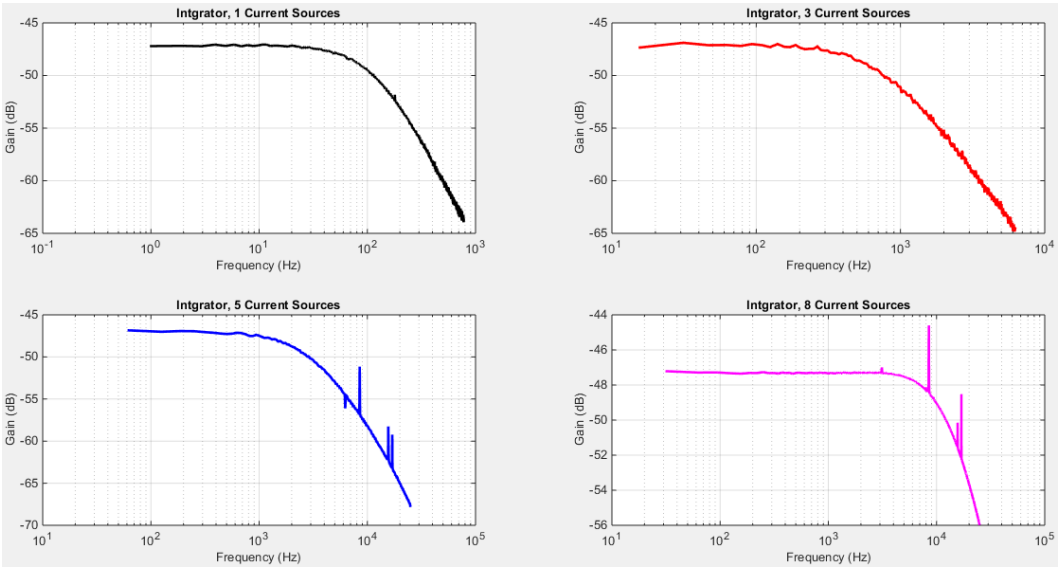


Figure 4.5: Integrator Corner Frequencies with Respect to Current Sources

Table 4.1: Integrator Comparison

Integrator	Power Consumption	Frequency Range	Dynamic Range
This work	730 nW	0.14 - 13.4 kHz	46 (THD=1%)
[8]	6 μW	1.6 - 8 kHz	58 (THD=2%)

4.2 HPF and Differentiator Results

The HPF test was run using the same method as the LPF test. The bits were loaded in via the MSP430 and the output was sent to the network analyzer through the SR570. However, the output looked like a LPF with a high corner frequency. It

seemed that the subtraction of the LPF signal from the original signal was not being completed. To further investigate the problem, a Monte Carlo simulation was run on the HPF. The results are shown in figure 4.6. This simulation shows that there was a possibility that the HPF would not work under certain circumstances. The Monte Carlo simulations that didn't perform as expected were traced to voltage headroom problems that occurred when running a "Slow NMOS, Slow PMOS,"(ss) simulation. In figure 3.17, the NMOS transistor that I_P flows into has a V_{DS} that is very small. This voltage causes the transistor to fall out of saturation mode and into linear mode. Therefore, the transistor properties necessary for the HPF no longer hold true. The differentiator results were the same as the HPF.

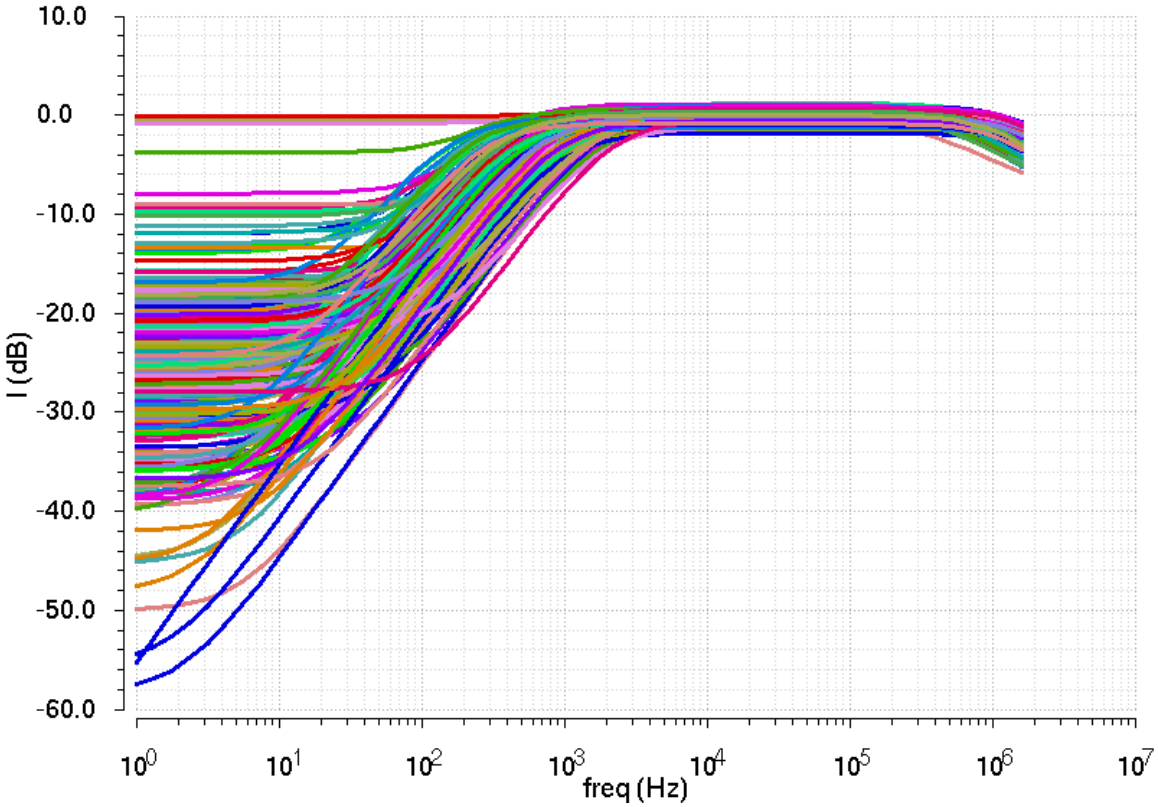


Figure 4.6: HPF Monte Carlo Simulation

4.3 OTA Results

The OTA receives a input voltage signal. The "IN-" input was connected to a 500 mV voltage; the "IN+" input was connected to a waveform generator with a 40 mV_{AC} plus 500 mV_{DC}. The output was sent to the SR570, which converted the current to voltage. The final voltage was observed on an oscilloscope. The same parameters from the simulation were used for comparison reasons. The output is shown in figure 4.7. After converting the voltage back to current, the output is 72.5 nA_{pp} with a DC offset of 113 nA. The power consumed by VDD was 1.18 μW.

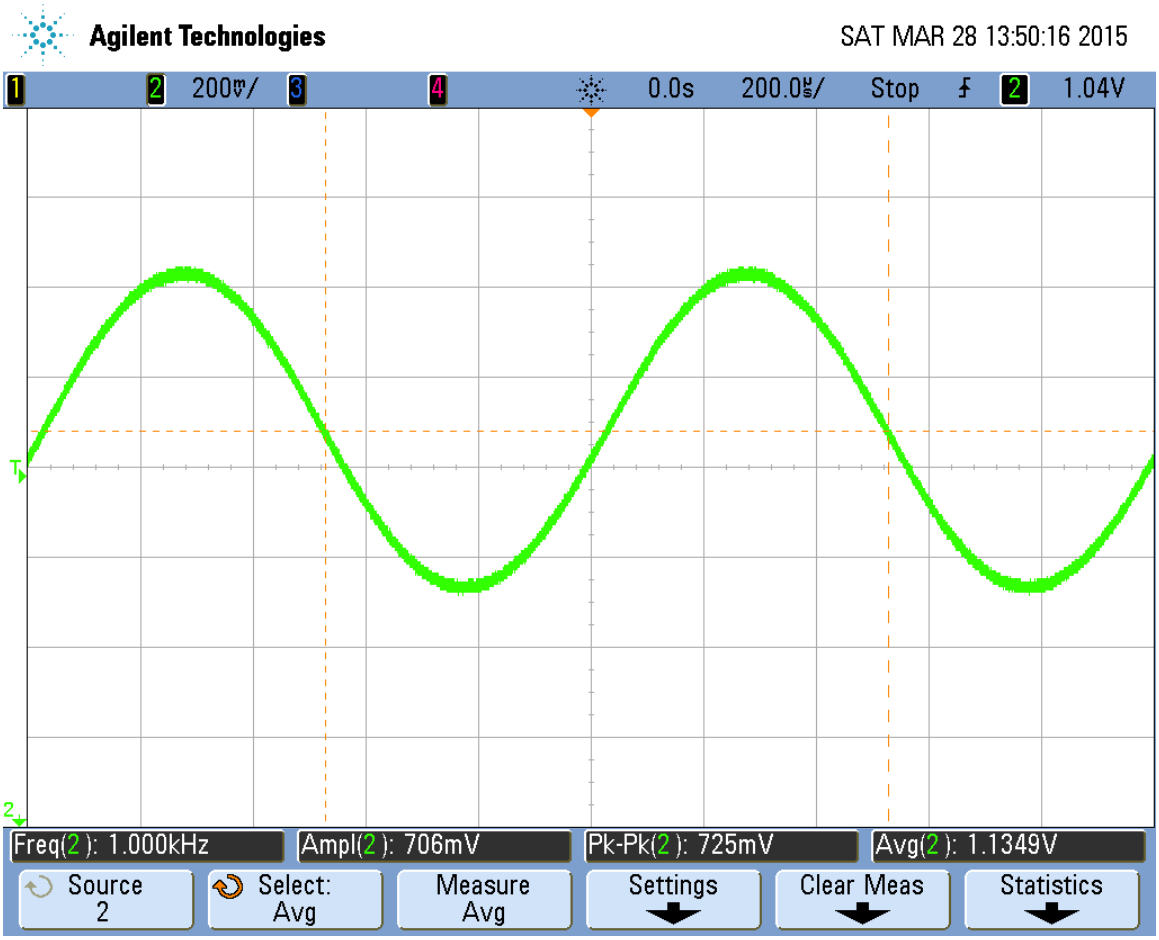


Figure 4.7: OTA Experimental Results

4.4 Multiplier Results

The multiplier was tested using several different configurations of bits; however, no combination ever produced the desired output. For having such good simulation results, its odd for the experimental results not to work. A Monte Carlo simulation of the transient response is shown in figure 3.31. The result is shown in figure 4.8. With respect to the Monte Carlo simulation, it is possible that the multiplier won't work due to process variations. The simulations that didn't perform multiplication are found when under "ff" and "fs" simulations. These simulations correlate with the HPF Monte Carlo simulation that didn't work with the "Slow PMOS" variation. Resizing the transistors to work better for all corners of process variation can fix the multipliers problems.

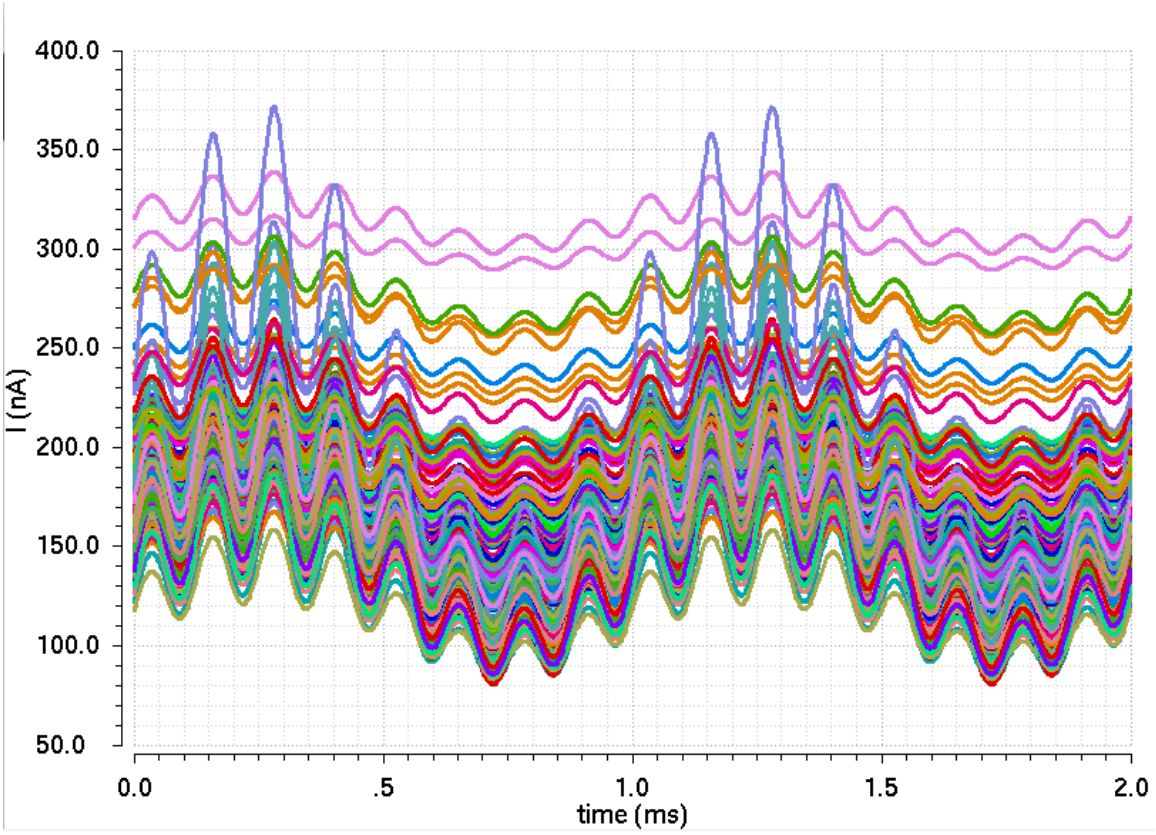


Figure 4.8: Multiplier Monte Carlo Simulation

4.5 Rectifier Results

The rectifier was a difficult block to test. The input stage built for testing only sources current, meaning all the current is in the same direction. However, the rectifier need a current that is positive and negative. This input current is difficult to create. The OTA output current signal was hooked up to the rectifier and a source-meter. The idea was to use the source-meter to sink the DC current off the output signal. This method didn't work because the added capacitance of the source-meter and OTA output caused the input of the rectifier to not work correctly. Therefore, this block wasn't tested.

4.6 Frequency Divider Results

The frequency divider was an all digital block. The experimental results were expected to work well and they did. The frequency was tested for all cases, divide by 2 to divide by 16. The divide by 2 and divide by 4 inputs were at 20 kHz and are shown in figure 4.9 and 4.10, respectively. The divide by 8 input was 100 kHz, shown in figure 4.11. Finally the divide by 16 input was set to 160 kHz, shown in figure 4.12; the oscilloscope says the input was 157 kHz because of its resolution limitation. The only improvement could be adding some buffers to the output to sharpen the edges.

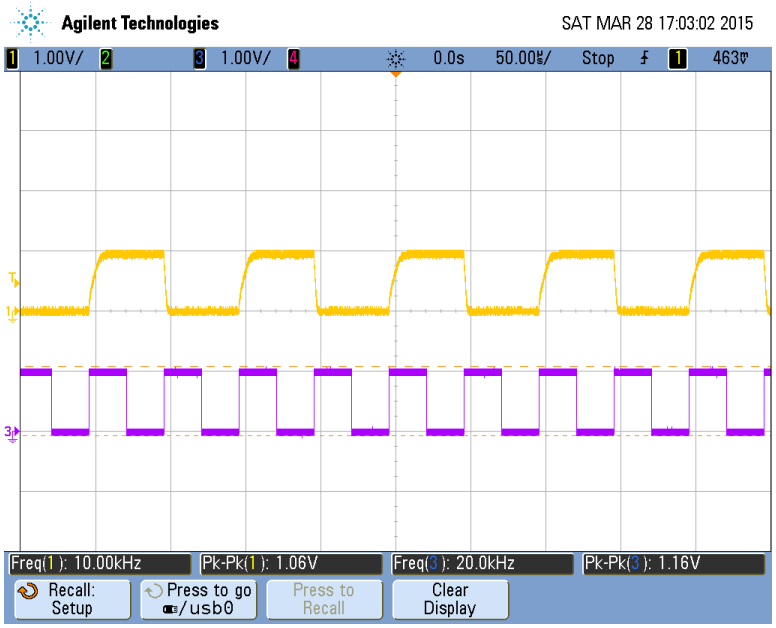


Figure 4.9: Frequency Divider - Divide by 2

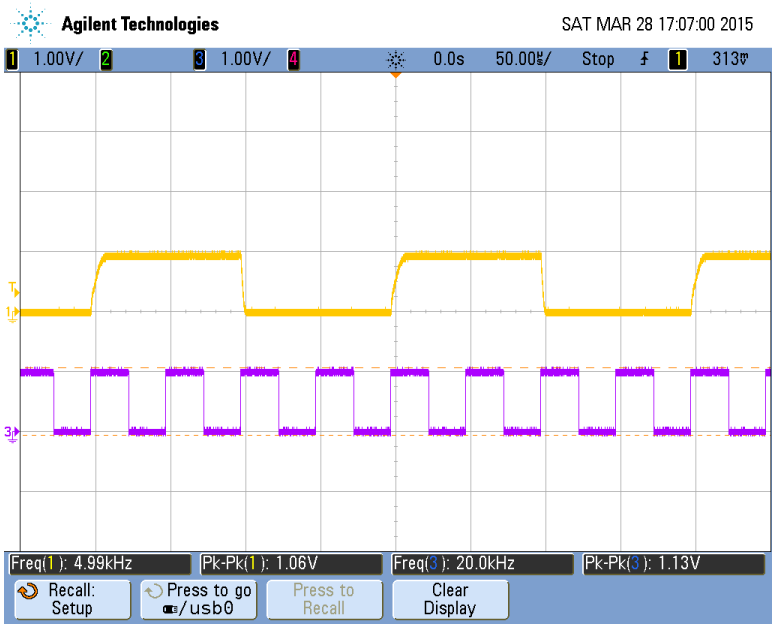


Figure 4.10: Frequency Divider - Divide by 4

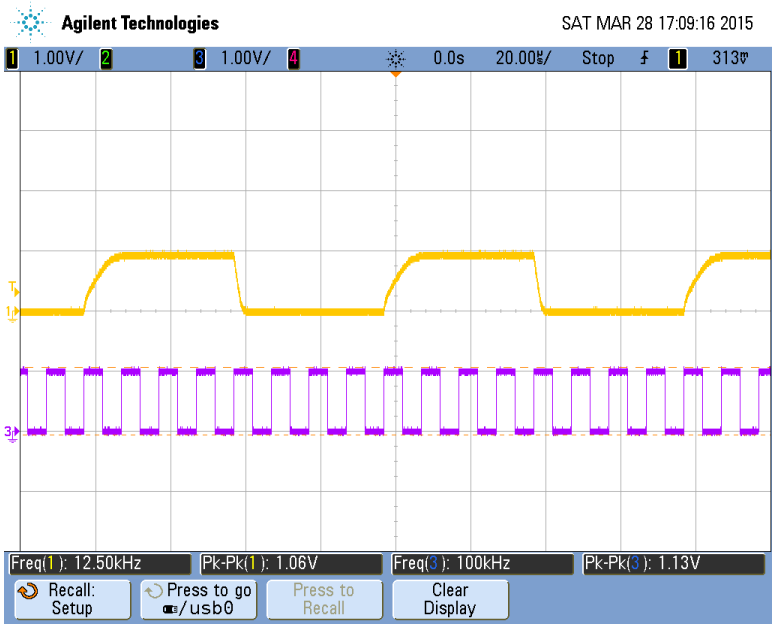


Figure 4.11: Frequency Divider - Divide by 8

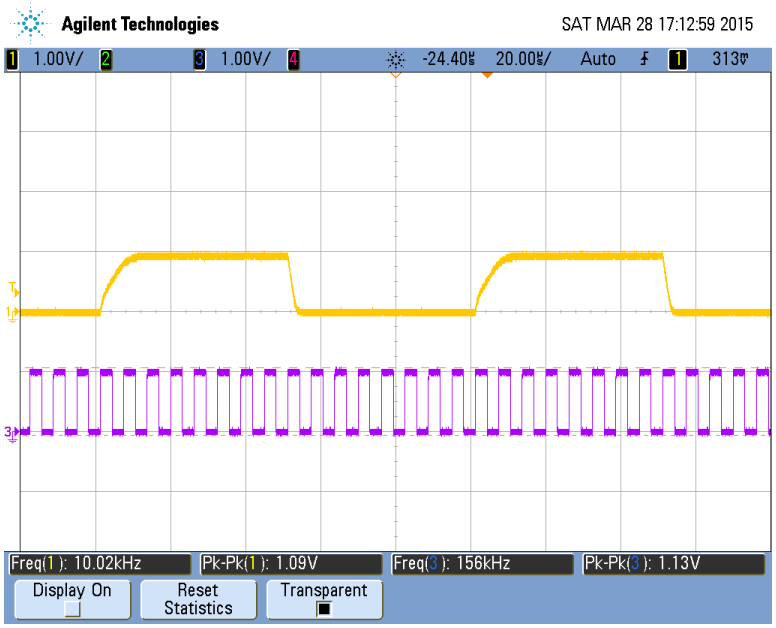


Figure 4.12: Frequency Divider - Divide by 16

Chapter 5

Conclusions

This thesis presents ultra-low-power analog signal processing techniques that could be used for wireless sensors. Seven analog signal processing blocks are able to complete functions at a lower power cost compared to digital signal processing. The move towards power efficiency creates a high demand for on-chip ASP's. CASP can successfully low-pass filter, integrate, and divide frequency with minimal power consumption. The goals were to be able to process neural recordings or acoustic signals coming from a bat. With some future work these goals should be met. The interconnect fabric for CASP was also a success. This interconnect design is ideal for small scale analog arrays that don't need large buses or global connections. For larger FPAA designs, global routing would be necessary to mitigate attenuation for signals traveling along long paths.

A second revision would allow CASP to perform more high level signal processing techniques such as: frequency compression, non-linear energy operator, and sigma-delta modulation. The high-pass filter and multiplier blocks need improvements for the second revision of CASP. Following the suggestions in the HPF and multiplier section about process variation, a future researcher could resize transistors for better results. Also bi-directional outputs would be helpful for certain processing techniques.

With these revision CASP could be an essential part integrated into a full system on chip used to monitor bat activity.

Bibliography

[1] Arindam Basu, Stephen Brink, Craig Schlottmann, Shubha Ramakrishnan, Csaba Petre, Scott Koziol, Faik Baskaya, Christopher M Twigg, and Paul Hasler. A floating-gate-based field-programmable analog array. *Solid-State Circuits, IEEE Journal of*, 45(9):1781–1794, 2010. 7

[2] Ravi Chawla, Abhishek Bandyopadhyay, Venkatesh Srinivasan, and Paul Hasler. A 531 nw/mhz, 128× 32 current-mode programmable analog vector-matrix multiplier with over two decades of linearity. In *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, pages 651–654. IEEE, 2004. 2

[3] Barrie Gilbert. Translinear circuits: A proposed classification. *Electronics Letters*, 11(1):14–16, 1975. 9, 18

[4] Reid R Harrison, Paul T Watkins, Ryan J Kier, Robert O Lovejoy, Daniel J Black, Bradley Greger, and Florian Solzbacher. A low-power integrated circuit for a wireless 100-electrode neural recording system. *Solid-State Circuits, IEEE Journal of*, 42(1):123–133, 2007. 2

[5] Ray Alan Hastings. *The art of analog layout*, volume 2. Prentice Hall, 2006. 5, 15, 21, 42

[6] Bradley Minch. Static and dynamic trnaslinear circuits. May 2010. 16

[7] Bogdan Pankiewicz, Marek Wojcikowski, Stanislaw Szczepanski, and Yichuang Sun. A field programmable analog array for cmos continuous-time ota-c filter applications. *Solid-State Circuits, IEEE Journal of*, 37(2):125–136, 2002. 7

[8] Wouter A Serdijn, Martijn Broest, Jan Mulder, Albert C van der Woerd, and Arthur HM van Roermund. A low-voltage ultra-low-power translinear integrator for audio filter applications. *Solid-State Circuits, IEEE Journal of*, 32(4):577–581, 1997. 62

[9] T. Serrano-Gotarredona, B. Linares-Barranco, and A. Andreou. A general translinear principle for subthreshold MOS transistors. *IEEE Transaction on Circuits and Systems–I: Fundamental Theory and Applications*, 46(5):607–616, 1999. 9

[10] Eric Vittoz and Jean Fellrath. Cmos analog integrated circuits based on weak inversion operations. *Solid-State Circuits, IEEE Journal of*, 12(3):224–231, 1977. 9

Appendix

Appendix A

A.1 C++ Code for Bit Stream Generation

```
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

//Function for interconnect block
vector <int> interConnect(string s1, string s2)
{
vector <int> out;
out.resize(8, 0);
if      (s1 == "IN1LtoIN1R") out[6]=out[1]=1;
else if (s1 == "IN1LtoOUT1T") out[6]=out[7]=1;
else if (s1 == "IN1LtoOUT1B") out[6]=out[4]=1;
else if (s1 == "IN1RtoOUT1T") out[1]=out[7]=1;
else if (s1 == "IN1RtoOUT1B") out[1]=out[4]=1;
else if (s1 == "OUT1TtoOUT1B") out[4]=out[7]=1;
else if (s1 == "off") out[1]=out[4]=out[6]=out[7]=0;
else if (s1 == "special") out[1]=out[4]=out[7]=1;
```

```
else out[0]=9;

if      (s2 == "IN2LtoIN2R") out[5]=out[2]=1;
else if (s2 == "IN2LtoOUT2T") out[5]=out[0]=1;
else if (s2 == "IN2LtoOUT2B") out[5]=out[3]=1;
else if (s2 == "IN2RtoOUT2T") out[2]=out[0]=1;
else if (s2 == "IN2RtoOUT2B") out[2]=out[3]=1;
else if (s2 == "OUT2TtoOUT2B") out[3]=out[0]=1;
else if (s2 == "off") out[0]=out[2]=out[3]=out[5]=0;
else out[7]=9;
return out;
}

//Function for current source block
vector <int> currentSource(string s0, string s1, string s2,
   string s3,
        string s4, string s5, string s6, string s7, string s8
           )
{
vector <int> out;
out.resize(9, 0);
if (s0 == "on") out[0]=1;
if (s1 == "on") out[1]=1;
if (s2 == "on") out[2]=1;
if (s3 == "on") out[3]=1;
if (s4 == "on") out[4]=1;
if (s5 == "on") out[5]=1;
if (s6 == "on") out[6]=1;
```

```
if (s7 == "on") out[7]=1;
if (s8 == "source") out[8]=1;
return out;
}

//Function for 9-bit Shift Register block
vector <int> shift9bit(string s0, string s1, string s2,
   string s3, string s4, string s5, string s6, string s7,
   string s8)
{
vector <int> out;
out.resize(9, 0);
if (s0 == "on") out[0]=1;
if (s1 == "on") out[1]=1;
if (s2 == "on") out[2]=1;
if (s3 == "on") out[3]=1;
if (s4 == "on") out[4]=1;
if (s5 == "on") out[5]=1;
if (s6 == "on") out[6]=1;
if (s7 == "on") out[7]=1;
if (s8 == "on") out[8]=1;
return out;
}

//Function for capacitor array
vector <int> capArray(string s0, string s1, string s2, string
    s3)
{
```

```
vector <int> out;
out.resize(4, 0);
if (s0 == "on") out[0]=1;
if (s1 == "on") out[1]=1;
if (s2 == "on") out[2]=1;
if (s3 == "on") out[3]=1;
return out;
}

//Function for frequency divider
vector <int> freqDivider(string s0, string s1, string s2,
   string s3)
{
vector <int> out;
out.resize(4, 0);
if (s0 == "divide2") out[0]=1;
if (s1 == "divide4") out[1]=1;
if (s2 == "divide8") out[2]=1;
if (s3 == "divide16") out[3]=1;
return out;
}

//Final Function for writing bits
int main () {
        vector <int> I0, I1, I2, I3, I4, I5, I6, I7, I8, I9,
           I10, I11, I12, I13, I14, I15, I16;
        vector <int> I17, I18, I19, I20, I21, I22, I23, I24,
           LPF, HPF, IB, IOUT, Ibias;
```

```
        ofstream myfile;
        myfile.open ("bitStream_abs.txt");
//Row 1
I0 = interConnect("OUT1TtoOUT1B", "off");
        for(int i=0; i<I0.size(); i++) myfile << I0[i];
I1 = interConnect("OUT1TtoOUT1B", "off");
        for(int i=0; i<I1.size(); i++) myfile << I1[i];
I2 = interConnect("off", "off");
        for(int i=0; i<I2.size(); i++) myfile << I2[i];

//Row 2
I3 = currentSource("off", "off", "off", "off", "off", "off",
   "off", "off", "source");    //Abs current
        for(int i=0; i<I3.size(); i++) myfile << I3[i];
I4 = currentSource("off", "off", "off", "off", "off", "off",
   "off", "off", "sink");      //LPF current
        LPF = capArray("off", "off", "off", "off");      //LPF
            caps
                for(int j=0; j<LPF.size(); j++) {int i = LPF[
                   j];
                I4.push_back(i);}
        for(int i=0; i<I4.size(); i++) myfile << I4[i];
I5 = currentSource("off", "off", "off", "off", "off", "off",
   "off", "off", "source");    //OTA current
        for(int i=0; i<I5.size(); i++) myfile << I5[i];
I6 = currentSource("off", "off", "off", "off", "off", "off",
   "off", "off", "source");                    //IA-
   multiplier
```

```
        IB    = currentSource("off", "off", "off", "off", "off
            ", "off", "off", "off", "source"); //IB-
            multiplier
                for(int j=0; j<IB.size(); j++) {int i = IB[j
                    ];              //Adds IB- onto end of I6
                I6.push_back(i);}
                IOUT = currentSource("off", "off", "off", "
                    off", "off", "off", "off", "off", "source"
                    ); //IOUT multiplier
                for(int j=0; j<IOUT.size(); j++) {int i =
                    IOUT[j];        //Adds IOUT onto end of I6
                I6.push_back(i);}
                Ibias = currentSource("off", "off", "off", "
                    off", "off", "off", "off", "off", "source"
                    );          //Ibias multiplier
                for(int j=0; j<Ibias.size(); j++) {int i =
                    Ibias[j]; //Adds Ibias onto end of I6
                I6.push_back(i);}
        for(int i=0; i<I6.size(); i++) myfile << I6[i];

//Row 3
I7 = interConnect("special", "off");
        for(int i=0; i<I7.size(); i++) myfile << I7[i];
I8 = interConnect("IN1LtoOUT1T", "off");
        for(int i=0; i<I8.size(); i++) myfile << I8[i];
I9 = interConnect("off", "off");
        for(int i=0; i<I9.size(); i++) myfile << I9[i];
```

```
//Row 4
I10 = currentSource("on", "on", "on", "on", "on", "off", "off
   ", "off", "source");          //OTA current
        for(int i=0; i<I10.size(); i++) myfile << I10[i];
I11 = currentSource("off", "off", "off", "off", "off", "off",
    "off", "off", "sink");       //Abs current
        for(int i=0; i<I11.size(); i++) myfile << I11[i];
I12 = freqDivider("off", "off", "off", "off");  //Frequency
   Divider
        for(int i=0; i<I12.size(); i++) myfile << I12[i];
I13 = currentSource("off", "off", "off", "off", "off", "off",
    "off", "off", "sink");       //LPF current
        LPF = capArray("off", "off", "off", "off");       //LPF
            caps
                for(int j=0; j<LPF.size(); j++) {int i = LPF[
                   j];
        I13.push_back(i);}
        for(int i=0; i<I13.size(); i++) myfile << I13[i];

//Row 5
I14 = interConnect("off", "off");
        for(int i=0; i<I14.size(); i++) myfile << I14[i];
I15 = interConnect("off", "off");
        for(int i=0; i<I15.size(); i++) myfile << I15[i];
I16 = interConnect("off", "off");
        for(int i=0; i<I16.size(); i++) myfile << I16[i];

//Row 6
```

```
I17 = currentSource("on", "on", "on", "on", "off", "off", "
    off", "off", "sink");                    //HPF current IOUT
        IB = currentSource("off", "off", "off", "off", "off",
            "off", "off", "off", "source");    //HPF current
            IBIAS
                for(int j=0; j<IB.size(); j++) {int i = IB[j
                    ];
        I17.push_back(i);}
        HPF = capArray("off", "off", "off", "off");       //HPF
            caps
                for(int j=0; j<HPF.size(); j++) {int i = HPF[
                    j];
        I17.push_back(i);}
                for(int i=0; i<I17.size(); i++) myfile << I17
                    [i];
I18 = currentSource("off", "off", "off", "off", "off", "off",
    "off", "off", "source");              //Differentiator
    current IOUT
        IB = currentSource("off", "off", "off", "off", "off",
            "off", "off", "off", "source");    //
            Differentiator current IBIAS
                for(int j=0; j<IB.size(); j++) {int i = IB[j
                    ];
        I18.push_back(i);}
                for(int i=0; i<I18.size(); i++) myfile << I18
                    [i];
I19 = currentSource("off", "off", "off", "off", "off", "off",
    "off", "off", "sink");                //Integrator current
```

```
        for(int i=0; i<I19.size(); i++) myfile << I19[i];
I20 = currentSource("off", "off", "off", "off", "off", "off",
    "off", "off", "source");            //Differentiator
   current IOUT
        IB = currentSource("off", "off", "off", "off", "off",
            "off", "off", "off", "source");    //
           Differentiator current IBIAS
                for(int j=0; j<IB.size(); j++) {int i = IB[j
                   ];
        I20.push_back(i);}
                for(int i=0; i<I20.size(); i++) myfile << I20
                   [i];

//Shift Registers
I21 = shift9bit("off", "off", "off", "off", "off", "off", "
   off", "off", "off"); //9 Bit Shift Register
        for(int i=0; i<I21.size(); i++) myfile << I21[i];
I22 = shift9bit("off", "off", "off", "on", "on", "on", "off",
    "off", "off");     //9 Bit Shift Register
        for(int i=0; i<I22.size(); i++) myfile << I22[i];
I23 = shift9bit("off", "off", "off", "off", "off", "off", "
   off", "off", "on");  //9 Bit Shift Register
        for(int i=0; i<I23.size(); i++) myfile << I23[i];
I24 = shift9bit("off", "off", "off", "off", "off", "off", "
   off", "off", "off"); //9 Bit Shift Register
        for(int i=0; i<I24.size(); i++) myfile << I24[i];
myfile << "00000000";
myfile.close();
```

```
return 0;
}
```

A.2 Python Code for MSP Code Generation

```
#!/usr/bin/env python
# code to create MSP main function code
bitName1 = '#define bit1'
bitName2 = '#define bit2'
bitName3 = '#define bit3'
bitName4 = '#define bit4'
bitName5 = '#define bit5'
bitName6 = '#define bit6'
bitName7 = '#define bit7'
bitName8 = '#define bit8'
bitName9 = '#define bit9'
bitName10 = '#define bit10'
bitName11 = '#define bit11'
bitName12 = '#define bit12'
bitName13 = '#define bit13'
bitName14 = '#define bit14'
bitName15 = '#define bit15'
bitName16 = '#define bit16'
bitName17 = '#define bit17'
bitName18 = '#define bit18'
bitName19 = '#define bit19'

bitString = open('bitStream_abs.txt', 'r')
f = bitString.readline()
```

```
bit1 = f[0:15]
bit2 = f[15:30]
bit3 = f[30:45]
bit4 = f[45:60]
bit5 = f[60:75]
bit6 = f[75:90]
bit7 = f[90:105]
bit8 = f[105:120]
bit9 = f[120:135]
bit10 = f[135:150]
bit11 = f[150:165]
bit12 = f[165:180]
bit13 = f[180:195]
bit14 = f[195:210]
bit15 = f[210:225]
bit16 = f[225:240]
bit17 = f[240:255]
bit18 = f[255:270]
bit19 = f[270:285]

bit1 = int(bit1,2)
bit2 = int(bit2,2)
bit3 = int(bit3,2)
bit4 = int(bit4,2)
bit5 = int(bit5,2)
bit6 = int(bit6,2)
bit7 = int(bit7,2)
```

```
bit8 = int(bit8,2)
bit9 = int(bit9,2)
bit10 = int(bit10,2)
bit11 = int(bit11,2)
bit12 = int(bit12,2)
bit13 = int(bit13,2)
bit14 = int(bit14,2)
bit15 = int(bit15,2)
bit16 = int(bit16,2)
bit17 = int(bit17,2)
bit18 = int(bit18,2)
bit19 = int(bit19,2)

bitStringWrite = open('bitStream_MSP_abs.txt', 'w')

bit1_final = bitName1 + '       ' + str(bit1) + '\n'
bit2_final = bitName2 + '       ' + str(bit2) + '\n'
bit3_final = bitName3 + '       ' + str(bit3) + '\n'
bit4_final = bitName4 + '       ' + str(bit4) + '\n'
bit5_final = bitName5 + '       ' + str(bit5) + '\n'
bit6_final = bitName6 + '       ' + str(bit6) + '\n'
bit7_final = bitName7 + '       ' + str(bit7) + '\n'
bit8_final = bitName8 + '       ' + str(bit8) + '\n'
bit9_final = bitName9 + '       ' + str(bit9) + '\n'
bit10_final = bitName10 + '     ' + str(bit10) + '\n'
bit11_final = bitName11 + '     ' + str(bit11) + '\n'
bit12_final = bitName12 + '     ' + str(bit12) + '\n'
bit13_final = bitName13 + '     ' + str(bit13) + '\n'
```

```
bit14_final = bitName14 + '      ' + str(bit14) + '\n'
bit15_final = bitName15 + '      ' + str(bit15) + '\n'
bit16_final = bitName16 + '      ' + str(bit16) + '\n'
bit17_final = bitName17 + '      ' + str(bit17) + '\n'
bit18_final = bitName18 + '      ' + str(bit18) + '\n'
bit19_final = bitName19 + '      ' + str(bit19) + '\n'

bitStringWrite.write(bit1_final)
bitStringWrite.write(bit2_final)
bitStringWrite.write(bit3_final)
bitStringWrite.write(bit4_final)
bitStringWrite.write(bit5_final)
bitStringWrite.write(bit6_final)
bitStringWrite.write(bit7_final)
bitStringWrite.write(bit8_final)
bitStringWrite.write(bit9_final)
bitStringWrite.write(bit10_final)
bitStringWrite.write(bit11_final)
bitStringWrite.write(bit12_final)
bitStringWrite.write(bit13_final)
bitStringWrite.write(bit14_final)
bitStringWrite.write(bit15_final)
bitStringWrite.write(bit16_final)
bitStringWrite.write(bit17_final)
bitStringWrite.write(bit18_final)
bitStringWrite.write(bit19_final)
```

Vita

James Kelly Griffin was born in Greenville, SC, on 10 November 1990, the son of Mary Griffin and Jimmie Griffin. After graduating from Baylor High school in Chattanooga, TN, he went on to attend the University of Tennessee (UTK) as an Engineering student in 2009. Initially he attended the university as a Materials Science Engineering student, but his interests moved quickly towards electronics. After joining the Electrical Engineering and Computer Science (EECS) department, he developed an interest in microelectronics. Following this path and receiving the Tennessee HOPE scholarship all four years, he graduated with his Bachelors of Science in Electrical Engineering in the spring of 2013.

During James's stint at the UTK, he obtained two internships (Summer 2013 and Summer 2014) with NASA's Jet Propulsion Laboratory (JPL) in Pasadena, California. While there he worked on the James Webb Space Telescope, a next-generation integrated camera, and an oil quality sensor.

During James's senior year at UTK, he spoke with Dr. Jeremy Holleman about graduate school. Dr. Holleman supported the idea of the 5 year MS/BS program, and James began work on the CASP project presented in this thesis. James is expected to graduate in the Spring of 2015. He will move to Dallas, TX in June and start working for Texas Instruments as a Verification Engineer for the Haptic Feedback team.