11-24-2014

Low-Power Analog Processing

Daniel J. White
University of Nebraska-Lincoln, dan@whiteaudio.com

LOW-POWER ANALOG PROCESSING

by

Daniel J. White

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Electrical Engineering

Under the Supervision of Professor Michael Hoffman

Lincoln, Nebraska

November, 2014

LOW-POWER ANALOG PROCESSING

Daniel J. White, Ph.D.

University of Nebraska, 2014

Adviser: Michael Hoffman

This dissertation presents the analog harmonic transform (AHT) and a first implementation in an integrated circuit. The transform is designed for a regular and simple hardware structure. It provides coefficients relating to an input signal's spectrum. These coefficients also have a simple relationship to the signal's Fouriér series coefficients.

The AHT is defined in its ideal form and evaluated for two example signal classification applications. Both military vehicle and bearing fault classification tasks are presented which validate the ability of a neural network to use the AHT coefficients to correctly classify the input signals. Because any real use of the AHT for classification would include various errors, a study determining the required hardware specifications is described.

These specifications are used to inform the design of a hardware implementation of the AHT coefficient generation. A prototype system in a 0.13 µm mixed-signal CMOS process was designed to confirm the new system's utility. The prototype chip included two separate blocks of AHT circuitry along with an on-board custom microprocessor to implement system control and supervision in a 4×4mm die area.

A new digitally-controlled operational transconductance amplifier (OTA) was designed as the core circuit element to support the AHT calculations. The OTA's offset and gain can be calibrated after fabrication to yield lower errors without significant increases in chip area or power consumption. This enables hardware imple-

mentation of applications, such as the AHT, which have strict offset requirements to maintain good system-level performance.

Testing of fabricated prototype chips confirms the ability of digital offset tuning to yield amplifiers having sub-10 mV output-referred offset with both low power and small die area. An algorithm was created to adaptively find the optimum tuning code, needed because the tuning characteristic is not guaranteed monotonic. Testing also confirms the reliable operation of OTAs with extremely large (gigaohm) output impedances. Low-frequency operation with long time constants requires these impedance levels to minimize integration capacitor size, the dominant factor in determining die area.

ACKNOWLEDGMENTS

I would like to thank my advisor Professor Michael W. Hoffman for bearing with two graduate degrees containing a heavy circuit emphasis. His perspective to keep the big picture in the thick of transistors has added to this work and shaped how I view electronic and signal-processing systems. His patience with my extended stay at UNL was appreciated and gave the space for me to find my professional calling.

Professor Sina Balkır and I share a passion for electronics that is built-in. Even though the University of Nebraska is not located in Texas or California, the chip design program has had a core thread of students under his influence. The "outside" perspective has encouraged more system-level views of electronic design which would be more difficult to achieve at other places.

The commission to teach EE 361 for the Electrical Engineering Department proved to be the catalyst to awaken my passion for teaching. It set the path to help me realize the type of institution I wanted to be with. Without that initial trust to have a M.S. student teach a core, junior-level course, I'm sure the current events would not have transpired. The opportunity to adjunct teach a similar electronics course at Nebraska Wesleyan University also provided valuable exposure to my current career path.

The support of my new colleagues in the Electrical and Computer Engineering department at Valparaiso University has been much appreciated. Their absolute dedication to each other and their students is still something to get used to. It is a privilege to be numbered among them. I thank them for taking a chance on me even when I did not have the Ph.D. degree in hand. Before interviewing there, I did not really believe such a place existed.

My daughter Elizabeth's questions about when Daddy will graduate are no

longer a subject. We missed many evenings of play time and running errands together while I worked on the dissertation after work. I thank her for understanding in her own way.

My wife Kathryn's support has been the key factor in this path. She bore with the uncertainty of a husband who didn't truly know what his calling was until a few engineering degrees in. Her work and private private practice provided the family income that was not covered by a teaching assistant's stipend. I thank her for sticking it out and hoping for the day of graduation destiny. Even with the difficulties of a longer time frame I appreciated her willingness to go down this path if it meant a university professor's life and flexible schedule. I hope to repay those lost years of late nights and dual commuting. She is truly a wonderful woman and it is a humbling privilege to have her as a life partner.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Sensor systems typically operate by transducing some physical quantity (e.g. pressure, velocity, flux) into the electrical domain and applying signal conditioning. They then compute "features" of the sensed signal relevant to its cause and make decisions or perform actions as a result of the extracted information. Because digital computers are best suited for information processing and decision-making tasks, there must be an analog-to-digital conversion (ADC) as part of the system-level operation. Where the domain conversion happens in the signal processing chain can affect implementation characteristics such as hardware/software complexity, energy consumption, and service lifetime.

The majority of signal detection and classification schemes first transform the acquired signal into a representation that can reveal significant characteristics in a relatively condensed feature vector. Fouriér and wavelet transforms have proven to generate suitable features for many signal detection and classification tasks including speech recognition, vehicle detection and classification, and bearing fault

detection. Several low power monitoring schemes use statistical parameters as features, including the signal mean, standard deviation, and extreme values. These parameters are obtained directly from the time domain signal to generate signal features for identification, but they are limited in the amount of classification information they contain compared with spectral features. Discrete Fouriér and wavelet features require sampling the signal at relatively high rates prior to processing.

Co-optimization of both the hardware and the signal processing it implements can achieve the best energy efficiency. This efficiency facilitates a wider application and deployment of small sensors. The term "Internet of Things" (IoT) is a current buzzword to describe the (future) ubiquity of these sensors [5]. Data from these "things" can be collected and used for monitoring, control, and analysis.

1.2 Contributions of this Dissertation

The primary contributions of this dissertation are the design, implementation, and qualification of a new analog transform for harmonic signals. More specific contributions include the following:

- Formulation of the analog harmonic transform (AHT) in its ideal form.
- Demonstration of the efficacy of AHT features for signal classification applications.
- Exploration of the effect of hardware-related imperfections on back-end system performance.
- Design, fabrication, and testing of an integrated circuit (IC) to calculate the AHT coefficients in parallel.

Several novel circuit techniques were created to support the AHT's hardware requirements:

- Digitally-controlled OTA for post-fabrication offset and gain calibration.
- Search algorithm for finding the optimal digital tuning code to minimize amplifier offset in the presence of a non-monotonic tuning characteristic.

Fabrication of the AHT integrated circuit prototype was subsidised with a fabrication grant through MOSIS [6]. The MOSIS Educational Program (MEP) research account program accepts proposals to assist in the fabrication of designs without external funding sources. For the 0.13 µm process and 4 mm × 4 mm die area utilized for the prototype, the un-subsidised fabrication charges are currently $75 000, not including wire-bonding and packaging costs.

Peer-reviewed publications related to this work:

[1] D. J. White, P. E. William, M. W. Hoffman, S. Balkir, and N. Schemm, "Analog Sensing Front-End System for Harmonic Signal Classification," in *Proc. IEEE Int Circuits and Systems (ISCAS) Symp*, May 2012, pp. 1155 –1158.

[3] D. J. White, P. E. William, M. W. Hoffman, and S. Balkir, "Low-Power Analog Processing for Sensing Applications: Low-Frequency Harmonic Signal Classification," *Sensors*, vol. 13, no. 8, pp. 9604–9623, July 2013.

[7] D. J. White, M. W. Hoffman, and S. Balkir, "Digital Offset Cancellation for Long Time-Constant Sub-Threshold OTA-C Integrators," submitted for publication in *IEEE Transactions on Circuits and Systems II – Express Briefs*, 2014

1.3 Other Public Contributions

Throughout the course of this work, several open source software packages were used. These packages were used for schematic drawing, circuit simulation, printed circuit board artwork, large simulation data analysis, and interfacing with USB ports.

Several packages had existing or new bugs or lacked functionality directly useful for design and analysis with this project. Due to the open, collaborative nature of these software projects, several patches and improvements were made to the code and were reported to the upstream authors and incorporated into subsequently released versions. A brief summary of these contributions follows:

- *gEDA - gschem* [8]: schematic drawing utility
  - Fix which prevents accidental data loss.
- *gEDA - gnetlist*: schematic data translation and manipulation engine
  - *makedepend*: new backend script which extracts multi-page and multi-level schematic dependencies.
  - *spice-sdb* backend script outputting SPICE netlists from schematics. Added capabilities for parameterized sub-circuits and flexible MOSFET model references.
- *gEDA - refdes_renum*: utility to re-number circuit element references. Safer support for renaming functional devices with multiple symbols.
- *gEDA - PCB*: PCB layout software
  - Patch to allow selection of multiple associated schematic sheet references.

  - *distaligntext*: plugin extension to align groups of text in various orientations.
  - *ElementUpdate*: plugin extension to update existing part footprints with updated library versions.

- *gwave - sp2sp* [9]: circuit simulation waveform viewer, data translation.
  - Option to directly write NumPy [10] arrays to disk including waveform metadata. This allowed processing simulation data which is larger than the available computer RAM via memory-mapped access.

- *libFTDI* [11]: C language library for using FTDI's Hi-Speed USB 2.0 UART/FIFO slave converter chips [12]. Added automatic extraction of in-source library documentation to the python language bindings. This enabled interactive documentation references using *IPython* [13].

- *IEEEtran BibTeX style* [14]: Updated the list of IEEE journal abbreviations to match the official preferred long and short versions.

1.4 Outline of Dissertation

This dissertation is organized into six chapters with Chapter 1 as this Introduction. Background and review of the existing literature on the major topics discussed here is in Chapter 2. Chapter 3 presents the new Analog Harmonic Transform (AHT) as a technique to extract spectral information from a signal. Chapter 4 presents the design of an integrated circuit system which can implement the AHT in hardware including a supervisory on-chip micro-processor. The chapter also includes details of a new wide-range digitally-tunable amplifier which forms the core active element

in a hardware-based AHT system. Chapter 5 details the testing of the prototype chip and the design of an automated testing fixture. Finally, Chapter 6 concludes the dissertation and outlines directions for future work.

Appendix A is a brief tutorial overview of MyHDL, the tool used to design the custom digital sections of the chip. This chip is the second known silicon-verified design using the tool. Appendices B and C contain the source code and schematics respectively of the chip and test fixture.

Chapter 2

Literature Review

2.1 Introduction

Three main issues provide background for this work: minimum energy signal processing, long time constant integrators in integrated circuits, and signal feature extraction. Each section gives an overview of the major work and problems in the subject areas.

2.2 Fundamental Signal Processing Energy Requirements

Low-power signal processing and circuit techniques are labeled as such in comparison to current implementations. The absolute minimum energy required for a given processing task can be derived from first principles and gives a gauge on how well new techniques achieve lowest-power operation.

In the context of signal processing, it is appropriate to normalize the power requirements by the signal bandwidth. This yields a metric in units of energy per

bandwidth. Communications signals are frequently mixed to higher carrier frequencies for various reasons while still only occupying a narrow range of frequencies. The following derivation assumes the signal to be at baseband, or centered around 0 Hz. These are either native baseband signals like audio or communications signals after their center frequencies have been translated to 0 Hz.

2.2.1 Analog

Continuous-time and continuous-amplitude, or analog, circuits always have at least one capacitance C forming a low-pass pole. Because any conductive object separated from another has a related mutual capacitance, this capacitor need only represent the various terminals of the circuit. It will be assumed that the active parts of the circuit operate with perfect efficiency and there is zero power dissipation with no signal present.

Driving an RMS signal voltage of V_{sig} across this capacitor through a non-zero real impedance at a frequency of f requires a power of at least

$$P = 8fCV_{sig}^2. \tag{2.1}$$

Accounting for only thermal noise, such a circuit has a minimum total voltage noise variance of

$$V_N^2 = \frac{kT}{C} \tag{2.2}$$

where k is the Boltzmann constant and T in kelvin. These can be expressed as the minimum energy per cycle and per pole as

$$\frac{P_a}{f} = 8kT\left(\frac{V_{sig}}{V_N}\right)^2. \tag{2.3}$$

This is independent of design or other technique and represents the analog energy limit for a required precision and bandwidth. It is achieved in practice by a simple RC low-pass filter. [15]

Adaptive biasing and operation outside of Class-A bias conditions can be used to assist analog circuits in lowering their total power dissipation without reducing signal ranges. Adaptive biasing refers to varying an amplifier's bias current in response to the input signal, thereby reducing the overall power consumption [16]. Extension of the technique has been on-going [17, 18, 19]. Related to adaptive biasing is the development of high-drive CMOS buffers [20, 21]. Operation in Class-AB and Class-B modes where individual output devices only handle positive or negative-going peaks is common in macro-scale power amplifiers but serves to enhance power efficiency on-chip as well [22, 23].

2.2.2 Digital

To compare digital and analog processing, express the resolution in bits as a ratio of signal to quantization noise power. As with analog processing, assume there is zero static power consumption with no signal present. For an ideal n-bit quantizer with a full-scale sinusoidal input, this yields a signal-to noise ratio of

$$\left(\frac{V_{sig}}{V_N}\right)^2 = \frac{3}{2} \cdot 2^{2n}. \tag{2.4}$$

The number of digital operations, m, required per input cycle depends on the bit-resolution of the signal and the specific algorithm. Each of these unit operations uses an energy of E_{tr} each and a dynamic power consumption per cycle of [24]

$$\frac{P_d}{f} = mE_{tr}. \tag{2.5}$$

For comparison purposes with the simple analog RC low-pass filter, consider the first-order IIR difference equation implementing a digital low-pass filter with input $x[n]$ and output sequence $y[n]$

$$y[n] = a_1 y[n-1] + b_0 x[n]. \tag{2.6}$$

Assume a unit operation is represented by a gate-level transition. A full adder then requires about 3 unit operations per bit, and a full $n \times n$ multiplication uses n^2 full adders. Computation of each sample of the difference equation requires two n-bit multiplications, an n-bit addition, and a n-bit shift. This then is about $3(n^2 + n) + n$ unit operations per sample. Assume also the sample rate is 10 times the signal frequency which allows for filtering of frequencies 2.3 octaves above the desired frequency. Under these conditions, the number of digital gate operations per signal cycle is

$$m = 30n^2 + 40n. \tag{2.7}$$

Figure 2.1 plots the two minimum processing energy curves from the above equations, similar to [15, 24, 25]. It graphically illustrates the relationship between the two processing modes. High precision or large signal-to-noise ratio processing is most efficiently performed with digital techniques. Analog processing, however, retains a theoretical advantage at low SNRs, even in the limit of minimum digital energy usage of $E_{tr} = 8kT$. This crossover occurs at an SNR of 28.2 dB or a digital resolution of 4.1 bit. Relative implementation efficiencies of real circuitry moves this boundary in either direction.

Sub-threshold and asynchronous digital circuit styles achieve the lowest energy usage per operation. At supply voltages in the hundreds of millivolt range, the optimum energy point depends on the relative significance of dynamic and leakage

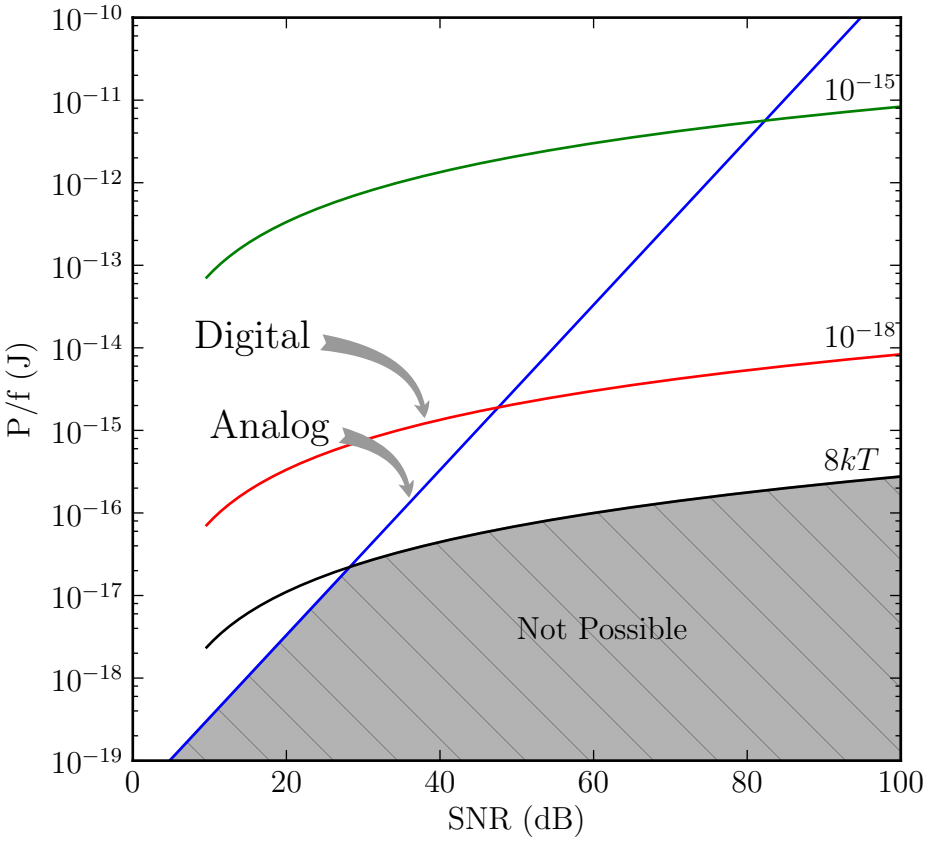


Figure 2.1: Analog and digital minimum energy requirements per pole per bandwidth. The digital curves are for the absolute minimum and several higher E_{tr} values. The hatched region is the unachievable energy/SNR region. The theoretical crossover between domains occurs at an SNR of 28.2 dB or 4.1 bit resolution.

components [26]. Source-coupled logic (SCL) circuits, formerly used for high-speed, find use in low power applications operating in weak inversion [27]. These circuits use PMOS devices in an isolated well and connect the body terminal not to the higher potential source but to the drain node to implement the load resistors of the traditional SCL style. Finally, fully asynchronous digital circuits avoid the clock distribution power overhead of synchronous circuits. Design techniques discussed in [28] allow reliable operation and timing using critical-path replica delay lines. This allows an inherent mitigation of process variations.

2.2.3 Alternative techniques

Systems are not strictly limited to the analog domain of continuous-time, continuous-values or the digital domain of discrete-time, discrete-values [25]. They may also operate in the discrete-time, continuous-value domain used by switched-capacitor circuits or the continuous-time, discrete-value domain used by neural-like systems. Processing approaches in these two domains follow similar derivations for minimum energy requirements.

Analog processing encodes the entire information content of a signal onto a single wire while digital techniques spread the information across several bit wires. Hybrid processing can balance the information transmitted per wire against parallel numbers of wires [25]. The curves of Figure 2.1 suggest the number of information bits transmitted per wire in such a scheme may be around 4 bits per wire.

2.3 Long time constants on ICs

Energy available for signal processing is necessarily limited in very low power systems. For both analog and digital signal processing systems, the power consumption is directly proportional to signal bandwidth. Integration is a common operation in signal processing, appearing at the end of the processing chain for correlation and projection operations. Realization of very long time constants for low-bandwidth signals is difficult for on-chip implementations.

The simplest resistor-capacitor low-pass filter operates as an integrator with an

integration time constant and operating frequency range of

$$\tau = R \times C \tag{2.8}$$

$$f_{\text{op}} > \frac{1}{2\pi RC} \tag{2.9}$$

This relationship serves as a basis for illustrating the issues inherent in building long time constant analog integrators with integrated circuits.

Construction of large-valued linear capacitors on chip is fundamentally limited by available area. For example, the IBM 8RF fabrication process limits the total capacitor area on a die to $2 \times 10^6\,\mu\text{m}^2$. Dual-MIM capacitors are the process' highest density capacitors at $4.10\,\text{fF}/\mu\text{m}^2$ This yields a design rule limited maximum total chip capacitance of 8.2 nF. Achieving a R-C integrator time constant of 1 s then requires at least a 121 MΩ resistance. Even at minimum width, this represents a resistor approximately 97 mm long and is clearly impractical. Available and practical capacitances in the process are only several hundred picofarads, which increases the need for extremely high equivalent resistances in order to achieve very long time constants.

Electro-chemical super capacitors can achieve charge storage densities many times greater than parallel-plate capacitors. Traditionally, these types of capacitors have not been possible to construct in silicon semiconductors because of chemical incompatibility. Recently, a process compatible with silicon substrates has been demonstrated [29]. That work suggests future IC processes with the capability of integrated super capacitors. For the time being, on-chip linear capacitors suitable for integrator construction are limited by available die area and high permittivity insulators.

Due to the inherent issues with fabricating large valued resistors, a common

alternative is to use operational transconductance amplifiers (OTA) to realize very low conductances or very high resistances. These OTA-C integrators have time constants of

$$\tau = \frac{C}{G_m} \tag{2.10}$$

where G_m is the OTA's transconductance. This suggests the need to design OTAs with transconductances on the same order as the available capacitance, e.g. 100 pS G_m and 100 pF capacitor for a 1 s time constant.

2.3.1 Low-G_m OTA Design

Since large capacitances are not realizable on integrated circuits, the design of large on-chip time constants focuses on lowering the transconductance of the associated amplifiers. Design issues such as achievable (low) G_m, small bias currents, linear input ranges, and statistical variations converge at these low values, giving a range of techniques. Though generally closely related, these issues will be discussed separately in the following sections.

2.3.2 G_m Reduction

Individual MOS transistor g_m scales with $(W/L)\sqrt{I_D}$ when biased into strong inversion. Reducing the bias current, in combination with the transistor aspect ratio, causes the FET to begin to operate in the weak inversion mode where $g_m = I_D/(nV_T)$ [30]. This decouples a transistor's size from its transconductance, leading to a focus on bias current reduction.

One of the earlier techniques aside from bias current reduction was an output current mirror division technique [31]. In this technique the goal is reduction of die area compared with previous techniques by using a traditional OTA biased in

strong inversion driving a current mirror with an input:output ratio of approximately $37\,500:1$. Arnaud, Fiorelli, and Galup-Montoro generalized and extended this mirror division technique using an OTA core operating in weak inversion [32]. Constructing each mirror input and output branch transistor from n/m series-parallel unit transistors allows well-controlled division factors with regular layout patterns.

Another reduction technique cascades amplifier stages in a g_m, $1/g_m$ series as implemented in [33, 34]. The $1/g_m$ stages are implemented by readily-available small-value resistors. With a variable bias current, this attenuation technique was shown to realize time constants ranging from 1 s to 17 s [34].

Finally, the implementation of large effective resistances has been used within OTAs. This technique applies the input signal across a circuit exhibiting an approximately linear V/I relationship made from one to several MOS transistors. Within an OTA, this is an extreme form of source degeneration where the amplifier's total G_m reduces to the effective conductance of the degeneration element. MOSFETs used as linear resistors provide a natural control terminal for large variations in conductance. Kwan and Martin [35] use a novel circuit de-coupling the MOS-resistor's gate voltage from the control port to allow a tunable, floating, resistor. Later variations from [36, 37, 38] extend the concept to progressively lower bias currents. Tajalli and Leblebici [38] retain the floating property and exploit weak inversion operation by connecting the transistors' body terminal to the drain instead of source.

The selection of a G_m-reduction technique is influenced by its sensitivity to random errors such as process variations. Pachnis et al. [39] investigated this sensitivity for the three main techniques of current mirror division, current cancellation, and cascaded $g_m - 1/g_m$ stages. It was shown that current division was the most reliable in terms of analog mismatches.

2.3.3 Small Bias Currents

The direct way to reduce an OTA's transconductance is to reduce its bias current. In weak inversion operation, transconductance is directly proportional to bias current [24]. Achieving small G_m with this strategy then focuses on the reliable generation of pA to nA currents.

Dividing a master reference current to very low values is, in principle, possible down to the magnitude of the transistor drain-body p-n junction's reverse leakage current. Linares-Barranco et al. discuss techniques to achieve pA-range currents in practice [40]. These techniques center on circuits which create negative gate-source potentials to bring MOS transistors closer to a true "off" state by raising the source terminal's potential. They also emphasize the difficulty of measuring on-chip pA and fA currents by bringing those signals off-chip as the pad connections themselves leak more than the entire measured signal magnitude.

Switched-capacitor circuits can be used to assist in generating bias currents as shown in [41]. In that paper, the resistor used in a constant-g_m bias generator is replaced by its switched-capacitor equivalent. This change allows easy bias current control by varying a clock frequency.

The ability of a MOS device to operate as a charge pump was first observed in 1969 [42]. Only relatively recently (2003) has this charge pumping effect been put to use in the generation of ultra-low current sources [43]. These structures utilize the normally undesirable charge-trapping behavior of an imperfect gate oxide/channel interface. Again, this allows varying the bias current easily and over wide ranges by a clock frequency. An OTA which uses the interface-trap charge pump for its bias source was subsequently demonstrated by Becker-Gómez, et al. for the design of a sub-hertz low-pass filter [44].

Continuous tuning of bias current and gain has also been achieved using floating-gate transistors [45]. Those transconductors were constructed with MOS second generation current conveyors (CCII) [46, 47] and MOS resistors.

2.3.4 OTA Linearization

In part because OTA integrators are used in open-loop configurations, the linear input range of an amplifier is frequently extended by several strategies. A direct linearization technique is to merely attenuate the input signal, therefore increasing the overall input linear range along with a proportional reduction in gain [48]. Firth and Andreou provided an overview of the major linearization techniques of degeneration via a single diffusor (resistance), degeneration via symmetric diffusors, and asymmetric differential pairs [49].

The first natural and common technique is the addition of degeneration resistance in the transconductor [35, 49, 36, 50, 38]. Degrauwe, et al. also used degenerated current mirrors to assist in linearizing amplifiers for switched-capacitor circuits [51]. In complement to G_m reduction, these added resistances need to be large for good linearization and small transconductance.

Adaptive or signal-dependent variation of the bias current can also extend the linear range [52, 36, 37, 53, 50]. These generally increase the core transconductor's current at larger signal amplitudes to avoid the normal g_m drop. Circuits for operation in both strong inversion [52] and weak inversion [48] use this type of linearization.

The third major technique is having several differential transconductors in parallel with asymmetric sizes [49, 54]. Koziel and Szczepanski use three cross-connected pairs, each biased at different currents, to strategically yield a wider overall linear

input range [54].

2.3.5 Offset, Mismatch, and Process Variations

Though an omnipresent consideration in all analog integrated circuit design, the impact of device mismatch and process-related parameter variations is enhanced at very low currents. MOS transistor drain current mismatch reaches its maximum in weak inversion [55]. These random variations limit the un-corrected precision of amplifier circuits in parameters such as gain and offset.

The noise and offset characterization for design is well-described by [56, 57, 58, 59] and others. Recent work demonstrates a ring-gate layout technique which specifically minimizes wafer-to-wafer matching in modern CMOS processes which use shallow trench isolation [60].

Low-frequency $1/f$ noise and offset can be mitigated by auto-zeroing, correlated double sampling, and chopper stabilizing techniques [61]. The chopper and other integrator based solutions operate continuously and are used in a wide range of applications such as radio receivers [62] and very low offset amplifiers [63].

The major technique for offset reduction in low-frequency amplifiers is to increase the gate area of matching transistors. This reduces the mismatch related to process variations to a lower fraction of the device's electrical parameters, reducing by the square-root of area [57]. Unfortunately, this technique cannot achieve arbitrary reduction in offset, which may still be too large for certain applications. Systems whose performance is critically dependent on amplifier offset may not be practical for low-frequency, low-power, on-chip implementation. This suggests the need for additional techniques to mitigate offset to retain the benefits of integrated circuit construction.

2.4 Signal Feature Extraction

Correlation-based transforms and coefficient series representations makeup a large class of available signal analysis tools. The results of these can be used as sets of signal features for the extraction of signal-related information. These analyses for continuous time domain input signals $s(t)$ have the form

$$a_i = K_i \int_{t_0}^{t_1} s(t) \cdot g_i(t - t_0) dt \tag{2.11}$$

where i is the coefficient index, K_i a scaling constant, and $g_i(t)$ a basis function for the particular transform. Similarly for discrete time signals

$$b_i = \kappa_i \sum_{n=n_0}^{n_1} s(n) \cdot g_i(n - n_0). \tag{2.12}$$

The properties of the transform are determined by the chosen set of basis functions g_i. This form of signal decomposition requires integration and multiplication, common operations possible in analog circuitry. As such, the calculation of the transform coefficients is a candidate for direct electronic implementation.

2.4.1 Fouriér Methods

By far the most popular class of spectral feature extraction methods is the Fouriér representation. Computation of Fouriér series coefficients is tractable in an analog circuit due to readily available multipliers and integrators. One example using analog computers is described in [64].

Discrete Fouriér transforms (DFT) and its more computationally efficient fast fourier transform (FFT) are used in varied applications. Hardware-based FFT

circuits minimize the energy required to perform the computation or maximize the speed of computation. Architectures for computing the real-valued FFT re-use logic blocks and other common circuits [65].

Analog circuits have been used to calculate the DFT with various strategies [66, 67, 68, 69, 70]. Replacing the FFT butterfly structure with a bank of tunable transconductors before analog-to-digital conversion gives a net dynamic range improvement for OFDM receivers [66, 68]. Similar work using analog butterfly implementation reduces the number of unique weighting factors to just three. The three weight factors were reduced to small rational divisors readily implemented by current mirror unit cells of 3/10, 7/10, and 9/10 [67]. Subsequent improvements on the technique uses current mirrors for all scaling and signal copying [69]. Floating gate-based arrays have also been used to implement analog DFTs [70]. This structure also uses current-mode operation for the addition of signals within the array.

Passive computation of the Fouriér transform using an LC lattice was simulated in [71]. There, the applied signal was modulated onto a high-frequency carrier and applied to one edge of the lattice. Analogous to thin-slit diffraction, the signal propagates across the lattice and arrives at the opposite edge as the spatial Fouríer transform of the spatial input. Correcting the phase shifting between thin-slit diffraction and transform is a "lens" in the middle of the lattice where the node capacitance is greater, with similar properties as the optical version.

2.4.2 Square Functions

Square basis functions using amplitudes of ± 1 only, possibly scaled by a constant. The Walsh set of waveforms are ordered according to sequency, or sign changes per transform length [72]. Arranged in sequency order and arranged into a matrix,

Walsh functions then become the Hadamard transform.

These waveforms may be generated with digital circuitry and computation of the basis function projections do not require multiplication. Circuit implementations of the transform utilize this property [73]. The functions also find use in communications context, for example, in coding the forward channel of the CDMA cellular phone scheme [74].

It is not the case that Walsh waveforms are merely the sign portion of a sinusoid. This is true for low sequency functions but not in general. Walsh waveforms form an orthogonal set just as the set of harmonic sinusoids of the Fouriér series, while waveforms constructed by the sign of a sinusoid do not form an orthogonal set.

The relationship between Walsh functions and their Fouriér series was described in [75]. A small number of superimposed square waveforms may be used to generate signals which have a suppressed harmonic. For example, the digital modulation scheme "HD" used in broadcast FM radio can result in an increase in noise for common sub-carrier analog demodulation techniques. These use a square wave locked to twice the 19 kHz pilot tone for demodulation of the stereo $L - R$ channel centered at 36 kHz. The digital modulation sidebands have significant energy at the fifth harmonic of this waveform 190 kHz causing the energy to be mixed with the stereo audio information. Using Walsh waveforms, the third and fifth harmonic of the mixer oscillator waveform is suppressed, eliminating this self-noise source [76].

Another use of Walsh functions for harmonic cancellation is in DC-AC power inverters [77, 78]. These systems arrange pulse-width modulation waveforms in Walsh sequences, both two state waveforms. The converse conversion, with an AC source can make use of active loads to control and minimize the current harmonics drawn from the source [79].

2.4.3 Compressed Sensing

The relatively new field of compressed sensing utilizes random-like and non-orthogonal basis functions. Through various back-end processing techniques, the outputs of these projections can be used to reconstruct a limited amount of information from the input signal. Its advantage is the ability to extract information spread across a wide bandwidth but containing a low information rate [80]. Nyquist sampling of these signals is prohibitive in relation to the information content.

The basis functions or chipping sequences for the random demodulator of [80] are operated at greater than twice the maximum signal frequency to achieve the required randomness. Constructing the modulating waveforms as run-length limited sequences can increase the available sensed bandwidth. This increased bandwidth comes at the expense of signal sparsity [81].

Circuit implementations have focused on random ± 1 basis sequences [82, 83, 84]. Circuit simulations for an implementation for detecting a $100, 200, 300$ MHz signal group used a 2 GHz basis sequence [82]. The system of Chen, et al. uses a PN sequence of up to 3 GHz and parallel projection channels generating multiple outputs in successive integration windows [83]. P parallel paths, each with Q windows, are equivalent to $P \cdot Q$ single-window paths, thus trading hardware area and complexity.

Similar to the successive windows of [83] is the parallel segmented compressed sensing scheme by [85]. Following its introductions, the authors simulated the system performance under circuit errors such as timing jitter and settling time. With a least mean-squared algorithm proposed to calibrate realized systems, they found that training the system on dense instead of sparse training signals accelerated the error convergence [86].

A modulated wideband converter system named “Xampling” is described in

[87, 84]. This scheme modulates several spectrum slices into a composite down-converted signal which is then sampled at a rate between the signal bandwidth and the signals' maximum frequency. Instead of a pseudorandom ± 1 sequence, the basis functions are only required to be periodic over the integration time with at least as many transitions as there are parallel channels [87].

2.5 Conclusion

For low precision computation, analog computation retains a theoretical energy efficiency advantage. Maintaining low power consumption also requires low bandwidth and low frequency operation. Circuit design at these low frequencies raises issues such as practical implementation and reliable operation with the larger errors. Techniques are needed to further reduce these amplifier errors, especially offset, to enable integrated circuit implementations.

Extraction of signal features centers on spectral techniques which all include an integration stage whose bandwidth is related to the received signal characteristics. Square basis functions and modulating waveforms are convenient to generate and have exhibited properties which relate well to several processing schemes. Design choices in light of the current art are discussed in Chapter 4.

Chapter 3

Analog Harmonic Transform

3.1 Introduction

Comparisons of the power and area required to implement signal-processing operations at a given precision between analog or digital integrated circuitry have been described by [24, 25], and others. Figure 3.1 plots the shape of power requirements for digital and analog computation as precision, given as SNR, is varied. Both power and area scale linearly with SNR for analog and as $\log_2(\text{SNR})$ for digital operations. The magnitude and crossover point is dependent on factors such as task, technology, and skill level of the designers [25].

Clearly from Figure 3.1, applications requiring high precision computation are best served by digital systems. However, systems which can tolerate lower SNRs can utilize an analog implementation's fundamental energy advantage, especially with SNRs below about 40 dB, indicated by the shaded region. The challenge for energy-efficient signal processing systems is then to find algorithms and architectures which maintain good system-level performance with low precision or noisy computations. The AHT focuses on the feature extraction phase of harmonic sig-

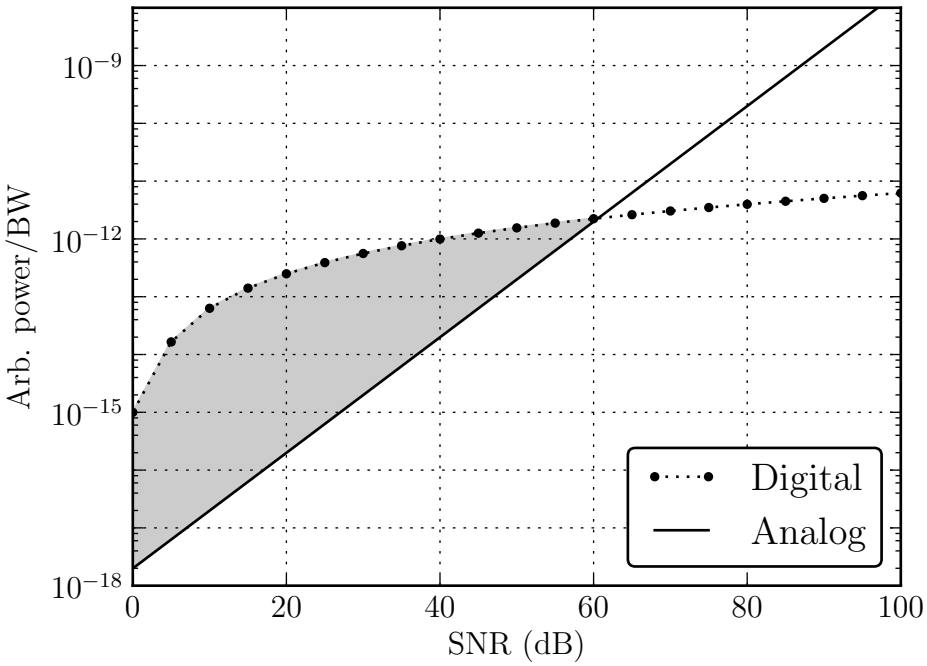


Figure 3.1: Analog and digital power requirements for signal processing as a function of SNR. Power is in arbitrary units and normalized to signal bandwidth.

nal classification tasks to take advantage of analog techniques, as suggested by the energy usage data from [88].

3.2 Harmonic Signals

Sensed harmonic signals originating from rotating machinery or other periodic phenomena may be modeled as a sum of two components: a deterministic harmonic signal model approximating the revolving parts and a non-deterministic component approximating all other components. Selective features extracted from these signals are sufficient for signal/source discrimination as shown in [89, 90, 91]. A harmonic

signal can be described as

$$x(t) = \sum_{k=1}^{M} \alpha_k \cos(2\pi k f_1 t + \phi_k) + n(t) \tag{3.1}$$

where α_k and ϕ_k are the amplitude and phase of the kth deterministic harmonic component, respectively, f_1 is the fundamental frequency (FF), M is the largest harmonic number, and $n(t)$ is the non-deterministic signal component. The signal's harmonic part is therefore completely defined by $2M+1$ parameters. If the FF and number of harmonics are known, the optimum solution in additive white noise for estimating the amplitude and phase set is the Least Squares (LS) solution, i.e. the signal's Fouriér series (FS) coefficients.

An alternate solution for estimating the harmonic parameters is to locate spectral peaks that maintain a line series, presented in [92, 93] as Harmonic Line Association (HLA). HLA, however, requires narrow frequency resolution (long FFT) and a complex approach for the selection of harmonically related peaks. In contrast, the time domain harmonics' amplitudes (TDHA) method extracts harmonic signal information with lower complexity than the FFT [91] but still operates in the digital domain and requires multiplication. Described here is a new transform for calculating these harmonic parameters well-suited for efficient analog-domain implementation.

3.3 Analog Basis Projection

To estimate the kth harmonic's amplitude α_k and phase ϕ_k, the input signal is first low-pass filtered to Mf_1 and then projected onto a pair of quadrature basis

functions with frequency kf_1 and integrated over $T = 1/f_1$ as

$$\begin{aligned} y_{Ik} &= \int_0^T x(t)\ \psi_{Ik}(t)dt \\ y_{Qk} &= \int_0^T x(t)\ \psi_{Qk}(t)dt \end{aligned} \tag{3.2}$$

where $k \in \{1, 2, \cdots, M\}$, with basis functions given as

$$\begin{aligned} \psi_{Ik}(t) &= \mathrm{sgn}\big(\cos(2\pi k f_1 t)\big) \\ \psi_{Qk}(t) &= \mathrm{sgn}\big(\sin(2\pi k f_1 t)\big), \end{aligned} \tag{3.3}$$

$$\mathrm{sgn}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}. \tag{3.4}$$

The AHT scheme takes the Fouriér series' sinusoidal basis functions and uses only their signs as shown in Equation (3.3). This change greatly simplifies analog implementation of the projection implementation as described in Chapter 4. Figure 3.2 plots the basis pairs for harmonic numbers $1, 2$, and 5 with $f_1 = 1/T$.

3.4 Feature Extraction

The harmonic part of the signal in Equation (3.1) can be expressed as in-phase and quadrature components by

$$x(t) = \sum_{k=1}^{M} \alpha_k \cos(\phi_k)\cos(2\pi k f t) - \sum_{k=1}^{M} \alpha_k \sin(\phi_k)\sin(2\pi k f t) \tag{3.5}$$

Substituting Equations (3.3) and (3.5) into (3.2) and evaluating the integration

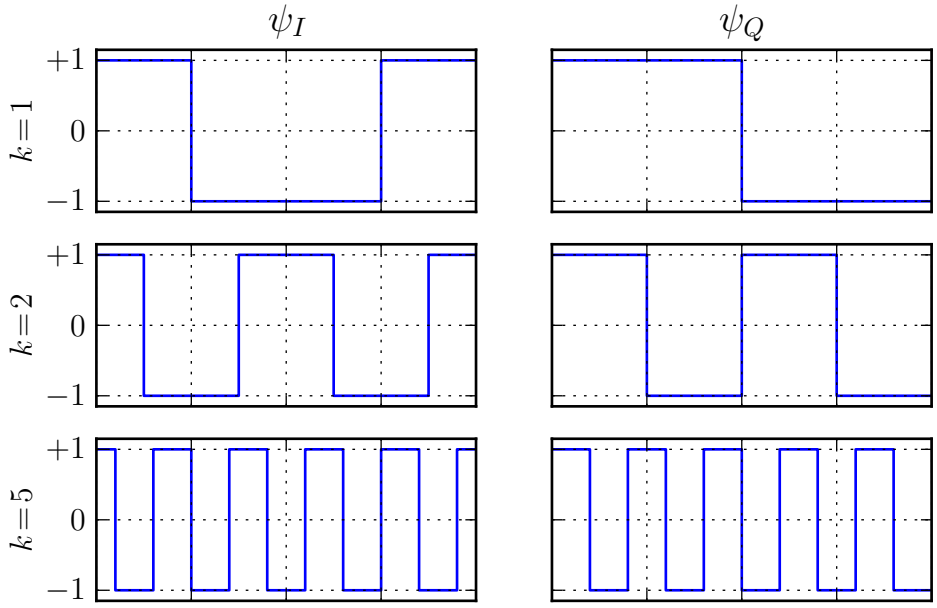


Figure 3.2: Quadrature basis function waveforms for $k=1,2,5$.

gives

$$\begin{aligned} y_{Ik} &= \frac{T}{2\pi}\sum_{p=1}^{M}\frac{\alpha_p\cos(\phi_p)}{p}\sum_{r=1}^{2k}(-1)^{r-1}\sin\left(\frac{(2r-1)\pi p}{2k}\right) \\ y_{Qk} &= \frac{-T}{2\pi}\sum_{p=1}^{M}\frac{\alpha_p\sin(\phi_p)}{p}\sum_{r=1}^{2k}(-1)^{r-1}\cos\left(\frac{2r\pi k}{2p}\right) \end{aligned} \tag{3.6}$$

The result of each in-phase and quadrature projection therefore represents the sum of scaled in-phase and quadrature harmonics' amplitudes, respectively. To better illustrate the relationship between the harmonic parameters (α_k, ϕ_k) and the basis projections (y_{Ik}, y_{Qk}), the parameter sets can be represented in column vector format as

$$\begin{aligned} \boldsymbol{y_I} &= \{y_{Ik}\} & \boldsymbol{y_Q} &= \{y_{Qk}\} \\ \boldsymbol{a_I} &= \{a_{Ik}=\alpha_k\cos(\phi_k)\} & \boldsymbol{a_Q} &= \{a_{Qk}=\alpha_k\sin(\phi_k)\}\,. \end{aligned} \tag{3.7}$$

The relation may then be written as

$$\boldsymbol{y_I} = \frac{T}{2\pi}\boldsymbol{U_I}\,\boldsymbol{a_I} \tag{3.8}$$

$$\boldsymbol{y_Q} = -\frac{T}{2\pi}\boldsymbol{U_Q}\,\boldsymbol{a_Q} \tag{3.9}$$

with

$$\boldsymbol{U_I} = \begin{bmatrix} 1 & 0 & -\frac{1}{3} & 0 & \frac{1}{5} & 0 & -\frac{1}{7} & 0 & \frac{1}{9} & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & -\frac{1}{3} & 0 & 0 & 0 & \frac{1}{5} & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{3} & 0 & \dots \\ \vdots & & & & & \ddots & & & & & \vdots \\ 0 & 0 & 0 & \dots & \dots & \dots & \dots & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

$$\boldsymbol{U_Q} = \begin{bmatrix} 1 & 0 & \frac{1}{3} & 0 & \frac{1}{5} & 0 & \frac{1}{7} & 0 & \frac{1}{9} & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{5} & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & \dots \\ \vdots & & & & & \ddots & & & & & \vdots \\ 0 & 0 & 0 & \dots & \dots & \dots & \dots & 0 & 0 & 0 & 1 \end{bmatrix} . \tag{3.11}$$

The in-phase and quadrature amplitude vectors may then be calculated by

$$\boldsymbol{a_I} = \frac{2\pi}{T}{\boldsymbol{U_I}}^{-1}\boldsymbol{y_I} \tag{3.12}$$

$$\boldsymbol{a_Q} = -\frac{2\pi}{T}{\boldsymbol{U_Q}}^{-1}\boldsymbol{y_Q}\,. \tag{3.13}$$

Individual harmonic magnitude and phase estimates may be calculated from the

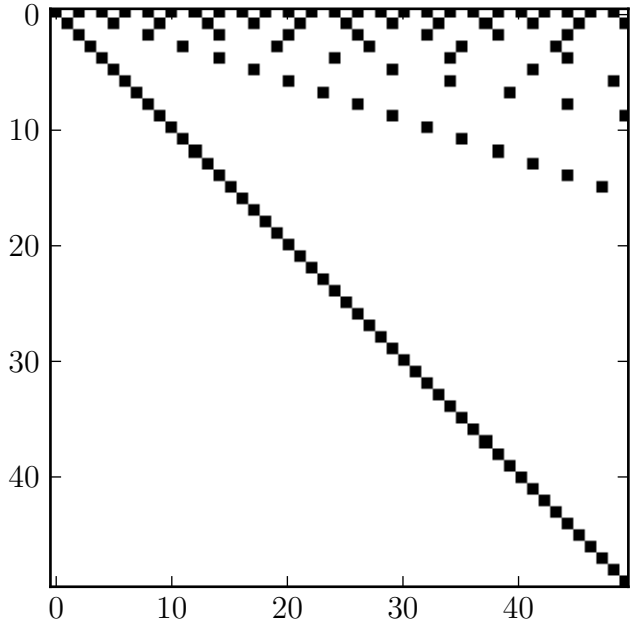


Figure 3.3: Structure of matrices $U_{I,Q}$, non-zero entries are black, and all main diagonal entries are 1.

rectangular parameters via

$$\alpha_k = \sqrt{a_{Ik}^2 + a_{Qk}^2} \qquad \phi_k = \tan^{-1}\left(\frac{a_{Qk}}{a_{Ik}}\right). \tag{3.14}$$

3.5 Computational Considerations

Matrices $\boldsymbol{U_I}$ and $\boldsymbol{U_Q}$ are sparse, upper-triangular, and unipotent. These properties ensure their inverses are well-conditioned and independent of any signal characteristics for a given M. From the upper-triangular property, a_{Ik} and a_{Qk} may be calculated by a simplified back-substitution while the sparsity greatly reduces the number of computations actually required. Figure 3.3 shows the non-zero matrix entries to illustrate the sparse and well-structured properties of the matrix; all main diagonal entries are 1.

Table 3.1: Comparison of digital real-valued operations to compute Fouriér series coefficients from the AHT back-substitution ($M = 32, 64$) or real-input FFT algorithm ($N = 64, 128$). From [3].

FS	AHT		RSR-FFT		Savings
Coeff's	mult+add	Total	mult+add	Total	Total
1...32	74+74	148	98+420	518	**71%**
3...32	52+52	104	98+420	518	**80%**
1...64	194+194	388	258+1028	1286	**70%**
4...64	82+82	164	258+1028	1286	**87%**

Note that the projection operation of Equation (3.2) requires no multiplications in practice due to the basis functions' shape. This is in contrast with the FFT and other methods like TDHA. Multiplication is only required to back-calculate the equivalent FS coefficients from the $\boldsymbol{y_I}$ and $\boldsymbol{y_Q}$ vectors. Calculating a pseudo-harmonic amplitude with the y_{Ik} and y_{Qk} values directly as $\sqrt{y_{Ik}^2 + y_{Qk}^2}$ bypasses even this operation and may be sufficient for many classification applications [4].

To illustrate the digital computation savings, Table 3.1 lists the number of real-valued multiplications and additions to recover the Fouriér series coefficients from the AHT projected values and compares with the equivalent FFT operation to yield the same coefficient set. The highest $^2/_3$ of the FFT coefficients are directly known from the corresponding AHT coefficients. Because the input data is purely real, the number of operations required for the FFT may be reduced by employing a specialized algorithm. The efficient Real Split-Radix FFT (RSR-FFT) algorithm discussed in [94] is therefore used for comparison purposes. Operation counts for the FFTs assume 3 multiplications and 3 additions per complex multiply by using Gauss' algorithm and observing that two of the additions may be pre-computed due to the constant twiddle factors [94]. The AHT back-substitution uses real values only.

Unlike the FFT, which produces all coefficients in the signal's bandwidth, it

is not necessary to calculate all the AHT coefficients if the back-end application will not use them. For example, if the lower 5% of the coefficients (3 harmonics for $M = 64$) do not increase the system-level performance, they need not be calculated. Due to the $U_{I,Q}$ matrix's structure, this would reduce the number of multiplication and addition operations by 30% and 58%, respectively, to calculate the top 95% of the coefficients. Advanced energy-aware detection and classification algorithms may selectively disable the unneeded harmonic projection channels for a further reduction in energy usage. Such fine-grained, adaptive energy management is not possible when generating frequency coefficients using FFT only.

Finally, it has been shown in [4] that calculating a pseudo-amplitude as $\sqrt{y_{Ik}^2 + y_{Qk}^2}$ in place of the FS amplitudes results in minimal system-level classification performance degradation. This means that the AHT coefficient to FS coefficient calculation may be skipped all-together, with its attendant energy savings.

3.6 Ideal System Evaluation

Two case studies are presented to validate the use of the AHT for harmonic signal classification applications. The first is classification of vehicle types from acoustic recordings while the second is the identification of machine bearing faults from attached accelerometer signals. System classification performance for both applications will be shown to be comparable to existing studies using considerably more complex front-end processing techniques on the same data sets. These studies were first presented in [4]; the major contribution in this work is a thorough discussion of the hardware error modelling for system-level simulation of the classification system in Section 3.7.

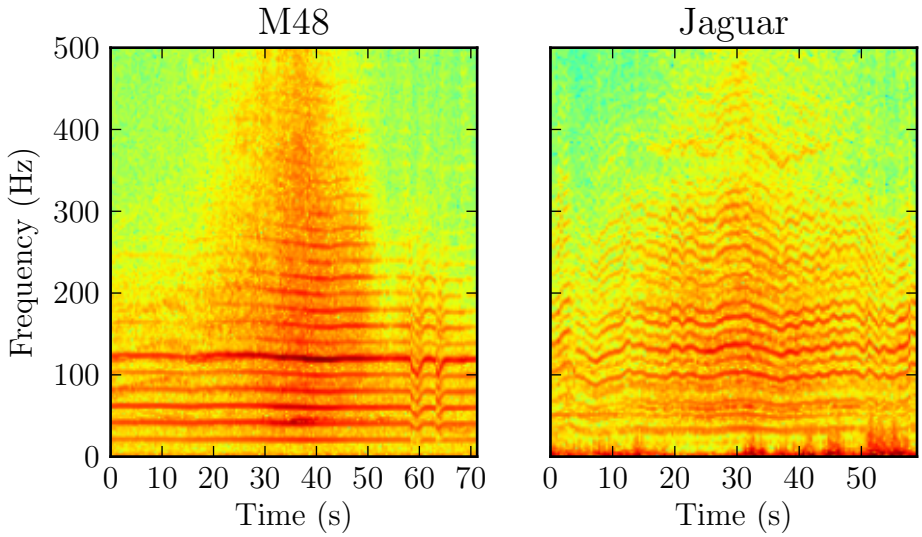


Figure 3.4: Time-Frequency acoustic response of two heavy-weight, tracked military vehicles.

3.6.1 Case Study I: Classification of military vehicles

Monitoring large regions for military vehicle activity for peacekeeping purposes is an application well-suited for wireless sensor modules. The acoustic emissions of such ground vehicles contain a wealth of information for purposes such as classification [95]. The main sources of acoustic emissions are from the engine and propulsion mechanism and can be approximated using a harmonic signal model [96, 97].

For ground vehicles, the engine-related FF of the acoustic signal typically lies within the range 8 Hz to 20 Hz [93, 96]. The time-frequency responses of sample runs of the acquired acoustic signals from two tracked military vehicles passing by a sensor node are shown in Figure 3.4. The harmonic structure and the time-varying nature of the signals are apparent.

The acoustic data of 9 different vehicles covering all combinations of wheeled/tracked and heavy/light-weight types (Leopard 1, Leopard 2, Wiesel, Jaguar, M48, Fuchs,

Hermelin, Unimog, and Mercedes-Benz 1017) were recorded by the Bochum Verification Project (BVP) during verification experiments in 2000 [89, 98]. The researchers equipped each of two stations with acoustic and seismic sensors to record signatures of the vehicles driving along four different lanes (paved and unpaved). Only the acoustic data is considered in this case study since the harmonic structure is more pronounced. Each run represents one vehicle passing by two stations placed 101.4 m apart on opposite sides of the lanes. More than 365 runs were recorded at variable speeds, from different directions and on different surfaces. The acoustic data was originally sampled at 20 kHz and re-sampled to 5 kHz prior to detection and feature extraction. Each recording was started manually when the vehicle entered within 200 m of the sensor stations.

To determine the presence of a vehicle, an adaptive Constant False Alarm Rate (CFAR) detector [91] was used to output a decision every 0.5 s, based on the average energy level of the acoustic signal. As an energy-based detector, it also detected events with no clear harmonic signature. Note that these events are included in the classification rate in the results presented. The number of detected event windows per run then depended on the vehicle itself and its speed. At normal speeds, medium and lightweight vehicles were detected within 50 m of the sensor station, while heavyweight tracked vehicles were detected beyond 100 m. Due to this variation, the total number of detection events per vehicle ranged between 1200 and 5500. Averaging over the nine vehicles, 96.64% of the acoustic energy for more than 32000 detected events was found to be in the range 0 Hz to 250 Hz.

For classification, a three-layer feed-forward neural network (FNN) was utilized with sigmoid neuron transfer functions. Harmonic amplitudes from each window (a_k or α_k) were used as the feature vectors and fed as the FNN input layer. Forty hidden neurons made up the middle layer, while the output layer consisted of 9 neu-

rons representing each of the vehicles. The network was trained using the resilient back-propagation algorithm (Rprop) [99]. For all results, 1/3 of the total number of detected events were randomly selected for network training with the remaining 2/3 used for testing.

Harmonic amplitudes were calculated using the AHT of Section 3.3 with three harmonic models each with an assumed $\widehat{\mathrm{FF}} = 5\,\mathrm{Hz}$ (less than half the expected range of 8 Hz to 20 Hz) and $M = \{25, 50, 100\}$. This approximated the deterministic signature in the bands 5–125 Hz, 5–250 Hz, and 5–500 Hz, respectively. Single-event detection, false alarm, and classification rates are shown in Table 3.2. Military vehicle acoustic signature single-event classification rates $\geq$ 80% are considered excellent [100].

From Table 3.2, we conclude the transform was capable of extracting distinctive features sufficient for acceptable vehicle discrimination. Note that this was without estimation of the fundamental frequency or exact number of harmonics. For constant spectral resolution (5 Hz), the capability to discriminate among military vehicles using harmonic amplitudes increased with bandwidth (increasing M) up to 250 Hz. Further increase in bandwidth beyond 250 Hz gave little improvement. This matches with the sample spectrograms in Figure 3.4 which show little signal energy above 250 Hz except when the vehicle is passing very near the station. Separate studies reducing the spectral resolution below 5 Hz for the same bandwidths did not yield significantly better classification rates considering the increase in feature vector length [4].

Previously published classification results from this data set include the original research [89] and more recent work [91]. The first study extracted spectral information with the FFT and employed learning vector quantization (LVQ) for classification. For 5 Hz resolution, a 88.02% average correct single-event classification rate

Table 3.2: Military vehicle single-event detection, false alarm and classification rates, from [4]

	Number of harmonics, M		25	50	100
	Vehicle	Type	$\widehat{\text{FF}} = 5$ Hz		
Detection rate (%)	Leopard 1	TH	92.87	96.20	95.83
	Leopard 2	TH	81.25	90.91	91.85
	Jaguar	TH	80.09	90.16	88.79
	M48	TH	88.09	95.62	95.47
	Wiesel	TL	77.42	82.95	86.62
	Fuchs	WH	81.89	88.79	87.66
	Hermelin	WH	53.04	66.77	64.82
	MB1017	WL	49.39	59.02	65.94
	Unimog	WL	56.81	64.20	63.77
False alarm rate (%)	Leopard 1	TH	1.33	0.89	1.09
	Leopard 2	TH	5.15	2.48	2.69
	Jaguar	TH	3.87	2.30	2.14
	M48	TH	2.08	0.63	0.55
	Wiesel	TL	3.29	2.27	2.21
	Fuchs	WH	2.49	1.98	1.64
	Hermelin	WH	1.58	1.26	0.89
	MB1017	WL	2.10	1.54	1.70
	Unimog	WL	1.04	0.56	0.59
Classification rate (%)			80.00	87.73	88.14

Type key: T=tracked, W=wheeled, H=heavy-weight, L=light-weight

was achieved. Both estimated and fixed fundamental frequencies were used with the time-domain harmonic amplitude (TDHA) spectral extraction of [91]; the classification rate with a fixed FF of 5 Hz was 85.20% while using an estimated $\widehat{\text{FF}}$ close to 5 Hz raised the rate to 90.38%. Feature vectors obtained from the first 50 coefficients of a 5 kHz, 1024-point FFT (4.88 Hz resolution) were also evaluated in [91] and gave a 88.02% classification rate. These results are comparable to the 88.14% single-event classification rate achieved here but with substantial reduction in computation complexity especially compared with FFT. Extensive cross-validation studies were done between randomly-selected sets of events for training and testing similar to those done in [91]. The processing approach described in this work

presents comparable performance with the promise of much lower power requirements.

3.6.2 Case Study II: Identification of bearing faults in rotating machinery

Induction motor failures may be classified as bearing, stator, broken rotor bar, end ring, or eccentricity-related faults [101]. These faults may lead to increased vibration and noise levels and can be detected by monitoring machine vibrations, acoustic emissions, or motor current signals.

Unlike the other fault classes which have signatures directly related to shaft speed, bearing-related faults are difficult to represent with a single harmonic model. The natural mechanical resonance frequencies of the machine are modulated by the defect frequency resulting in spectral components that are not harmonics of either the defect frequency or the machine's natural resonance frequencies [101]. The defect frequencies generated from specific faults depend on their location within the bearing structure (inner or outer race, ball, cage) and the bearing assembly's geometry. Amplitudes of the defect frequencies have been shown to be an indication of the severity in [102].

The data set from [103] is a collection of accelerometer data from the introduction of single-point faults to test bearings mounted in a 3-horsepower induction motor. Bearings were separately prepared with 7, 14, and 21 mil diameter faults on a ball, inner race, or outer race. The signal from an accelerometer mounted on the motor housing at the drive end was recorded at motor loads of $0 - 3$ horsepower. Figure 3.5 shows representative vibration spectra of the signals for normal operation and with the different bearing fault types.

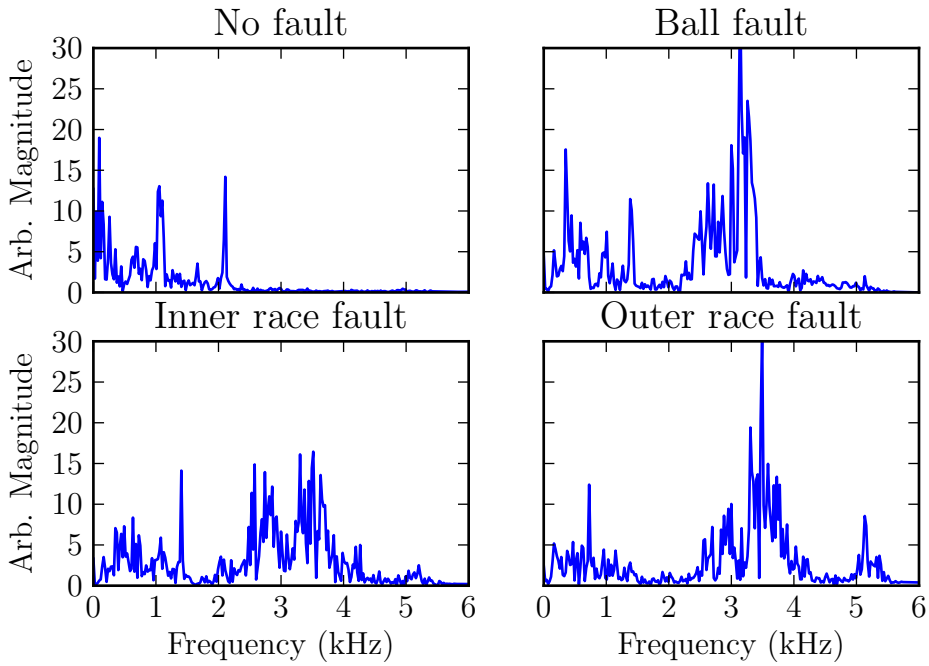


Figure 3.5: Vibration spectra of an electric motor with various drive-end bearing faults at 1772 RPM and 2 HP load.

Under normal operation, most energy is concentrated below about 2 kHz while the presence of faults moves this energy into the 2 kHz to 4 kHz band. A single harmonic model with $\widehat{\mathrm{FF}} = 100\,\mathrm{Hz}$ was chosen to give sufficient resolution to extract the spectral envelope without attempting to identify and match individual intermodulation components. Doing so would require sub-Hertz resolution and knowledge of the exact shaft speed and bearing geometry as shown in [104]. A similar neural network and training procedure to the one used in Case Study I in Section 3.6.1 was used for classification of the bearing fault.

Table 3.3 shows the classification results for three values of $M = \{10, 20, 40\}$ corresponding to upper frequencies of 1, 2, and 4 kHz, respectively. It is clear that the lower 10 harmonic amplitudes are sufficient for discriminating a healthy bearing but not for identifying the type of defect. To identify the defect type, at least 20 har-

Table 3.3: Bearing fault single-event detection, false alarm, and classification rates, from [4]

	Number of harmonics, M	10	20	40
	Bearing Fault	$\widehat{\text{FF}} = 100\,\text{Hz}$		
Det. (%)	No fault	99.68	99.83	100.00
	Ball fault	83.80	94.34	98.79
	Inner race fault	86.70	96.04	98.93
	Outer race fault	76.73	94.90	98.49
F.A. (%)	No fault	0.10	0.03	0.00
	Ball fault	9.28	2.16	0.45
	Inner race fault	1.22	0.85	0.33
	Outer race fault	6.23	1.66	0.39
	Classification rate (%)	87.20	96.42	99.09

monics are required to reliably approximate motor vibrations. A harmonic model with more than 40 harmonics has little advantage since there was little vibration energy above 4 kHz. Previously published classification rates using this data set were in the range 82.8–100%, as summarized in [104].

3.7 Feasibility of Hardware Implementation

The Analog Harmonic Transform has features well-suited to analog-domain implementation. The results of using this approach in Section 3.6 show that it is competitive in terms of classification performance with state-of-the-art techniques, while presenting the promise of very low-power analog implementations. This section explores an example system model which exploits these analog-friendly features and deterimines relevant hardware specifications that would be required to maintain good system-level performance.

Figure 3.6 shows a top-level block diagram of such a system. The input amplifier low-pass filters the signal to the maximum expected harmonic frequency and distributes the resulting signal to the projection blocks. Each harmonic projection

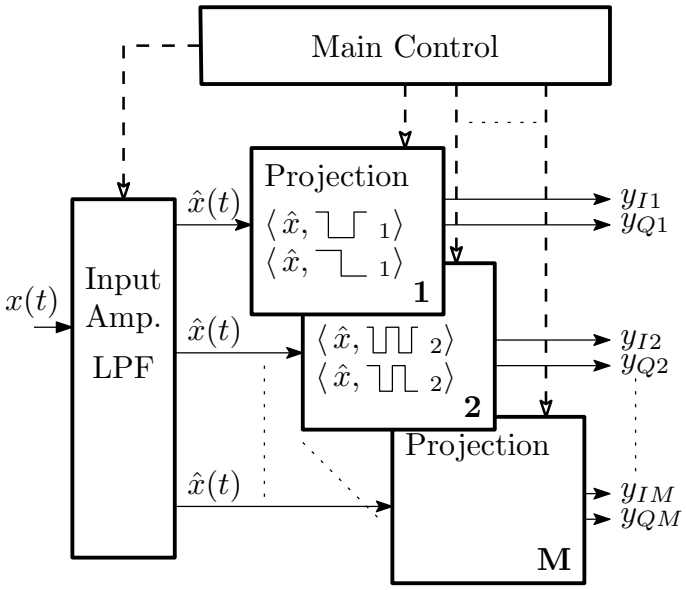


Figure 3.6: Analog projection system block diagram.

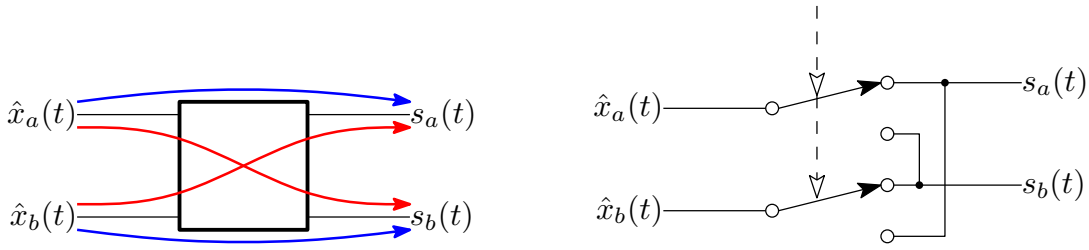


Figure 3.7: Differential ±1 multiplier.

shares the common input and global timing signals, and receives individual configuration information from the Main Control. The circuitry for each projection block can then be identical, allowing a highly regular circuit implementation to minimize inter-channel differences.

3.7.1 Transform Features and Architecture

Chapter 4 presents a complete, integrated-circuit design of a system which can implement the AHT. Here, the major and common characteristics of any circuit-level AHT implementation are discussed.

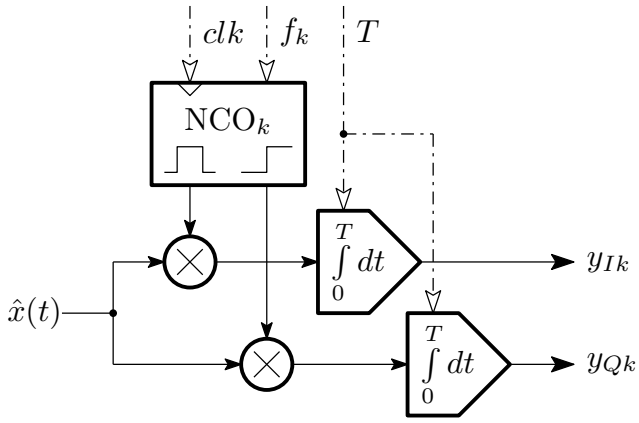


Figure 3.8: Harmonic projection channel block diagram.

Multiplication of the input signal by the basis function values of ± 1 may be viewed as a conditional signal pass-through or inversion. Using a differential signal path, this inversion is simply a re-labeling of the signal branches as illustrated in Figure 3.7. This reduces the signal-basis multiplication operation to a double-pole double-throw switch which is readily implemented with analog switches. Without this simplification, the necessary continuous-valued four-quadrant analog multiplier would dominate the noise and distortion performance of the signal path. In addition, the real-time basis function generation circuitry does not need to create synchronized sets of quadrature sinusoids of sufficient purity. The required switch timing signals may be readily generated by several analog or digital techniques such as multiplying phase-locked loops or numerically-controlled oscillators (NCO).

Figure 3.8 shows the general contents of each harmonic projection block. A digital NCO is shown generating the basis functions under control of a system clock and frequency control word. The modulated signal is then integrated by a number of techniques which could be as simple as an RC filter of appropriate time constant. Depending on the integrator implementation, the time constant can be made to span orders of magnitude, supporting applications with wide-ranging fundamen-

tal frequencies such as vehicle classification (8–20 Hz) and bearing fault detection (100–1000 Hz) with the same circuitry.

Other projection systems use a similar hardware topology but employ Compressive Sensing concepts for basis function generation [105, 80, 84, 83]. However, the basis function generators in [80] must be operated at greater than twice the maximum signal frequency to achieve the required randomness for CS-based reconstruction. The chipping rate can be reduced to sub-Nyquist [81] under certain signal-dependent assumptions but is still tied to the signal's maximum frequency. For the AHT, the basis functions are always less than or equal to the maximum signal frequency and equal to the desired harmonic frequency which is, in general, substantially lower than the maximum frequency.

Also featured in this topology is the low bandwidth requirements placed on the active circuitry; only the input buffer amplifier must operate over the entire signal bandwidth. The integrators in the projection blocks only need response on the order of the integration time window T. Power dissipation in many integrators is inversely proportional to their time constant, leading to inherently low-power operation.

3.7.2 Hardware Error Sources

Errors introduced into the computation of Equation (3.2) by hardware may result in degraded system-level performance. The effects of non-ideal computation must be accounted for in order to both determine the system's feasibility and to set the required hardware design specifications which maintain acceptable system-level performance. These errors may be combined into five classes: timing, distortion, random noise, gain, and offset. Figure 3.9 shows a simulation model of the har-

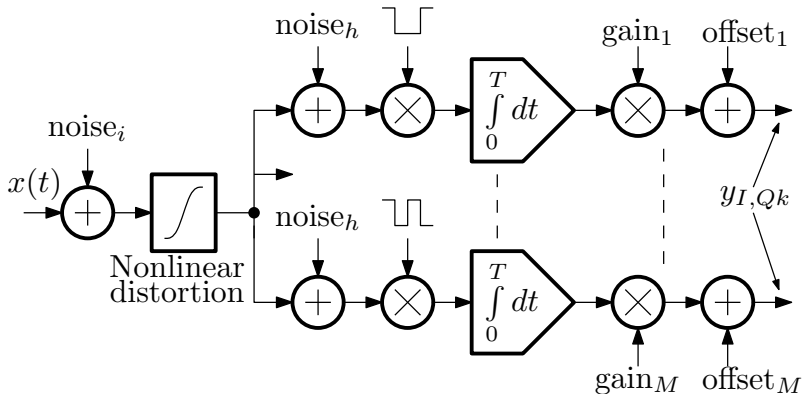


Figure 3.9: Simulation model of hardware error sources.

monic projection paths which model the major sources of errors. Varying the random noise magnitude, the gain variance, and the offset variance and evaluating the resulting simulated system-level performance allows mapping a set of hardware specifications to achieve for a successful circuit design.

3.7.2.1 Timing Errors

These errors come from basis function generation and reset/readout delays in the integrators. Jitter in the basis function waveforms broadens the spectral sensitivity of the channel. For an NCO-based function generator, employing a sufficient number of phase accumulator bits and reducing the ratio of highest frequency harmonic to digital clock rate f_M/f_{clk} can render these errors insignificant. For example, if a system clock of 32.768 kHz is used in the system from Case I with a 16-bit phase accumulator, the highest harmonic of 200 Hz would have a peak cycle-to-cycle variation of only 0.6%. Using $n \cdot$ FF multiplying PLLs would give exact timing at the expense of increased startup time to achieve phase lock and larger power consumption. Due to the general ease of minimizing timing-related errors compared to the other types, they are not included in the simulation modelling.

3.7.2.2 Distortion Errors

Waveform distortion from input amplifier and integrator nonlinearity generates additional signal-related frequency content in addition to that of the original input signal. The net effect of this is additional terms in the matrices $\boldsymbol{U_{I,Q}}$ in Equation (3.8) below the diagonal entries, invalidating its upper-triangular property, and compression of the diagonal entries at large input amplitudes.

The input amplifier and the integrators are the only active elements in the signal path which would typically contribute significant waveform distortion. Generally, distortion is to be avoided in the signal path and may be assumed to be mild and primarily related to the amplifiers' finite input range. This suggests a frequency-independent model which describes output clipping behavior. The specific function shape, $v_{out} = f(v_{in})$, is extracted from transistor-level simulation of the hardware design from Chapter 4. Because this is independent of time and a function of amplitude only, it has the same effect at any point before the idealized integrator in the simulated signal path. To ease the computational burden, the distortion function is placed at the input to the system. Figure 3.10 plots the transfer characteristic used in the study.

3.7.2.3 Random Noise Errors

Random noise including $1/f$, thermal, and switching noise will add random variation to the input signal. The total effect, after projection, may be modelled as a random variable added to each output y_{Ik} and y_{Qk}. An alternate model, and preferred in this case, of these sources is to collapse all noise sources into a source at the signal input with an equivalent total spectral density. Each projection channel shares the common input signal but is operated independently from the others. Therefore,

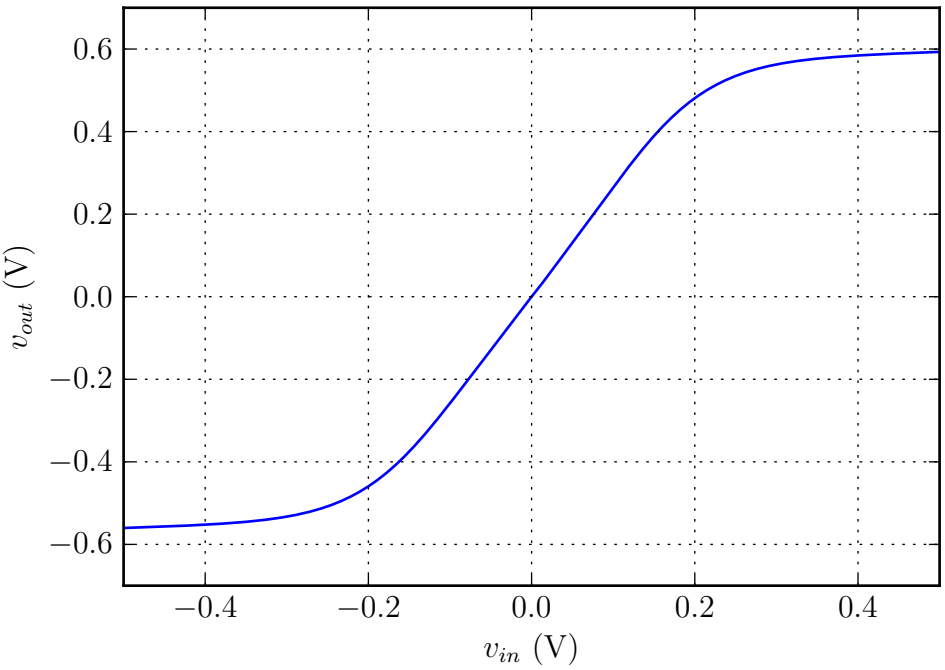


Figure 3.10: Static $v_{out} = f(v_{in})$ transfer function extracted from circuit simulation of the designed amplifier.

one noise source at the signal input models the equivalent input noise of the input buffer and low-pass filter while a source at the input to each projection channel models their equivalent input noise. The spectral densities of the channel noise (noise_h) are identical because they represent identical circuits, but the time-domain noise samples must be generated independently. Noise source (noise_i) then only represents the noise contribution of the input amplifier. Figure 3.11 plots the noise spectral density used for simulation. This density represents the output current noise of the designed amplifier.

3.7.2.4 Gain Errors

Gain errors arise from unequal amplifier gains from the input to the individual projection blocks and integrator time constant variations; the latter may vary by

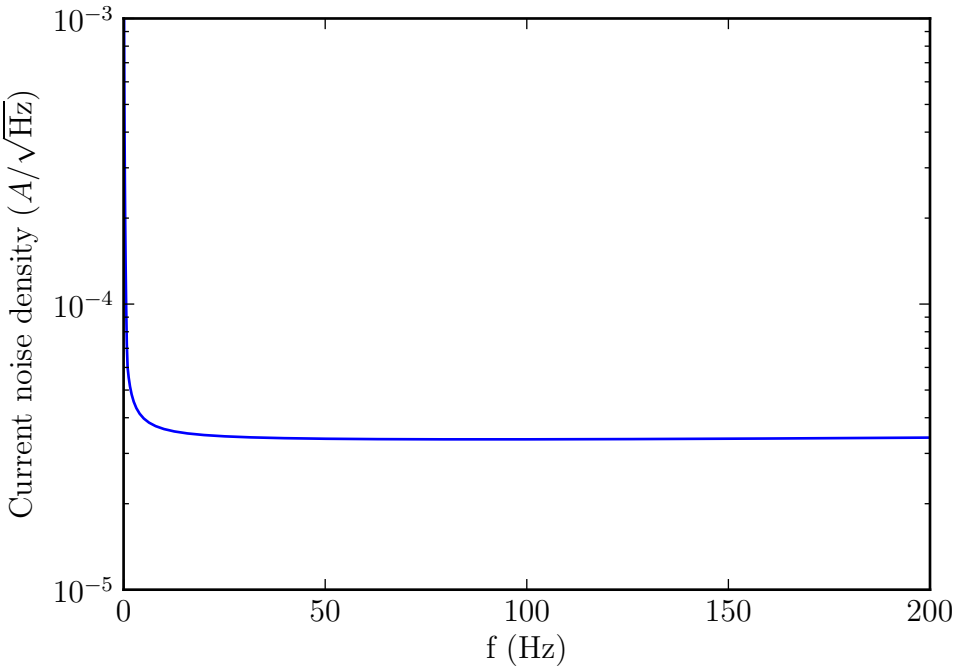


Figure 3.11: Spectral density vs. frequency for noise modelling.

as much as a factor of three with poor design and physical layout. The cumulative effect is a random scaling of each harmonic amplitude. Due to their nature, these errors may be considered fixed for a given IC sample and operating conditions. This error type directly challenges the assumption that each hardware projection channel operates identically. Fortunately, once the gain-related error for a given channel has been characterized, its output may be scaled by the inverse of the gain error.

The modelling of this error simulates selecting a "virtual chip instance" (VCI) which represents a unique set of channel gains or gain vector. Any usage of this VCI in subsequent system testing scales the individual channel outputs by the instance's set of channel gains. They are assumed to be samples of a random variable with mean of one and variance describing the across-chip, or across-wafer variations expected or measured for the design. For this study the random variable was

assumed to be normally distributed.

3.7.2.5 Offset Errors

Each projection path output will also yield a non-zero output for a zero input signal due to DC shifts and offsets accumulated through the signal path. Transistor mismatch in the integrator, residual charge injection from the multiplier switches and *reset*/*read* integrator switches contribute to this error. Harmonic amplitude outputs then appear with a static shift in value. Amplifier offset calibration along with correlated double-sampling techniques can be effective for reducing this type of static error.

Similar to the gain errors, these offsets are relatively fixed for a particular VCI channel. These are samples of a zero-mean random variable. Additionally, they are are assumed to be normally distributed in this study.

3.7.3 MATLAB Modelling

Modelling of both random noise and random fabrication errors was included in MATLAB by system instance generator function `makesystem`, whose call signature is summarized in Table 3.4 and example usage in Table 3.5. Each call to `makesystem` yields a randomly-generated set of gain and offset coefficients which corresponds to a single VCI. This allows investigation of classifier robustness to training and testing on the same simulated chip instance, training and testing on different hardware instances, or training on an ideal system and testing a hardware sample.

System instance noise generators `sys.inoise` and `sys.hnoise` use spectral densities and power levels obtained from transistor-level simulations. At the low frequencies used for the example applications, circuit noise will be dominated by $1/f$

Table 3.4: `makesystem` function signature description.

```
sys = makesystem(measfile, nChan, Fs, seed)

Inputs:
    measfile        - data from analog simulations in .mat file
    nChan           - number of channels to instantiate (= 2 * nHarmonics)
    Fs              - sample rate
    seed            - initialization seed for random number generator
Output sys structure members:
    .h              - nChannels length vector of structures
      .h(n).gain      - instance gain, N(1, gain_variance)
      .h(n).offset    - instance gain, N(0, offset_variance)
    .inoise(k)      - function yielding k samples of input amp noise spectrum
    .hnoise(k)      - function yielding k samples of channel noise spectrum
    .amplifier(vec) - function applying amplifier nonlinearity to vec
```

Table 3.5: Example harmonic coefficient generation including hardware modelling.

```
inst_seed = rand();
sys = makesystem('amp_meas.mat', 50, 44100, inst_seed);
vec = vec + sys.inoise(nsamples); % add input amp noise
vec = sys.amplifier(vec);         % amplifier nonlinearity

for n = 1:nHarmonics
    % dot product, mult-int
    c(n) = basis_i(n) * (vec + sys.hnoise(nsamples));
    s(n) = basis_q(n) * (vec + sys.hnoise(nsamples));

    % instance gain + offset errors
    c(n) = (sys.h(n).gain * c) + sys.h(n).offset;
    s(n) = (sys.h(n).gain * s) + sys.h(n).offset;
end
```

noise and it is therefore important to use colored noise for modelling. Individual channel gain and offset errors are dominated by geometry and doping variations and are considered fixed for a given instance. Temperature variations are not modelled.

3.7.4 System Classification Rates with Hardware Errors

Verification of the technique including estimated analog hardware error sources was conducted by replacing the explicit computation of Equation (3.2) with the system described by Figure 3.9 and calculated similar to the pseudo-code in Table 3.5. Instances of $noise_h$ have identical spectral densities but are generated independently.

The low FF of the vehicle classification case study presents very severe hardware requirements (much longer time constant) and correspondingly larger potential errors than those of bearing fault detection in Section 3.6.2. Initialization data for the error modelling was obtained from a system design implemented in a standard 0.13 µm CMOS process [1]. Because the amplifier/integrator distortion can be estimated *a priori*, event windows randomly selected for training the FNN were subjected to the same memoryless nonlinear distortion function, extracted from transistor-level simulation.

Neural network training was performed on a system instance which included the distortion but whose σ_{gain}, σ_{offset}, and noise magnitudes were set to zero. Signal-to-noise ratios (SNR) were set by scaling the input amplitude with respect to the noise density and power extracted from transistor-level simulations. Therefore, the total vehicle recording is considered "signal" for these simulations even though the recordings contain additional environmental and other noise. System-level vehicle classification rates were then simulated at two noise levels and over a range of gain

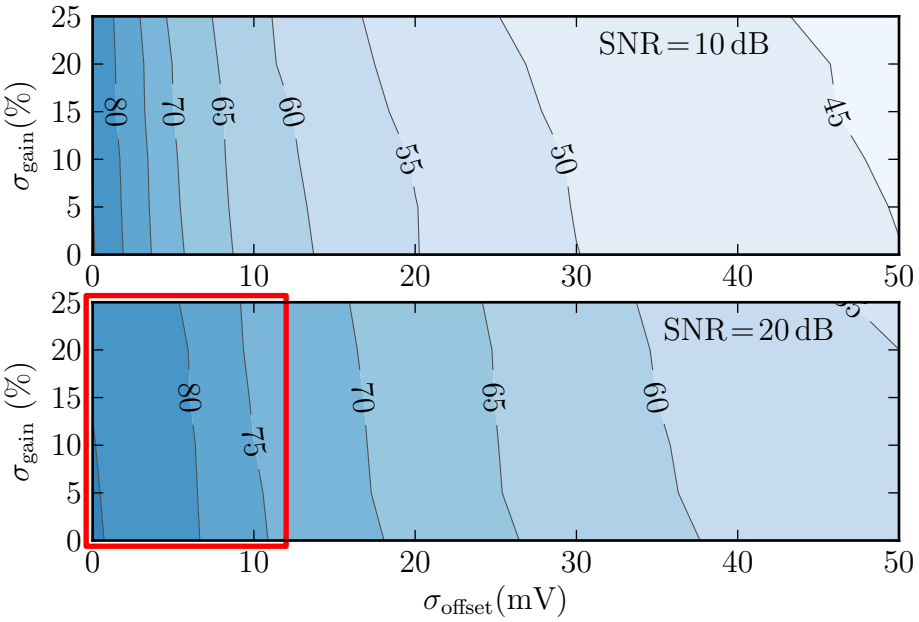


Figure 3.12: Average classification rate variation over gain/offset standard deviation and added noise values.

and offset error standard deviations.

Figure 3.12 plots a contour map of average system-level classification results for a range of gain/offset standard deviations and noise levels. It is clear that gain variations with standard deviations up to 25% have little effect on classification performance. However, classification rates are much more sensitive to offset variations. Offsets have the effect of consistently over-estimating the signal energy at that harmonic, even if there is insignificant signal content at that particular frequency.

The boxed region of Figure 3.12 then indicates the range of errors allowable to maintain good system-level performance. For full-scale circuit outputs of $\pm 1.2\,\text{V}$, this represents relative offset variations on the order of 1%. This region therefore sets the target hardware design parameters for the vehicle classification application of Section 3.6.1.

Chapter 4

Hardware Design

4.1 Introduction

This chapter describes the design of the major sub-modules of a prototype chip which can perform the AHT operations. It includes extensive digital control to achieve the required performance and to facilitate collection of repeatable, documented testing conditions and data.

Before more detailed design and chip layout activities were done, the primary digitally-tuned OTA-C integrator was designed and simulated to determine the feasibility of the system in Section 4.2. The prototype chip's overall architecture is described in Section 4.3 while the AHT system circuitry is found in Section 4.4.

Section 4.5 describes the AHT hardware design in increasing levels of detail. A comprehensive description of the OTA as fabricated is provided in Section 4.6.

To assist with managing the many digital tuning control points of the system, a micro-processor was included on the chip. The adaptation of a MSP430-compatible design fabricated in a 0.18 µm process to this 0.13 µm chip forms the content of Section 4.7.

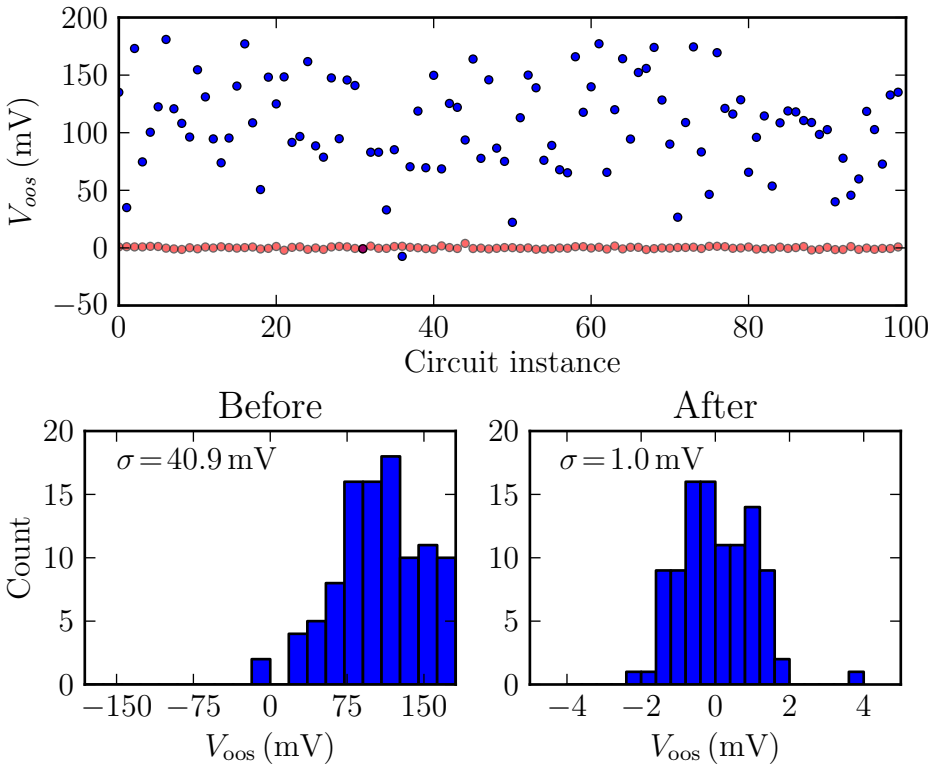


Figure 4.1: Scatter plot (top) and histograms showing pre- and post-calibration integrator offset error for 100 Monte Carlo simulations. Pre-calibration values in the scatter plot are in blue while the post-calibration offset values are red. Note the horizontal scale change for the right post-calibration histogram. Adapted from [1]

Appendices A and B include additional information relating to the use of MyHDL [106] for the design and unit-testing of the custom digital modules. This is a hardware description language (HDL) based on the Python programming language which eases the creation and testing of digital designs.

4.2 Initial Design Exploration

To determine whether the target offset values generated in Chapter 3 are feasible, we performed Monte Carlo simulations of a transistor-level integrator design which included offset calibration circuitry. Figure 4.1 summarizes the offset error from 100

random instances both before and after internal calibration [1]. Circuit calibration brought the unacceptable offset standard deviation of 41 mV down to around 1 mV. Because this simulated post-calibration error is well below the estimated 10 mV upper bound, it is expected that most chip instances from this design would be able to achieve acceptable system-level performance [1]. Such on-chip calibration is necessary to achieve (offset) errors within the feasible region of Figure 3.12.

The single-channel integrator used for this study dissipated 200 nW when tuned for a 5 Hz FF [1]. Thus, with two projections per harmonic and the parameters $M = 50$ with a 200 ms integration time of Case Study I, the feature vector computation would consume 4 µJ of energy. This energy consumption is three orders of magnitude less than the 5100 µJ FFT computation energy measured in [88] for a similar military vehicle classification application. After projection, the ADC would sample the 100 quadrature projection results and pass the data to the classifier or other back-end system processing.

On-chip custom FFT implementations can naturally use less energy for computation, such as [65] using 116 nJ per 128-point transform. This does not include the system overhead for loading the input data and reading the result. Common processors utilized for wireless sensor systems include the MSP430 series [107] which requires approximately 1.5 nJ per instruction to move data in memory. For the 128-point custom hardware, this requires loading 128 real values and reading 64 complex values for a total energy use of 0.69 µJ. Circuitry used in the AHT can be re-purposed for other signal processing tasks when not actively projecting, reducing the increased die area penalty. Due to their continuous-time operation, they can be also used as a real-time spectral energy detector to trigger further digital processing; FFT-based techniques by their nature require the system processor to be active.

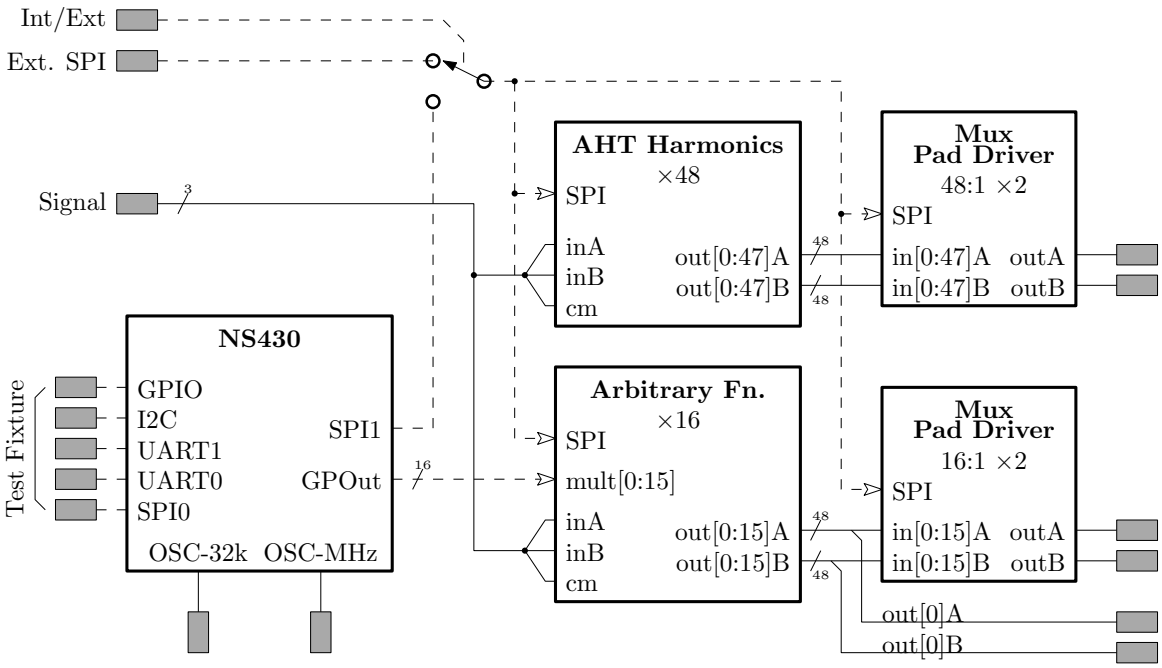


Figure 4.2: Prototype chip block diagram and internal interconnections. Filled rectangles represent functional pads for connecting signals on/off-chip, not necessarily individual package pins. Dashed lines are digital signals for control and basis functions while solid lines are analog signal paths.

4.3 Prototype chip architecture

A prototype chip was created to evaluate the hardware-level performance of the AHT system, its sub-parts had several features to enable testing. A functional block view of the chip is shown in Figure 4.2.

A 0.13 µm mixed-signal CMOS process from IBM, "8RF," was used for the prototype. The initial part of the design effort was to update the UNL Advanced Chip Design Group's portfolio of capabilities to include this process. Prior to this project, the most advanced process available for research prototyping was the previous-generation IBM 0.18 µm CMOS process, "7RF."

Critical to configuring the software environment for the new process design kit

(PDK) was verifying the proper setup to integrate both full-custom schematic and hand layout with an automated digital design flow. Some differing conventions between the full-custom and digital portions of the PDK were un-documented and found to be in error. These errors were fixed to bring the PDK configuration in line with IBM's master process documentation.

Finally, the design rule checking scripts provided in the PDK were inconsistent with the published design rules. One such configuration error caused the digital design flow to generate artwork guaranteed to fail the design rule checks. Another necessary change was the addition of rules to properly check layouts which included black-box intellectual property (IP) cells whose complete layout was not available.

After the PDK setup and validation with this project, the IBM "8RF" process is available for other advanced prototype chip designs. For example, this accelerated the design of the "PIRANHA" imager with focal-plane digital processing. This design is inherently an analog/digital hybrid and therefore required a PDK with consistent and correct checking and setup.

The core of the design is the "AHT Harmonics" block which contains a parallel bank of 48 quadrature harmonic paths. These, and the other bank, share a common differential input, *inA / inB*, and common-mode input, *cm*. The architecture and design of this module is described in later Sections 4.4, 4.5, and 4.6. Analog coefficient signal outputs from each of the integrators is applied to a multiplexer and buffered by a pad driver for off-chip measurements.

A second bank of 16 projection channels is included as "Arbitrary Fn." For these channels, the internal digital NCO basis function generator is disconnected from the multiplier. The multiplier inputs are provided from the on-chip micro-

processor to use arbitrary ±1 functions besides the AHT's 50% duty-cycle quadrature square waves. Also, two of the integrator outputs are routed to both the multiplexer and directly to pads. This allows observation signals before the multiplexer, bypassing the pad drivers.

Block "NS430" is a custom micro-processor which is code-compatible with the TI MSP430 family of commercial micro-controllers. Through a switch, it can control the rest of the chip via SPI, a serial data connection. Section 4.7 describes the processor in more detail.

4.4 AHT System Architecture

Figure 4.3 shows the AHT system architecture implemented in the hardware, first shown in Figure 3.6. The basis function projection blocks represent the majority of the system design work. For the prototype, the Input Amplifier and signal preprocessing functions are performed off-chip and the Main Control is implemented in software with a serial communication link to the controlled projection blocks.

4.5 Basis Function Projection

Each quadrature projection section consists of a local digital block and two identical multiply-integrate channels. The digital section uses the 1.2 V nominal digital supply while the channels operate from the 2.5 V nominal analog supply rail. Logical control signals cross the boundary through a level translator block.

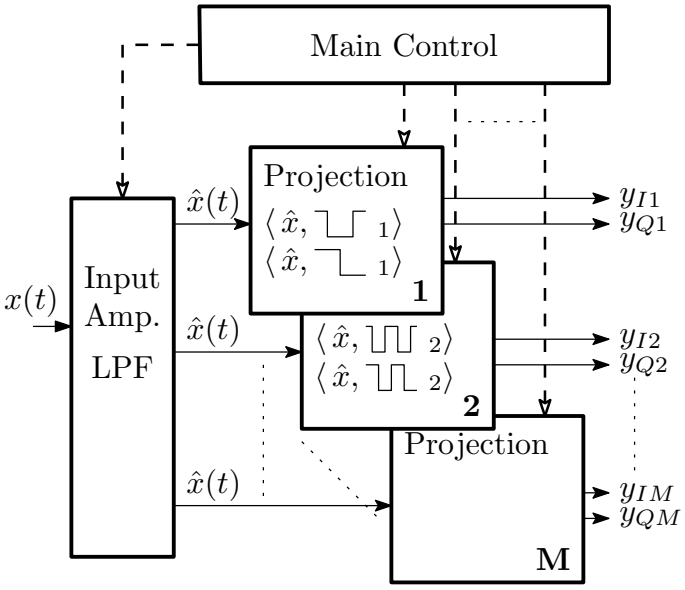


Figure 4.3: Analog projection system block diagram. The Input Amplifier and low-pass filter (LPF) block is not integrated into the prototype chip but is provided by the test fixture.

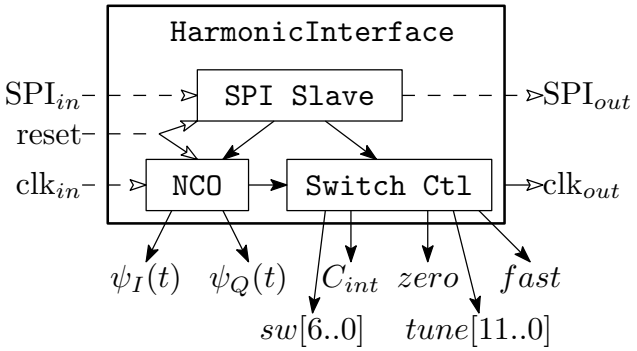


Figure 4.4: Quadrature harmonic projection digital architecture.

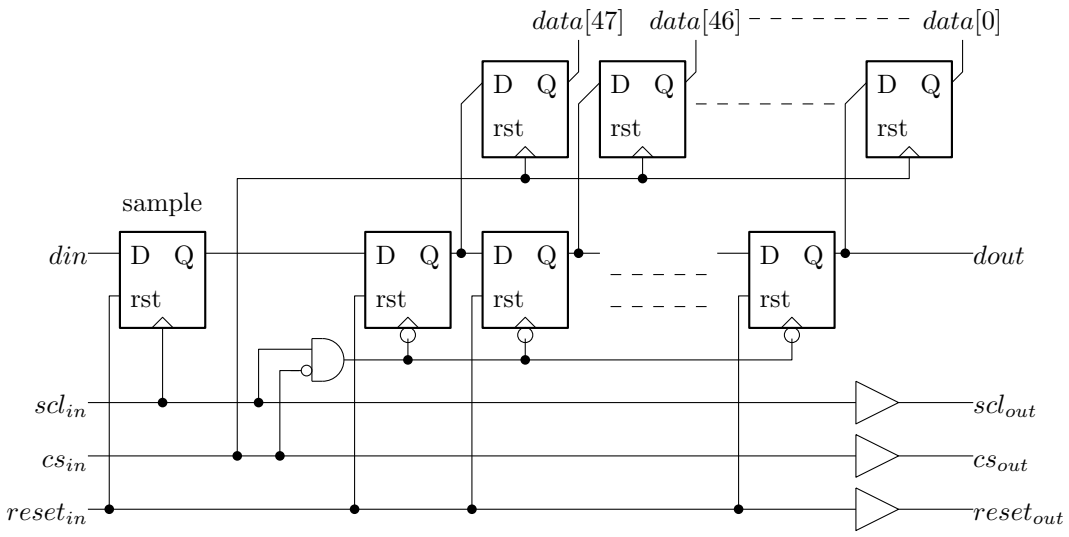


Figure 4.5: SPI architecture showing latched data output, rising-edge sample, falling-edge shift, and buffered through signals.

4.5.1 Local Digital and Switching

The digital section contains the quadrature basis function generator, two OTA tuning registers, integrator control, and a serial communication port. Figure 4.4 illustrates the hardware description level (HDL) design hierarchy and inter-connections. These modules used MyHDL [106] as the design input language and made use of the software's automatic Verilog or VHDL output conversion.

All configuration information is passed into the SPI register and distributed to the NCO and switch control. The reference clock for the NCO is separately passed in. To facilitate abutting the harmonic blocks in the circuit layout, the SPI and NCO clock signals pass through each block on opposite edges. The following sections describe the digital sub-blocks in more detail.

4.5.1.1 Serial Communication

The serial peripheral interface (SPI) was chosen for simple communication of the harmonic control parameters from the system host processor. It is a serial interface with a dedicated clock signal *scl*, serial data input *din*, active-low chain select *cs*, and serial data output. Daisy-chaining the next harmonic's *din* to the previous' *dout* causes the interface to appear to the processor as a long shift register. The register is 48-bits wide with the logical layout given later in Table 4.1.

Figure 4.5 shows a schematic representation of the serial peripheral interface (SPI) for loading configuration values from the processor. The configuration corresponds to the standard SPI mode denoted by `CPOL=0` and `CPHA=0`, or mode 0. This mode corresponds to the clock line *scl* which idles low (`CPOL=0`), and which samples data on the clock's rising edge while propagating data on the clock's falling edge (`CPHA=0`). Start of a data transfer is indicated by a falling edge on the *cs* line. Data is then applied to the *din* pin synchronized with the *scl* data clock line. When all bits have been shifted in, the controller raises the *cs* line which latches the current state of the shift register into the *data* output storage. This ensures that the data output bits do not change as a new data word is being shifted in.

The relative timing or skew between the SPI signals can degrade as they propagate through a long chain of harmonic sections, possibly violating register setup/hold times at the end of the chain. Also, the clock *scl*, chain-select *cs*, and *reset* signals are common to all sections and could present a large load to the input driver. Buffers inserted in these signal lines minimize the load and allow the signal to propagate evenly down the chain. Finally, the D flip-flop labeled "sample" in Figure 4.5 samples the incoming data bit on the rising edge of *scl* while the data is shifted on the falling edge of *scl*. This bi-phase sample-shift operation ensures there will be

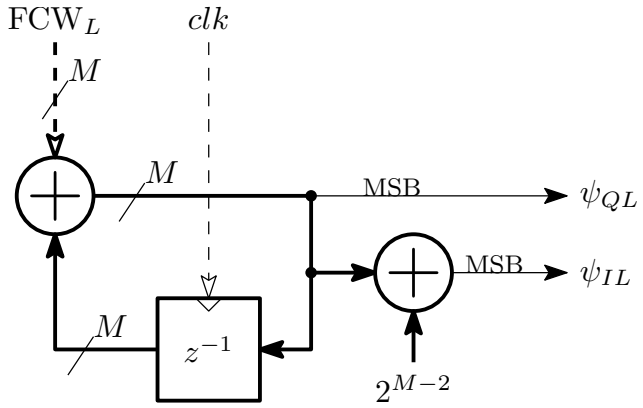


Figure 4.6: Numerically-controlled oscillator with M-bit phase accumulator, M-bit phase increment word, and quadrature output bits.

no skew-related timing errors at any point in the chain if the condition that the *scl* period is longer than twice the longest propagation delay through the entire chain is satisfied — the interface has a maximum shift rate but by design it has no lower rate limit and is inherently immune to timing skew.

Appendix A describes the MyHDL code for the SPI slave module in detail as an example of digital module development using the tool. One advantage with using MyHDL is the ease of implementing test benches for verification of modules. The fact that the tool is written in python, a general-purpose programming language, with features added to describe digital hardware at the register transfer level enables natural description of digital behavior. This is in contrast with the prevailing HDL languages Verilog and VHDL, which must use the limitations of a hardware-oriented language to describe high-level behavior.

4.5.1.2 Basis Function Generator

The quadrature basis function generator is implemented as a numerically-controlled oscillator (NCO). Figure 4.6 shows the functionality of the NCO, consisting of a phase accumulator (PA) register which holds the current phase value, an adder

for incrementing the phase by the value FCW_L, and an adder which increments a static 1/4 of a phase accumulator offset. The most-significant bit of the phase accumulator is used as the quadrature output. Calculation of the in-phase offset in practice does not require a full-width N-bit adder, it only uses a 2-bit adder for the upper two bits of the PA and therefore has little hardware impact.

For a given reference clock frequency f_{ref}, N-bit phase accumulator width, and phase increment FCW, the average output frequency and available frequency resolution are

$$f_{\mathrm{out}} = f_{\mathrm{ref}} \frac{\mathrm{FCW}}{2^N} \tag{4.1}$$

$$\Delta f = \frac{f_{\mathrm{ref}}}{2^N}. \tag{4.2}$$

While the average period of the output signals are as described by Equation (4.1), the instantaneous periods vary between the two integer multiples of the reference clock period f_{ref} on either side of the average value.

4.5.1.3 Tuning and Projection Control

The digital block `SwitchCtl` from Figure 4.4 is a combinational logic block which translates the multiplier command and settings from the SPI register and outputs control signals to the various switches. Table 4.1 describes the bit layout of the 48-bit SPI configuration register. For each quadrature harmonic projection block, there are two instances of projection channels and one digital control block which handles both channels.

Besides the global chip reset, the NCO basis function generator has a reset configuration bit *rst* to ensure all the generators operate with a known phase relationship to other harmonic projection blocks. The projection channels are configured to

bit #	47	[46:32]		[31:16]					[15:0]
block	Ch.-A,B	NCO		Channel-A					Channel-B
name	cal	rst	$FCW_{13:0}$	cintA	zeroA	seA	fastA	$tuneA_{11:0}$	(same)

Table 4.1: Bit layout of the 48-bit SPI register for each harmonic projection.

cal	*se*	Description
0	0	Differential input signal multiplied by basis function. OTA in open-loop.
0	1	*sigA* or *sigB* connected to OTA + input based on *mult* state. OTA in unity-gain feedback.
1	0	Both OTA inputs connected to *CMI*. No feedback.
1	1	*CMI* connected to OTA + input. OTA in unity-gain feedback.

Table 4.2: Projection channel operation mode as a function of the *cal* and *se* configuration bits.

be run in both single-ended and differential input modes during operation according to the state of the *se* bit. Configuration bit *cal* turns on the offset calibration mode. The OTA's switchable mirror ratio discussed in the next section is controlled by the *fast* bit. Finally, bit *Cint* controls whether the integration capacitor is connected to the amplifier output and bit *zero* resets the integration capacitor's voltage to a mid-supply level.

Nine transmission gates switch signals around the OTA and integration capacitor under control of the `SwitchCtl` block according to the selected mode. Table 4.2 describes the four operation modes provided by the *cal* and *se* configuration bits.

Digital offset and gain tuning control for each OTA are provided by the two 12-bit *tuneA* and *tuneB* words. For the current design, the gain and offset tuning code values used 4- and 8-bit resolutions, respectively. This feature is described in more detail later in Section 4.6.1.

Figure 4.7 shows a higher-level overview of the analog portion of the projection

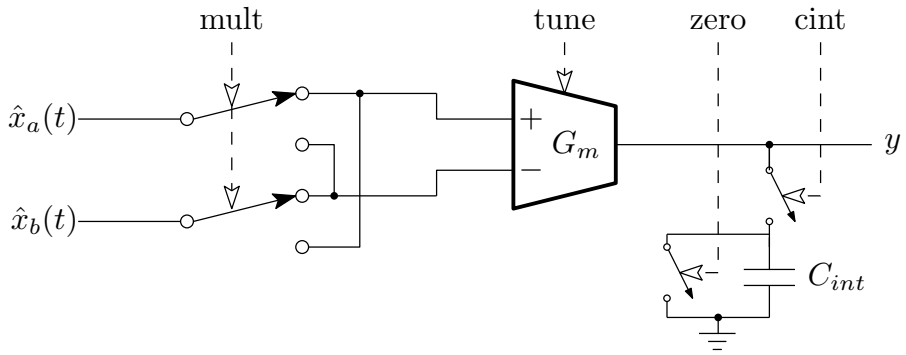


Figure 4.7: Conceptual schematic of analog projection channel showing multiplier, OTA, and integrating capacitor along with control signals.

channel. Two of these channels are included to implement the in-phase and quadrature portions of the harmonic projection. The input signal x is applied differentially as $x = x_a - x_b$, multiplying by ± 1 under control of the *mult* signal. The *zero* and *cint* switches allow disconnection of the integration capacitor from the output node and to reset its voltage to the common-mode voltage (the functional "zero" level for bi-polar integration outputs). Figure 4.8 shows the schematic of the projection channel circuitry with the nine transmission gate switches, OTA, and integration capacitor.

4.5.1.4 Level Translators

The digital portion operates from a nominal 1.2 V supply rail while all analog circuitry operates from a separate 2.5 V nominal supply rail. Control signals which cross the digital-analog boundary must then be translated to the appropriate logic levels. Each translation cell handles a single, differential pair of control signals.

Figure 4.9 shows a single cell of the level-translation circuitry as a bi-stable latch structure. Digital inputs are applied as signals *inA* and *inB* to the 2.5 V-capable input transistors T2 and T3. Cross-coupled transistor pair T1 and T0 provide a positive-feedback action to quickly snap the output nodes *outA* and *outB* to either

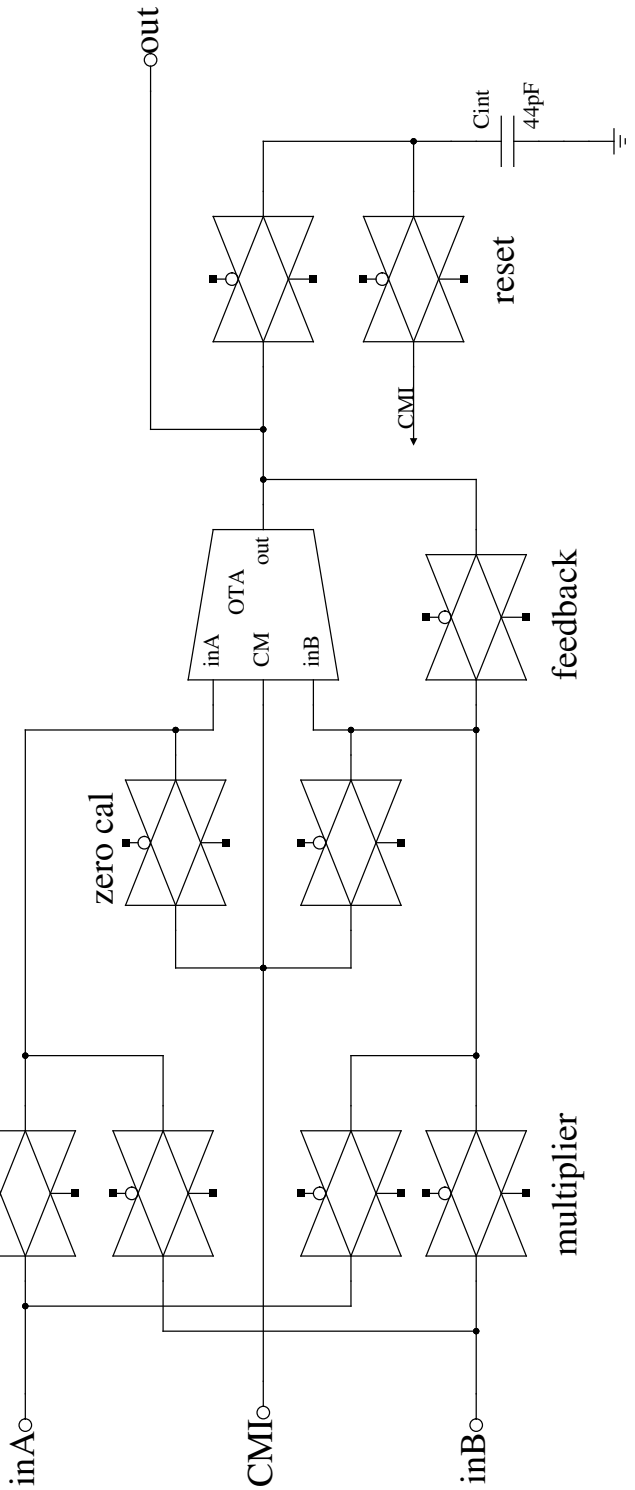


Figure 4.8: Schematic of the analog portion of a single projection channel. Differential signal input is applied at nodes *inA, CMI, inB* and integrator output at node *out*.

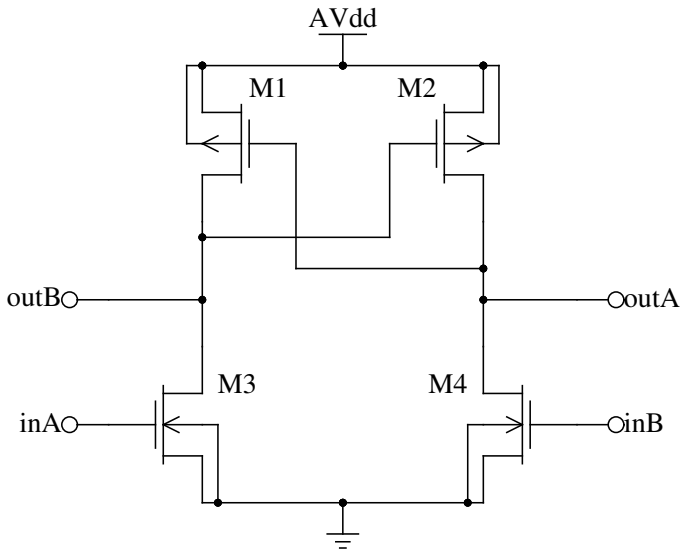


Figure 4.9: Level translator schematic.

the analog supply voltage $AVDD$ or zero. Input transistors are sized to apply a strong enough pull-down to force the latch out of its current state. The PMOS cross-coupled pair are sized to provide a very weak pull-up, with a weak aspect ratio of $(W/L)_p = 0.48/2.40\mu m$.

4.5.2 Analog Integrator

The multiplied waveform is integrated in the analog domain with an OTA-C type integrator. A voltage input signal is converted to an output current with the operational transconductance amplifier (OTA) with a transfer ratio of G_m A/V as in Equation (4.3). This current is applied directly to a linear capacitor C_{int} and the output V_{out} is taken as the capacitor voltage. Equation (4.4) shows the relationship between an input signal integrated for time T and the resulting output voltage value.

$$i_{out}(t) = G_m V_{in}(t) \tag{4.3}$$

$$V_{out} = \frac{1}{C_{int}} \int_0^T i_{out}(t)dt = \frac{G_m}{C_{int}} \int_0^T V_{in}(t)dt \tag{4.4}$$

Figure 4.7 shows a representative schematic of the tunable OTA-C integrator. The integrator time constant required, $\tau = C_{int}/G_m$, depends on the integration time, input voltage amplitude, and output voltage range. At very long time constants, the capacitor should be as large as possible to relax the corresponding OTA's ultra-low G_m requirement.

The IBM-8RF process used for the prototype has a dual-MIM, or metal-oxide-metal-oxide-metal, parallel-plate capacitor option which allows relatively high density, linear capacitors. The lower and upper plates are shorted together to make one plate while the middle metal forms the second plate of the capacitor. Active circuitry is allowed below the capacitors in the process and thus allows area-efficient implementations utilizing large capacitors. For this design, the integration capacitor C_{int} was set to 44 pF. This value corresponds to a capacitor which is only slightly larger than the rest of the OTA layout. Such a capacitor size ensures maximum utilization of available chip area. Due to the complexity of the OTA designed for this system, it is described separately in Section 4.6.

4.5.3 Off-Chip Analog Output

Each of the 48 quadrature harmonic projection channels has two analog outputs. It is not generally feasible to bring all 96 analog voltages off-chip for analog-to-digital conversion at the end of an integration period. A multiplexer and analog pad buffer are used to route each quadrature set signals to a pair of analog output pins on the chip. These two output pins are routed to external ADCs under control of the system processor.

One analog output path multiplexes the in-phase harmonic output while the second routes the quadrature channels. This configuration allows the simultaneous sampling of a quadrature harmonic pair of channels to minimize time-dependent leakage errors. The output to route off-chip and other settings in configured through an SPI-based digital block similar to the OTA configuration block.

Each multiplexer output feeds the input to an amplifier which isolates the sensitive on-chip nets from external influence. To reduce design effort, this buffer was implemented as a copy of the channel OTA with a different digital `SwitchCtl` block to allow unity-gain buffering and local offset calibration. This buffer is sub-optimal due to its high output impedance and slow step response but is sufficient for prototype evaluation.

4.6 Wide-Range Digitally-Tunable OTA

Figure 4.10 shows the OTA implementation, described in detail in the following sections. Bias currents I1 and I2 are digitally-tunable to change the OTA's G_m and to zero the offset current/voltage, described in Section 4.6.1. Differential input transistors M1 and M2, along with linearization elements Ma and Mb perform the main V/I conversion. Output current is generated as the difference between the M1 and M2 drain currents scaled by the mirroring ratios of (M3:M5×M6:M7) and (M4:M5), respectively. For this design, the mirror ratio (M3:M5) was set to unity, while the output node mirrors were constructed with a switchable transfer ratio. The mirror ratio switching is described in Section 4.6.2.

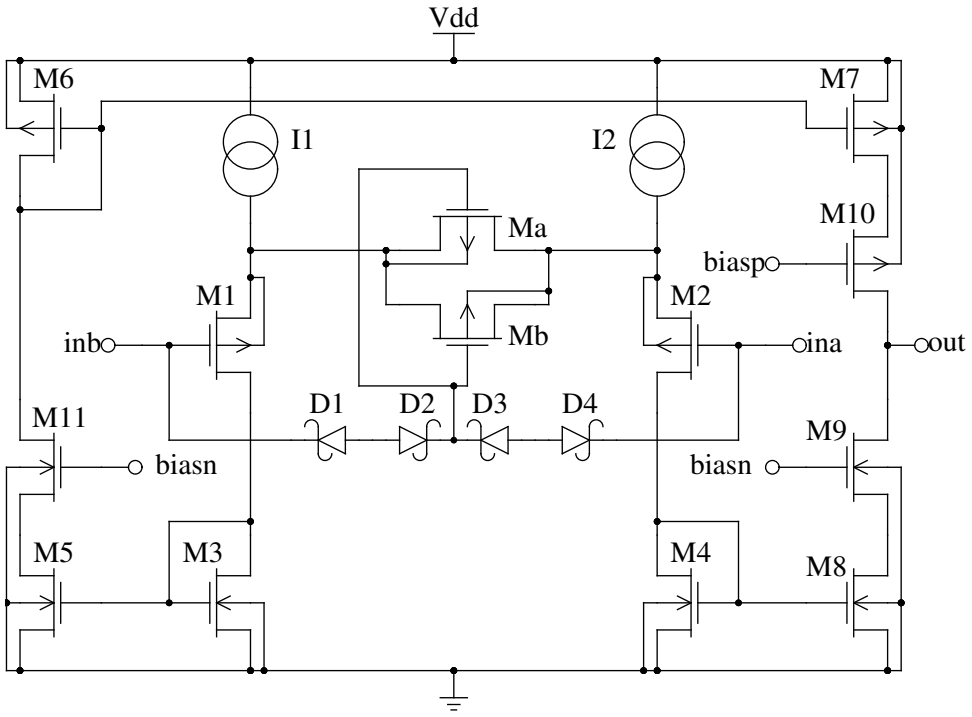


Figure 4.10: OTA core schematic with common-mode generation.

4.6.1 Gain and offset digital bias current source

Because the major source of errors within the OTA are fabrication-related mismatch, geometrical, and doping variations, they can be considered relatively constant for a particular die-amplifier instance. Tuning techniques such as laser-trimming and fusible links are not feasible for this type of system due to the large number of instances present on a given die — it would be cost-prohibitive to measure and tune each OTA after fabrication for small, inexpensive sensor node applications.

Due to the strict offset requirements of the application from Chapter 3, the circuit offset must be tuned after fabrication to achieve acceptable system-level performance. Though there are several offset-reduction techniques available as discussed in Chapter 2, the one here features a binary interface compatible with a software-controlled tuning using the system's already-present processor. In contrast

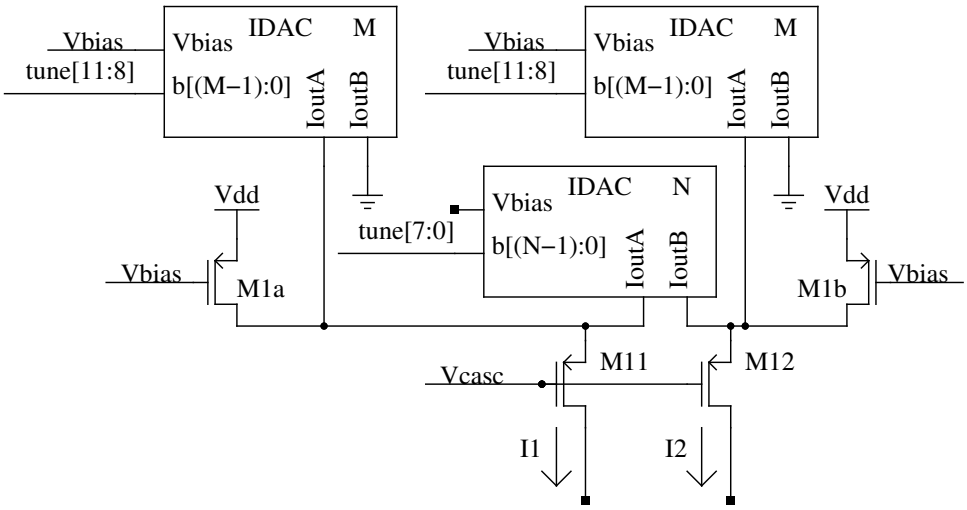


Figure 4.11: Magnitude and offset bias current DAC.

with chopper-based and other self-adaptive techniques, there is no tuning-related switching activity after the tuning procedure has finished. This reduces power consumption and eliminates the need to shield or isolate the amplifier from the tuning circuitry. The results of post-fabrication tuning will only vary as a result of environmental factors such as temperature changes and common-mode variations, the latter being minimized by the input conditioning circuitry.

Post-fabrication G_m and V_{os} tuning is achieved in this case by varying the two core transconductor bias currents I_1 and I_2 in Figure 4.10. Increasing both currents increases G_m only, to a first order. Skewing the values effectively changes the output offset voltage and current. If the current sources are implemented as current-output DACs, a digital system controller has direct and repeatable control of an individual amplifier's G_m and offset. In systems already utilizing an analog-to-digital converter and digital processor, calibration of the amplifiers requires no additional hardware and can be performed on-line as needed when environmental conditions change.

The amplifier bias current sources are composed of three coupled switched-

current cells. Figure 4.11 illustrates the cell connections. Transconductance tuning cells are M-bits wide and shunt the "off" current as shown in Figure 4.11. The offset-tuning DAC has N-bits of resolution and steers its current to either the I_1 or I_2 output. Transistors M1a and M1b of Figure 4.11 add a constant "pedestal" current to each branch to narrow the G_m tuning range and increase resolution. Finally, cascode devices M11 and M12 buffer current to the OTA.

Partitioning a $(M+N)$-bit tuning word into unsigned M-bit G_m (m) and signed N-bit offset (n) values causes the two branch currents to vary as

$$I_1 = I_{\text{ref}}\left(k_s + k_g\frac{m}{2^M} + k_o\frac{n}{2^{N-1}}\right), \tag{4.5}$$

$$I_2 = I_{\text{ref}}\left(k_s + k_g\frac{m}{2^M} - k_o\frac{n}{2^{N-1}}\right), \tag{4.6}$$

The terms k_s, k_g, and k_o represent the ratios between the pedestal current, maximum gain, and maximum offset current magnitudes to a global reference I_{ref}. Resolution and span of the G_m and offset tuning ranges are varied by appropriate sizing of the corresponding DAC sections.

Design of typical DAC cells expend significant effort on ensuring a linear and monotonic output-versus-code behavior [108]. The DACs here operate within the overall amplifier tuning system, meaning their errors simply contribute to the overall amplifier errors. They may then be compactly constructed with minimum-sized transistors provided their range and resolutions are designed to be sufficient to tune the composite gain/offset errors. Gradient-based tuning methods may settle in a local minimum if the controlling DAC does not have a monotonic output versus code characteristic.

Figure 4.12 shows the implementation of the switched-current cell. The voltage V_{bias} is generated by the global reference generator and is hence set by I_{ref}.

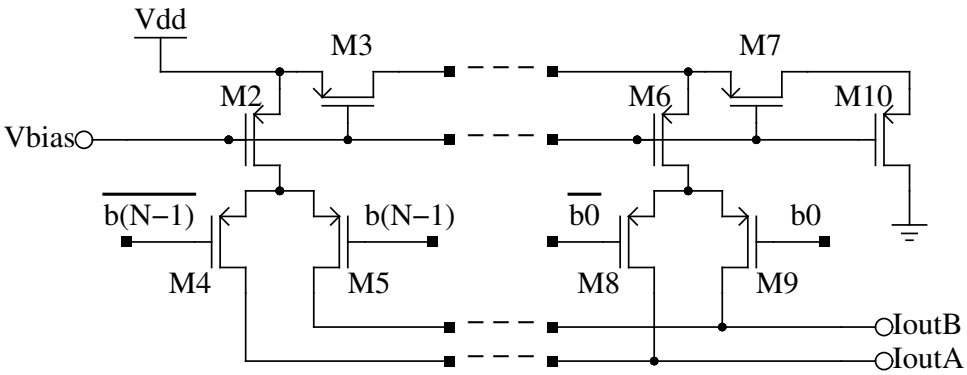


Figure 4.12: Current-steering I-DAC implementation schematic.

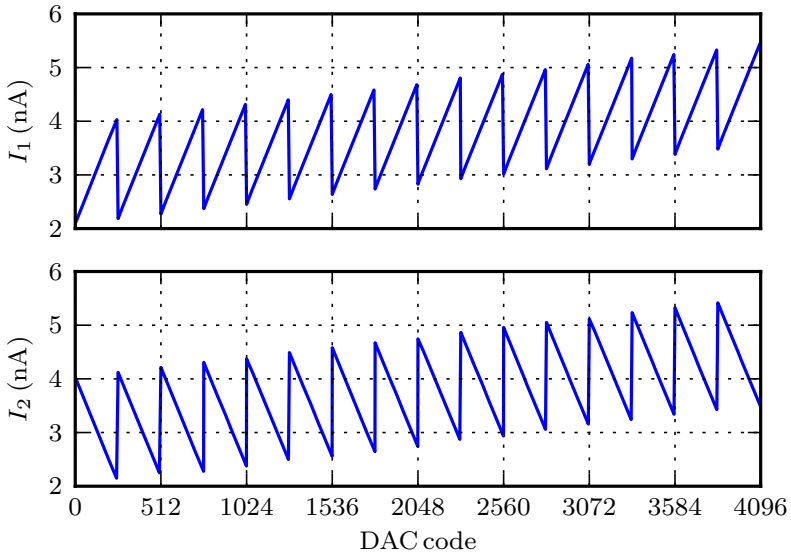


Figure 4.13: IDAC outputs versus input code. Each of the 16 gain steps increase both currents while there are 256 steps for each gain value which skew the two output currents to tune the offset.

Binary-weighted currents are generated by the N-bit M/2M ladder with transistors M2,3,6,7,10. Each current output is switched between nodes IoutA and IoutB according to the state of the gate voltages of transistors M4,5 and M8,9. For example, as drawn, each control bit generates true and inverse signals for the switch cells. If bit *b0* is low, M8 is switched on while M9 is off, steering the least-significant current magnitude to the *IoutA* node.

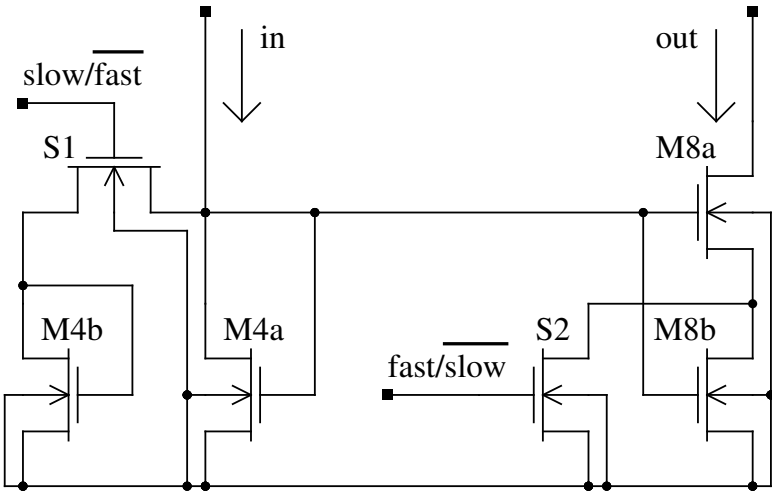


Figure 4.14: Current mirror ratio switching schematic.

Figure 4.13 plots simulation results of the designed gain/offset IDAC. The "pedestal" current provided by transistors M11 and M12 in Figure 4.11 was set to about 2 nA, demonstrated in Figure 4.13 as the minimum, non-zero value of each of the two output currents.

4.6.2 Switched-ratio current mirror

The OTA transconductance is directly proportional to its bias current when the input transistors are operated in weak inversion. Making the G_m range decades-wide then requires stable master reference current variation over the same range. Because the OTA reduces its transconductance by the current mirror technique, it is relatively easy to change the output mirror ratio, allowing the bias reference to operate over a smaller range.

Figure 4.14 illustrates the NMOS version of the switched mirror ratio. Transistors M4a and M4b are composed of M_a and M_b unit-sized transistors in parallel,

respectively, while transistors M8a and M8b are composed of the N_a and N_b unit-sized transistors in series, respectively, with a common gate connection. When the switch signal *fast* is asserted, the gate of S1 is low and the gate of S2 is high. This turns off switch S1 and forces all mirror input current through M4a. Switch S2 is turned on and nearly all the mirror output current flowing through M8a effectively bypasses M8b. When the switch signal *slow* is asserted, the gate of S1 is high and the gate of S2 is low. Switch S1 is then on, allowing the input current to flow through the parallel combination of the diode-connected transistors M4a and M4b. Since switch S2 is now off, the output current flows through the series combination of M8a and M8b.

The resulting mirror transfer ratio in each mode is then

$$\frac{I_{out}}{I_{in}}(fast) = \frac{1}{N_a M_a} \tag{4.7}$$

$$\frac{I_{out}}{I_{in}}(slow) = \frac{1}{(N_a + N_b)(M_a + M_b)}. \tag{4.8}$$

Achieving a factor of x decrease in output current (transconductance) from *fast* to *slow* mode results in the relations

$$N_a = (\sqrt{x} - 1)N_b \tag{4.9}$$

$$M_a = (\sqrt{x} - 1)M_b. \tag{4.10}$$

For the design, an approximately decade change was desired and the numbers of unit transistors were set as

$$N_a = 2.2N_b \tag{4.11}$$

$$M_a = 2.2M_b, \tag{4.12}$$

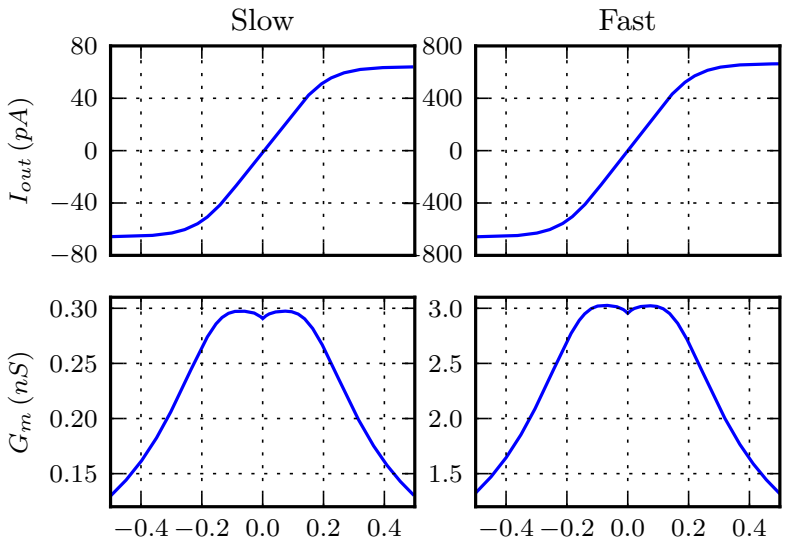


Figure 4.15: Simulated OTA output current and calculated transconductance versus differential input voltage for the *slow* and *fast* modes, 4 nA reference current bias.

resulting in a factor of 10.89 change between *fast* and *slow* modes.

Figure 4.15 plots I_{out} and G_m versus differential input voltage at a 4 nA nominal bias current in *slow* and *fast* modes. The designed OTA had sizing relationships of $L_{8b} = 2.2L_{8a}$ and $W_{4b} = 2.2W_{4a}$, mode switches then cause an approximately 10× change in output current and transconductance. The plot also shows the characteristic "dip" in G_m when both linearization diffusors are in triode mode. The range for $< 1\%$ G_m variation is typically $|V_{id}| = 115$ mV.

Figure 4.16 shows simulated G_m versus bias current I_{ref} with 10× range switching. Sub-pS operation indicated at pS-range reference currents suffer from numerical simulation issues. The device models used in these simulations are not qualified at these extremely low currents. In *fast* mode and at the higher reference currents transistors begin to transition into moderate inversion operation and reduce their g_m/i_D efficiency as shown. This OTA is intended to operate at G_m's from 10 pS to

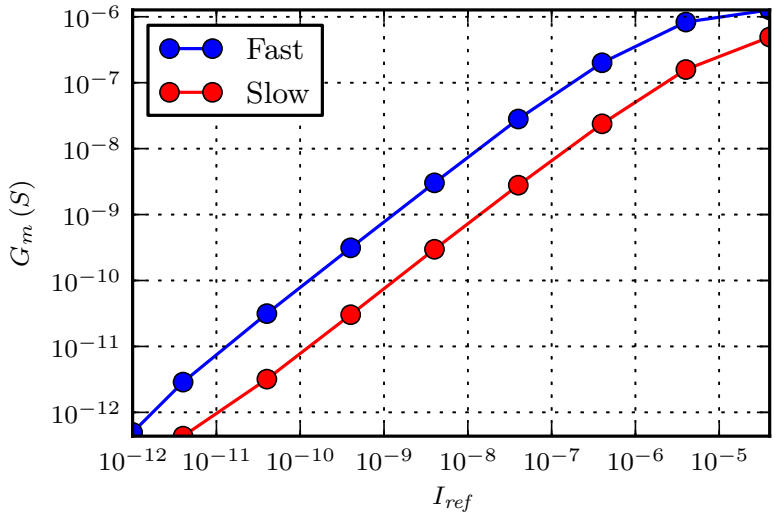


Figure 4.16: Simulated transconductance versus reference current for both modes. Transistors leave the weak inversion region at the larger currents. Available device models from the foundry are not characterized at pA current levels.

100 nS. Mode-switching ratios greater than 10× may be designed to allow narrower ranges of reference biases.

4.6.3 Linearization

Sub-threshold operation, required to achieve the low G_m, severely restricts the OTA's linear range. The single-diffusor, triode transistor was shown to be capable of the largest linear range, its disadvantage being the required generation of a common-mode bias voltage for the triode transistor gate [49].

Differential pair degeneration simultaneously reduces G_m and moderately increases the transconductor linear range. Large linearity increases require auxiliary bias- or input-shaping [109] circuitry with corresponding increase in total amplifier bias current and power, reducing efficiency.

For this design, two triode-mode transistors, Ma and Mb, are in parallel as the diffusors. The body connections allow transistor groups M1,Ma and M2,Mb to

share floating N-wells.

To generate the required diffusor gate bias with no external connections, we employ the inherent leakage and parasitic capacitance of back-to-back Schottky diodes D1-4 available in the CMOS fabrication process employed in the design. Internal nodes between D1/D2 and D3/D4 vary between $[V_{inb}, V_{cm}]$ and $[V_{cm}, V_{ina}]$, respectively, but the center node between D2 and D3 extracts the common-mode voltage due to the structural symmetry. The overall OTA schematic in Figure 4.10 shows the linearization transistors and common-mode biasing diodes.

4.7 NS430 Local processor

Included on the chip is an embedded processor which is code-compatible with the TI MSP430x series of micro-controllers which include the 20-bit extended memory addressing instruction extensions. Original design and implementation of the processor is described by Schemm in [110]. This processor core differs from the TI version by adding some extended addressing and post-increment instructions. Though the CPU instruction set itself is compatible with the TI parts, most of the included peripherals are different in function and configuration. Only the differences from the last 0.18 µm CMOS version in [110] are discussed here, the primary being porting the design to a new 0.13 µm CMOS process.

4.7.1 MSP430X-compatible CPU

The processor implementation is code-compatible with the TI MSP430X instruction set, which adds 20-bit memory addressing to the original MSP430 features. Notable exceptions to the compatibility is an addressing mode enhancement and a faster multiplier module.

Program code words beginning with the bits `00011` indicate the word is an extension and provides either the highest 4-bits of a 20-bit source/destination address or specifies additional addressing modes and other features [107]. This processor adds "indirect register" and "indirect auto-increment" addressing modes to those specified by the original TI version [110]. Indirect auto-increment modes support single-instruction, multiple-data operation, useful for multiply-accumulate operations of vectorized data.

The hardware multiplier peripheral for the processor is one clock cycle faster than the TI version [110]. This eliminates the need to have a 1-cycle delay instruction following a multiply operation with certain addressing modes. Combined with the addressing mode, this processor yields slightly higher performance for signal-processing tasks.

Besides a small mask-programmable read-only-memory (ROM), described in Section 4.7.3, the only program or data storage space is provided by 48 KiB static RAM (SRAM).

4.7.2 Clock Selection Module

Two crystal oscillators were included on the chip. One intended for a 32.768 kHz timing crystal and the second for an operation frequency in the tens of megahertz. Both timing sources may be used as the CPU's master clock. Bits 7:6 of the CPU's status register are named `SCG1` and `SCG0`, respectively. The first bit in this implementation enables the high-frequency clock while the second bit selects the clock source between the two options.

Clock switching circuitry must insure that glitches do not occur in the master clock signal. These short-duration transitions can violate timing constraints of the

CPU's various registers and cause unpredictable behavior. Figure 4.17 shows the schematic of a clock selection circuit which is guaranteed glitch-free from [2].

A multiplexer for data only uses only the last *AND-OR* stage of the figure. Each clock is applied to a multiplexer input and is allowed to propagate to the *OR* gate when the appropriate *select* code is applied. The modification of Figure 4.17 for clock switching uses D flip-flop (DFF) synchronizers to re-time the one-hot select signal into the respective input clock timing domain. For an arbitrary number of clock inputs, the input to each synchronizer is a "1" only after the previous one-hot select signal's $1 \rightarrow 0$ transition has propagated through the last-selected clock synchronizer. After a rising and falling edge of the previously-selected clock, a "1" propagates through the newly-selected clock synchronizer on a rising then falling edge and finally allows the output multiplexer to pass the new clock signal. The net effect is to first disable the previous clock and then enable the next clock signal in sequence, thus preventing glitches. Metastability is still possible at the input to the first DFF in the chains, but has such a low probability to be negligible for most applications – one in 150 years for two 100 MHz clocks [111]. A second DFF stage exponentially reduces the occurrence of metastability problems in the synchronizer.

Appendix B.3 contains the myHDL code to generate a clock selection module for an arbitrary number of clock inputs. The file also instantiates a 2-input version which was included in the NS430's digital design modules.

4.7.3 Boot ROM

Unlike the TI processors, this version does not include programmable non-volatile memory such as EEPROM or flash. At power-up, all RAM locations are in an unknown state and therefore unsuitable for code execution. A small mask-programmable

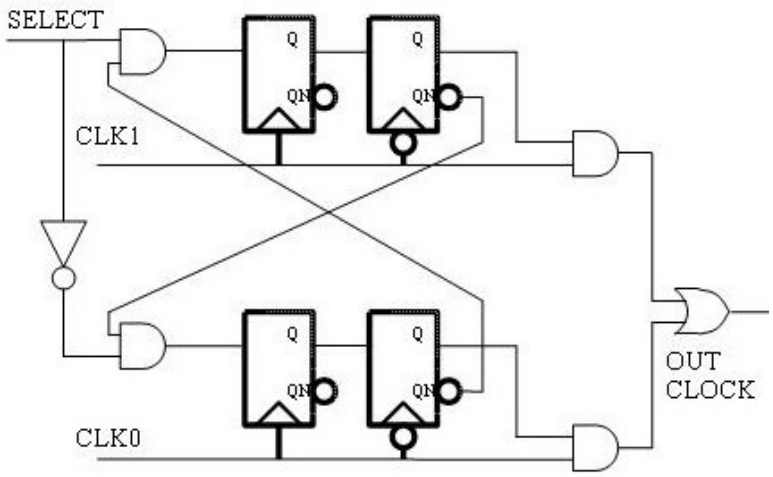


Figure 4.17: Glitch-free clock switching for two inputs. Image from [2].

ROM is present to assist in initializing the processor and RAM to a known state.

The program counter is loaded with the first ROM address location by design and code execution proceeds with the ROM contents. Code stored in the prototype chip's ROM was able to initialize an externally-connected flash chip via SPI port 0, copy its contents into the static RAM, and then jump to the first RAM location. This feature allows programming the flash chip by other means and also pre-filling the processor's memory with custom code and data.

4.8 Conclusion

This chapter described the design of the prototype chip and its various digital and analog sub-modules. The initial design exploration showed that an OTA with digital offset tuning was capable of reducing post-fabrication offsets to a level to maintain good system-level performance. Successive sections described the top-down design of the hardware from the overall layout to the core tunable OTA of the AHT coefficient calculation circuitry. Finally, aspects of the NS430 processor not de-

scribed in previous documents were discussed.

Chapter 5

Prototype System Testing

5.1 Introduction

The design described in Chapter 4 was fabricated in a 0.13 µm CMOS process and received for testing. Figure 5.1 is a composite microscope photograph of a fabricated die. The layout follows the architecture description of Section 4.3, shown in Figure 4.2. The NS430 processor is clearly seen in the lower-left while the regular structure of the 48 main harmonic channels and 16 auxiliary channels wrap around it.

Testing a prototype of this complexity required the design of a custom chip test fixture to control, stimulate, and measure the chip's function. The fixture and its capabilities are described in Section 5.2. Software drivers and packages written to provide a unified environment for testing sequences are separately covered in Section 5.3. Appendix C includes schematics for the test fixture boards for reference.

Nearly all of the prototype chip sub-systems described in Chapter 4 were tested and verified functional. Table 5.1 summarizes the state of each major or tested aspect of the system. The only failure was related to the mask-programmable ROM

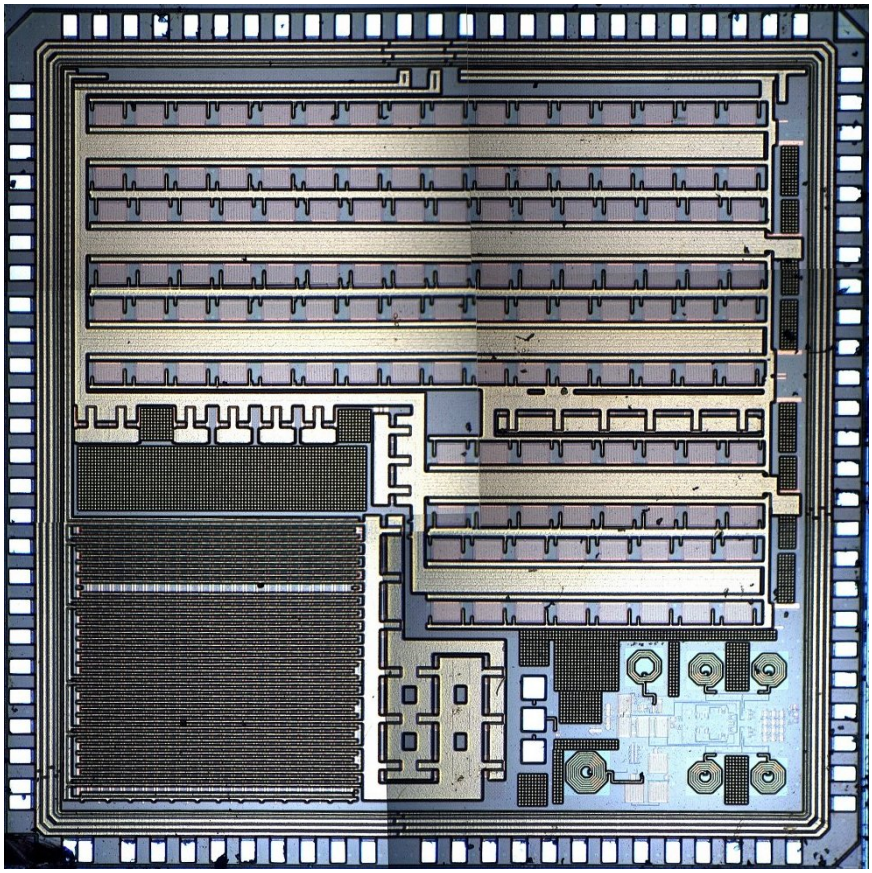


Figure 5.1: Die photograph assembled from a composite of optical microscope pictures. The outer dimensions are 4 mm × 4 mm. Lower-left quarter of the die is the NS430 processor with 48 kiB static RAM. The three full-width rows above the processor are the 48 harmonic projection blocks while the shorter rows to the right of the processor are 16 additional projection blocks whose multiplier inputs are directly connected to the processor. To the right of the projection groups are the four analog output multiplexers and pad buffers.

code in the NS430 which prevented execution of the ROM-resident interactive command line interpreter. The bugs in the ROM were discovered after the chip submission by disassembling the provided ROM bit file by hand. All other aspects of the prototype chip were either fully verified or performed with reduced functionality.

Section 5.4 discusses the analog signal path's measurement. A full characterization of the parallel AHT bank of channels was not possible due to a pad driver issue. However, the ability for the amplifier offset to be digitally calibrated was verified functional. This offset tuning is critical to achieve good system-level performance in the classification applications described in Chapter 3.

5.2 Chip Test Fixture

Due to the large number of control and measurement points designed into the prototype chip, a general-purpose testing fixture was also designed and constructed. This fixture provided the following features, most programmable through serial interfaces accessible by either a USB-connected PC or the on-chip processor:

- Six voltage regulators, four of which are independently-programmable power supply voltages for the chip's analog and digital sub-systems.
- Power supply current monitoring on all four test chip supply rails.
- Analog system master bias current control.
- Jumper selection of AHT control between the on-chip processor or an external serial port.
- Eight digital-to-analog converters to generate analog input signals and bias control tuning values.

Table 5.1: Summary of prototype chip functionality. Section references point to the component's design description.

Name	Good	Partial	Bad
Local Digital and Switching, 4.5.1			
SPI Slave, 4.5.1.1	X		
NCO, 4.5.1.2	X		
SwitchCtl, 4.5.1.3	X		
Level translators, 4.5.1.4	X		
Transmission gates, 4.5.1.3	X		
Off-Chip Analog Output, 4.5.3			
Mux input selection	X		
Pad buffer		X	
Wide-Range Digitally-Tunable OTA, 4.6			
OTA gain tuning, 4.6.1	X		
OTA offset tuning, 4.6.1	X		
OTA diff/S.E connections, 4.5.1.3	X		
OTA fast/slow switching, 4.6.2	X		
OTA G_m measurement, 4.5.2	X		
Auxiliary channels, 4.3			
NS430 mult inputs,	X		
`Arb0,1` direct amp access,	X		
NS430 Local Processor, 4.7			
Load code from flash	X		
Boot to ROM interpreter			X
SRAM read/write	X		
Multiplier module	X		
Clock switching module	X		
SPI 0,1 modules	X		
UART 0,1 modules	X		
I2C module	X		
Control of AHT config.	X		

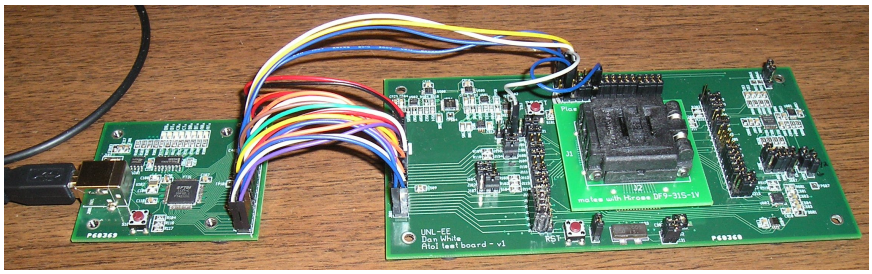


Figure 5.2: Chip test fixture and USB PC interface.

- Eight analog-to-digital channels to measure analog chip outputs, power supply currents, and generated bias voltages.
- High-frequency crystal clock oscillator monitor and injection.
- Ultra-low leakage amplifiers to buffer the high-impedance analog chip outputs.
- Non-volatile flash memory for processor code and general data storage.

While USB-based single-port asynchronous serial port (RS-232) converters are readily available, there are few multi-port, flexible general digital interfaces available which use a host's USB port for control. Such flexible interfaces were readily implemented using a PC's parallel port for bi-directional communication and many examples are available. Modern PCs, however, have long since ceased including parallel ports in their construction, opting for the now-ubiquitous USB port. The FT4232H chip made by FTDI provides four 8-bit I/O ports with flexible configuration with USB communication to a host. Two of the ports include hardware serial engines which can handle synchronous serial communication and be configured to be compliant with most serial interfaces. All four ports can also act as RS-232 asynchronous serial ports or be configured as 8 general-purpose input and/or output pins. This chip was selected to link the test fixture to a host PC for all communication and control. There were no commercially-available USB interfaces which

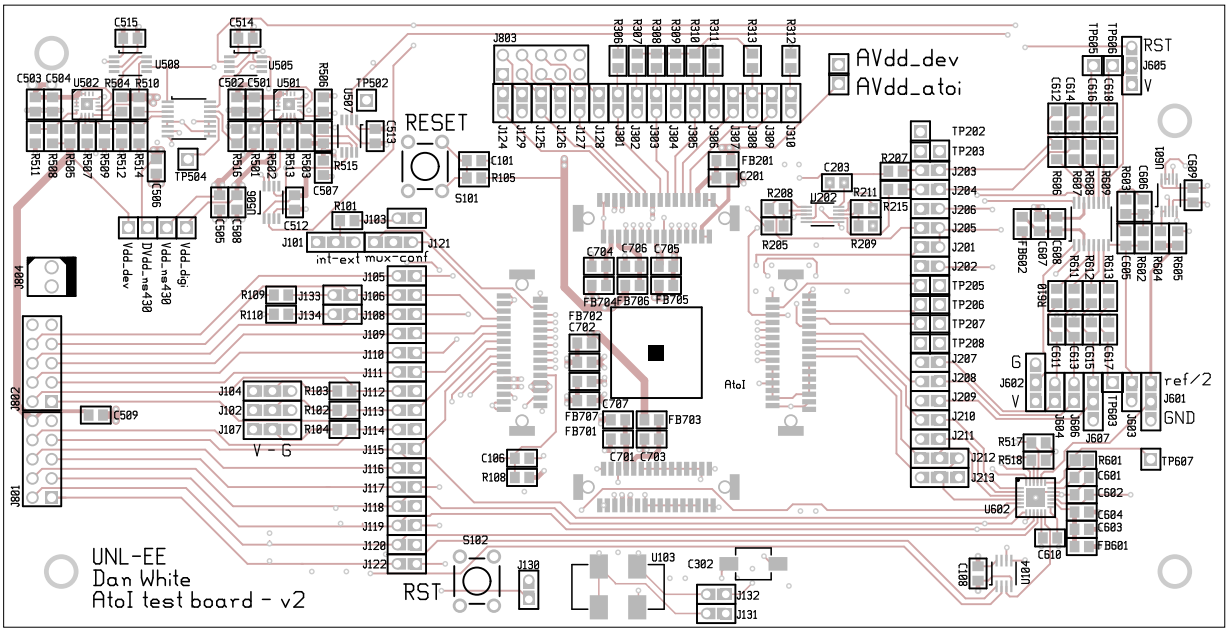


Figure 5.3: Test fixture printed circuit board layout, 1:1 scale, top view. The small filled square in the center represents the location of the test chip die and the surrounding square is the size of the QFN package (12 mm × 12 mm). The upper-left section has the 6 power supply regulators, with circuitry to digitally control four. The left connectors provide power and external serial communications. An 8-channel ADC is in the bottom-left while the 8-channel DAC is on the right-center side. The pads for the 100-pin test chip package socket interposer surround the center section of the board.

used this chip without compromising its flexibility. Therefore, a new USB interface board was designed and constructed for this project.

Figure 5.2 is a photograph of the USB interface and test fixture. The fixture sub-systems are described in more detail in the following sub-sections. Figure 5.3 shows a top view of the test fixture's printed circuit board design while Figure 5.4 shows the corresponding bottom view. Schematics for this board are included in Appendix C.1.

A socket was used to connect the prototype chips to the test fixture to facilitate measurements on each of the copies. The socket used is pictured in Figure 5.5,

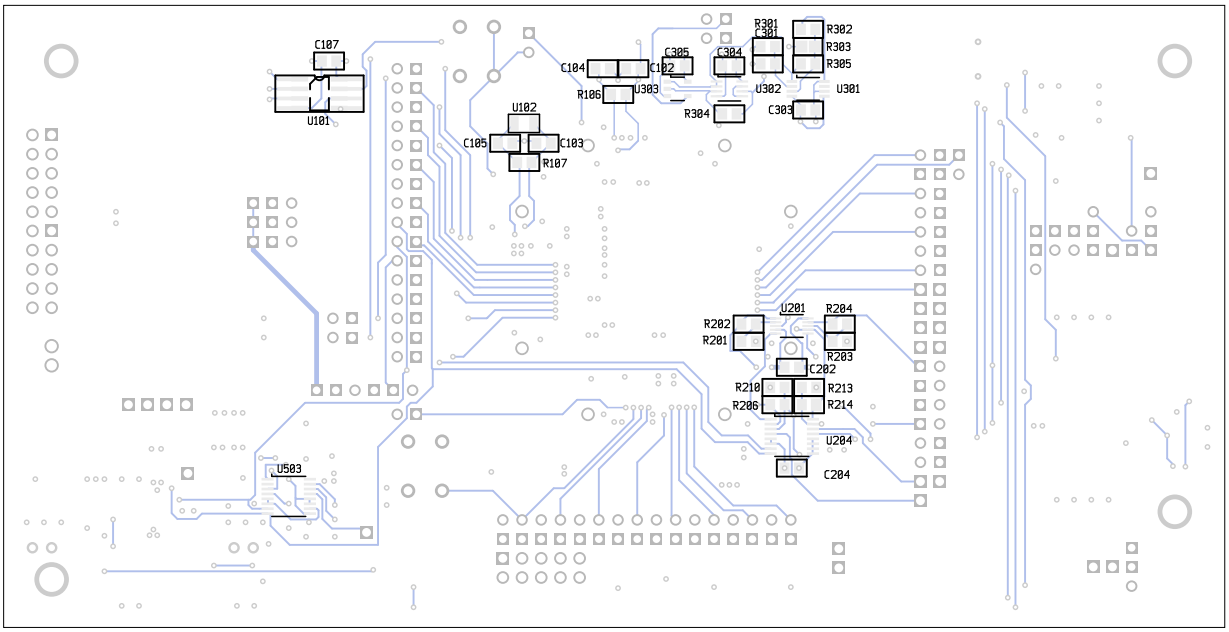


Figure 5.4: Bottom view of the test fixture circuit board. The upper-left device is the 1 megabit flash memory. The upper-center group of parts implement the clock oscillator monitor and auxiliary clock generator. Parts near the center implement the programmable bias current generator control.

Figure 5.5: 100-pin QFN socket manufactured by Plastronics.

shown with the lid open. Each pad of the 100-pin QFN package rests on a tiny spring-loaded pin and when the lid is closed a spring-loaded slug compresses the package into the bottom pins for a solid connection. Connections are routed from the package pins to an array of through-hole pins on the bottom of the socket. An interposer printed circuit board was designed to hold the socket and mate with four 25-position sockets which mate to the test fixture board. This arrangement allows the (expensive) socket to be re-used in other test fixtures.

5.2.1 Programmable power supplies

All six power supply rails are derived from an external 3.3 V source. Two triple-output low dropout (LDO) linear regulators then drop this voltage to the individual power domains. Components implementing these supplies are in the upper-left region of the board shown in Figure 5.3. The power rail names, ranges, and purposes are summarized in Table 5.2.

Table 5.2: Test fixture power supply names, ranges, and usage.

Name	Range (V)	Purpose
AVdd_dev	2.7	ADC, DAC, DigiPot, flash
Vdd_dev	2.5	Auxiliary for external devices
Vdd_digi	$0.50 - 1.30$	AHT digital
AVdd_atoi	$1.30 - 2.75$	AHT analog
Vdd_ns430	$0.50 - 1.30$	NS430 core
DVdd_ns430	$1.30 - 2.75$	NS430 pad I/O

Varying the LDO output is accomplished by a pair of resistors and yields the following voltage

$$V_{\text{reg}} = 0.5 \left(1 + \frac{R_2}{R_1}\right) \text{V} \,. \tag{5.1}$$

In all cases, resistor R_1 is fixed-value. Resistor R_2, however, is implemented by a combination of a fixed resistor R_2 and a digitally-controlled potentiometer con-

nected as a variable resistor. If x is the fractional position of the potentiometer's wiper with an end-to-end resistance of R_{pot}, the output voltage then becomes

$$V_{\text{reg}}(x) = 0.5\left(1 + \frac{R_2 + xR_{\text{pot}}}{R_1}\right)\text{V}\,. \tag{5.2}$$

Appropriate selection of the resistances ensures the voltage spans the required range and, most importantly, never allows an over-voltage condition at any potentiometer setting. The digital potentiometers used in the fixture have 8-bit resolution and are controlled over a common I2C bus with hard-wired addresses.

Each of the four power supplies for the AHT and NS430 include a current shunt for monitoring the supply current. The shunt voltage is amplified by individual OTAs with outputs which may be disabled. Only two ADC channels were available for monitoring the four currents and the enable inputs of the OTAs provide a convenient method for multiplexing the measurements. These enable pins are controlled by general-purpose output pins provided with the digital potentiometers and are hence under I2C control.

Figure 5.6 plots the measured supply current as a function of programmed voltage for the AHT digital circuitry named Vdd_digi in Table 5.2. The plot shows an unexpectedly-high current which varied strongly with voltage. This output only drives static CMOS logic gates and should have a sub-1 mA current. This excess supply current did not seem to affect the operation of the AHT digital circuitry, however. The plot suggests a forward-biased diode characteristic with a high series resistance. It is unknown the source of this error, but is presumed to be a layout error as it was observed on all chips tested.

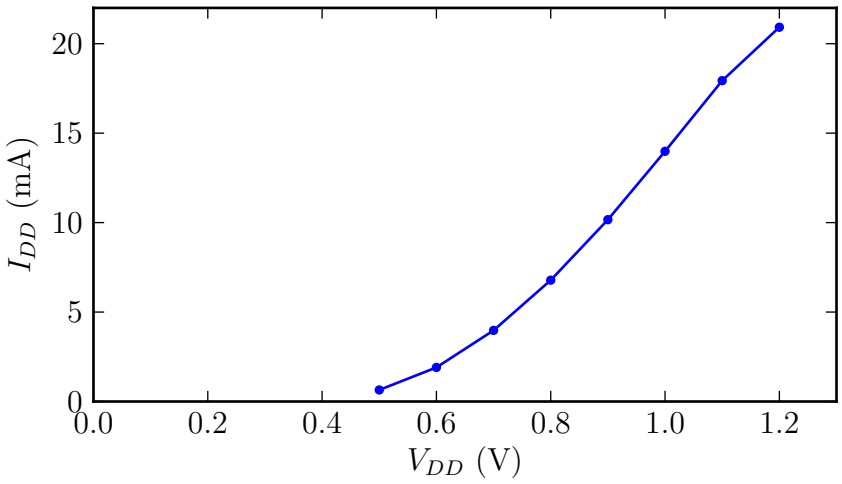


Figure 5.6: AHT digital circuitry 1.2 V nominal power supply current over the entire variable supply voltage range. Curve suggests a forward-biased p-n junction. The excess current did not seem to affect the digital circuit functionality.

5.2.2 Analog-to-digital converters

Eight channels of analog digitizers are provided by a single chip in the lower-right of Figure 5.3. The device chosen has 12-bit resolution and includes a programmable-gain amplifier. Multiple operating modes allow triggered burst sampling and data storage for later retrieval. All operation is controlled over a SPI bus interface. Table 5.3 lists the ADC channels and the connected signal.

Table 5.3: ADC input channels.

Channel	Signal
1	Main pad multiplexer A
2	Main pad multiplexer B
3	Auxiliary pad multiplexer A
4	Auxiliary pad multiplexer B
5	Supply current OTA outputs 1 and 2
6	Supply current OTA outputs 3 and 4
7	AHT analog bias generator monitor 1
8	AHT analog bias generator monitor 2

5.2.3 Digital-to-analog converters

The test board employs an 8-channel, 16-bit DAC for test signal generation, shown in Figure 5.3 on the right-center of the board. Flexible loading and updating of output values is done over an SPI port. A voltage reference is provided on the chip. It is used as the master reference for both the DAC and ADC to reduce measurement errors. With the 2.5 V reference, the DAC's resolution is approximately 38 µV per count. Table 5.4 lists the signals generated by the DAC.

Table 5.4: DAC output channels.

Channel	Signal
1	Main differential input, $V_{\text{in,A}}$
2	Main differential input, $V_{\text{in,B}}$
3	Common-mode input, V_{cm}
4	Core bias current generation
5	Pad buffer bias current generation
6	Unused
7	Unused
8	Unused

5.2.4 Ultra-high impedance buffer

Initial testing of the pad buffer amplifiers revealed that their output impedances, even in unity-gain buffer mode, was extremely high and on the order of the simulated value. Attaching a 10 MΩ oscilloscope probe to the output pins would indicate a characteristic capacitor discharge and a maximum output of around 250 mV. The output responded proportionally to tuning commands and input voltages but would remain very low, closely matching the effect a 10 : 1 attenuator in the circuit. Switching to a 1 MΩ probe confirmed the indication that the pad buffer amplifiers had a minimum output impedance of an estimated 100 MΩ.

Computer simulation of the pad buffer OTA circuit yielded an open-loop small-signal output resistance of 32 000 MΩ and a gain of about 320 V/V or 50 dB. An amplifier under unity-gain feedback reduces its effective output impedance from the open-loop value by the loop gain. This then gives a predicted unity-gain buffer output impedance of 32 000 MΩ/320 = 100 MΩ, which closely matches the observed behavior. Such an extremely high resistance is a direct effect of the cascode, long channel length, low current output stage. It was nevertheless surprising to observe such a high-impedance output which also matched the simulator's prediction so closely.

Since the amplifier was behaving properly as designed, it was necessary to measure voltages at the pad buffer pins with a loading impedance significantly greater than 100 MΩ. A simple non-inverting op-amp buffer was added to the test fixture board to buffer between the on-chip pad buffers and the ADC. The opamp selected was designed for ultra-low input bias current and leakage at the input nodes, with specified values of $I_{ib} < 1$ pA and $R_{icm} > 1$ GΩ. Installation of this additional buffer allowed observation of the full-range of output voltages from the chip.

5.3 Digital Testing

Nearly every aspect of the prototype system was designed to be digitally controlled or instrumented. This control on-chip was provided through a single SPI compatible serial port with four selectable destinations: the main 48-harmonic AHT chain, the main chain's output multiplexers and pad buffers, the secondary arbitrary-AHT chain, the secondary chain's output multiplexers and pad buffers. One pin on the chip was used to switch control of the SPI port to either the on-chip processor or an external PC connection via the chip test fixture. Because of the complexity of

coordinating the control points, most system testing was done using the PC connection.

5.3.1 PC libraries

The python programming language was used to develop the testing software in a hierarchical method. The software provided an object-oriented interface to the channel calibration and control bits and the chip test fixture's control points. When running the code under IPython [13], the entire environment is real-time interactive. This software-based, interactive control has the following benefits:

- Exploration of the chip's response to control commands.
- Real-time visualization with an oscilloscope of pin voltages while changing parameters.
- Documentation of the test procedures and conditions.

Included here are two listings which demonstrate the ease of accessing and modifying any accessible digital control point on both the fixture and the chip in an integrated fashion. The first listing loads a default set of fixture and chip parameters, routes the desired outputs to the pad buffer, and places both OTAs into calibration mode. From then, the amplifier offsets can be tuned to zero by writing appropriate values into the `a0.otaA.offset` and `a0.otaB.offset` parameters and measuring the offset with either the test fixture's ADC or by external metering.

```
1  #
2  # Put the first two OTAs in the auxiliary chain into calibration mode
3  #
4  import devboard as dev
5
6  # load default values for every subsystem
7  dev.init_breadboard('devboard-defaults.yaml')
8
9  arb = dev.arb    # auxilary chain class
10 amux = dev.amux  # aux mux + pad buffer
```

```
11|
12|a0 = arb.h[0]    # first harmonic in the chain
13|
14|#route the first harmonic outputs to the pad buffer, off-chip
15|amux.selA = 0
16|amux.write()    # send the configuration to the chip
17|
18|# set both integrators to calibration mode
19|a0.cal = 1
20|
21|a0.otaA.se = 0
22|a0.otaA.fast = 1
23|a0.otaA.gain = 10
24|a0.otaA.cint = 1
25|
26|a0.otaB.se = a0.otaA.se     # same settings for both
27|a0.otaB.fast = a0.otaA.fast # quadrature paths
28|a0.otaB.gain = a0.otaA.gain
29|a0.otaB.cint = a0.otaA.cint
30|
31|arb.write()     # send and activate the configuration
32|
33|# zero then release the outputs
34|a0.otaA.zero = 1
35|a0.otaB.zero = 1
36|arb.write()
37|
38|a0.otaA.zero = 0
39|a0.otaB.zero = 0
40|arb.write()
```

The following listing was used to ensure correct functioning of the test fixture's ADC and DAC. It sets up the ADC into full manual control mode, writes sequential values to one channel of the digital-to-analog converter. This output is directly connected to an input channel of the ADC with a jumper wire. Proper operation of the system yields a staircase plot of 2^{12} levels with 16 samples at each level. This is the expected result of measuring a 16-bit DAC with a 12-bit ADC.

```
1|# devboard jumper:
2|#   dac.vina (ch 2) --> adc.ch4
3|#   J604 -- J210
4|
5|from matplotlib import *  # plotting
6|import devboard
7|
8|adc = devboard.adc
9|
10|adc.triggerMode(adc.MODE_IDLE)  # config must happen in idle mode
11|adc.average(16, True)   # average 16 sequential conversions
12|adc.convst_spi(1)       # trigger conversion from SPI activity
13|
14|
15|#ch4 setup
16|adc.mux(4)
17|adc.channelMode(4, adc.SE)
18|adc.channelGain(4, 1)
19|
```

```
# go to mode 2 (full manual control)
adc.triggerMode(adc.MODE_MANUAL_MANUAL)

n = []
r = []
nbits = 16   # DAC resolution
for i in range(2**nbits):
    dac.set(2, i)
    adc.read()      #sham to trigger conversion
    sleep(160e-6)   #ensure 160us conversion time delay
    n.append(i)
    v = adc.read()

# should show a rising staircase from 0 to 2**12-1
# ADC resolution is only 12 bits compared to the DAC's 16-bit.
plot(n, v)
```

5.3.2 USB Interface Drivers

The manufacturer of the USB interface chip provides a few libraries for communicating with the FT4232H. Unfortunately, they do not make the source code available for the libraries nor is there a Python binding available. The `libFTDI` [11] project is an alternate library for driving FTDI's series of USB interface chips. It includes bindings for several languages, including Python, in addition to the normal C library.

A convenient feature of IPython when in interactive mode is instant access to function documentation while typing. The system displays either the short or long documentation included in the function's source code. While documentation is included within the C source code in `libFTDI`, the text was stripped off and therefore unavailable when constructing the python bindings. I added a step to the library build system to automatically extract and include the documentation in the python version. The modification reduced the need to have the library's documentation open in a second window while writing code, speeding development.

Available I2C or SPI libraries built on either of these base libraries were variously deficient in several aspects. Because the test fixture requires the use of all the

flexibility of the FTDI interface chip, custom I2C and SPI libraries were written on top of the `libFTDI` base library.

I2C driver. Capabilities included in the new library and not completely present in other available software includes:

- Bus clock stretching. Slave devices can implement a type of flow control by holding the clock line low during a bus transaction. Other libraries were not compliant with this I2C specification and would corrupt the sent or received data.

- Bus arbitration. The I2C specification allows multiple master nodes on a bus. When two devices attempt to use the bus at the same time, they are required to detect the collision and yield the bus to the "winning" master. The new library would properly detect a collision, properly release the bus, and report the loss of control to the calling code to allow a later transaction re-try.

- Open-collector pins. The two pins involved in the I2C connection are driven low but never explicitly driven high by any device on the bus, they are pulled high by a resistor. This allows the clock stretching and arbitration features with the "wired-OR" connection. It also allows I2C connections between devices having different power supply voltages; the USBIO board used 3.3 V while the test fixture had several power supplies around 2.5 V. Other libraries could only implement an I2C connection on the pins allocated to the hardware serial engine, which was not capable using open-collector outputs while enabled. The new library can implement an I2C interface on any of the 8 pins on any of the 4 ports available on the FT4232H chip and still leave the remaining pins available for general purpose I/O."

SPI driver. Similar to the I2C driver, the SPI driver has features necessary for control of the test fixture but not available in existing drivers. The features of the new library include:

- Chip select, CS. All other libraries used a single pin to activate the CS pin of a connected device, the same pin described in FTDI's application notes. This usage was not strictly required by the hardware architecture of the chip. Any general bit pattern can be used as a chip select with the new driver. This includes the common single-pin-per-device connection to select the destination of the bus' communication. It also allows external decoding of the applied bit pattern to control up to $2^5 - 1$ or 31 SPI devices from one 8-bit port. The latter feature was used to select between 4 devices with the 2 available pins on the interface.

- Pins not used in any of the SPI activities are available for general-purpose I/O. None of the other libraries could do this without significant modification.

- The communication clock rate could be requested to be any value. The software would calculate the required setup parameters and report back the actual rate being used. Bit rates up to 30 MHz were possible.

- Simultaneous read/write. No other libraries would both write a command to an SPI slave device and read the device's output bits at the same time. This capability is essential for SPI slaves configured as shift registers. An `exchange()` function was added to the `libMPSSE` library as a contribution back to the community, but the library itself was not used for the above reasons.

5.3.3 Chip digital testing

This section describes the procedures and results of testing the prototype chip's digital sections. In summary, all the tested design elements were verified through either direct measurement or by their effect on the system operation.

SPI interface. The USB interface was used to send data into the chip's SPI port. As the input bits were shifted in, the output bits were recorded also. Due to the shift-register operation of the AHT digital section, the shifted-out block of bits should be equal to the bits shifted in during the previous operation.

A test script repeatedly shifted into all four of the selectable configuration ports random data and recorded the output data. The script then compared the received data with the expected values and would abort operation on any bit errors. Bit rates up to the USB interface's maximum of 30 MHz were used with zero observed errors over operation times of several hours. The chip's maximum bit rate is unknown. The longest configuration has 48 harmonics, each with 48 bits, or 2304 bits per transaction. At 30 MHz, a full bus transaction would only take 76.8 µs, a small fraction of the typical 400 ms integration time used in the vehicle classification application.

NCO. The outputs of the numerically-controlled oscillator digital block were not directly accessible through any pins. In the standard harmonic correlation mode, the NCO outputs determine the analog ± 1 multiplier state. As such, the OTA outputs were observed while integrating a DC input voltage. This would yield steadily increasing or decreasing output voltages depending on the state of the NCO output.

The clock supplying all the NCOs has its own input pin on the chip and could

be directly clocked at arbitrary rates down to zero. The procedure used to verify correct NCO operation was as follows:

- Reset the NCO internal state to zero via the SPI port and load the NCO with a *fcw* value.
- Calculate the expected period on the output in terms of the number of input clock cycles.
- Apply a DC input voltage.
- Clock the NCO *clk* input pin.
- Count the clock rising edges until the OTA output changes direction.
- Continue clocking until the output changes back to the original direction.

The test procedure was repeated for several representative frequency control words. Each test yielded the expected number input clocks for the given control word. The digital behavior of the NCO had been thoroughly tested during the design phase by the `myHDL` test bench simulations, co-simulation between `myHDL` and two Verilog simulators, and simulation of the final logic including parasitic delays by a Verilog simulator.

SwitchCtl and level translators. The logic accepting the configuration and NCO outputs and outputting the appropriate control signals to the analog switches was not directly observable. Testing the NCO by its effect necessarily used both the `SwitchCtl` logic and the level translators. Because changing configuration bits and observing the OTA output voltages resulted in the expected behaviors with regard to operation modes, these design elements were implicitly verified.

5.4 Analog Measurements

This section describes and reports the measurements performed on the prototype chips to characterize and verify the operation of the AHT harmonic channels and associated support circuitry. The operation and performance of the digitally-controlled OTA calibration is included in this section.

5.4.1 Pad Buffers

The pad buffers utilized the existing OTA core design instead of a new buffer design to save development time. They are normally used in a closed-loop, unity-gain feedback configuration where the output node is connected to the inverting input. Output from the multiplexer is connected to the non-inverting OTA input.

It was discovered that in addition to the large output impedance of the pad buffer amplifier (from earlier Section 5.2.4), the amplifier had a restricted common-mode input range. This was a direct consequence of connecting the common-mode input signals for both the harmonic amplifiers and the pad buffers to the same signal. Only one pin on the chip was allocated to provide the input common-mode voltage for the OTA's linearization transistors – all OTAs were connected to this signal, but it is inappropriate for the pad buffer OTAs to be connected to this signal. When operated in unity-gain feedback mode, the buffers need the common-mode input voltage to track the input to properly supply a gate voltage to the G_m-reduction transistors in the input differential circuit. The effect of this is varying the transconductance of the input stage as the input varies. Since the input pair and diffuser transistors are p-type, the input stage only functions when the input is around or above the common-mode input level.

Figure 5.7 shows the output of the pad buffer attempting to track a sinusoidal

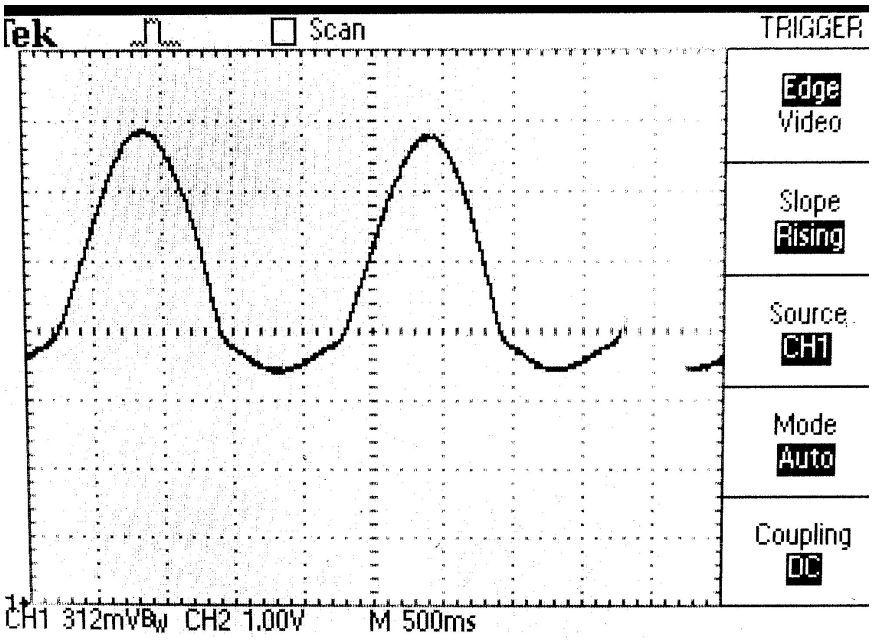


Figure 5.7: Output of pad buffer with multiplexer set to the output of a channel which was integrating a sinusoid with a 2 s period. This corresponds to an OTA output current of about ±40 pA peak-to-peak. The internal OTA output node is assumed to trace a complete sinusoid, with the lower-half saturation showing the pad buffer's inability to track inputs lower than the common-mode voltage.

input signal. The amplifier correctly buffers the input signal when above the common-mode voltage of 1.25 V, the mid-level between the power supply voltages of 0 V and 2.5 V. Below the common-mode level, the buffer fails to track the input signal because the input stage transistors are turned off as predicted by analysis.

Further investigation of the pad buffer's performance limitations is shown in Figure 5.8 which plots raw data obtained from the following repetitive sequence of operations while continuously sampling the output voltage:

- Zero the integrating capacitor.
- Apply a differential input voltage.
- Release the integrating capacitor zero switch.

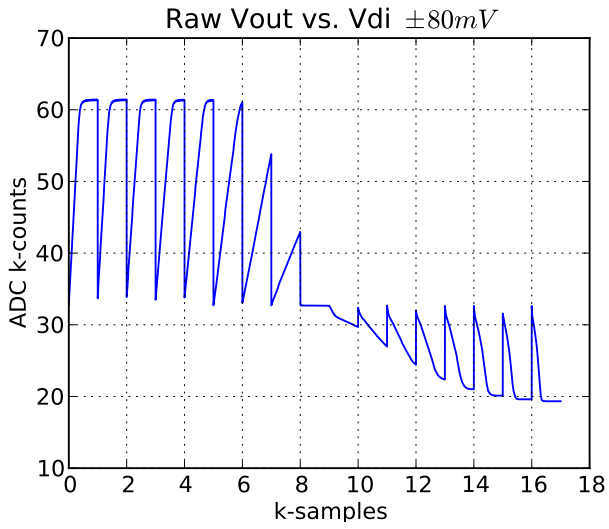


Figure 5.8: Sampled time-domain data from sequential integrate-reset sequences for a differential input voltage varying from −80 mV to 80 mV. For these measurements, the pad buffer was run in single-ended, unity-gain mode. The OTA outputs as measured through the pad buffer are inverted in relation to the internal signals. Due to the discussed limitation of the buffer's common-mode connection, its open-loop gain and output range are severely compromised for inputs below the half-supply level, causing the pad buffer's output to no longer track the on-chip multiplexer output.

- Wait for a fixed delay.
- Set the differential input to zero.

Figure 5.9 shows the calculated harmonic channel OTA's incremental output current inferred from the data of Figure 5.8, measured through the pad buffer and assuming an integrating capacitance of 44 pF. Differential inputs from −80 mV up through around +20 mV show a linear relationship between applied voltage and output current, giving a relatively constant G_m as expected. Inputs greater than +20 mV cause output voltages to move outside the range able to be buffered by the pad buffer and show no dependence on input magnitude, leading to a zero calculated incremental G_m. This effectively does not allow direct measurement of

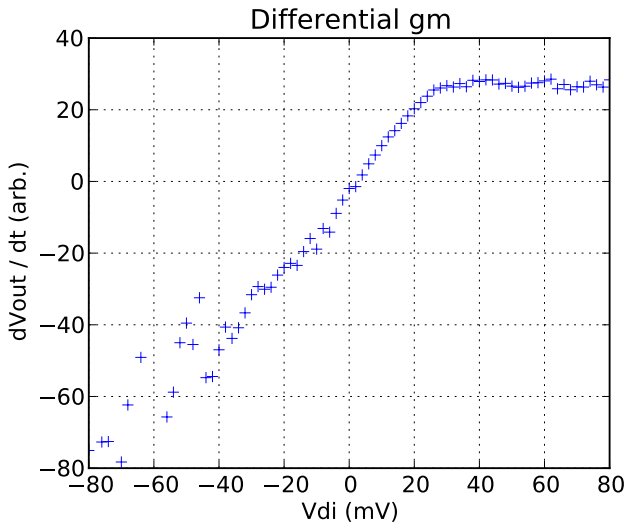


Figure 5.9: Shape of the G_m versus differential input voltage of an internal OTA integrator extracted from the data of Figure 5.8. The effect of the pad buffer's limited negative-going common-mode input range is also evident in the flat-topping for positive output swings. The reported and effective differential input voltage are inverted with respect to each other.

the harmonic channel OTAs.

5.4.2 OTA G_m characterization

The auxiliary chain of harmonic channels had two internal OTA outputs wired both to the output multiplexer and directly off-chip. This allowed direct measurement of two OTAs per chip copy by bypassing the multiplexer pad buffer. Figure 5.10 shows the raw data obtained from integrating DC differential inputs to these accessible OTAs. Due to the extremely long time constants created by these integrators, an adaptive measurement scheme was used.

At large magnitudes, the output current is relatively large and the output rises/falls from the zero reference relatively quickly. Avoiding output saturation then requires limiting the integration time. When the input magnitude is smaller, the output

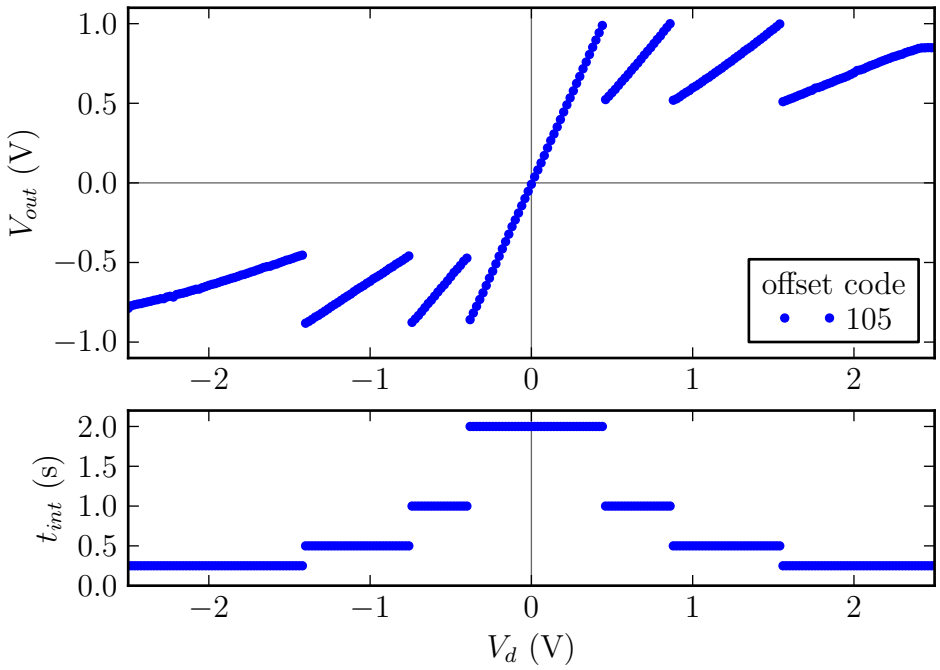


Figure 5.10: Raw measured data from measuring the integrated output current versus differential input voltage of OTA "`arb0`" on chip #14. The upper plot records the measured output voltage by the test fixture's ADC through the high-impedance buffer while the integration time used in the adaptive algorithm is shown in the lower plot. The full, power-supply-limited differential input voltage is shown, demonstrating the very wide linear input range of the OTA.

changes proportionally less rapidly. Adaptively changing the integration time such that the output voltage neither saturates nor is too small for a precise measurement gives a more complete picture of the amplifier's G_m. The lower plot of Figure 5.10 records the integration time used for a given differential input while the upper plot shows the resulting output voltage at the end of the integration time.

The data from Figure 5.10 was post-processed to infer the output current charging the capacitor from the resulting final integrated voltage using the equation

$$I_{out} = \frac{C_{int} V_{out}}{t_{int}} \,. \tag{5.3}$$

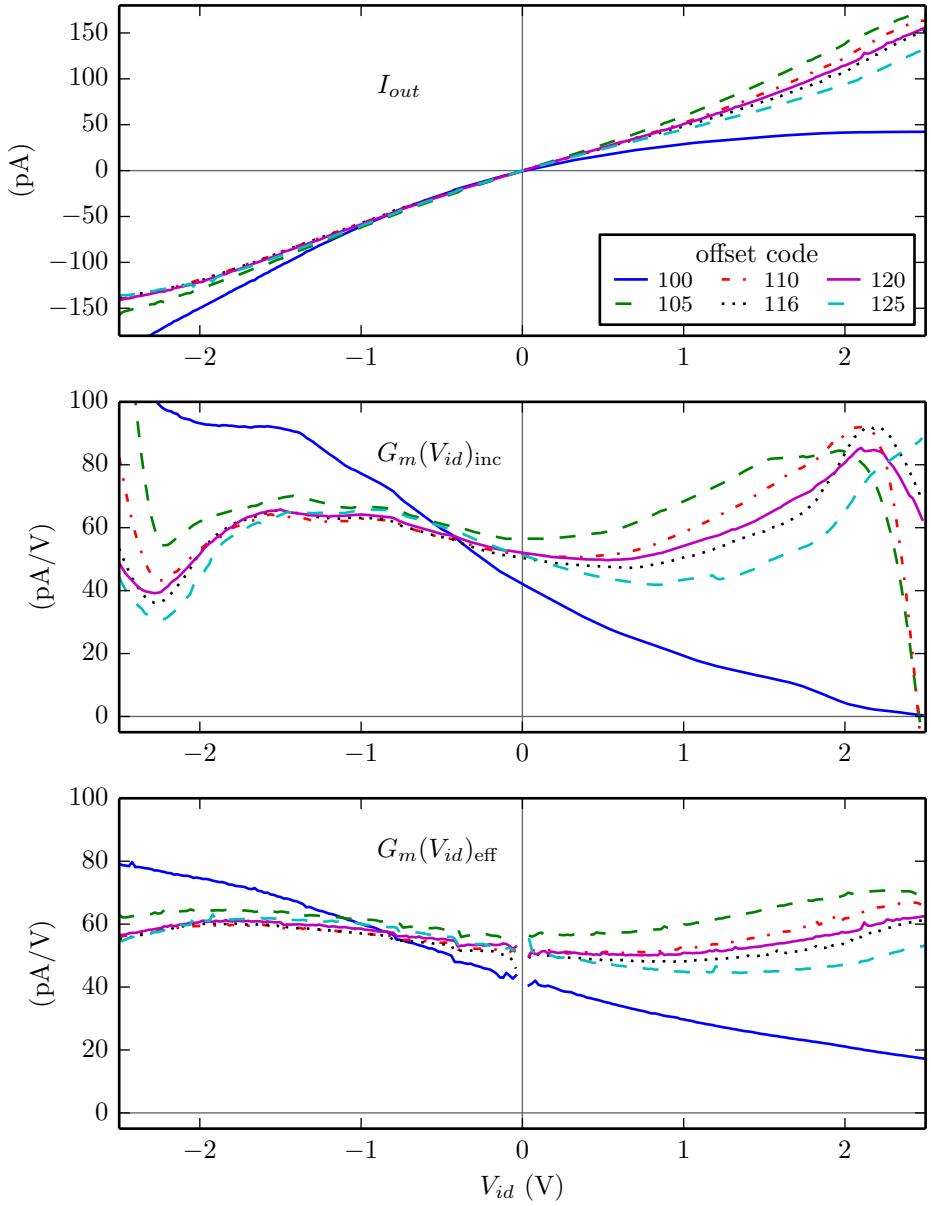


Figure 5.11: From top to bottom: calculated output current, incremental G_m, and effective G_m versus differential input voltage for the OTA "`arb0`" on chip #14. The apparent reduction of non-linearity in the bottom plot is a direct consequence of the definition. For input signals with an average value of 0 V, the lower plot best represents the non-linearity observed at the output. An offset code of 100 provides insufficient bias to the OTA, resulting in severe non-linearity. Other offset values show the characteristic G_m "wobble" and wide input range predicted by simulation.

The integration capacitance was taken to be 50 pF, which includes the 44 pF integrating capacitor and an estimated 6 pF parasitic capacitance added by bringing the internal OTA output node to a package pin. These extracted values are plotted against the input V_{id} in the top plot of Figure 5.11 for several values of the offset-tuning code.

Calculation of the overall OTA's transconductance may take two forms: the incremental G_m and the effective G_m. The incremental form is the slope of the I_{out} vs. V_{id} curve as a function of differential input voltage V_{id}, or merely the derivative. This is the middle plot of Figure 5.11 and is calculated from the data as

$$G_m(V_{id})_{\text{inc}} = \frac{dI_{out}}{dV_{id}} = \frac{C_{int}}{t_{int}} \cdot \frac{dV_{out}}{dV_{id}} \tag{5.4}$$

Effective transconductance is defined here as the output current which flows in response to a given input voltage. By this definition, G_m is simply the ratio of the quantities I_{out} and V_{id}, given as

$$G_m(V_{id})_{\text{eff}} = \frac{C_{int}}{t_{int}} \cdot \frac{V_{out}}{V_{id}} \tag{5.5}$$

The effective transconductance of Equation (5.5) is the one experienced by an input signal centered around zero as it is converted to an output current and is the most appropriate version for calculating non-linearity. Ignoring the case with the offset code of 100, the variation in G_m about the mean value for the lower plot of Figure 5.11 is approximately ±15% over the entire input signal range and much less for inputs restricted to ±1.5 V.

The shape of the measured G_m curve is consistent with simulations. For this circuit topology, the transconductance is not necessarily maximum around zero

input, but has a "wobble" on the outer regions of the plot.

5.4.3 Offset calibration

Initial design hardware requirements discussed in Chapter 3 indicated offset as the most severe performance specification on the integrators. To maintain good performance for the vehicle classification application, the amplifier offsets should have a standard deviation of 10 mV or less. Chapter 4 describes the OTA in this project which included post-fabrication digital offset calibration. This section presents results of tuning the offset on the set of 96 amplifiers in each chip tested.

The process of tuning each channel is a search for the digital offset code which minimizes the output current of an amplifier with a zero differential input voltage. Due to the extremely high output impedance of the OTA (estimated $> 32\,000\,\mathrm{M\Omega}$ from Section 5.2.4), the offset current is considered to be proportional to offset voltage. Minimizing the measured output offset voltage would then also minimize the desired output offset current. This simplification allows disconnecting the OTA's integrating capacitor C_{int}, speeding up the settling time by several orders of magnitude.

Because each amplifier operates independently, all channels may be calibrated simultaneously. This then limits the minimum calibration time to be proportional to the settling time of the output multiplexer and associated amplifiers. With the described test fixture, it was necessary to wait approximately 400 ms for the output channel A and B amplifiers to settle after changing the multiplexer input. Measuring the two output channels simultaneously doubles the effective measurement throughput.

Process variations in the individual amplifiers caused some calibration values to

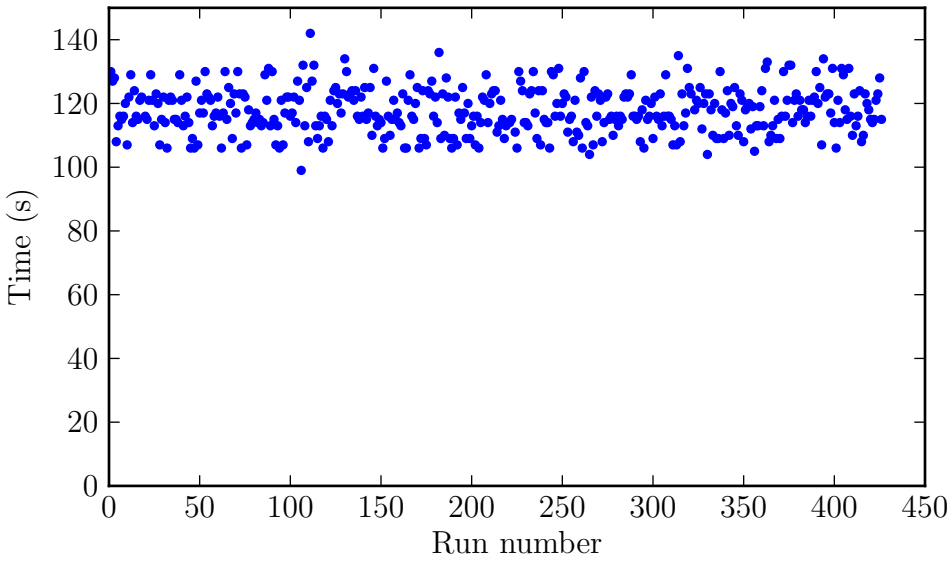


Figure 5.12: Time to convergence of repeated calibration runs for one chip. These times are for calibrating the offset of the two multiplexer pad buffers and all 96 OTAs in the main bank of 48 harmonic channels. The majority of the time is waiting for the pad buffers' outputs to settle to make a measurement.

converge faster than others. In these cases, the algorithm would no longer change or measure the offset of a declared "converged" OTA. This avoids unnecessary measurements and therefore speeds up calibration of the entire system. Figure 5.12 plots the time to calibration convergence for many repeated calibration runs for a single chip.

As an optimization problem, there are a multitude of ways to search for the code giving the minimum offset besides brute-force search. Restricting the hardware design to guarantee a monotonic code versus offset function shape allows utilizing the large class of optimization routines which expect "smooth" objective functions. This restriction is not strictly necessary as it effectively increases the chip area of the calibration circuitry. This area increase is multiplied by the number of integrators in the system, which with the AHT scheme can be large.

Several algorithms were tried for the offset code search including binary search,

direct Newton's method, and related quasi-Newton methods like BFGS [112]. Due largely to the fact that the hardware was designed for minimum calibration circuitry area and not guaranteed monotonicity, none of these algorithms gave acceptable overall performance. For OTAs which did have monotonic tuning curves, they found the minimum offset in the least number of steps. However, when a curve was not monotonic, the algorithms would diverge and sometimes enter large limit cycles.

The secant method [112] with a convergence detection heuristic was found to have stable performance and generally resulted in convergence to the best tuning value in fewer steps than a binary search during testing. Though both the measured offset values from the ADC and the calibration codes are integers, internal calculations should use higher precision. Rounding to the next x_{n+1} value should only be done at the end. Not doing so greatly increased the likelihood that the algorithm would enter a limit cycle with an amplitude greater than one.

Table 5.5 describes the core algorithm, omitting rounding, saturation, and range checking. According to the algorithm, the first step is a fixed guess which is the value of the pad buffer's offset code and the second step is another guess as a combination of the offset's sign and proportion of available tuning range. After these first two steps, the secant method is used to search for the minimum-offset code.

For the last steps in the convergence process, the algorithm typically makes code changes of ± 1 only. This is the feature which ensures a reliable convergence – the selected code will oscillate between two adjacent codes. The current termination condition detects only one oscillation cycle to balance the time to convergence with the certainty of finding the true optimum code.

Table 5.5: Secant method algorithm for calibrating the OTA offset.

- Start with the ending code from a previous calibration. Typically this code was that of the pad buffer.
- Wait for the buffer to settle after changing the multiplexer.
- Measure the offset with this initial value.
- Calculate the next input value by the following equation where $k \in (0, 1)$ (typically 0.1) and sgn$(\cdot)$ returns the argument's sign:

$$x_1 = x_0 - k \cdot (x_{max} - x_{min}) \cdot \text{sgn}\left(f(x_0)\right) \tag{5.6}$$

- Measure the offset after the buffer settling time.
- Repeat the following steps until declared convergence or the maximum number of steps have been reached:
  - Calculate the next value by:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} \tag{5.7}$$

  - Measure the offset $f(x_{n+1})$.
  - If one of the two following conditions are true, declare the channel converged:
    - The input has remained the same for three iterations.
    - The input has changed $+1, -1, +1$, or $-1, +1, -1$ between the last three iterations.

■

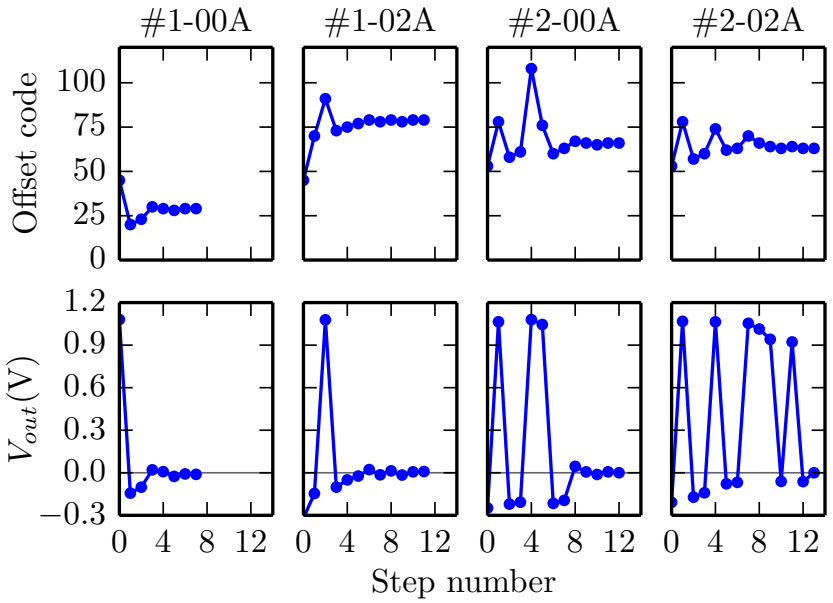


Figure 5.13: Representative calibration trajectories from chips `#1` and `#2` for channels `00A` and `02A`. Data for each OTA is arranged vertically. The upper row plots the input calibration code integer while the bottom row plots the resulting measured offset at each step of the algorithm.

Figure 5.13 shows representative calibration trajectories for an example run for two channels on each of two chips. The upper row displays the offset code while the bottom row plots the resulting measured offset at each algorithm step. Due to the high gain, observing the output voltage with a zero input allows rapid determination of the required offset tuning value. Trajectories for the tuning value and output offset followed the same pattern for all the channels on the chip.

Histograms of the converged offset code and resulting offset voltage are shown in Figure 5.14 for many calibration runs, 428 for chip #1 and 114 for chip #2. These were performed sequentially – when the calibration for all the channels finished, the test software would cycle power to the chip, load a default set of parameters, and begin another calibration run. As indicated, even for similar environmental conditions, an individual channel would not always converge to the same value. Typical

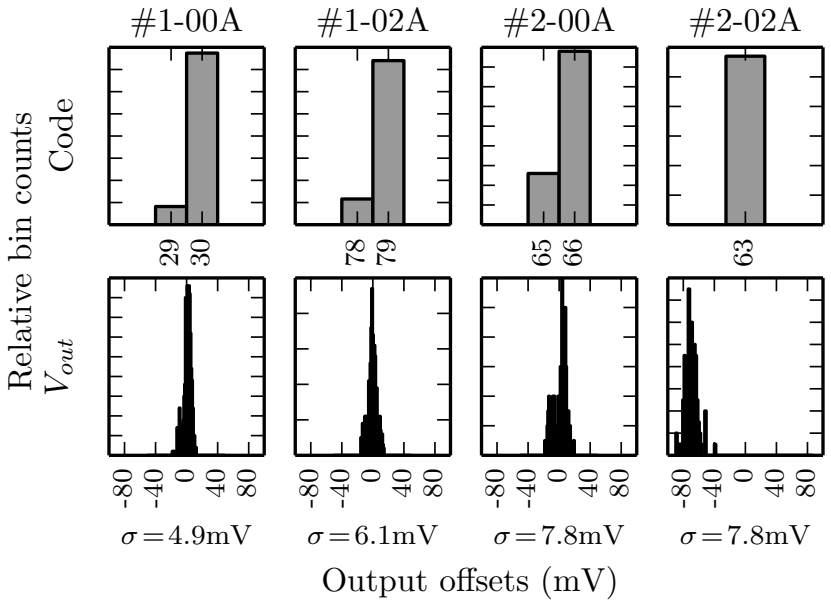


Figure 5.14: Histograms of calibration results for channels `#1,00-A`, `#1,02-A`, `#2,00-A`, and `#2,02-A` from many runs. The upper row is the tuning code while to lower row shows the post-calibrating residual offsets. All channels in this plot met the AHT application's requirement of offset standard deviations below 10 mV. Data for Chip #1 is for 428 runs while Chip #2 is for 114 runs.

channels converged to two adjacent values whose proportion was related to how close each corresponding offset was to zero. Noise and the convergence heuristic account for these variations.

Data from Channel `#2-02A` in both Figures 5.13 and 5.14 show the typical behavior of a marginal channel. The specific set of fabrication-related variations for this channel instance did not have a good offset code available to reach a zero offset voltage. One effect was the channel took more steps to converge than other more "well-behaved" channels. According the histogram of Figure 5.14, however, the algorithm declared the code `63` to be the best value in all 114 runs. It is unsurprising to observe this behavior at the current-output DAC (IDAC) code boundaries where many bits change in the code, in this case from a tuning code of `00111111` (63) to `01000000` (64). Such changes code boundaries magnify the effect of transistor

variations in binary-coded digital-to-analog converters.

The resulting offset from Channel `#2-02A`'s code of `63` had a mean residual offset of around −80 mV but with an acceptable standard deviation of 7.8 mV. Higher-level software could observe this residual offset on this and similar channels and apply an post-integration offset to shift the mean back to zero. This would increase the yield of usable channels on a given chip.

Figures 5.15, 5.16, 5.17, and 5.18 show a different view of the calibration results in light of anticipated performance in the vehicle classification application of Chapter 3. Each figure plots the mean and $\pm 1\sigma$ standard deviation levels for individual channels. The horizontal lines are at ±10 mV to indicate the range of offset allowable to maintain "good" classification performance. Titles on the figures report the number of passing channels, according to the magnitude of their offset variations, on that particular chip.

This type of testing data is rarely reported in the literature which includes prototype testing results. It is typical to show measurements of a "golden" chip and not comment on the prototype batch yield. These figures highlight the main issue of offset performance across a large number of channels as critical to keep good system-level performance. The amplifier tuning circuitry design can be optimized for specific applications to increase yield without unnecessary increase in the required chip area.

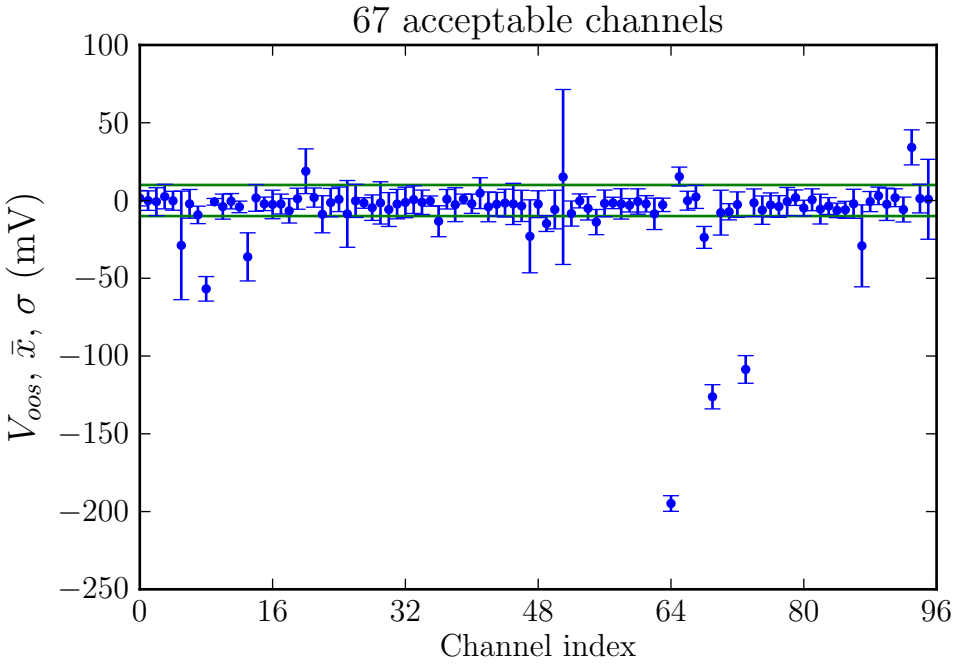


Figure 5.15: Chip #01 individual channel calibration statistics. The horizontal line represents the ±10 mV maximum acceptable offset value for "good" classification performance per Chapter 3, section 3.7.4.

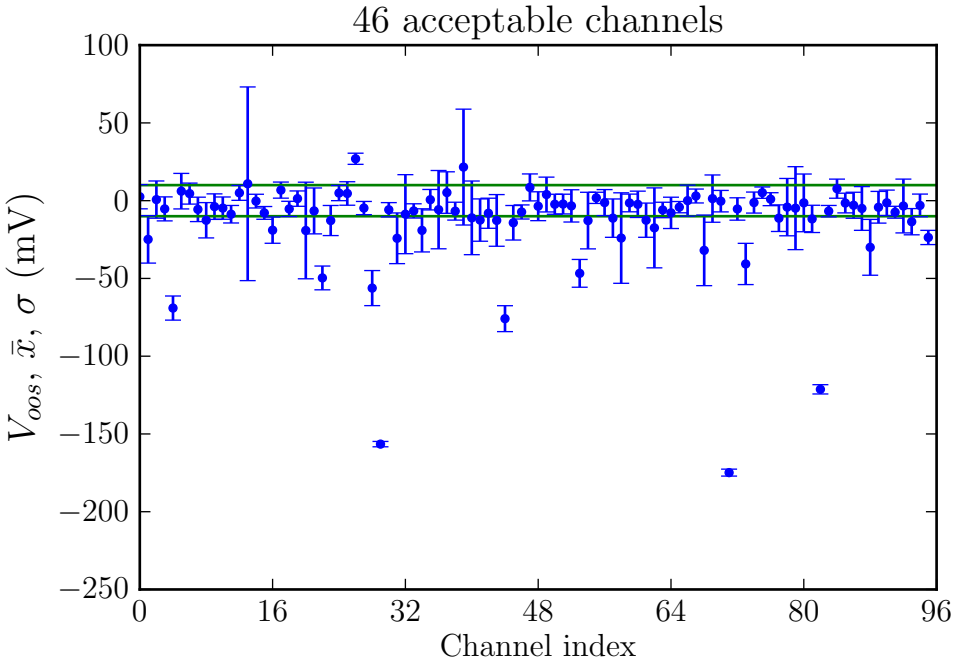


Figure 5.16: Chip #02 individual channel calibration statistics.

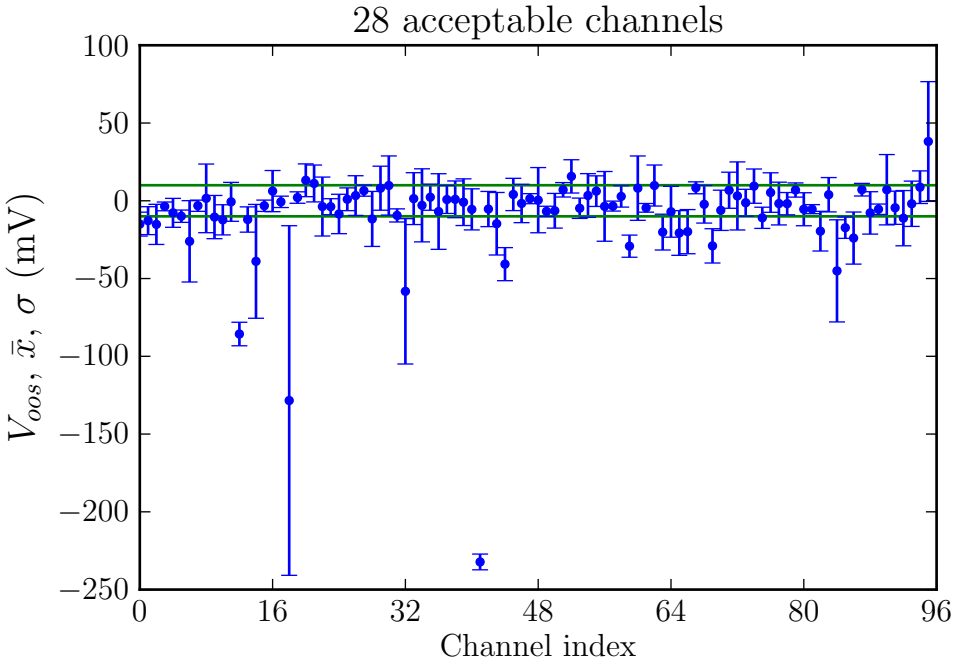


Figure 5.17: Chip #03 individual channel calibration statistics.

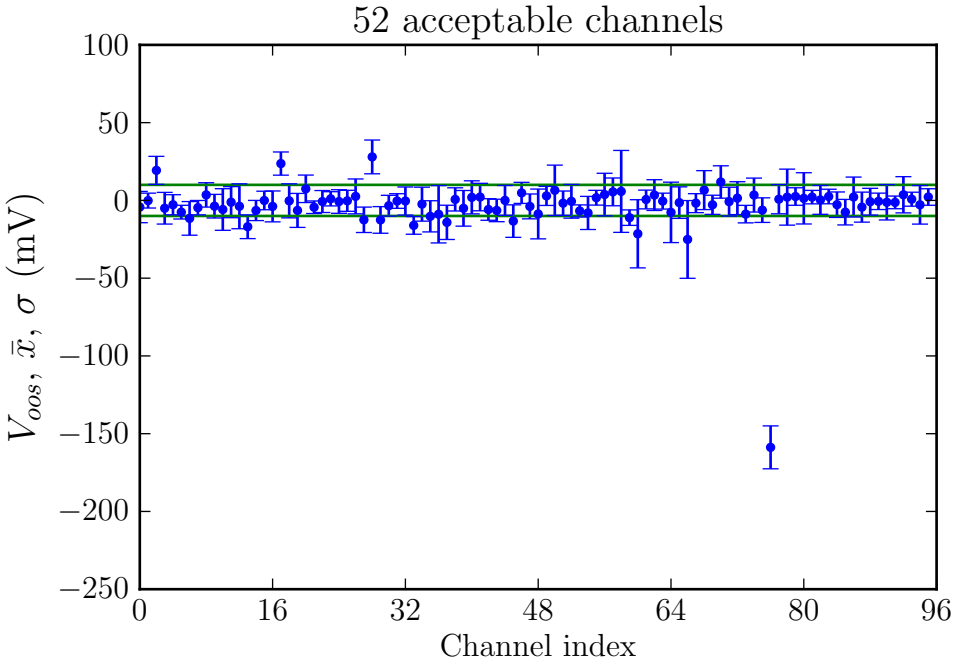


Figure 5.18: Chip #04 individual channel calibration statistics.

Figure 5.19 shows a final view of the calibration data which illustrates the reduction in offset variance after tuning. It shows the same four chips as the previous figures, but instead plots histograms of the converged tuning codes instead of the resulting output offset for all available calibration data. Each chip has clearly different mean code values, but the spans and distributions are similar. Optimum values span a majority of the available tuning range. This indicates the tuning range is a good match to the range of un-tuned offsets.

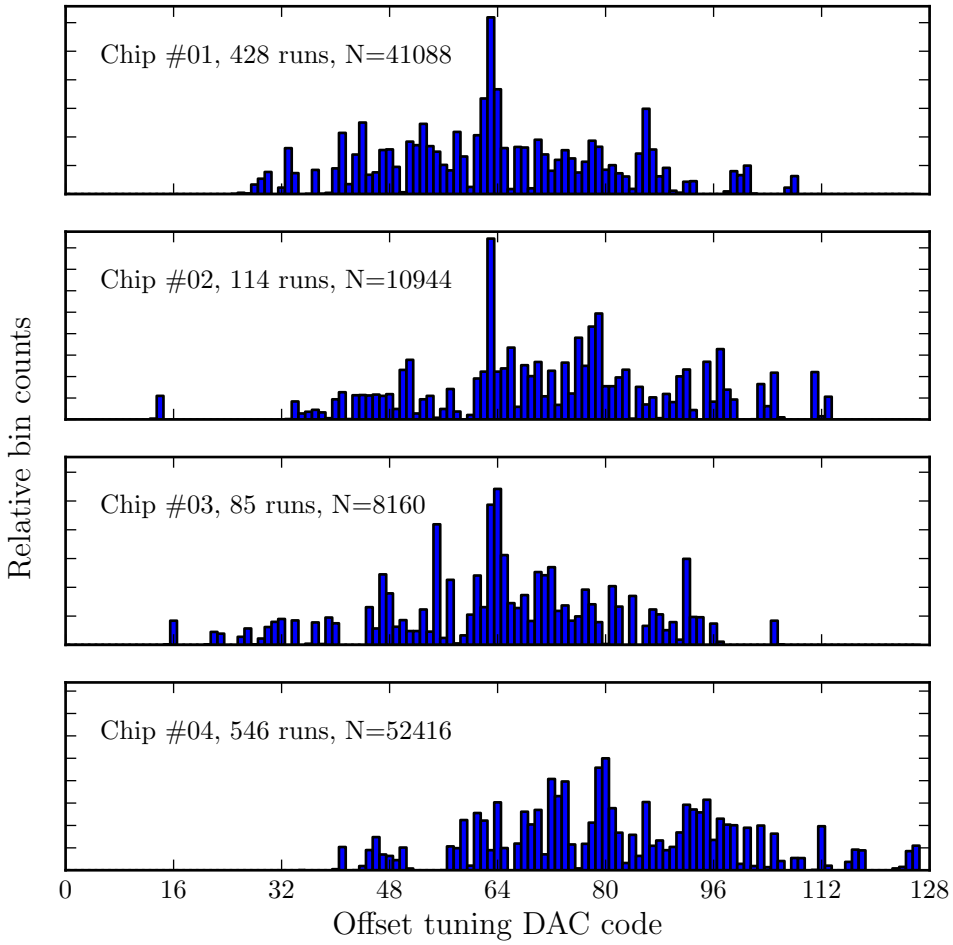


Figure 5.19: Aggregate histograms of the converged offset tuning DAC code for chips #1-4. Data is from consecutive calibration runs and includes all 96 OTAs in the main AHT group of channels. Across the four chips, this is data for 384 fabricated OTAs.

It should be noted that no particularly great effort was used in the design of the OTAs with regard to a low inherent offset deviation. The point was to demonstrate the feasibility of on-chip, on-line digital calibration of critical analog parameters. Chips #01 and #04 had a high number of usable channels while chip #03 demonstrated considerable variations. The fabrication service did not provide information on the location of particular dies within a wafer or even if the chips were from different wafers, so no conclusions can be drawn about cross-wafer variations.

An advantage of having parallel hardware paths is that re-tuning can be performed on a rolling basis. Since each channel is topologically identical, a small fraction of channels may be taken "out of service" for several integration times in order to re-zero the channel. With a small fraction increase in die area, such a system can continuously operate with good performance even during environmental changes such as temperature variations.

5.4.4 Harmonic projections

Figure 5.20 shows transistor-level time-domain simulations of the expected outputs of a harmonic channel's two quadrature outputs for three input frequencies. Equation (3.6) in Chapter 3 describes the expected output at the end of the integration period, indicated by the solid blue lines and filled red squares in Figure 5.20. Letting C be a constant including the $T/2\pi$ term and signal path gain, the y_Q coefficients reduce to the following for inputs at frequencies at n multiples of the basis function frequency with relative starting phases of $\Delta\phi$:

$$y_{Q,n}(\Delta\phi) = \begin{cases} \dfrac{C}{n}\cos(\Delta\phi) & n \in 1, 3, 5, \cdots \\ 0 & n \in 2, 4, 6, \cdots \end{cases} \tag{5.8}$$

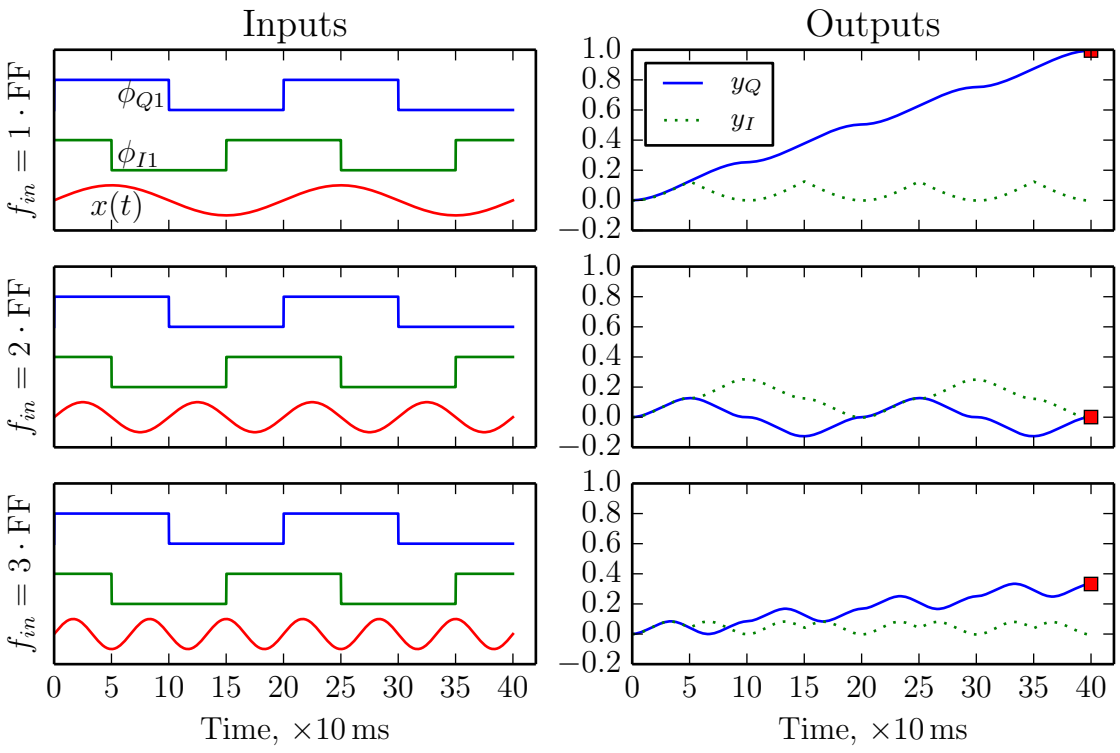


Figure 5.20: Transistor-level simulation of y_Q and y_I outputs during a 400 ms integration time for a 5 Hz harmonic channel. The first row is for an $x(t)$ input at the harmonic frequency with a starting phase equal to that of the ψ_Q basis function. The second and third rows are for input frequencies at twice and three times the harmonic frequency of 10 Hz and 15 Hz, respectively. Filled red squares are the values read out at the end of the period as the y_Q coefficients. The amplifier gain was set so the y_Q output at a 0° phase difference was unity for a full-scale input.

Access to the two direct integrator outputs on the auxiliary bank of 16 harmonics allowed testing of the signal path's performance in calculating AHT coefficients. Figure 5.21 plots the coefficients obtained from a 5 Hz harmonic with an input sinusoid of 5 Hz with varying relative starting phases. These are the $y_{Q,1}(\Delta\phi)$ values predicted by Equation (5.8). At 0° and 180°, the signal is in phase with the basis function and gives maximal or minimal coefficient values. At relative phases of 90° and 270°, the two functions are in quadrature and should result in a zero output

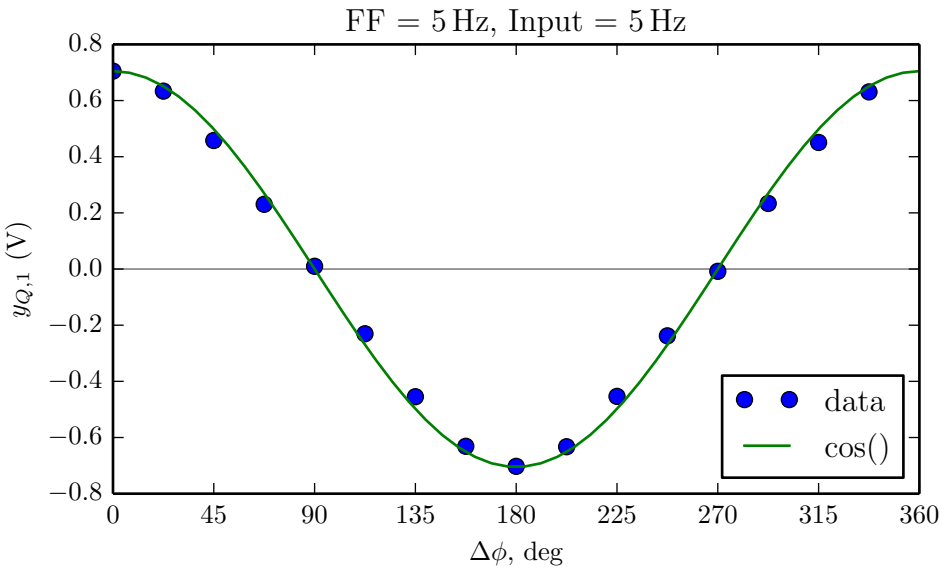


Figure 5.21: AHT y_Q coefficients resulting from correlating an input 5 Hz sinusoid with a 5 Hz harmonic frequency at various relative starting phase differences $\Delta\phi$. The data is plotted against the expected cosine function scaled to match the zero phase measurement.

voltage. For reference, the filled red squares in Figure 5.20 plot the values of $y_{Q,1}(0)$, $y_{Q,2}(0)$, and $y_{Q,3}(0)$.

Not shown are the similar measurements for the y_I values. For single-tone inputs, they will be maximum at 90° and 270° and zero at 0° and 180°. Taken as a complex pair, their magnitude should remain constant, or $\sqrt{y_I^2 + y_Q^2} = C/n$.

Figure 5.22 plots the measured outputs for input frequencies at the first seven harmonics of the 5 Hz basis function. The first plot, for $n = 1$, is the same data as Figure 5.21. According to Equation (5.8), the outputs should be all zeros for even harmonics and vary as the cosine of the relative phase for odd harmonics. Errors of various origin contribute to the even harmonic outputs being non-zero. Similarly, for the odd harmonics dynamic errors vary the amplitudes from the expected $1/n$ magnitudes as the harmonic number increases.

These dynamic errors increase when the fundamental frequency and the input

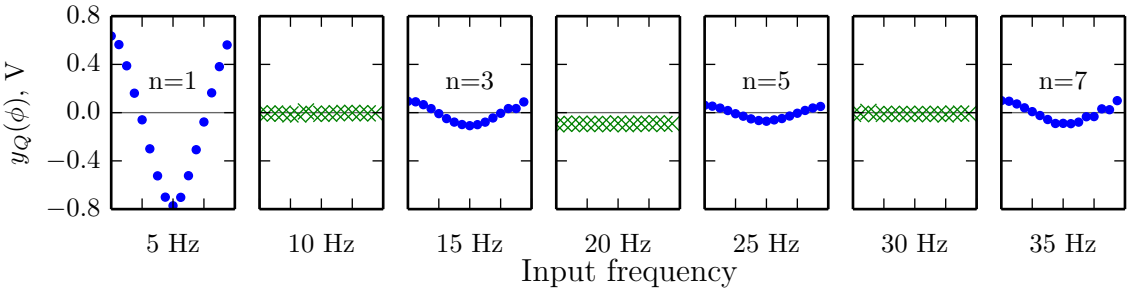


Figure 5.22: Measured integrator output voltage for input frequencies in steps of 5 Hz for a 5 Hz basis function. Each plot's horizontal axis is the relative starting phase between the basis function and input sinusoid over a complete cycle. Plots with blue circles indicate input frequencies where the results are expected to be non-zero.

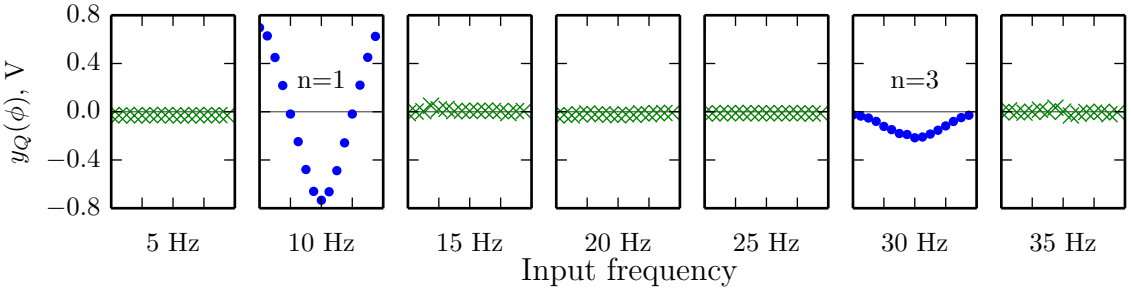


Figure 5.23: Measured integrator output voltage for input frequencies in steps of 5 Hz for a 10 Hz basis function. Each plot's horizontal axis is the relative starting phase between the basis function and input sinusoid over a complete cycle. Plots with blue circles indicate input frequencies where the results are expected to be non-zero. Higher frequencies for both the basis function and input reveal the presence of a dynamic error source caused by the non-symmetric single-ended integrator.

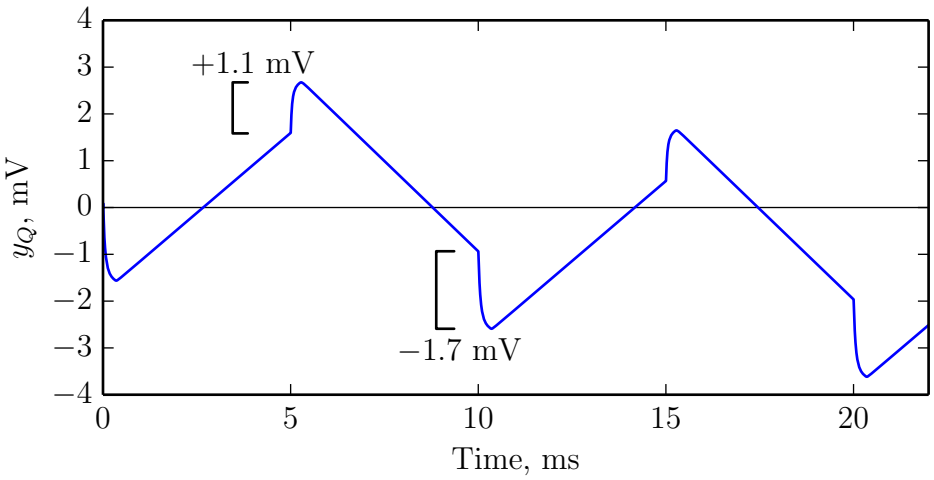


Figure 5.24: Time-domain simulation of a harmonic channel with 200 mV DC input and 100 Hz basis function. The ideal shape should be triangular only. Asymmetrical slew rate and switching time is evident by the output steps at each basis function edge.

frequency increase. Figure 5.23 uses the same input frequencies, but uses a 10 Hz basis function. All outputs are then expected to be zero except for the first harmonic input of 10 Hz and the third harmonic of 30 Hz. Especially for the third harmonic, signal-dependent errors accumulate to shift the outputs lower to a non-zero mean value over the changes in relative starting phase.

These errors are directly related to the size and number of waveform discontinuities applied to the OTA-C integrator. Non-zero input voltages at the time of a ± 1 edge of the basis function can exceed the OTA's slew rate. When this happens, the unequal delays through the OTA from each input to the single-ended output generate a charge pulse into the capacitor. For the present circuit, the pull-down path is faster, which explains the downward shift of the outputs at certain input/basis function frequency combinations.

The time-domain simulation of Figure 5.24 shows a channel's output waveform when a DC input is applied. Instead of a triangular shape, the waveform has dif-

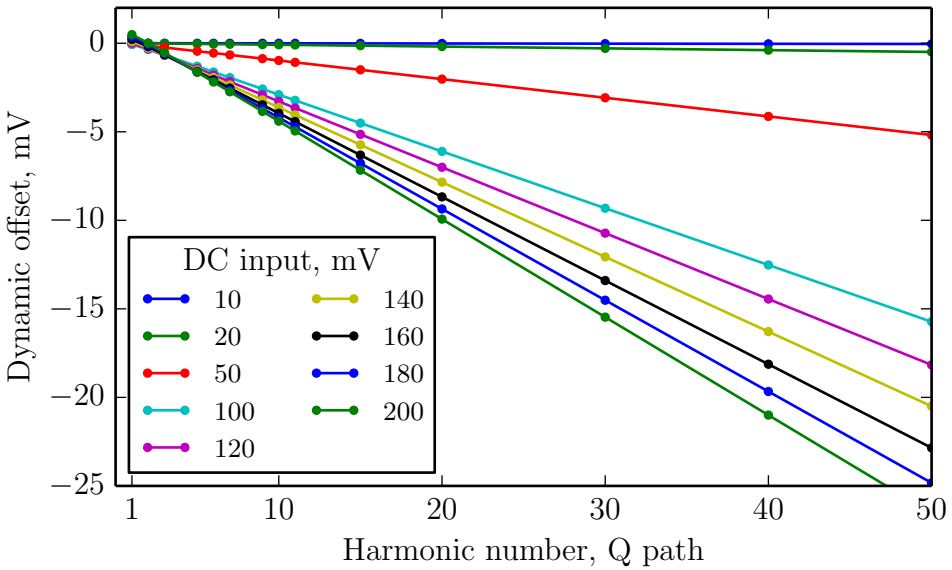


Figure 5.25: Simulation of a harmonic channel's output after an integration period. The dynamic offset is linearly dependent on both the input step size and the number of rise/fall transitions in a period (harmonic number). The basis function frequency was 5 Hz with a 400 ms integration time.

ferent glitch amplitudes depending on the sign and magnitude of the integrator's input change. Because the fast path pulls down the output for a longer time than it pulls up, the effect is to introduce a net negative trend to the output in addition to the normal integrating shape.

This dynamic error magnitude should increase linearly with both the number of transitions and for larger input amplitudes at the moment of switching. Figure 5.25 shows simulations performed to verify that this effect was indeed the dominant offset source. While this dynamic error can be well-characterized for DC inputs and by extension for known input amplitudes at the switching instant, its effect cannot be removed for arbitrary signal inputs.

Since the dynamic offset is the result of asymmetry in the integrator, the solution is to make the integrator path fully differential. Slewing delays in each half of the circuit would be matched by design to minimize this error source. A second

advantage for using a differential circuit is a factor of two increase in signal range and a simlar increase in dynamic range.

A fully-differential circuit requires more chip area. However, for low fundamental frequency operation, the integration capacitor requires the most die area. It is expected that a new OTA structure would still be able to be implemented completely under this capacitor, resulting in better performance with still no additional die area.

5.5 Conclusion

This chapter described the prototype chip testing, including its support circuitry. Most of the chip's modules were either directly accessible and functioned properly or were verified to be working in cooperation with other modules. Table 5.1 from the beginning of the chapter summarized those results.

The chip test fixture was designed to allow digitally-controlled and repeatable measurements of the chip. Having a software-controlled testing environment allowed documentation of both the output data and the specific conditions and sequence under which the measurements were taken. Because of this consistency, the recorded data could be analyzed across chip copies and over time.

Due to the pad buffer's common-mode connection, a limited amount of measurements could be performed on the parallel AHT channels. This was mostly limited to offset tuning and characterization and could not include the full, parallel, AHT coefficient generation for an input signal.

The offset measurements confirmed both the proper operation and necessity of a post-fabrication tuning step. As described in Chapter 3, Section 3.7.4, the vehicle classification task required strict limits for the hardware offset. Digital offset tuning

circuitry which could meet the specification was validated through simulation and verified with the measurements of Section 5.4.3.

Chapter 6

Conclusions and Future Work

This chapter briefly summarizes and draws some conclusions from the work as well as presents some future exploration opportunities relating to implementation of the AHT and its underlying circuitry.

6.1 Conclusions

The analog harmonic transform presented in this dissertation is an alternative spectral feature extraction technique to traditional FFT or other transforms. It is specifically designed to have a simple, regular hardware implementation.

The aspects of the AHT which set it apart from previous approaches are:

- Fouriér series coefficients may be calculated from the AHT coefficients by simple back-substitution. Under the assumption of a band-limited input signal and precision computations, the calculated FS coefficients are exact.
- Each quadrature set of AHT coefficient calculations are mutually independent. This is in strong contrast to most efficient digital transforms where all coefficients must be calculated simultaneously. Because of this independence, all

hardware associated with coefficients which are not used in back-end processing may be powered off to save energy.

In addition to the transform design, the corresponding hardware implementation prototyped a new digitally-controlled OTA tuning structure. The offset tuning was shown to reduce an amplifier's DC offset to levels acceptable for long time constant integrators. These levels of offset are below the offset achievable without a post-fabrication calibration step. Because the control and tuning of the amplifiers is digital, a system processor can tune channels on-demand in response to environmental or application needs.

6.2 Future Work

Broadly, future work can focus on two aspects: development of back-end algorithms and applications which use the AHT's ability to extract only selected coefficients, and improving the range of available amplifier tuning techniques suited to use in AHT hardware.

6.2.1 AHT Applications

Sections 3.6.1 and 3.6.1 of Chapter 3 detailed only two specific applications. It was shown that the AHT provided sufficient information to a neural network back-end to achieve similar classification rates compared with existing techniques using the same data sets. It is of interest to determine other application areas which can directly benefit from the AHT's unique feature set. These areas would typically be sensor systems with severe energy constraints.

Also, this work focused on AHT for signal analysis only. For band-limited signals, it would be interesting to evaluate and compare using the AHT and reconstruction with scaled basis functions to more traditional filtering operations. An application which may benefit from this technique is the signal path of hearing aids.

6.2.2 Amplifier Calibration

A large emerging area focused on small, low-energy sensors is "internet of things" (IoT). Wide deployment of simple sensors benefits from circuitry which implements only the required precision for the application. Such circuitry can be made smaller and lower-power with the addition of post-fabrication calibration such as provided with the prototype OTA. Other methods of tuning besides the current-steering DAC used in this work are worthy of study in light of these new application areas.

Re-design of the OTA to be fully-differential (FD) will eliminate the dynamic offset error source caused by waveform discontinuities arising from the switching-type ± 1 multiplication. This FD structure is better suited to the current-steering DAC structure for offset and gain tuning.

To optimally include post-fabrication tuning circuitry, it will be useful to generate a higher level analysis of the circuitry's range and precision. The range and bits of precision allocated to the gain and offset tuning of the OTA were informally determined through simulation. An optimal balance of die area and power allocated to the amplifier and to its tuning circuitry could be found for certain combinations. This would facilitate automated methods of generating low-power, precision amplifier schematics and layouts.

Bibliography

[1] D. J. White, P. E. William, M. W. Hoffman, S. Balkir, and N. Schemm, "Analog Sensing Front-End System for Harmonic Signal Classification," in *Proc. IEEE Int Circuits and Systems (ISCAS) Symp*, May 2012, pp. 1155 –1158. (document), 1.2, 3.7.4, 4.1, 4.2

[2] R. Mahmud. (2003) Techniques to make clock switching glitch free. EE Times article. Archived copy, alt: http://www.eetimes.com/story/OEG20030626S0035. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1202359 (document), 4.7.2, 4.17

[3] D. J. White, P. E. William, M. W. Hoffman, and S. Balkir, "Low-Power Analog Processing for Sensing Applications: Low-Frequency Harmonic Signal Classification," *Sensors*, vol. 13, no. 8, pp. 9604–9623, July 2013. (document), 1.2, 3.1

[4] P. E. William, "Low Complexity Feature Extraction for Classification of Harmonic Signals," Ph.D. dissertation, University of Nebraska-Lincoln, 2011. (document), 3.5, 3.6, 3.6.1, 3.2, 3.3

[5] K. Ashton. (2009, June) That 'Internet of Things' Thing. RFID Journal. Accessed August 2011. [Online]. Available: http://www.rfidjournal.com/articles/view?4986 1.1

[6] The MOSIS Service. [Online]. Available: http://www.mosis.com 1.2

[7] D. J. White, M. W. Hoffman, and S. Balkir, "Digital Offset Calibration for Long Time-Constant Sub-Threshold OTA-C Integrators," *Submitted to IEEE Transactions on Circuits and Systems II: Express Briefs*, 2014. 1.2

[8] The gEDA project. Accessed: August 31, 2013. [Online]. Available: http://www.geda-project.org 1.3

[9] Gwave - a waveform viewer. Accessed: August 31, 2013. [Online]. Available: http://gwave.sourceforge.net/ 1.3

[10] NumPy. Accessed: August 31, 2013. [Online]. Available: http://www.numpy.org/ 1.3

[11] libFTDI - FTDI USB driver with bitbang mode. Accessed: August 31, 2013. [Online]. Available: http://www.intra2net.com/en/developer/libftdi/ 1.3, 5.3.2

[12] Future Technology Devices International, Ltd. Accessed: August 31, 2013. [Online]. Available: http://www.ftdichip.com 1.3

[13] F. Pérez and B. E. Granger, "IPython: a System for Interactive Scientific Computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org 1.3, 5.3.1

[14] M. Shell. The IEEEtran Homepage. Accessed: August 31, 2013. [Online]. Available: http://www.michaelshell.org/tex/ieeetran/ 1.3

[15] E. A. Vittoz, "Future of analog in the VLSI environment," in *Proc. IEEE International Symposium on Circuits and Systems*, 1990, pp. 1372–1375 vol.2. 2.2.1, 2.2.2

[16] M. Degrauwe, J. Rijmenants, E. Vittoz, and H. De Man, "Adaptive biasing CMOS amplifiers," *IEEE J. Solid-State Circuits*, vol. 17, no. 3, pp. 522–528, 1982. 2.2.1

[17] M. Degrauwe and W. Sansen, "Novel adaptive biasing amplifier," *Electronics Letters*, vol. 19, no. 3, pp. 92–93, 1983. 2.2.1

[18] H. Parzhuber and W. Steinhagen, "An adaptive biasing one-stage CMOS operational amplifier for driving high capacitive loads," *IEEE J. Solid-State Circuits*, vol. 26, no. 10, pp. 1457–1460, 1991. 2.2.1

[19] S. Baswa, A. Lopez-Martin, R. Carvajal, and J. Ramirez-Angulo, "Low-voltage power-efficient adaptive biasing for CMOS amplifiers and buffers," *Electronics Letters*, vol. 40, no. 4, pp. 217–219, 2004. 2.2.1

[20] A. Nosratinia, M. Ahmadi, G. Jullien, and M. Shridhar, "High-swing, high-drive CMOS buffer," *IEE Proceedings - Circuits, Devices and Systems*, vol. 142, no. 2, pp. 109–112, 1995. 2.2.1

[21] S. Kim, Y.-S. Son, and G.-H. Cho, "Low-power high-slew-rate CMOS buffer amplifier for flat panel display drivers," *Electronics Letters*, vol. 42, no. 4, pp. 214–216, 2006. 2.2.1

[22] A. Lopez-Martin, S. Baswa, J. Ramirez-Angulo, and R. Carvajal, "Low-Voltage Super class AB CMOS OTA cells with very high slew rate and power efficiency," *IEEE J. Solid-State Circuits*, vol. 40, no. 5, pp. 1068–1077, 2005. 2.2.1

[23] J. A. Galan, A. J. Lopez-Martin, R. G. Carvajal, J. Ramirez-Angulo, and C. Rubia-Marcos, "Super Class-AB OTAs With Adaptive Biasing and Dynamic

Output Current Scaling," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 3, pp. 449–457, 2007. 2.2.1

[24] E. Vittoz, "Low-power design: ways to approach the limits," in *1994 IEEE International Solid-State Circuits Conference. Digest of Technical Papers. 41st ISSCC.*, 16-18 Feb. 1994, pp. 14–18. 2.2.2, 2.2.2, 2.3.3, 3.1

[25] R. Sarpeshkar, "Analog Versus Digital: Extrapolating from Electronics to Neurobiology," *Neural Computation*, vol. 10, no. 7, pp. 1601 – 1638, 1998. 2.2.2, 2.2.3, 3.1

[26] B. Calhoun and A. Chandrakasan, "Characterizing and Modeling Minimum Energy Operation for Subthreshold Circuits," *Proceedings of the 2004 International Symposium on Low Power Electronics and Design, 2004. ISLPED '04.*, pp. 90 – 95, 2004. 2.2.2

[27] A. Tajalli, E. Brauer, Y. Leblebici, and E. Vittoz, "Subthreshold Source-Coupled Logic Circuits for Ultra-Low-Power Applications," *IEEE J. Solid-State Circuits*, vol. 43, no. 7, pp. 1699–1710, 2008. 2.2.2

[28] I. J. Chang, S. P. Park, and K. Roy, "Exploring Asynchronous Design Techniques for Process-Tolerant and Energy-Efficient Subthreshold Operation," *IEEE J. Solid-State Circuits*, vol. 45, no. 2, pp. 401 –410, Feb. 2010. 2.2.2

[29] L. Oakes, A. Westover, J. W. Mares, S. Chatterjee, W. R. Erwin, R. Bardhan, S. M. Weiss, and C. L. Pint, "Surface engineered porous silicon for stable, high performance electrochemical supercapacitors," *Scientific Reports*, vol. 3, Oct 2013. [Online]. Available: http://dx.doi.org/10.1038/srep03020 2.3

[30] E. Vittoz and J. Fellrath, "CMOS analog integrated circuits based on weak inversion operations," *IEEE J. Solid-State Circuits*, vol. 12, no. 3, pp. 224–231, 1977. 2.3.2

[31] M. Steyaert, P. Kinget, and W. Sansen, "Full integration of extremely large time constants in CMOS," *Electronics Letters*, vol. 27, no. 10, pp. 790–791, 1991. 2.3.2

[32] A. Arnaud, R. Fiorelli, and C. Galup-Montoro, "Nanowatt, Sub-nS OTAs, With Sub-10-mV Input Offset, Using Series-Parallel Current Mirrors," *IEEE J. Solid-State Circuits*, vol. 41, no. 9, pp. 2009–2018, 2006. 2.3.2

[33] R. Rieger, A. Demosthenous, and J. Taylor, "A 230-nW 10-s time constant CMOS integrator for an adaptive nerve signal amplifier," *IEEE J. Solid-State Circuits*, vol. 39, no. 11, pp. 1968–1975, 2004. 2.3.2

[34] I. F. Triantis and A. Demosthenous, "An improved, very long time-constant CMOS integrator for use in implantable neuroprosthetic devices," in *Proc. European Conf. Circuit Theory and Design*, vol. 3, 2005. 2.3.2

[35] T. Kwan and K. Martin, "An adaptive analog continuous-time CMOS biquadratic filter," *IEEE J. Solid-State Circuits*, vol. 26, no. 6, pp. 859–867, 1991. 2.3.2, 2.3.4

[36] K.-C. Kuo and A. Leuciuc, "A linear MOS transconductor using source degeneration and adaptive biasing," *IEEE Trans. Circuits Syst. II*, vol. 48, no. 10, pp. 937–943, 2001. 2.3.2, 2.3.4

[37] A. A. Fayed and M. Ismail, "A low-voltage, highly linear voltage-controlled transconductor," *IEEE Trans. Circuits Syst. II*, vol. 52, no. 12, pp. 831–835, 2005. 2.3.2, 2.3.4

[38] A. Tajalli and Y. Leblebici, "A widely-tunable and ultra-low-power MOSFET-C filter operating in subthreshold," *Custom Integrated Circuits Conference, 2009. CICC '09. IEEE*, pp. 593 –596, sept. 2009. 2.3.2, 2.3.4

[39] I. Pachnis, A. Demosthenous, and N. Donaldson, "Comparison of Transconductance Reduction Techniques for the Design of a Very Large Time-Constant CMOS Integrator," in *Proc. 13th IEEE Int. Conf. Electronics, Circuits and Systems ICECS '06*, 2006, pp. 37–40. 2.3.2

[40] B. Linares-Barranco and T. Serrano-Gotarredona, "On the design and characterization of femtoampere current-mode circuits," *IEEE J. Solid-State Circuits*, vol. 38, no. 8, pp. 1353–1363, 2003. 2.3.3

[41] B. Gregoire and U.-K. Moon, "A Sub 1-V Constant Gm ndash; C Switched-Capacitor Current Source," *IEEE Trans. Circuits Syst. II*, vol. 54, no. 3, pp. 222–226, 2007. 2.3.3

[42] J. Brugler and P. Jespers, "Charge pumping in MOS devices," *IEEE Transactions on Electron Devices*, vol. 16, no. 3, pp. 297–302, 1969. 2.3.3

[43] U. Cilingiroglu, A. Becker-Gomez, and K. T. Veeder, "An evaluation of MOS interface-trap charge pump as an ultralow constant-current generator," *IEEE J. Solid-State Circuits*, vol. 38, no. 1, pp. 71–83, 2003. 2.3.3

[44] A. Becker-Gomez, U. Cilingiroglu, and J. Silva-Martinez, "Compact Sub-Hertz OTA-C Filter Design with Interface-Trap Charge Pump," *IEEE J. Solid-State Circuits*, vol. 38, no. 6, pp. 929–934, 2003. 2.3.3

[45] A. J. Lopez-Martin, J. Ramirez-Angulo, R. Gonzalez Carvajal, and L. Acosta, "CMOS Transconductors With Continuous Tuning Using FGMOS Balanced Output Current Scaling," *IEEE J. Solid-State Circuits*, vol. 43, no. 5, pp. 1313–1323, 2008. 2.3.3

[46] K. Smith and A. Sedra, "The current conveyor – A new circuit building block," *Proceedings of the IEEE*, vol. 56, no. 8, pp. 1368–1369, 1968. 2.3.3

[47] A. Sedra and K. Smith, "A second-generation current conveyor and its applications," *IEEE Trans. Circuit Theory*, vol. 17, no. 1, pp. 132–134, 1970. 2.3.3

[48] P. Bruschi, N. Nizza, F. Pieri, M. Schipani, and D. Cardisciani, "A Fully Integrated Single-Ended 1.5–15-Hz Low-Pass Filter With Linear Tuning Law," *IEEE J. Solid-State Circuits*, vol. 42, no. 7, pp. 1522–1528, 2007. 2.3.4

[49] P. M. Furth and A. G. Andreou, "Linearised differential transconductors in subthreshold CMOS," *Electronics Letters*, vol. 31, no. 7, pp. 545–547, 1995. 2.3.4, 4.6.3

[50] L. Acosta, M. Jimenez, R. G. Carvajal, A. J. Lopez-Martin, and J. Ramirez-Angulo, "Highly Linear Tunable CMOS G_m-C Low-Pass Filter," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 10, pp. 2145–2158, 2009. 2.3.4

[51] M. Degrauwe, E. Vittoz, and I. Verbauwhede, "A Micropower CMOS-Instrumentation Amplifier," *IEEE J. Solid-State Circuits*, vol. 20, no. 3, pp. 805–807, 1985. 2.3.4

[52] A. M. Ismail and A. M. Soliman, "Novel CMOS Wide-Linear-Range Transconductance Amplifier," *IEEE Trans. Circuits Syst. I*, vol. 47, no. 8, pp. 1248–1253, 2000. 2.3.4

[53] S. Sengupta, "Adaptively biased linear transconductor," *IEEE Trans. Circuits Syst. I*, vol. 52, no. 11, pp. 2369–2375, 2005. 2.3.4

[54] S. Koziel and S. Szczepanski, "Design of Highly Linear Tunable CMOS OTA for Continuous-Time Filters," *IEEE Trans. Circuits Syst. II*, vol. 49, no. 2, pp. 110–122, 2002. 2.3.4

[55] E. A. Vittoz, "The Design of High-Performance Analog Circuits on Digital CMOS Chips," *IEEE J. Solid-State Circuits*, vol. 20, no. 3, pp. 657–665, 1985. 2.3.5

[56] P. Kinget, "Device mismatch and tradeoffs in the design of analog circuits," *IEEE J. Solid-State Circuits*, vol. 40, no. 6, pp. 1212–1224, 2005. 2.3.5

[57] A. Arnaud and C. Galup-Montoro, "Consistent noise models for analysis and design of CMOS circuits," *IEEE Trans. Circuits Syst. I*, vol. 51, no. 10, pp. 1909–1915, 2004. 2.3.5

[58] D. M. Binkley, *Tradeoffs and Optimization in Analog CMOS Design*. Wiley, 2008. 2.3.5

[59] P. Jespers, *The gm/ID Methodology, a sizing tool for low-voltage analog CMOS Circuits: The semi-empirical and compact model approaches*, ser. Analog Circuits and Signal Processing, 2010, Ed. Springer, 2009. 2.3.5

[60] K. Sakakibara, T. Kumamoto, and K. Arimoto, "Impact of subthreshold hump on bulk-bias dependence of offset voltage variability in weak and moderate inversion regions," in *2012 IEEE Custom Integrated Circuits Conference (CICC)*, 2012, pp. 1–4. 2.3.5

[61] C. Enz and G. Temes, "Circuit techniques for reducing the effects of op-amp imperfections: autozeroing, correlated double sampling, and chopper stabilization," *Proc. IEEE*, vol. 84, no. 11, pp. 1584–1614, 1996. 2.3.5

[62] P.-I. Mak, U. Seng-Pan, and R. Martins, "On the Design of a Programmable-Gain Amplifier With Built-In Compact DC-Offset Cancellers for Very Low-Voltage WLAN Systems," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 2, pp. 496 –509, Mar. 2008. 2.3.5

[63] J. Witte, K. Makinwa, and J. Huijsing, "A CMOS Chopper Offset-Stabilized Opamp," *IEEE J. Solid-State Circuits*, vol. 42, no. 7, pp. 1529–1535, 2007. 2.3.5

[64] D. E. Dick and H. J. Wertz, "Analog and Digital Computation of Fouriér Series and Integrals," *IEEE Trans. Electron. Comput.*, vol. EC-16, no. 1, pp. 8–13, Feb. 1967. 2.4.1

[65] A. Wang and A. Chandrakasan, "Energy-aware architectures for a Real-Valued FFT implementation," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design, 2003. ISLPED '03*, aug. 2003, pp. 360 – 365. 2.4.1, 4.2

[66] M. Lehne and S. Raman, "An Analog/Mixed-Signal FFT Processor for Wideband OFDM Systems," in *Proc. IEEE Sarnoff Symp*, 2006, pp. 1–4. 2.4.1

[67] K. Boyle, P. Mercier, N. Sadeghi, V. Gaudet, C. Schlegel, C. Winstead, and M. Kashyap, "Design and implementation of an all-analog fast-fourier transform processor," in *50th Midwest Symposium on Circuits and Systems, 2007. MWSCAS 2007*, 2007, pp. 1532–1535. 2.4.1

[68] M. Lehne and S. Raman, "A Prototype Analog/Mixed-signal Fast Fouriér Transform Processor IC for OFDM receivers," in *Proc. IEEE Radio and Wireless Symp*, 2008, pp. 803–806. 2.4.1

[69] N. Sadeghi, V. Gaudet, and C. Schlegel, "Analog DFT Processors for OFDM Receivers: Circuit Mismatch and System Performance Analysis," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 9, pp. 2123–2131, 2009. 2.4.1

[70] S. Suh, A. Basu, C. Schlottmann, P. E. Hasler, and J. R. Barry, "Low-Power Discrete Fourier Transform for OFDM: A Programmable Analog Approach," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 2, pp. 290–298, 2011. 2.4.1

[71] E. Afshari, H. S. Bhat, and A. Hajimiri, "Ultrafast Analog Fouriér Transform Using 2-D LC Lattice," *IEEE Trans. Circuits Syst. I*, vol. 55, no. 8, pp. 2332–2343, 2008. 2.4.1

[72] J. L. Walsh, "A Closed Set of Normal Orthogonal Functions," *American Journal of Mathematics*, vol. 45, no. 1, p. 5, Jan 1923. [Online]. Available: http://dx.doi.org/10.2307/2387224 2.4.2

[73] I. Sasaki, K. Matsui, S. Yoneda, and T. Kasai, "Switched-capacitor realization of a Walsh transform circuit and its application to sequency filtering," in *IEEE*

International Symposium on Circuits and Systems, 1988., 1988, pp. 2477–2480 vol.3. 2.4.2

[74] Qualcomm Inc., "An Overview of the Application of Code Division Multiple Access (CDMA) to Digital Cellular Systems and Personal Cellular Networks," Document Number EX60-10010, 1992. 2.4.2

[75] N. M. Blachman, "Sinusoids versus Walsh functions," *Proc. IEEE*, vol. 62, no. 3, pp. 346 – 354, Mar. 1974. 2.4.2

[76] B. Beezley. (2011, Mar.) Harmonic Cancellers for HD Radio Signals. Accessed Dec. 2013. [Online]. Available: http://www.ham-radio.com/k6sti/ibocharm.htm 2.4.2

[77] F. Swift and A. Kamberis, "A new Walsh domain technique of harmonic elimination and voltage control in pulse-width modulated inverters," *IEEE Trans. Power Electron.*, vol. 8, no. 2, pp. 170–185, 1993. 2.4.2

[78] T.-J. Liang, R. O'Connell, and R. Hoft, "Inverter harmonic reduction using Walsh function harmonic elimination method," *IEEE Trans. Power Electron.*, vol. 12, no. 6, pp. 971–982, 1997. 2.4.2

[79] F. Freijedo, J. Doval-Gandoy, O. Lopez, P. Fernandez-Comesana, and C. Martinez-Penalver, "A Signal-Processing Adaptive Algorithm for Selective Current Harmonic Cancellation in Active Power Filters," *IEEE Trans. Ind. Electron.*, vol. 56, no. 8, pp. 2829–2840, 2009. 2.4.2

[80] J. A. Tropp, J. N. Laska, M. F. Duarte, J. K. Romberg, and R. G. Baraniuk, "Beyond Nyquist: Efficient Sampling of Sparse Bandlimited Signals," *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 520–544, 2010. 2.4.3, 3.7.1

[81] A. Harms, W. Bajwa, and R. Calderbank, "Beating Nyquist through correlations: A constrained random demodulator for sampling of sparse bandlimited signals," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),*, May 2011, pp. 5968 –5971. 2.4.3, 3.7.1

[82] J. Laska, S. Kirolos, M. Duarte, T. Ragheb, R. Baraniuk, and Y. Massoud, "Theory and Implementation of an Analog-to-Information Converter using Random Demodulation," in *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, 2007, pp. 1959–1962. 2.4.3

[83] X. Chen, Z. Yu, S. Hoyos, B. Sadler, and J. Silva-Martinez, "A Sub-Nyquist Rate Sampling Receiver Exploiting Compressive Sensing," *IEEE Trans. Circuits Syst. I*, vol. 58, no. 3, pp. 507 –520, march 2011. 2.4.3, 3.7.1

[84] M. Mishali, Y. Eldar, O. Dounaevsky, and E. Shoshan, "Xampling: Analog to digital at sub-Nyquist rates," *IET Circuits, Devices and Systems*, vol. 5, no. 1, pp. 8–20, january 2011. 2.4.3, 3.7.1

[85] Z. Yu, S. Hoyos, and B. Sadler, "Mixed-signal parallel compressed sensing and reception for cognitive radio," in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2008. ICASSP 2008*, 2008, pp. 3861–3864. 2.4.3

[86] Z. Yu and S. Hoyos, "Digitally assisted Analog Compressive Sensing," in *2009 IEEE Dallas Circuits and Systems Workshop,(DCAS)*, 2009, pp. 1–4. 2.4.3

[87] M. Mishali, Y. Eldar, and A. Elron, "Xampling: Signal Acquisition and Processing in Union of Subspaces," *IEEE Trans. Signal Process.*, vol. 59, no. 10, pp. 4719 –4734, oct. 2011. 2.4.3

[88] F. Pianegiani, H. Mingqing, A. Boni, and D. Petri, "Energy-Efficient Signal Classification in Ad Hoc Wireless Sensor Networks," *IEEE Trans. Instrum. Meas.*, vol. 57, no. 1, pp. 190–196, 2008. 3.1, 4.2

[89] J. Altmann, S. Linev, and A. Wei, "Acoustic-seismic detection and classification of military vehicles–developing tools for disarmament and peace-keeping," *Applied Acoustics*, vol. 63, no. 10, pp. 1085–1107, Oct. 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0003682X0200021X 3.2, 3.6.1, 3.6.1

[90] P. E. William and M. W. Hoffman, "Efficient sensor network vehicle classification using peak harmonics of acoustic emissions," in *Proc. SPIE 6963, 69630P*, ser. Unattended Ground, Sea, and Air Sensor Technologies and Applications X, E. M. Carapezza, Ed., vol. 6963, no. 1. SPIE, 2008, p. 69630P. 3.2

[91] P. William and M. Hoffman, "Classification of Military Ground Vehicles Using Time Domain Harmonics' Amplitudes," *IEEE Trans. Instrum. Meas.*, vol. 60, no. 11, pp. 3720–3731, Nov 2011. 3.2, 3.2, 3.6.1, 3.6.1

[92] J. A. Robertson and B. Weber, "Artificial Neural Networks for Acoustic Target Recognition," joint report by ARL and ITT Research, Tech. Rep., 1993. 3.2

[93] M. C. Wellman, N. Srour, and D. B. Hillis, "Acoustic Feature Extraction for a Neural Network Classifier." Army Research Laboratory - Technical Report ARL-TR-1166, Tech. Rep. ARL-TR-1166, Jan 1997. [Online]. Available: http://handle.dtic.mil/100.2/ADA320924 3.2, 3.6.1

[94] H. Sorensen, D. Jones, M. Heideman, and C. Burrus, "Real-valued fast Fourier transform algorithms," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 35, no. 6, pp. 849–863, 1987. 3.5

[95] N. Srour and J. Robertson, "Remote netted acoustic detection system: Final report," Army Research Laboratory - Technical Report ARL-TR-706, Tech. Rep., May 1995. 3.6.1

[96] D. Lake, "Efficient maximum likelihood estimation for multiple and coupled harmonics," Army Research Laboratory - Technical Report ARL-TR-2014, Tech. Rep., 1999. 3.6.1

[97] D. A. Depireux, S. Varma, J. Baras, N. Srour, and T. Pham, "Vehicle classification using acoustic data based on biology hearing model and multi-scale vector quantization," in *ARL Federal Laboratory, 4th Annual Symposium*, Mar. 2000, pp. 83 – 87. 3.6.1

[98] J. Altmann, "Acoustic and seismic signals of heavy military vehicles for co-operative verification," *Journal of Sound and Vibration*, vol. 273, no. 4-5, pp. 713–740, Jun. 2004. 3.6.1

[99] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," in *IEEE International Conference on Neural Networks*, 1993, pp. 586 –591 vol.1. 3.6.1

[100] H. Wu and J. M. Mendel, "Classification of Battlefield Ground Vehicles Using Acoustic Features and Fuzzy Logic Rule-Based Classifiers," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 1, pp. 56 –72, feb. 2007. 3.6.1

[101] S. Nandi, H. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors–a review," *IEEE Trans. Energy Convers.*, vol. 20, no. 4, pp. 719–729, Dec. 2005. 3.6.2

[102] N. Gebraeel, M. Lawley, R. Liu, and V. Parmeshwaran, "Residual life predictions from vibration-based degradation signals: a neural network approach," *IEEE Trans. Ind. Electron.*, vol. 51, no. 3, pp. 694 – 700, Jun. 2004. 3.6.2

[103] Bearing Data Center. Case Western Reserve University. Accessed: August 31, 2013. [Online]. Available: http://csegroups.case.edu/bearingdatacenter 3.6.2

[104] P. William and M. Hoffman, "Identification of bearing faults using time domain zero-crossings," *Mechanical Systems and Signal Processing*, vol. 25, no. 8, pp. 3078–3088, 2011. 3.6.2, 3.6.2

[105] S. Kirolos, J. Laska, M. Wakin, M. Duarte, D. Baron, T. Ragheb, Y. Massoud, and R. Baraniuk, "Analog-to-Information Conversion via Random Demodulation," in *Proc. IEEE Dallas/CAS Workshop Design, Applications, Integration and Software*, 2006, pp. 71–74. 3.7.1

[106] MyHDL. Accessed: August 31, 2013. [Online]. Available: http://myhdl.org 4.1, 4.5.1, A

[107] *MSP430FR573x Mixed Signal Microcontroller Datasheet (Rev. D)*, Texas Instruments, Aug. 2012. [Online]. Available: http://www.ti.com/product/msp430fr5739 4.2, 4.7.1

[108] J. Schoeff, "An inherently monotonic 12 bit DAC," *IEEE J. Solid-State Circuits*, vol. 14, no. 6, pp. 904–911, 1979. 4.6.1

[109] C. Sawigun, D. Pal, and A. Demosthenous, "A wide-input linear range sub-threshold transconductor for sub-Hz filtering," in *Proc. IEEE Int Circuits and Systems (ISCAS) Symp*, 2010, pp. 1567–1570. 4.6.3

[110] N. R. Schemm, "A Single-Chip Ultra-Wideband Based Wireless Sensor Network Node," Ph.D. dissertation, University of Nebraska-Lincoln, 2010. 4.7, 4.7.1

[111] R. Senger, E. Marsman, G. Carichner, S. Kubba, M. McCorquodale, and R. Brown, "Low-latency, HDL-synthesizable dynamic clock frequency controller with self-referenced hybrid clocking," in *2006 IEEE International Symposium on Circuits and Systems. ISCAS 2006. Proceedings.*, May 2006, pp. 4 pp.–778. 4.7.2

[112] A. L. Peressini, F. E. Sullivan, and J. J. U. Jr., *The Mathematics of Nonlinear Programming*, 2nd ed., ser. Undergraduate Texts in Mathematics. Springer, 1993. 5.4.3

[113] pytest. Accessed: August 31, 2013. [Online]. Available: http://pytest.org A.0.4

Appendix A

Digital Design With MyHDL

MyHDL [106] is an open-source package for using the Python programming language as a hardware description and verification language. It utilizes Python generators to model hardware concurrency. Each hardware module is described by a function which returns a group of generators which implement its functionality. In addition to a built-in simulator with waveform tracing, it can perform co-simulation with an external Verilog or VHDL simulator. A subset of the functionality provided in MyHDL can be converted to either Verilog or VHDL files for hardware synthesis. These converted files may then be used as part of a hardware design flow.

Python and associated modules were extensively used for general scripting, data analysis and plotting, and circuit simulation control. Version 0.7 of MyHDL was available at the beginning of the project; but later the developmental code for version 0.8 was used for the design, simulation, and testing of all the local digital blocks except for the MSP430-compatible processor itself.

The code implementing the complete digital hierarchy is included in the Appendices. Here, two blocks are described which together utilize the major features provided by MyHDL.

A.0.3 SPI Slave module

Each module was written in separate files for this design, though it is not necessary to do so.

The SPI slave module is specifically described to illustrate the procedure of developing a digital module with MyHDL. Its conceptual design was shown in Figure 4.5 and described in Section 4.5.1.1.

```
1|from myhdl import *
2|
3|ACTIVE_LOW, ACTIVE_HIGH = 0,1
4|
```

The first statement populates the root namespace with the `myhdl` module contents. Only the `Signal` class, the `always()`, `always_seq()`, and `always_comb()` function decorators, and the `instances()` helper function are actually used in this case. Line 3 defines a symbolic constant to represent symbolic logic states – this is useful for easily changing the logic sense of e.g. reset signals which are typically asserted with a low voltage.

```
5|def SPISlave(reset, scl, cs, din, dout, data):
6|    """Harmonic digital control
7|
8|    Signals:
9|    reset    - async reset to default values
10|   scl      - input clock
11|   cs       - Chip Select: low:shift in data, high:latch to outputs
12|   din      - serial data in
13|   dout     - serial data out
14|
15|   data     - latched data (read-only)
16|   """
17|
```

This is the sole function defined in this file. It is not the SPI module, instead, it returns a group of generators which implement the module. This difference is a primary reason which allows the full power of the Python language to make configurable and parameterized digital module constructors. The function definition signature and docstring together describe the usage and function of the parameters. By personal convention, the first arguments give specific parameters for the

following parameterized module generator. The last set of arguments are the Signal objects which represent, in order, the input and output communication ports of the module.

```
18|     N = len(data)
19|     reg = Signal(intbv(0)[N:])
20|     sample = Signal(intbv(0)[0])
21|
```

The width of the SPI register is implicitly defined by the width of the *data* signal; line 18 extracts this parameter for later use in constructing the module functionality. Lines 19–20 define two internal Signal objects. The `intbv(0)[N:]` creates an `intbv` object initialized to zero then only takes the last N bits, implicitly creating an unsigned, N-bit integer object. Signal `sample` is defined as only bit-0 of the initial zero value, or a 1-bit signal.

```
22|     @always_seq(scl.posedge, reset=reset)
23|     def SampleInput():
24|         sample.next = din
25|
```

This defines the first concurent block named `SampleInput`, the `always_seq()` decorator specifies the generator's signal sensitivity and the reset signal for the inferred registers. The generator is activated on the positive edge of the *scl* signal when the reset signal is not active. It stores the current value of the *din* signal into the 1-bit register `sample`.

```
26|     @always_comb
27|     def wire_dout():
28|         dout.next = reg[N-1]
29|
```

This is a combinational logic block which wires the most-significant bit of the shift register `reg` to the output signal *dout*.

```
30|     @always_seq(scl.negedge, reset=reset)
31|     def InputRegister():
32|         if cs == ACTIVE_LOW:
33|             reg.next = concat(reg[N-1:], sample)
34|
```

On the opposite (negative) edge of the input *scl* signal, the `InputRegister` block left-shifts the sampled data bit into the internal register `reg`. The input bit is placed into the least-significant bit position.

```
35|     @always(cs.posedge)
36|     def ChipSelect():
37|         if cs == ACTIVE_HIGH:
38|             data.next = reg
39|
```

When the *cs* signal rises, the internal data register contents are latched into the output data register `data`.

```
40|     return instances()
```

The `instances()` function automatically introspects the containing function definition and returns the defined generators. In this case, it is a convenient equivalent for the return statement

```
40|     return (SampleInput, wire_dout, InputRegister, ChipSelect)
```

A.0.4 Test bench for `SPISlave` module

MyHDL may be used as a front-end to help generate parameterized synthesizable Verilog or VHDL code. It's feature set beyond the convertible subset is what allows constructing digital systems with MyHDL to shine.

Verification of hardware modules is an essential part of a digital design flow to ensure correct behavior. Test benches are constructed which instantiate a module under test, exercise its inputs, and check for the expected outputs. The test benches are then run in a digital simulator.

Writing these test benches directly in Verilog or VHDL can be difficult due to the lack of high-level language features and general programming constructs. With MyHDL, the entire expressivity of the Python programming language may be utilized to construct simulations and describe complex behavior. Additional Python

packages can be used to e.g. construct plots, interact with a network, or perform other analyses.

The verification for the `SPISlave` module is described here to demonstrate the features which allow complete testing of parameterized modules. The `pytest` [113] software was used to assist in collecting and running the test elements. It inspects the input file for class definitions beginning with the name `Test` instantiates each class and calls the class methods beginning with the name `test_`, reporting the number of passing and failing tests. It also looks for for top-level function definitions named `test_` for additional tests to run.

```
1 import random
2
3 from myhdl import *
4
5 from SPISlave import SPISlave
6
7
8 PERIOD = 10
9
```

Python coding convention places most package and module imports at the top of a file. Built-in packages are loaded first, followed by additional packages, then local modules. The constant `PERIOD` defines the clock period in simulation steps.

```
10 # SPI bus transaction helper
11 def start(spi):
12     spi.cs.next = 0
13     yield spi.clk.posedge
14
15 def sendBit(spi, b):
16     spi.scl.next = 0
17     spi.din.next = b
18     yield spi.clk.posedge
19     spi.scl.next = 1
20     yield spi.clk.posedge
21
22 def stop(spi):
23     spi.scl.next = 0
24     yield spi.clk.posedge
25     spi.cs.next = 1
26     yield spi.clk.posedge
27
28 def tx(spi, word):
29     yield start(spi)
30     for i in downrange(len(word)):
31         yield sendBit(spi, word[i])
32     yield stop(spi)
33
34
```

These define several Python generators which define the functional behavior of an SPI bus transaction. They accept an *spi* object which contains the SPI bus signals *cs*, *scl*, and *din*. Because these generators directly, they are not convertible to Verilog or VHDL and are used for MyHDL simulation only. They do, however, model the high-level functional behavior of a SPI bus master.

Each bus transaction begins with the *cs* line going low, modelled by calling the `start(spi)` function which sets the *cs* line low and waits to return until global simulation clock *clk* line rises. Ending an SPI bus transaction is modelled by the `stop(spi)` function which sets the *scl* line low (to the SPI mode 0 clock idle state), waits for a simulation clock time step, then un-asserts the Chip Select line *cs*.

For SPI mode 0 transactions, as discussed in Section 4.5.1.1, a bit is sent to a slave module by changing the input bit *din* while the bus clock line *scl* is low and sampling its value on the *scl* rising edge. This is modelled by generator `sendBit(spi, b)` whose second parameter is the current bit value to send. Sending a bit therefore requires waiting for two rising edges of the simulation clock *clk* to complete, making the SPI clock rate (*scl*) one-half the *clk* rate.

Finally, the `tx(spi, word)` function models a complete bus transaction given a container holding the given SPI bus lines and the word to send. The transaction is functionally described by first performing a bus start operation, sequentially sending the word bits starting with the most-significant, then ending the bus transaction. MyHDL convenience function `downrange(x)` returns a descending list of integers like $\{x-1, x-2, \cdots, 0\}$.

```
35| class TestShiftRegister:
```

This `TestShiftRegister` class wraps several test routines which use similar functionality.

```
36|     def makeN_tester(self, N):
37|         reset = ResetSignal(0, active=0, async=True)
```

```
38|         clk = Signal(bool(0))
39|         scl, cs = [Signal(bool(0)) for i in range(2)]
40|         din = Signal(intbv(0)[0])
41|         dout = Signal(intbv(0)[0])
42|
43|         class SPI:
44|             pass
45|         spi = SPI()
46|         spi.clk = clk
47|         spi.scl = scl
48|         spi.cs = cs
49|         spi.din = din
50|         spi.dout = dout
51|
52|         # dependent signals
53|         data = Signal(intbv(0)[N:])
54|         indata = Signal(intbv(0)[N:])
55|
56|         spislave = SPISlave(
57|                 reset, scl, cs, din,
58|                 dout, data)
59|
60|
61|         # system clock generator
62|         @always(delay(PERIOD//2))
63|         def clkgen():
64|             clk.next = not clk
65|
66|         # feed some random input words
67|         @instance
68|         def tester():
69|             spi.cs.next = 1
70|             reset.next = 1
71|             for iteration in range(10):
72|                 collector = intbv(0)[N:]
73|                 reset.next = 0
74|                 indata.next = intbv(random.randrange(2**32))[N:]
75|                 yield clk.posedge
76|
77|                 reset.next = 1
78|                 yield clk.posedge
79|
80|                 yield tx(spi, indata)
81|                 assert data == indata
82|
83|                 # shift out data
84|                 yield start(spi)
85|                 for i in downrange(N):
86|                     yield sendBit(spi, 0)
87|                     collector[i] = dout
88|                 yield stop(spi)
89|
90|                 assert collector == indata
91|
92|             raise StopSimulation
93|
94|         return instances()
95|
```

The `makeN_tester(self, N)` method constructs a simulation clock generator and a single `SPISlave` instance which has a N-bit wide data register. Lines 37–41 create the serial input and output bit lines for communicating with the module. Lines 43–

50 defines an empty class `SPI`, instantiates it, then assigns named attributes. This provides a convenient logical container which holds the signals related to a single SPI bus communication lines.

The output register `data` is created along with a second signal for holding an input word. Their bit widths are set to the given desired width of the module instance to be tested. Lines 56–58 create an instance of the `SPISlave` module with the appropriate input and output signals whose register size is implicitly determined by `data`'s width. Lines 61–65 define a concurrent module which simply toggles the simulation system clock every half-period time steps.

Finally, the simulation block `tester` is created using the `instance` function decorator. This identifies the following generator to be elaborated and a part of the simulation. Such generators are not convertible to Verilog or VHDL and are for describing complex behavior within a simulation.

Lines 69–70 ensure the chip-select and reset lines are not asserted. The `collector` variable is initialized to zeros and will collect the output bits of the SPI slave as they are clocked out. The reset signal is asserted and a random N-bit vector is generated to transmit; testing operations resume on the rising edge of the next simulation *clk*. The reset signal is un-asserted and allowed to propagate through the system by waiting for another *clk* rising edge, simulating a system reset.

Line 80 simulates sending the complete vector `indata`'s contents across the SPI bus by calling the helper function `tx(spi, indata)`. When the transmission is finished, `tester` resumes execution and checks that the latched output of `SPISlave` matches the random data sent. Failure of this condition raises an exception, effectively causing the test to fail.

Because the SPI slave is implemented as a shift register, a second way to test proper operation is to shift in additional bits and check that the output bits match

the previously input bits in the proper order. Lines 83–90 perform this operation by starting a second transaction, sending N zero bits while storing each output bit in the expected bit position in `collector`, then stopping the transaction. Again, the shifted-out data should match the data sent originally.

Ten test iterations are performed with random data. A complete test of this functional aspect would require 2^N iterations, sending in all possible data values. The behavior of the module may be sufficiently evaluated by only a few non-trivial (e.g. all-zero or all-one) inputs. When the iterations finish, the special `StopSimulation` exception is raised which indicates successful termination of the simulation. The method `makeN_tester(self, N)` returns, in line 94, the simulation generators which implement the above-described test for a single register width N.

```
 96|     def test_dataOut(self):
 97|         for N in [2,4,8,16,32,48]:
 98|             tb = self.makeN_tester(N)
 99|             sim = Simulation(tb)
100|             sim.run()
101|
```

The `test_dataOut()` method is called by the `py.test` framework with no arguments. It creates and runs several simulations of differing bit-width `SPISlave` modules for proper functionality. The `sim.run()` statement exits when a `StopSimulation` exception is raised, exiting the containing function as well. By convention of the test framework, `test_*` functions pass when they do not propagate an un-handled exception.

```
102|     def bench_seriesDevices(self):
103|         reset = ResetSignal(0, active=0, async=True)
104|         clk = Signal(bool(0))
105|         scl, cs = [Signal(bool(0)) for i in range(2)]
106|         din = Signal(intbv(0)[0])
107|         d0 = Signal(intbv(0)[0])
108|         d1 = Signal(intbv(0)[0])
109|         d2 = Signal(intbv(0)[0])
110|         dout = Signal(intbv(0)[0])
111|
112|         class SPI:
113|             pass
114|         spi = SPI()
115|         spi.clk = clk
```

```
116|         spi.scl = scl
117|         spi.cs = cs
118|         spi.din = din
119|         spi.dout = dout
120|
121|         N = 16
122|         N_SERIES = 4
123|         # intbv is 64bit max...
124|         assert (N*N_SERIES) <= 64
125|
126|         # dependent signals
127|         data0 = Signal(intbv(0)[N:])
128|         data1 = Signal(intbv(0)[N:])
129|         data2 = Signal(intbv(0)[N:])
130|         data3 = Signal(intbv(0)[N:])
131|
132|         spi0 = SPISlave(reset, scl, cs, din, d0, data0)
133|         spi1 = SPISlave(reset, scl, cs, d0, d1, data1)
134|         spi2 = SPISlave(reset, scl, cs, d1, d2, data2)
135|         spi3 = SPISlave(reset, scl, cs, d2, dout, data3)
136|         indata = Signal(intbv(0)[N_SERIES*N:])
137|
138|
139|         # system clock generator
140|         @always(delay(PERIOD//2))
141|         def clkgen():
142|             clk.next = not clk
143|
144|         # feed some random input words
145|         @instance
146|         def tester():
147|             collector = intbv(0)[N_SERIES*N:]
148|             assert len(indata) == len(collector)
149|             for iteration in range(10):
150|                 indata.next = random.randrange(2**(N_SERIES*N))
151|                 reset.next = 1
152|                 yield tx(spi, indata)
153|
154|                 # shift out data
155|                 # sample dout when SCL == 1
156|                 yield start(spi)
157|                 for i in downrange(N_SERIES*N):
158|                     yield sendBit(spi, 0)
159|                     collector[i] = dout
160|                 yield stop(spi)
161|                 assert collector == indata
162|             raise StopSimulation
163|
164|         return instances()
165|
```

Method `bench_seriesDevices` tests proper shifting when four `SPISlave` modules are chained in series. In this implementation, the data to end up in all four modules is created as a single bit-vector. Line 124 ensures that this large vector will be a maximum of 64 bits wide, which is the width limitation of the `intbv` object type. Lines 126–135 create output data registers and instantiate four modules with serial connection of their *din* and *dout* ports. The shift behavior is evaluated in the same

way as the previous test.

```
166|     def test_seriesDevices(self):
167|         tb = self.bench_seriesDevices()
168|         sim = Simulation(tb)
169|         sim.run()
170|
171|     def vcd_test_timing(self):
172|         def bench_SPISlave():
173|             return self.makeN_tester(8)
174|
175|         tb = traceSignals(bench_SPISlave)
176|         sim = Simulation(tb)
177|         sim.run()
178|
179|
180| if __name__ == '__main__':
181|     TestShiftRegister().vcd_test_timing()
```

Appendix B

MyHDL Code Listings

B.1 src/HarmonicInterface.py

Top-level assembly of the harmonic projection channel pair digital functions.

```
#!/usr/bin/env python

from myhdl import *

from SPISlave import SPISlave
from NCO import NCO
from SwitchCtl import SwitchCtl

def HarmonicInterface(clk_in, reset_in, scl_in, cs_in, din,
        nco_i, nco_q, multA, multB,
        clk_out, reset_out, scl_out, cs_out, dout,
        swAp, swAn,
        cintAn, zeroAn, fastAn, tuneAn,
        cintAp, zeroAp, fastAp, tuneAp,
        swBp, swBn,
        cintBn, zeroBn, fastBn, tuneBn,
        cintBp, zeroBp, fastBp, tuneBp):
    """Harmonic digital interface

    clk     - input clock
    reset   - async reset to default values
    scl     - SPI clock
    cs      - SPI chip select
    din     - SPI MOSI
    dout    - SPI MISO

    swXx     - multiplier switches
    cintXx   - Cap on
    zeroXx   - Reset cap to Vcm
    fastXx   - gm x10
    tuneXx   - 12bit IDAC word
    """

    N = 16
```

```
35|     N_DATA_BITS = 16 + 2*N
36|
37|     # SPI
38|     cdata = Signal(intbv(0)[N_DATA_BITS:])
39|     spiSlave = SPISlave(reset_in, scl_in, cs_in, din, dout, cdata)
40|
41|     # pull out slices of SPI words
42|     cal = cdata(47)
43|     rst = cdata(46)
44|     seA = cdata(29)
45|     seB = cdata(13)
46|     fcw = cdata(46,32)
47|
48|     # NCO
49|     nco = NCO(N, clk_in, reset_in, rst, fcw, nco_i, nco_q)
50|
51|     # Channels
52|     sA = Signal(intbv(0)[7:])
53|     sB = Signal(intbv(0)[7:])
54|     channelA = SwitchCtl(multA, cal, seA, sA)
55|     channelB = SwitchCtl(multB, cal, seB, sB)
56|
57|     @always(clk_in.posedge)
58|     def switchOut():
59|         swAn.next = sA
60|         swAp.next = ~sA
61|         swBn.next = sB
62|         swBp.next = ~sB
63|
64|     @always(cdata)
65|     def passthru():
66|         cintAn.next = cdata[31]
67|         cintAp.next = not cdata[31]
68|
69|         zeroAn.next = cdata[30]
70|         zeroAp.next = not cdata[30]
71|
72|         fastAn.next = cdata[28]
73|         fastAp.next = not cdata[28]
74|
75|         tuneAn.next = cdata[28:16]
76|         tuneAp.next = ~intbv(cdata[28:16], max=2**12)
77|
78|         cintBn.next = cdata[15]
79|         cintBp.next = not cdata[15]
80|
81|         zeroBn.next = cdata[14]
82|         zeroBp.next = not cdata[14]
83|
84|         fastBn.next = cdata[12]
85|         fastBp.next = not cdata[12]
86|
87|         tuneBn.next = cdata[12:0]
88|         tuneBp.next = ~intbv(cdata[12:0], max=2**12)
89|
90|     @always_comb
91|     def thrulines():
92|         clk_out.next = clk_in
93|         reset_out.next = reset_in
94|         scl_out.next = scl_in
95|         cs_out.next = cs_in
96|
97|     return instances()
98|
99|
100| def convert():
```

```
    clk_in, reset_in, scl_in, cs_in = [Signal(intbv(0)[0]) for i in range(4)]
    clk_out, reset_out, scl_out, cs_out = [Signal(intbv(0)[0]) for i in range(4)]
    din, dout = [Signal(intbv(0)[0]) for i in range(2)]
    nco_i, nco_q = [Signal(intbv(0)[0]) for i in range(2)]
    multA, multB = [Signal(intbv(0)[0]) for i in range(2)]

    swAn = Signal(intbv(0)[7:])
    swAp = Signal(intbv(0)[7:])
    swBn = Signal(intbv(0)[7:])
    swBp = Signal(intbv(0)[7:])

    cintAn, zeroAn, fastAn = [Signal(intbv(0)[0]) for i in range(3)]
    cintAp, zeroAp, fastAp = [Signal(intbv(0)[0]) for i in range(3)]

    cintBn, zeroBn, fastBn = [Signal(intbv(0)[0]) for i in range(3)]
    cintBp, zeroBp, fastBp = [Signal(intbv(0)[0]) for i in range(3)]

    tuneAn = Signal(intbv(0)[12:])
    tuneAp = Signal(intbv(0)[12:])

    tuneBn = Signal(intbv(0)[12:])
    tuneBp = Signal(intbv(0)[12:])

    toVerilog(
        HarmonicInterface,
        clk_in, reset_in, scl_in, cs_in, din,
        nco_i, nco_q, multA, multB,
        clk_out, reset_out, scl_out, cs_out, dout,
        swAp, swAn,
        cintAn, zeroAn, fastAn, tuneAn,
        cintAp, zeroAp, fastAp, tuneAp,
        swBp, swBn,
        cintBn, zeroBn, fastBn, tuneBn,
        cintBp, zeroBp, fastBp, tuneBp)

if __name__ == '__main__':
    convert()
```

B.1.1 src/SPISlave.py

Serial Peripheral Interface slave module.

```
from myhdl import *

ACTIVE_LOW, ACTIVE_HIGH = 0,1

def SPISlave(reset, scl, cs, din, dout, data):
    """Harmonic digital control

    Signals:
    reset    - async reset to default values
    scl      - input clock
    cs       - Chip Select: low:shift in data, high:latch to outputs
    din      - serial data in
    dout     - serial data out

    data     - latched data (read-only)
    """

    N = len(data)
    reg = Signal(intbv(0)[N:])
    sample = Signal(intbv(0)[0])
```

```
21|
22|     @always_seq(scl.posedge, reset=reset)
23|     def SampleInput():
24|         sample.next = din
25|
26|     @always_comb
27|     def wire_dout():
28|         dout.next = reg[N-1]
29|
30|     @always_seq(scl.negedge, reset=reset)
31|     def InputRegister():
32|         if cs == ACTIVE_LOW:
33|             reg.next = concat(reg[N-1:], sample)
34|
35|     @always(cs.posedge)
36|     def ChipSelect():
37|         if cs == ACTIVE_HIGH:
38|             data.next = reg
39|
40|     return instances()
41|
42| # temp typesetting column-width ruler
43| #        1         2         3         4         5         6         7         8
44| # 345678901234567890123456789012345678901234567890123456789012345678901234567890
```

B.1.2 src/NCO.py

Numerically-controlled oscillator with quadrature outputs.

```
1| #!/usr/bin/env python
2|
3| from myhdl import *
4|
5| ACTIVE_LOW, ACTIVE_HIGH = 0,1
6|
7| def NCO(N, clk, reset, rst, fcw, outi, outq):
8|     """Numerically controlled oscillator.
9|
10|     N       - phase accumulator bit-width CONFIG
11|     clk     - clock input
12|     reset   - global actLow asynchronous reset-to-zero
13|     rst     - local reset-to-zero
14|     fcw     - phase increment tuning word
15|     outi    - in-phase output
16|     outq    - quadrature output
17|     """
18|     assert(N >= 2)
19|
20|     acc_max = 2**N
21|     offset = 2**(N-2)
22|
23|     MSB = N-1
24|
25|     phase = Signal(intbv(0)[N:])
26|     phase_delay = Signal(intbv(0)[N:])
27|     x = Signal(intbv(0)[N:])
28|
29|     @always_seq(clk.negedge, reset=reset)
30|     def loopdelay():
31|         phase_delay.next = phase
32|
33|     @always_seq(clk.posedge, reset=reset)
```

```
34|     def ncoLogic():
35|         if rst == ACTIVE_LOW:
36|             phase.next = 0
37|             outi.next = 0
38|             outq.next = 0
39|         else:
40|             phase.next = (phase_delay + fcw) % acc_max
41|             tmp = (phase_delay + offset) % acc_max
42|             outi.next = intbv(tmp)[MSB]
43|             outq.next = phase[MSB]
44|
45|     return instances()
```

B.1.3 src/SwitchCtl.py

Transmission gate switch control logic.

```
1|from myhdl import *
2|
3|
4|def SwitchCtl(mult, cal, se, sw):
5|     """TX gate multiplier control
6|
7|     mult    - 0=+1, 1=-1
8|     cal     - 0:normal, 1:short inputs to CM
9|     se      = 0:diff, 1:feedback
10|
11|     Switch control outputs, 1 == on
12|     a       - ina-siga switch
13|     b       - inb-siga switch
14|     c       - ina-sigb switch
15|     d       - inb-sigb switch
16|     e       - cm-siga
17|     f       - cm-sigb
18|     g       - out-sigb
19|     """
20|
21|     @always_comb
22|     def logic():
23|         if cal == 0:
24|             # normal +-1 diff mult
25|             if se == 0:
26|                 if mult:
27|                     sw.next = 0b0110000
28|                 else:
29|                     sw.next = 0b1001000
30|
31|             # SE mult of A only, B-out
32|             else:
33|                 if mult:
34|                     sw.next = 0b0100001
35|                 else:
36|                     sw.next = 0b1000001
37|
38|         else:
39|             # Calibrate to CM open-loop
40|             if se == 0:
41|                 sw.next = 0b0000110
42|
43|             # Calibrate to CM closed-loop
44|             else:
45|                 sw.next = 0b0000101
46|
47|     return instances()
```

B.2 src/Chain0Ctl.py

Main 48-harmonic block output multiplexer and pad driver control.

```
#!/usr/bin/env python

from myhdl import *

from SPISlave import SPISlave
from AnalogMuxCtl import AnalogMuxCtl
from BufferCtl import BufferCtl

N = 16
N_DATA_BITS = 16 + 2*16
N_MUX_INPUTS = 49 #48 harmonics + CMI

def Chain0Ctl(
        reset, scl, cs, din,
        dout,
        txAn, txAp,
        txBn, txBp,
        swAn, swAp,
        swBn, swBp,
        fastAn, fastAp,
        fastBn, fastBp,
        tuneAn, tuneAp,
        tuneBn, tuneBp):
    """Harmonic chain 0 multiplexer and pad buffer control

    Inputs:
    reset   - reset SPI register
    scl     - SPI SCLK
    cs      - SPI CS
    din     - SPI MOSI

    Outputs:
    dout    - SPI MISO
    txXn    - one-hot mux A/B NMOS control
    txXn    - one-cold mux A/B PMOS control
    swXn    - buffer A/B mode NMOS switches
    swXp    - buffer A/B mode PMOS switches
    fastXn  - buffer A/B gm config
    fastXp  - buffer A/B gm config
    tuneXn  - IDAC A/B switches
    tuneXp  - IDAC A/B complementary switches
    """

    # SPI
    cdata = Signal(intbv(0)[N_DATA_BITS:])
    spiSlave = SPISlave(reset, scl, cs, din, dout, cdata)

    # pull out slices of SPI words
    unusedA  = cdata(48,46)
    muxSelA  = cdata(46,40)

    unusedB  = cdata(40,38)
    muxSelB  = cdata(38,32)

    bufModeA = cdata(32,29)
    fastA    = cdata(28)
    tuneA    = cdata(28,16)

    bufModeB = cdata(16,13)
    fastB    = cdata(12)
```

```
61      tuneB    = cdata(12,0)
62
63
64      # Analog Mux
65      tAn, tAp, tBn, tBp = [Signal(intbv(0)[N_MUX_INPUTS:]) for i in range(4)]
66      muxA = AnalogMuxCtl(N_MUX_INPUTS, 0, muxSelA, tAn, tAp)
67      muxB = AnalogMuxCtl(N_MUX_INPUTS, 0, muxSelB, tBn, tBp)
68
69      # Buffer switch control
70      sA, sB = [Signal(intbv(0)[4:]) for i in range(2)]
71      bufSwCtlA = BufferCtl(bufModeA, sA)
72      bufSwCtlB = BufferCtl(bufModeB, sB)
73
74
75      @always_comb
76      def muxbits():
77          txAn.next = tAn
78          txAp.next = tAp
79          txBn.next = tBn
80          txBp.next = tBp
81
82      @always_comb
83      def passthru():
84          swAn.next = sA
85          swAp.next = ~sA
86
87          swBn.next = sB
88          swBp.next = ~sB
89
90          fastAn.next = fastA
91          fastAp.next = not fastA
92
93          fastBn.next = fastB
94          fastBp.next = not fastB
95
96          tuneAn.next = tuneA
97          tuneAp.next = ~tuneA
98
99          tuneBn.next = tuneB
100         tuneBp.next = ~tuneB
101
102     return instances()
103
104
105 def convert():
106     reset, scl, cs, din, dout = [Signal(intbv(0)[0]) for i in range(5)]
107     txAn, txAp, txBn, txBp = [Signal(intbv(0)[N_MUX_INPUTS:]) for i in range(4)]
108     swAn, swAp, swBn, swBp = [Signal(intbv(0)[4:]) for i in range(4)]
109     fastAn, fastAp, fastBn, fastBp = [Signal(intbv(0)[0]) for i in range(4)]
110     tuneAn, tuneAp, tuneBn, tuneBp = [Signal(intbv(0)[12:]) for i in range(4)]
111
112     toVerilog(
113         ChainOCtl,
114         reset, scl, cs, din,
115         dout,
116         txAn, txAp,
117         txBn, txBp,
118         swAn, swAp,
119         swBn, swBp,
120         fastAn, fastAp,
121         fastBn, fastBp,
122         tuneAn, tuneAp,
123         tuneBn, tuneBp)
124
125 if __name__ == '__main__':
126     convert()
```

B.2.1 src/SPISlave.py

Serial Peripheral Interface slave module, an instance of the module in Appendix B.1.1.

B.2.2 src/AnalogMuxCtl.py

Transmission gate multiplexer control decoder. Invalid selector codes use an explicitly-defined default selection.

```
#!/usr/bin/env python

from math import ceil, log

from myhdl import *


def AnalogMuxCtl(N, default, sel, swN, swP):
    """TX gate multiplexer control

    Construction:
    N       - N switches (constructor)
    default - output if sel>=N

    Inputs:
    sel     - ceil(log2(N))-bit selector

    Outputs:
    swN     - N-long one-hot output vector for NMOS switches
    swP     - N-long one-cold output vector for PMOS switches
    """

    Nbits = int(ceil(log(N, 2)))
    SELECTOR = [2**i for i in range(N)]

    # explicitly fill in unused cases with default value
    for i in range(N, 2**Nbits):
        SELECTOR.append(int(default))

    SELECTOR = tuple(SELECTOR)

    x = Signal(intbv(0)[N:])

    @always_comb
    def logic():
        x.next = SELECTOR[sel]

    @always(x)
    def outputs():
        swN.next = x
        swP.next = ~x

    return instances()


def convert():
    N = 48
    Nbits = ceil(log(N, 2))
    default = intbv(0)[Nbits:]

    sel = Signal(intbv(0)[Nbits:])
```

```
52|     swN = Signal(intbv(0)[N:])
53|     swP = Signal(intbv(0)[N:])
54|
55|     toVerilog(AnalogMuxCtl, N, default, sel, swN, swP)
56|
57| if __name__ == '__main__':
58|     convert()
```

B.2.3 src/BufferCtl.py

Pad driver OTA control. The analog circuit is the same OTA used in the harmonic channel but with a different input switching configuration.

```
1|  #!/usr/bin/env python
2|
3|  from myhdl import *
4|
5|
6|  ACTIVE_HIGH, INACTIVE_LOW = 1,0
7|  ACTIVE_LOW, INACTIVE_HIGH = 0,1
8|
9|  def BufferCtl(mode, sw):
10|     """TX gate multiplier control
11|
12|     cint     - unused
13|     zero     - +input to CMI
14|     se       = 0:openloop, 1:follower
15|
16|     mode     - [cint, zero, se] bit vector (as integer)
17|
18|     Switch control outputs, 1 == on
19|     a        - mux-inA
20|     b        - CMI-inA
21|     c        - CMI-inB
22|     d        - inB-out
23|     """
24|
25|     # mode[2] is unused, don't care
26|     # only mode[1:0] used
27|     MODE_SWITCHES = (
28|             0b1010, #mux cmp
29|             0b1010, #mux cmp
30|
31|             0b1001, #mux buff
32|             0b1001, #mux buff
33|
34|             0b0110, #tune fast
35|             0b0110, #tune fast
36|
37|             0b0101, #tune slow
38|             0b0101, #tune slow
39|             )
40|
41|     @always_comb
42|     def logic():
43|         sw.next = MODE_SWITCHES[int(mode)]
44|
45|     return instances()
46|
47|
48|  def convert():
49|     mode = Signal(intbv(0)[3:])
```

```
50|     sw = Signal(intbv(0)[4:])
51|
52|     toVerilog(BufferCtl, mode, sw)
53|
54|
55| if __name__ == '__main__':
56|     convert()
```

B.3 src/CpuClkSel.py

Additional module added to implement clock source switching in the NS430. The converted Verilog was included into the processor module synthesis within Cadence's Encounter tool.

```
1|  #!/usr/bin/env python
2|
3|  from myhdl import *
4|
5|  from ClockMux import ClockMux
6|
7|  # FIXME: give a Fail-Safe state, or not-allowed 'sel' combinations
8|  # to avoid shooting one's self in the foot.
9|  #
10| # Here, sel=10b disables the HFxtal and also selects it (<-- BAD)
11| # DO NOT rely on proper coding to ensure this state is never reached,
12| # do this in hardware!!
13| # (the ns430 bootloader code does exactly this, compiled to ROM code
14| #  before Dan completely audited the boot code)
15| #
16| def CpuClkSel(reset, sel, hfxtal, lfxtal, hf_en, cpu_clk):
17|     """NS430 system clock select, HF enable
18|
19|     reset   - Nrst system reset
20|     sel     - SysClkSel<1:0> from NS430
21|     hfxtal  - HF crystal output
22|     lfxtal  - 32k crystal output
23|
24|     hf_en   - Enable HF crystal
25|     cpu_clk - Main clock for NS430
26|
27|     sel[1] is ~hf_en or "Disable HFXTAL"
28|     sel[0] selects [hf, lf] xtal inputs
29|     """
30|
31|     clk_sel = Signal(intbv(0)[0])
32|     in_clocks = Signal(intbv(0)[2:])
33|
34|     @always_comb
35|     def cheat():
36|         in_clocks.next[0] = hfxtal
37|         in_clocks.next[1] = lfxtal
38|
39|     clkMux = ClockMux(2, reset, clk_sel, in_clocks, cpu_clk)
40|
41|     @always_comb
42|     def hf_en_logic():
43|         hf_en.next = ~sel[1]
44|
```

```
45|     @always_comb
46|     def clk_sel_logic():
47|         if sel == 0:
48|             clk_sel.next = 0
49|         else:
50|             clk_sel.next = 1
51|
52|     return instances()
53|
54|
55| def convert():
56|     reset = Signal(intbv(0)[0])
57|     sel = Signal(intbv(0)[2:])
58|     hfxtal = Signal(intbv(0)[0])
59|     lfxtal = Signal(intbv(0)[0])
60|     hf_en = Signal(intbv(0)[0])
61|     cpu_clk = Signal(intbv(0)[0])
62|
63|     toVerilog(CpuClkSel, reset, sel, hfxtal, lfxtal, hf_en, cpu_clk)
64|
65|
66| if __name__ == '__main__':
67|     convert()
```

B.3.1 src/ClockMux.py

Generalized N-input glitch-free clock multiplexer. The clock selector's control lines are mutually synchronized to ensure no glitches appear at the clock output. This module makes use of the elaboration phase of MyHDL module construction. Function `Synchronizer()` returns a group of generators which implement a single clock-enable synchronizer. Likewise, function `EnableLogic()` returns a combinational block which detects the condition when only the select bit corresponding to its own index is asserted and none others. This condition only occurs when all the enable signals have propagated according to their respective clocks.

```
1| #!/usr/bin/env python
2|
3| """
4| Idea from: Techniques to make clock switching glitch free
5|            by Rafey Mahmud
6|
7| http://www.eetimes.com/electronics-news/4138692/Techniques-to-make-clock-switching-glitch-fre
8|
9| Adapted to arbitrary number of inputs.
10| """
11|
12| import sys
13| from math import ceil, log
14|
15| from myhdl import *
16|
```

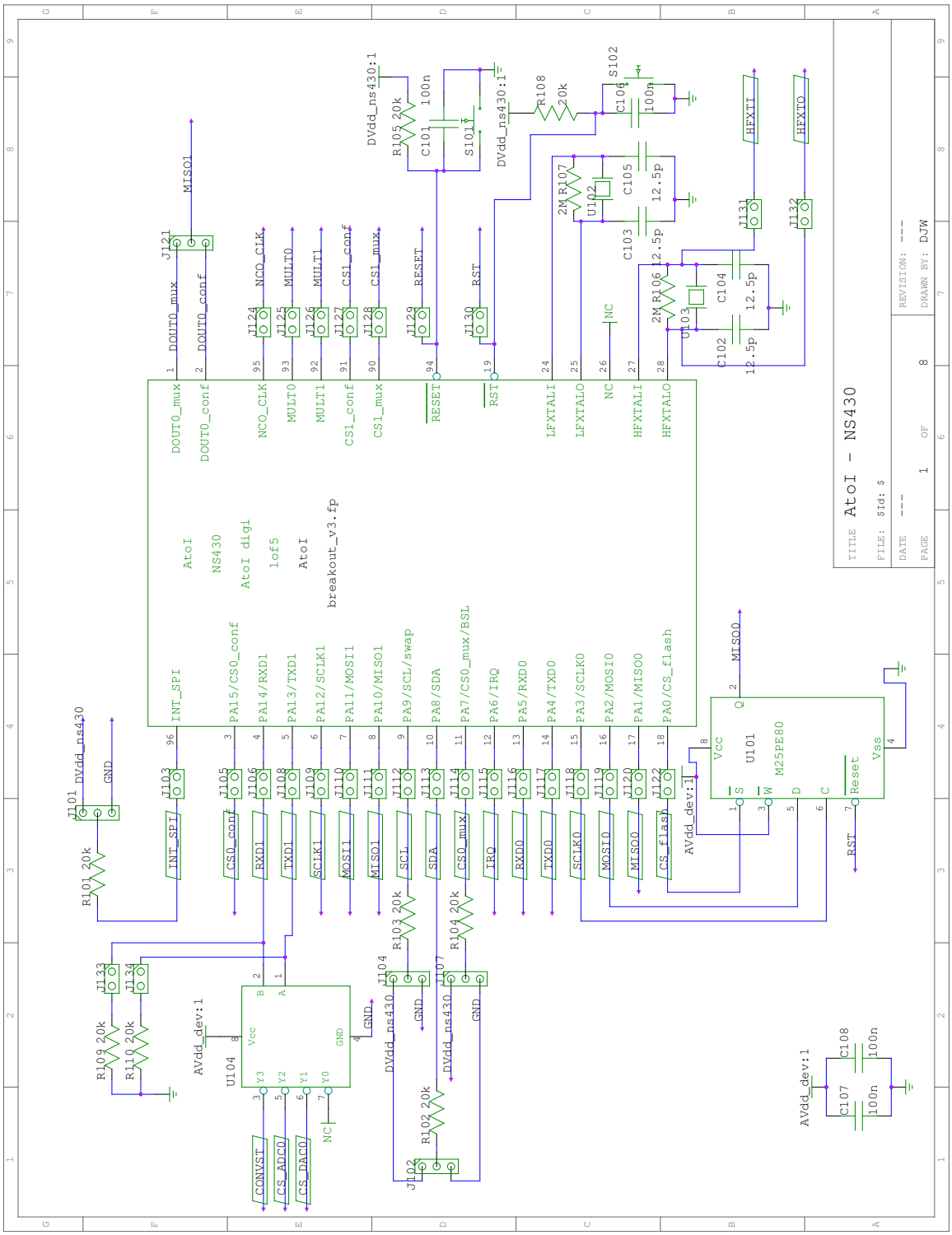
```
17|
18|ACTIVE_LOW, ACTIVE_HIGH = 0,1
19|
20|
21|def ClockMux(N, reset, sel, in_clocks, out_clock):
22|    """Glitch-free clock multiplexer
23|
24|    N         - number of clocks
25|
26|    reset     - async actLow reset
27|    sel       - clock select index
28|    in_clocks - vector of input clock lines
29|
30|    Outputs:
31|    out_clock    - selected output clock
32|    """
33|
34|    assert int(ceil(log(N, 2))) == len(sel)
35|
36|    # internal signals
37|    in_enables = [Signal(intbv(0)[0]) for i in range(N)]
38|    out_enables = [Signal(intbv(0)[0]) for i in range(N)]
39|    sync_clocks = [Signal(intbv(0)[0]) for i in range(N)]
40|    sync_clocks_concat = ConcatSignal(*reversed(sync_clocks))
41|
42|
43|    def Synchronizer(reset, clk_in, en_in, en_out, clk_out):
44|        """Single synchronizer stage.  Only enables clock iff en is asserted,
45|        which is iff it is the only one asserted.
46|
47|        clk_in  - input clock
48|        en      - enable this clock
49|
50|        not_en  - delayed, inverted enable
51|        """
52|        d0, d1 = [Signal(intbv(0)[0]) for i in range(2)]
53|
54|        @always_seq(clk_in.posedge, reset=reset)
55|        def stage0():
56|            d0.next = en_in
57|
58|        @always_seq(clk_in.negedge, reset=reset)
59|        def stage1():
60|            en_out.next = d0
61|
62|        @always_comb
63|        def clkOut():
64|            clk_out.next = (en_out and clk_in)
65|
66|        return instances()
67|
68|
69|    def EnableLogic(index, sel, en):
70|        """Exclusive enable
71|        output = en and (not any(others))
72|        """
73|        them = [out_enables[j] for j in range(N) if j != i]
74|        if len(them) == 1:
75|            others = them[0]
76|        else:
77|            others = ConcatSignal(*reversed(them))
78|
79|        @always_comb
80|        def enLogic():
81|            if ((sel == index) and
82|                (others == 0)):
83|                en.next = 1
```

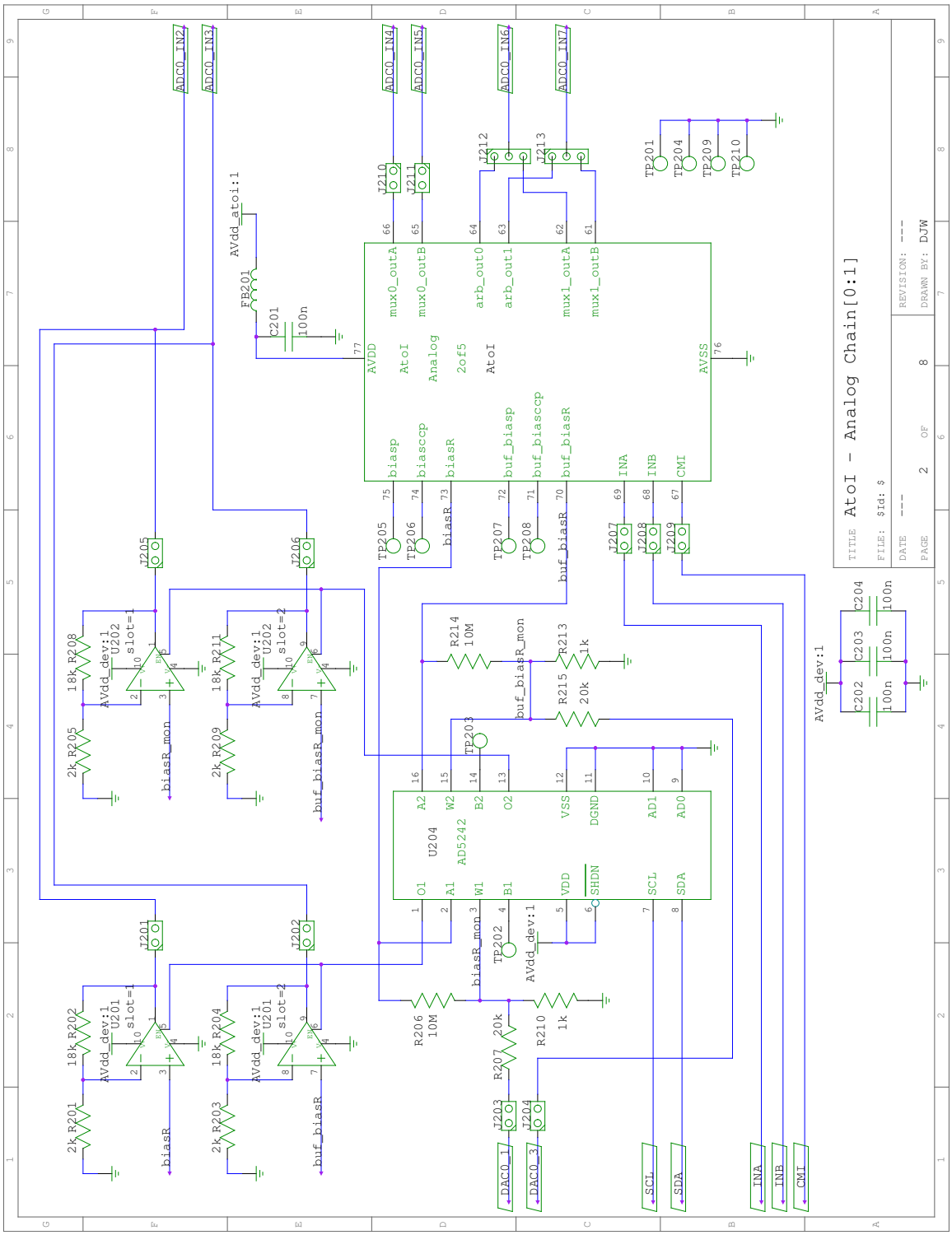
```
84|             else:
85|                 en.next = 0
86|         return instances()
87|
88|     #make the exclusive-enable combinational blocks
89|     enableBlocks = []
90|     for i in range(N):
91|         enableBlocks.append(EnableLogic(i, sel, in_enables[i]))
92|
93|     #make the synchronizers
94|     syncBlocks = []
95|     for i in range(N):
96|         syncBlocks.append(
97|                 Synchronizer(
98|                     reset,
99|                     in_clocks(i),
100|                    in_enables[i],
101|                    out_enables[i],
102|                    sync_clocks[i]
103|                    )
104|                )
105|
106|    @always_comb
107|    def clockOr():
108|        out_clock.next = (sync_clocks_concat != 0) # OR of all vectors
109|
110|    return instances()
111|
112|
113|def convert():
114|    N = 3
115|    reset = Signal(intbv(0)[0])
116|    sel = Signal(intbv(0)[2:])
117|    in_clocks = Signal(intbv(0)[N:])
118|    out_clock = Signal(intbv(0)[0])
119|
120|    toVerilog(ClockMux, N, reset, sel, in_clocks, out_clock)
121|
122|
123|if __name__ == '__main__':
124|    convert()
```

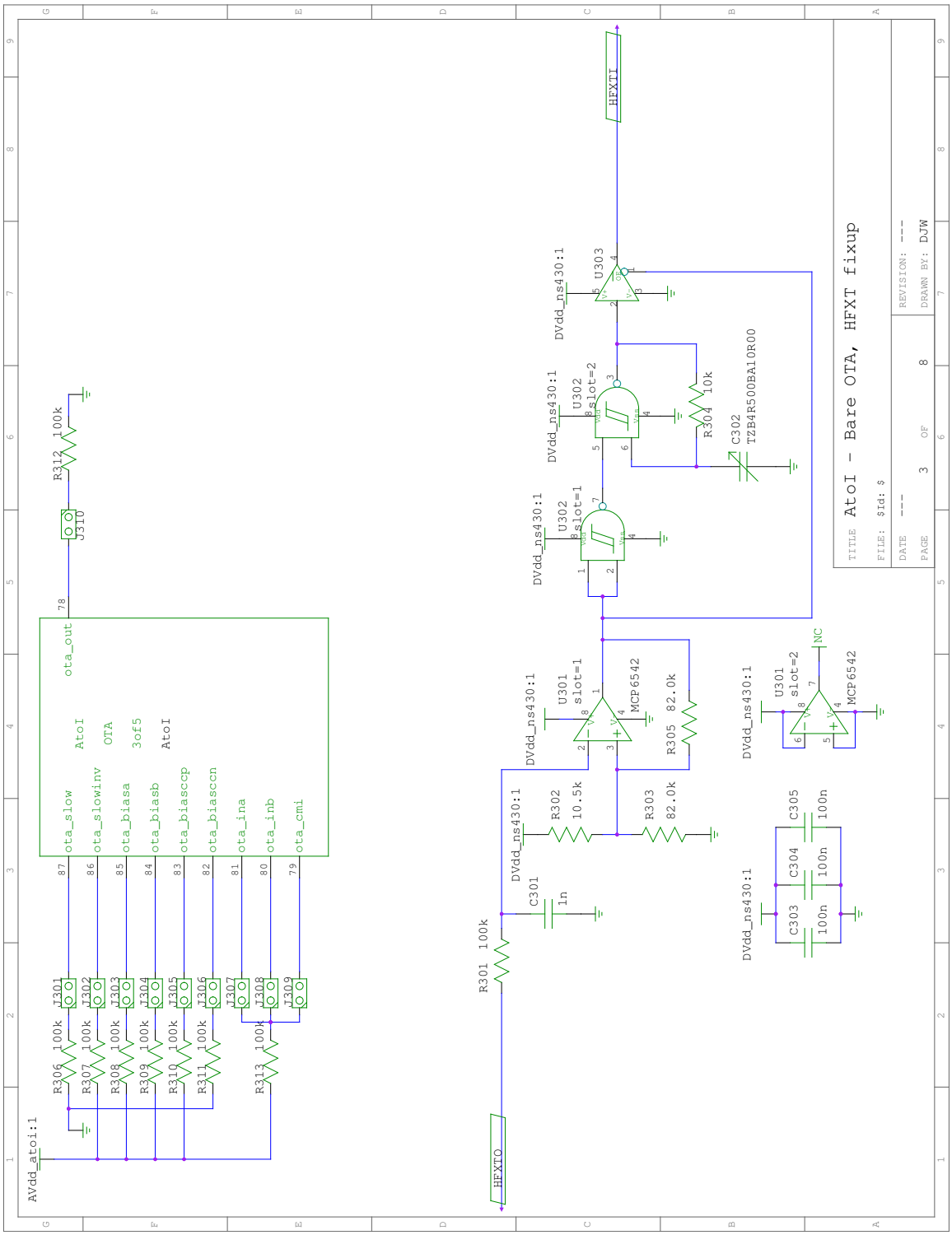
Appendix C

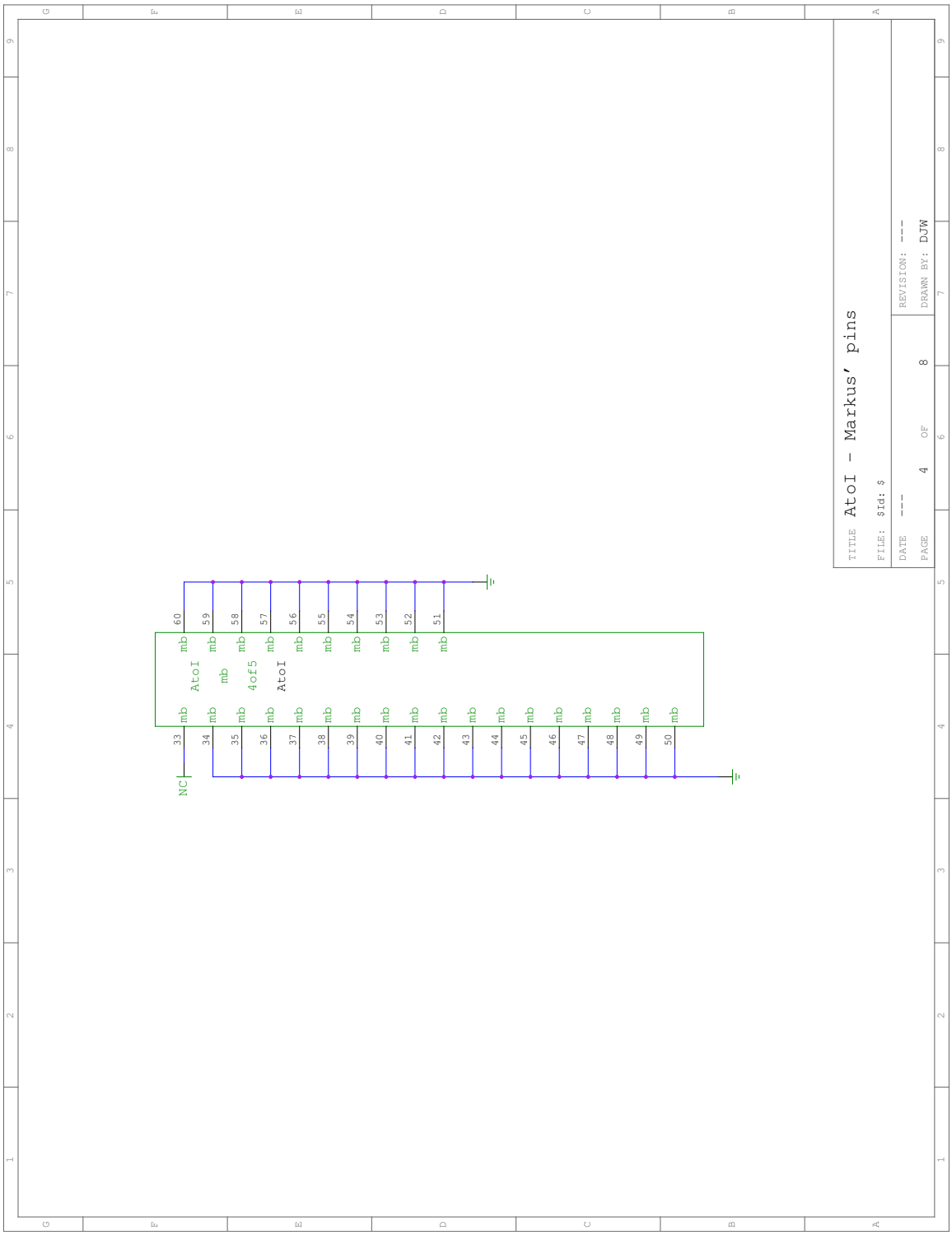
Schematics

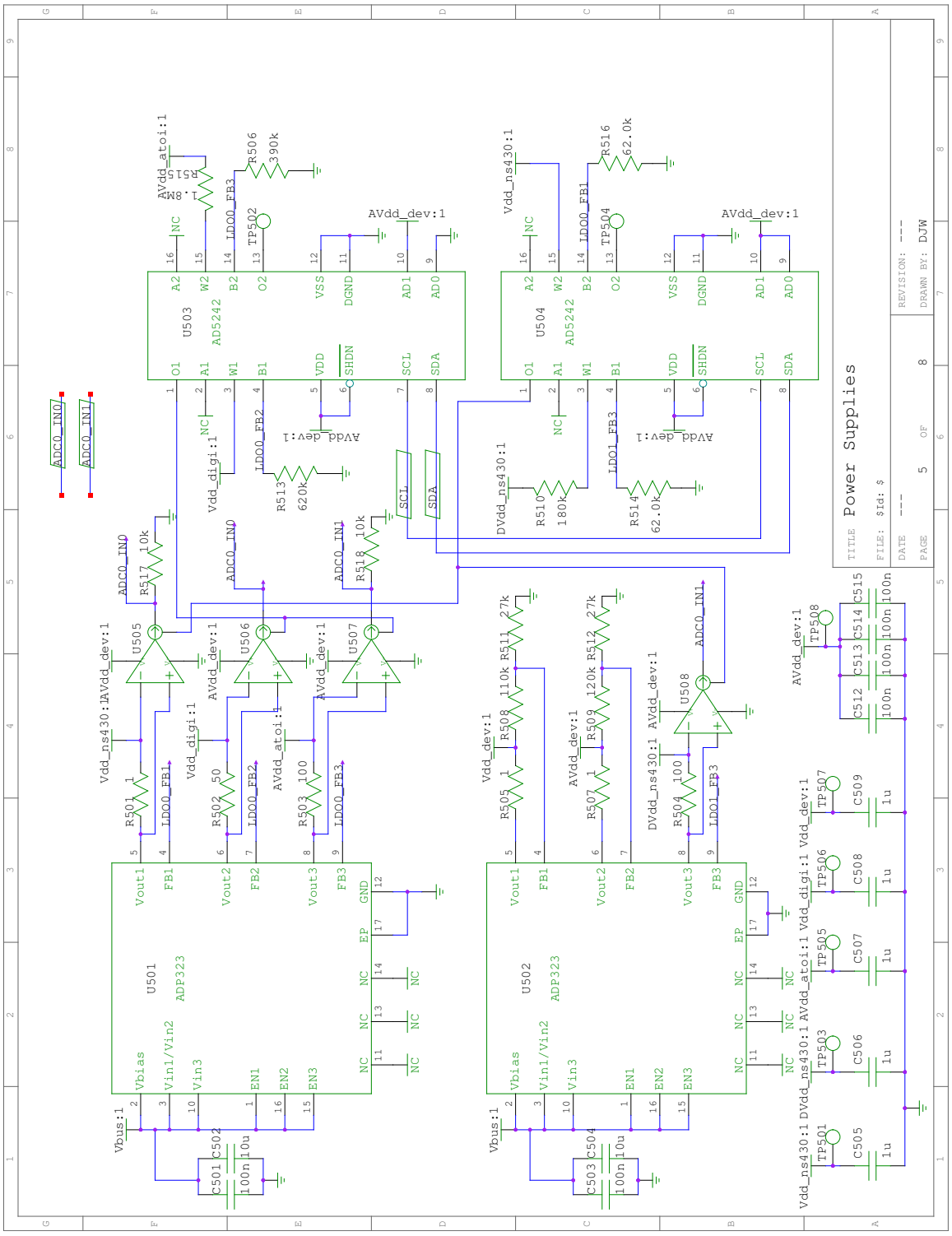
C.1 Test Fixture Board

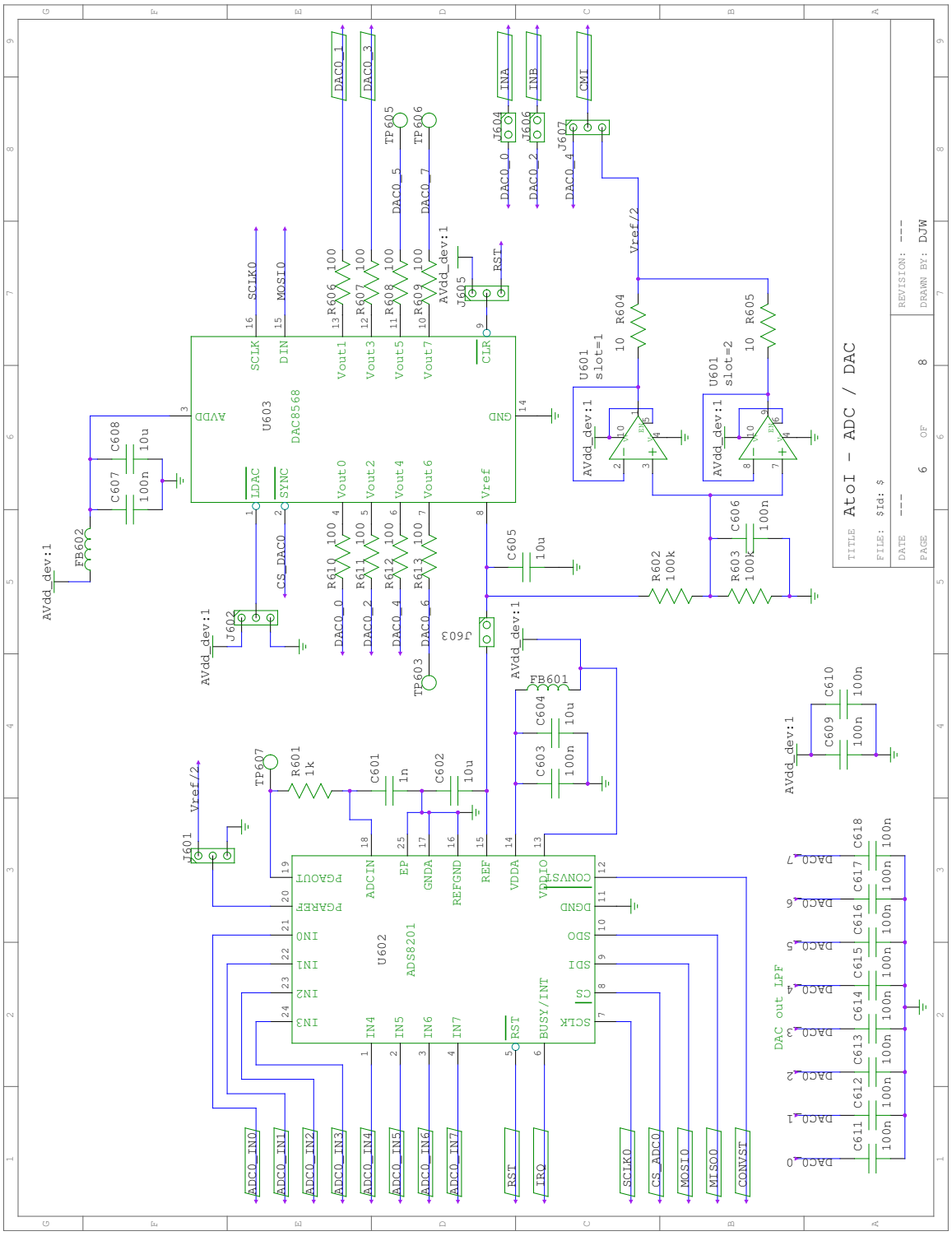


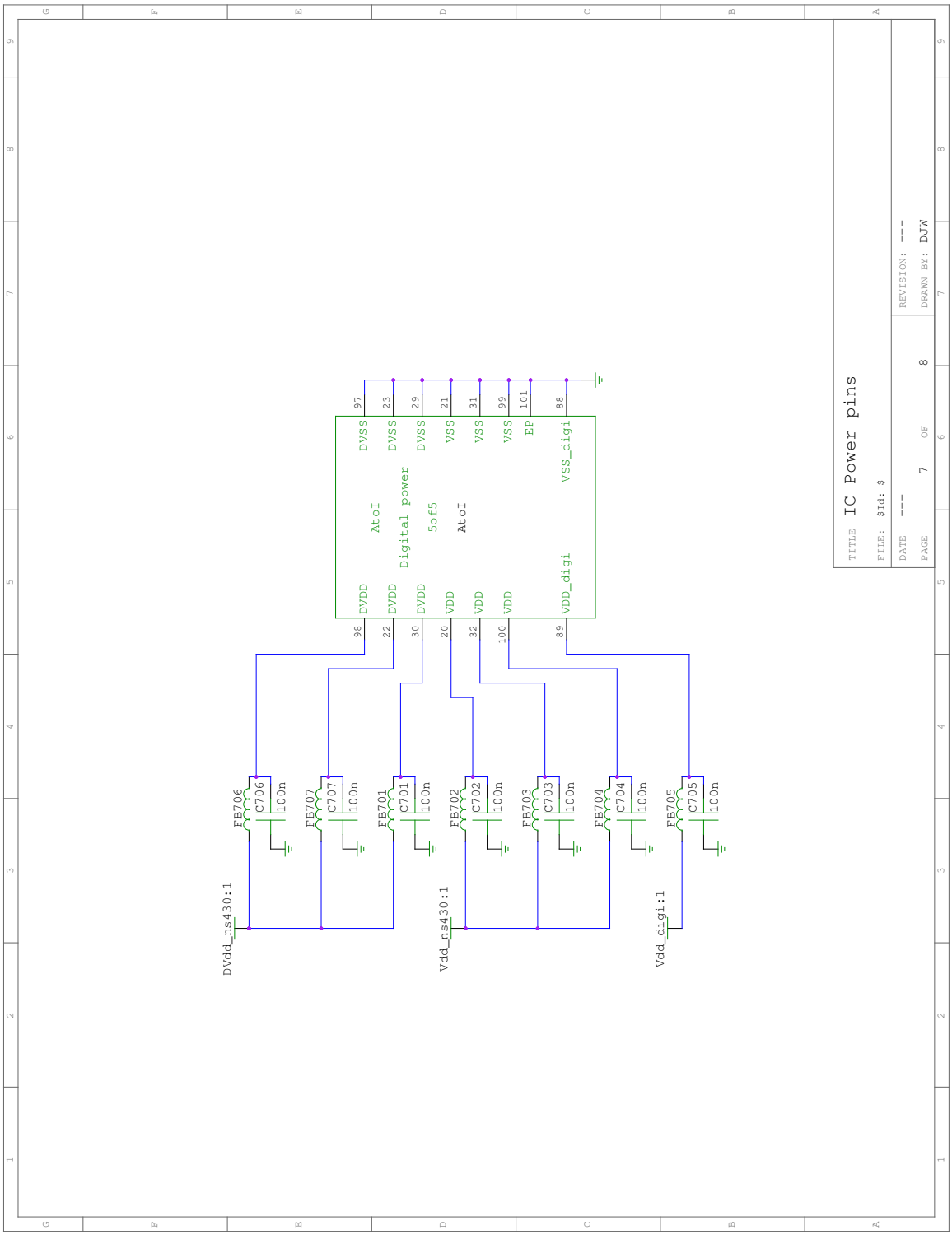

AtoI - Analog Chain[0:1]
TITLE
FILE: $Id: $
DATE ---
PAGE 2 OF 8
REVISION: ---
DRAWN BY: DJW
U204
AD5242
U201
U202
AVdd_dev:1
AVdd_atoi:1
ADC0_IN2
ADC0_IN3
ADC0_IN4
ADC0_IN5
ADC0_IN6
ADC0_IN7
FB201
C201
100n
AVDD
AVSS
AtoI
Analog
2of5
mux0_outA
mux0_outB
arb_out0
arb_out1
mux1_outA
mux1_outB
biasp
biascp
biasR
buf_biasp
buf_biascp
buf_biasR
INA
INB
CMI
J201
J202
J203
J204
J205
J206
J207
J208
J209
J210
J211
J212
J213
TP201
TP202
TP203
TP204
TP205
TP206
TP207
TP208
TP209
TP210
R201 2k
R202 18k
R203 2k
R204 18k
R205 2k
R206 10M
R207 20k
R208 18k
R209 2k
R210 1k
R211 18k
R213 1k
R214 10M
R215 20k
C202 100n
C203 100n
C204 100n
biasR
biasR_mon
buf_biasR
buf_biasR_mon
DAC0_1
DAC0_3
SCL
SDA
INA
INB
CMI

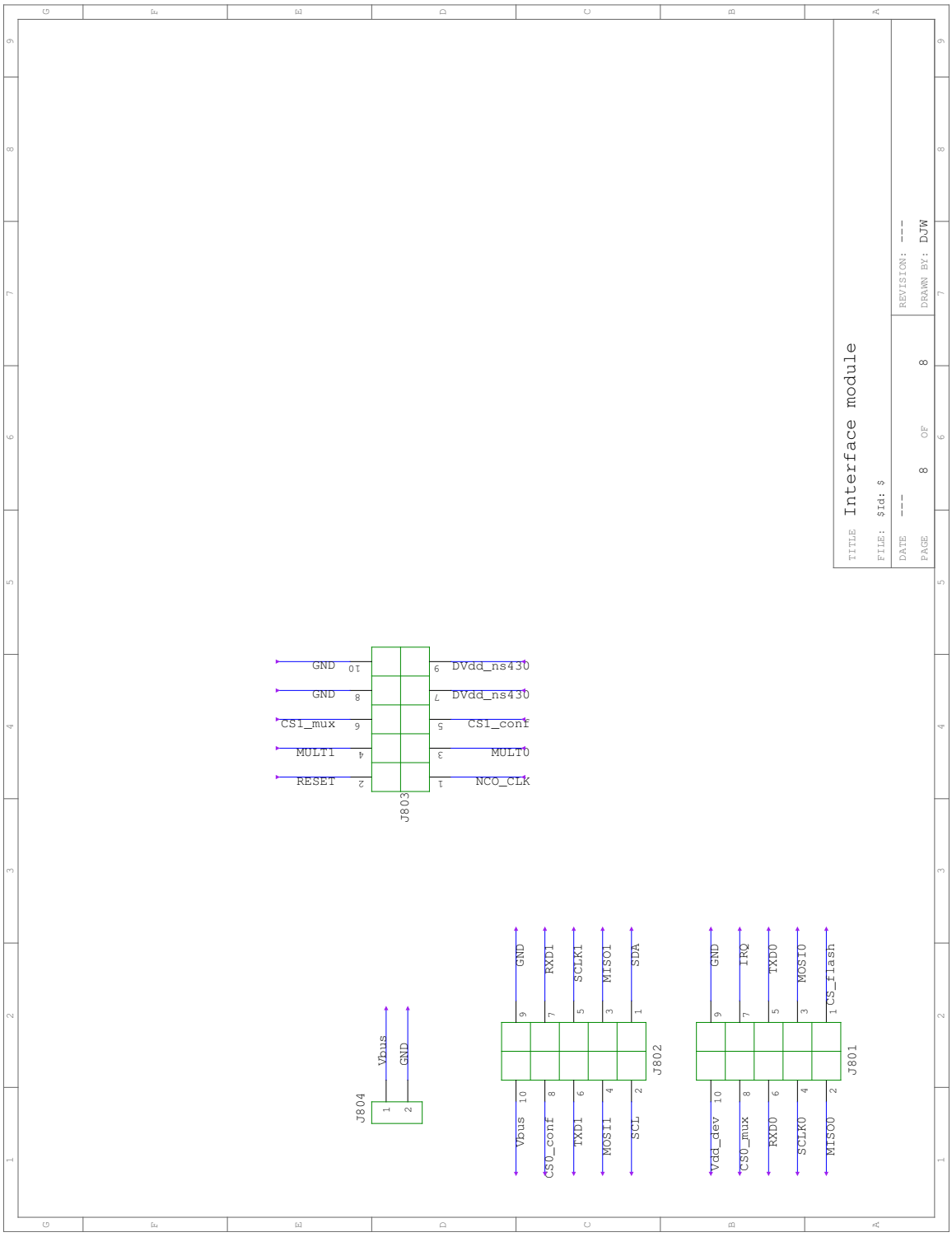

AtoI - Bare OTA, HFXT fixup
TITLE
FILE: $Id: $
DATE ---
PAGE 3 OF 8
REVISION: ---
DRAWN BY: DJW
HFXTI
HFXTO
AVdd_atoi:1
ota_out
ota_slow
ota_slowinv
ota_biasa
ota_biasb
ota_biasccp
ota_biasccn
ota_ina
ota_inb
ota_cmi
AtoI
OTA
3of5
AtoI
R312 100k
J310
R306 100k
R307 100k
R308 100k
R309 100k
R310 100k
R311 100k
R313 100k
J301
J302
J303
J304
J305
J306
J307
J308
J309
R301 100k
C301 1n
R302 10.5k
R303 82.0k
R304 10k
R305 82.0k
C302 TZB4R500BA10R00
C303 100n
C304 100n
C305 100n
U301 slot=1 MCP6542
U301 slot=2 MCP6542
U302 slot=1
U302 slot=2
U303
DVdd_ns430:1
NC


TITLE AtoI - Markus' pins
FILE: $Id: $
DATE ---
PAGE 4 OF 8
REVISION: ---
DRAWN BY: DJW
AtoI
mb
4of5
AtoI
NC

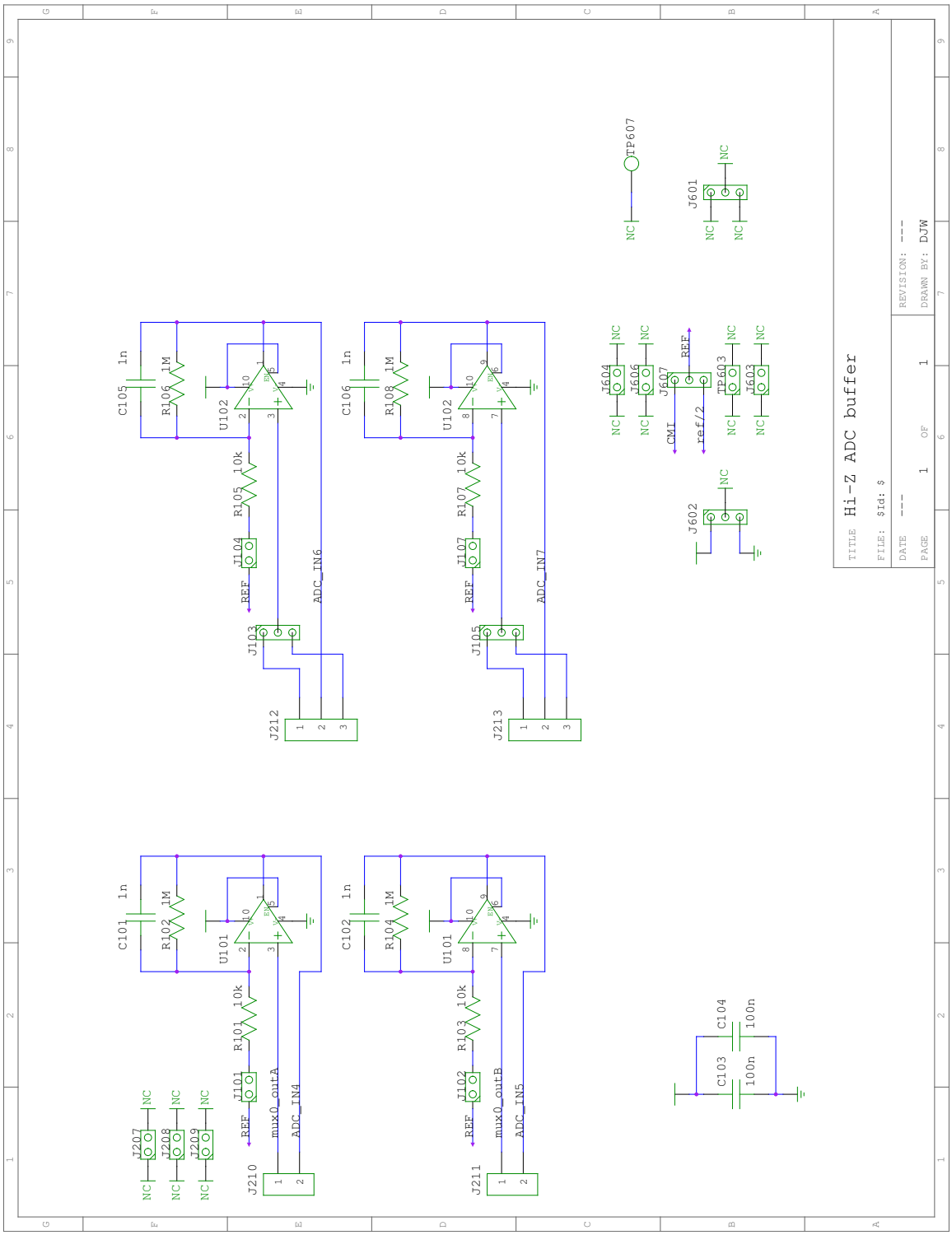

Power Supplies
TITLE
FILE: $Id: $
DATE ---
PAGE 5 OF 8
REVISION: ---
DRAWN BY: DJW
U501 ADP323
U502 ADP323
U503 AD5242
U504 AD5242
U505
U506
U507
U508
ADC0_IN0
ADC0_IN1
SCL
SDA
AVdd_dev:1
AVdd_ns430:1
AVdd_atoi:1
Vdd_dig:1
Vdd_ns430:1
DVdd_ns430:1
Vbus:1
R501 1
R502 50
R503 100
R504 100
R505 1
R506 390k
R507 1
R508 1
R509 120k
R510 180k
R511 27k
R512 27k
R513 620k
R514 62.0k
R515 1.8M
R516 62.0k
R517 10k
R518 10k
C501 100n
C502 10u
C503 100n
C504 10u
C505 1u
C506 1u
C507 1u
C508 1u
C509 1u
C512 100n
C513 100n
C514 100n
C515 100n
TP501
TP502
TP503
TP504
TP505
TP506
TP507
TP508


AtoI - ADC / DAC
U603
DAC8568
U602
ADS8201
U601
slot=1
slot=2
DAC out LPF
REVISION: ---
DRAWN BY: DJW

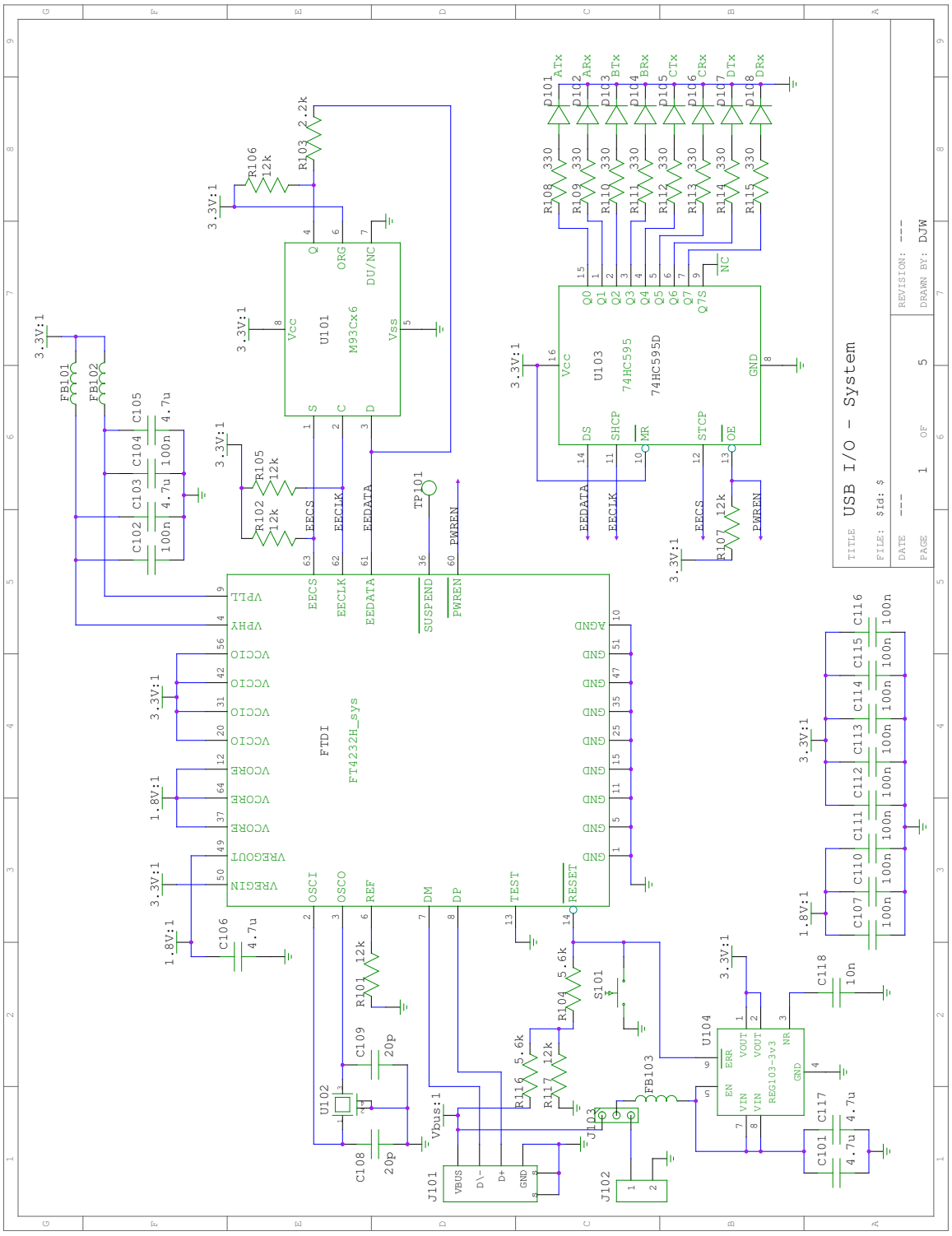

TITLE IC Power pins
FILE: $Id: $
DATE ---
PAGE 7 OF 8
REVISION: ---
DRAWN BY: DJW
DVSS
AtoI
Digital power
5of5
AtoI
VSS_digi
DVDD
VDD
VDD_digi
97
23
29
21
31
99
101
88
98
22
30
20
32
100
89
FB706
C706
100n
FB707
C707
100n
FB701
C701
100n
FB702
C702
100n
FB703
C703
100n
FB704
C704
100n
FB705
C705
100n
DVdd_ns430:1
Vdd_ns430:1
Vdd_digi:1

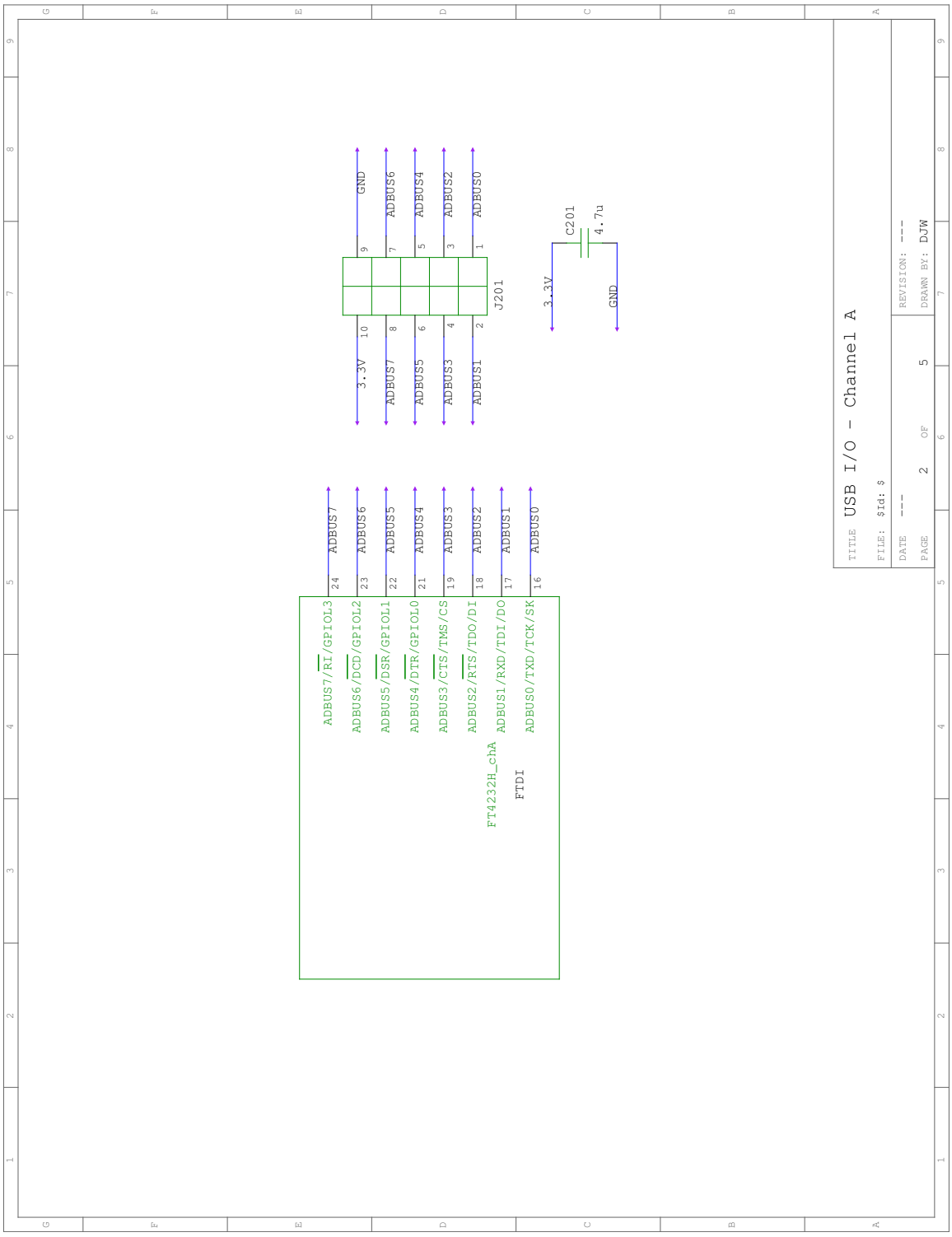

Interface module
TITLE
FILE: $Id: $
DATE ---
PAGE 8 OF 8
REVISION: ---
DRAWN BY: DJW
GND
GND
CS1_mux
MULT1
RESET
DVdd_ns430
DVdd_ns430
CS1_conf
MULT0
NCO_CLK
J803
Vbus
GND
J804
GND
RXD1
SCLK1
MISO1
SDA
Vbus
CS0_conf
TXD1
MOSI1
SCL
J802
GND
IRQ
TXD0
MOSI0
CS_flash
Vdd_dev
CS0_mux
RXD0
SCLK0
MISO0
J801

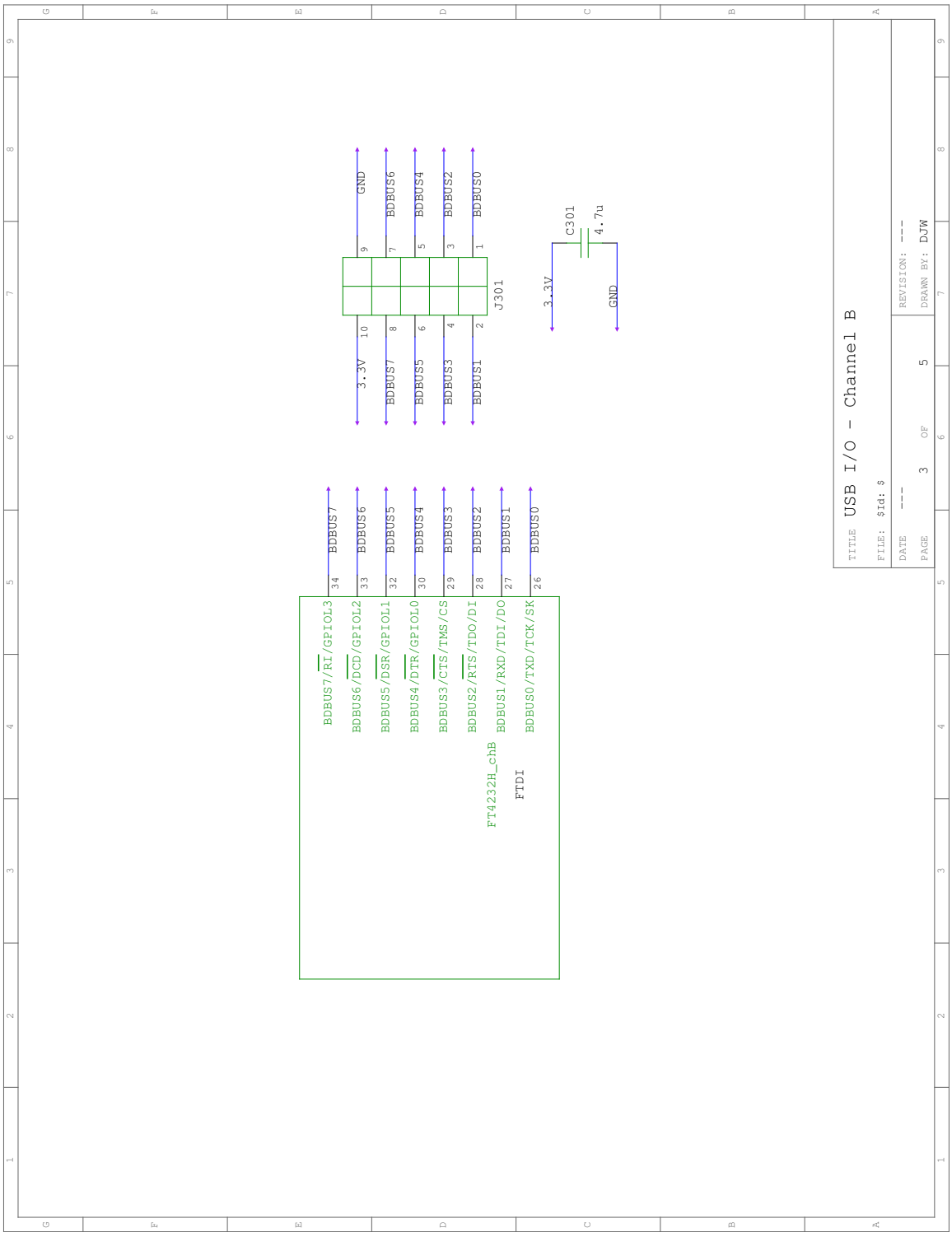
C.2 High-impedance Buffer

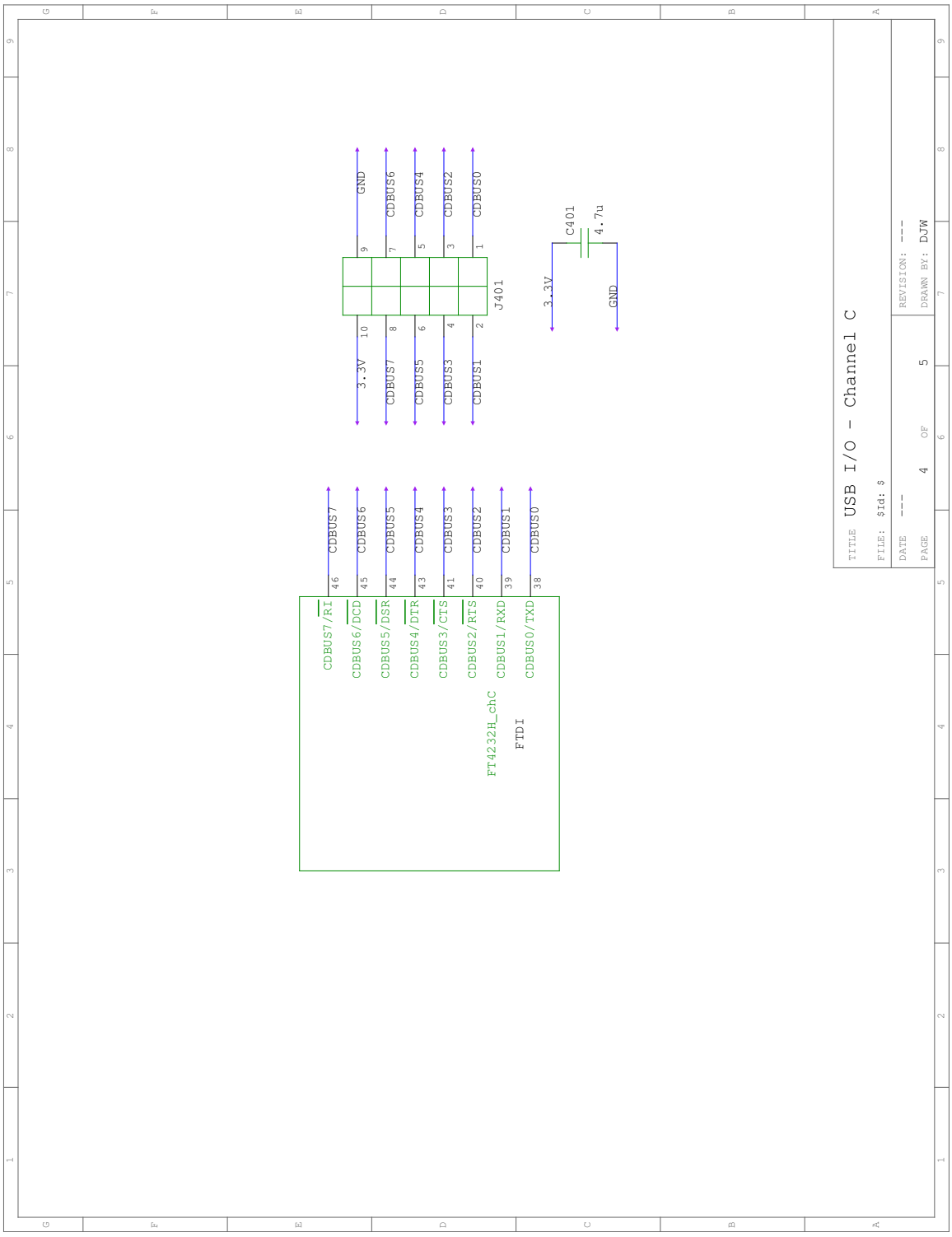


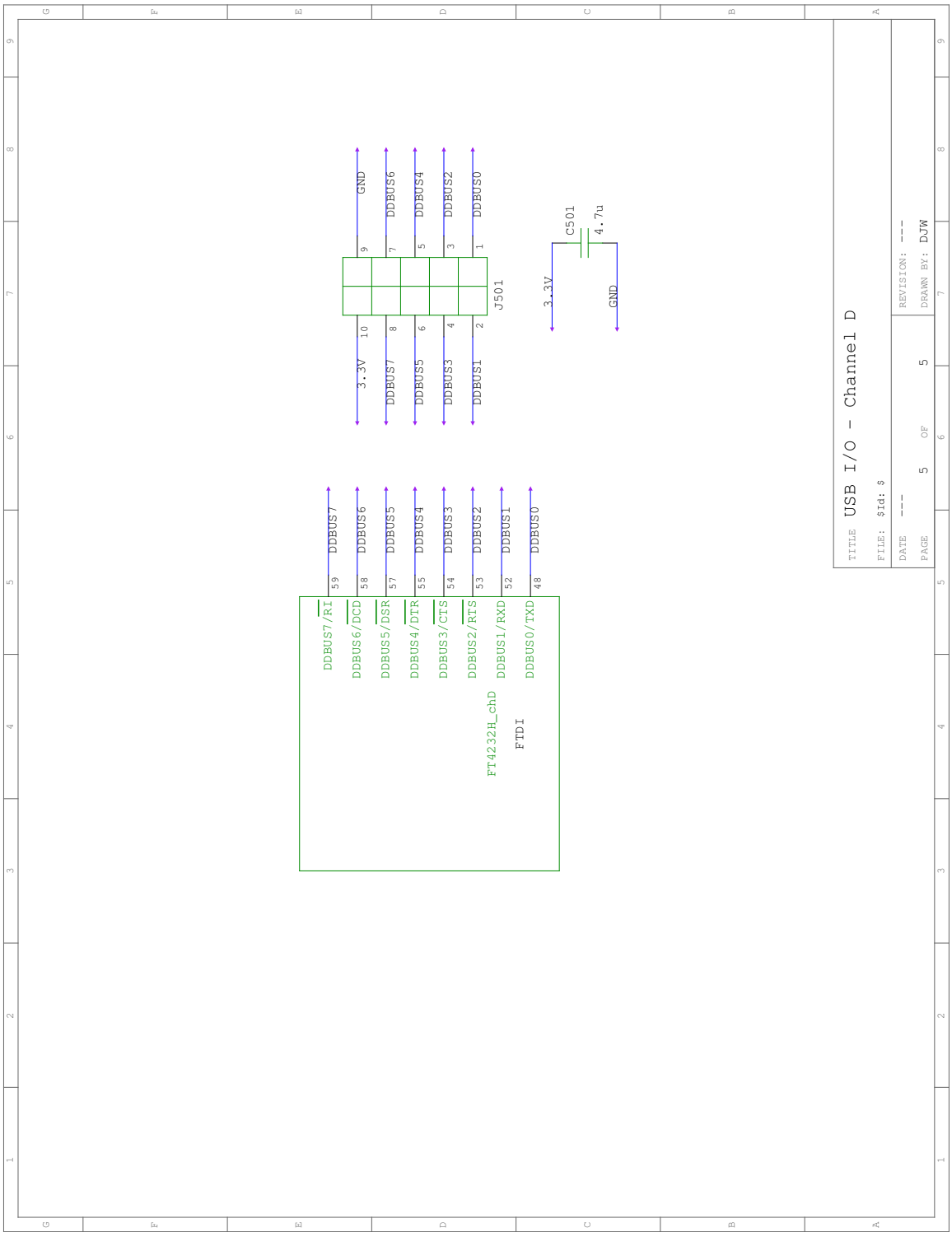
C.3 USB Interface




TITLE USB I/O - Channel A
FILE: $Id: $
DATE ---
PAGE 2 OF 5
REVISION: ---
DRAWN BY: DJW
FT4232H_chA
FTDI
ADBUS7/RI/GPIOL3
ADBUS6/DCD/GPIOL2
ADBUS5/DSR/GPIOL1
ADBUS4/DTR/GPIOL0
ADBUS3/CTS/TMS/CS
ADBUS2/RTS/TDO/DI
ADBUS1/RXD/TDI/DO
ADBUS0/TXD/TCK/SK
24 ADBUS7
23 ADBUS6
22 ADBUS5
21 ADBUS4
19 ADBUS3
18 ADBUS2
17 ADBUS1
16 ADBUS0
J201
10 3.3V
9 GND
8 ADBUS7
7 ADBUS6
6 ADBUS5
5 ADBUS4
4 ADBUS3
3 ADBUS2
2 ADBUS1
1 ADBUS0
C201
4.7u
3.3V
GND


TITLE USB I/O - Channel B
FILE: $Id: $
DATE ---
PAGE 3 OF 5
REVISION: ---
DRAWN BY: DJW
J301
GND
BDBUS6
BDBUS4
BDBUS2
BDBUS0
3.3V
BDBUS7
BDBUS5
BDBUS3
BDBUS1
C301
4.7u
3.3V
GND
FT4232H_chB
FTDI
BDBUS7/RI/GPIOL3
BDBUS6/DCD/GPIOL2
BDBUS5/DSR/GPIOL1
BDBUS4/DTR/GPIOL0
BDBUS3/CTS/TMS/CS
BDBUS2/RTS/TDO/DI
BDBUS1/RXD/TDI/DO
BDBUS0/TXD/TCK/SK


TITLE USB I/O - Channel C
FILE: $Id: $
DATE ---
PAGE 4 OF 5
REVISION: ---
DRAWN BY: DJW
J401
GND
CDBUS6
CDBUS4
CDBUS2
CDBUS0
3.3V
CDBUS7
CDBUS5
CDBUS3
CDBUS1
C401
4.7u
3.3V
GND
FT4232H_chC
FTDI
CDBUS7/RI
CDBUS6/DCD
CDBUS5/DSR
CDBUS4/DTR
CDBUS3/CTS
CDBUS2/RTS
CDBUS1/RXD
CDBUS0/TXD


TITLE USB I/O - Channel D
FILE: $Id: $
DATE ---
PAGE 5 OF 5
REVISION: ---
DRAWN BY: DJW
J501
GND
DDBUS6
DDBUS4
DDBUS2
DDBUS0
3.3V
DDBUS7
DDBUS5
DDBUS3
DDBUS1
C501
4.7u
3.3V
GND
FT4232H_chD
FTDI
DDBUS7/RI
DDBUS6/DCD
DDBUS5/DSR
DDBUS4/DTR
DDBUS3/CTS
DDBUS2/RTS
DDBUS1/RXD
DDBUS0/TXD