VARIABLE PRECISION TANDEM ANALOG-TO-DIGITAL CONVERTER (ADC)

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Engineering

by

Colton Parsons

June 2014

COMMITTEE MEMBERSHIP

TITLE:	Variable Precision Tandem Analog-to-Digital Converter (ADC)
AUTHOR:	Colton Parsons
DATE SUBMITTED:	June 2014
COMMITTEE CHAIR:	Dr. Wayne Pilkington, Associate Professor Electrical Engineering Department
COMMITTEE MEMBER:	Dr. Vladimir Prodanov, Assistant Professor Electrical Engineering Department
COMMITTEE MEMBER:	Dr. Tina Smilkstein, Assistant Professor Electrical Engineering Department

ABSTRACT

Variable Precision Tandem Analog-to-Digital Converter (ADC)

Colton Parsons

This paper describes an analog-to-digital signal converter which varies its precision as a function of input slew rate (maximum signal rate of change), in order to best follow the input in real time. It uses Flash and Successive Approximation (SAR) conversion techniques in sequence.

As part of the design, the concept of "total real-time optimization" is explored, where any delay at all is treated as an error (Error = Delay × Signal Slew Rate). This error metric is proposed for use in digital control systems. The ADC uses a 4-bit Flash converter in tandem with SAR logic that has variable precision (0 to 11 bits). This allows the Tandem ADC to switch from a fast, imprecise converter to a slow, precise converter. The level of precision is determined by the input's peak rate of change, optimized for minimum real-time error; a secondary goal is to react quickly to input transient spikes.

The implementation of the Tandem ADC is described, along with various issues which arise when designing such a converter and how they may be dealt with. These include Flash ADC inaccuracies, rounding issues, and system timing and synchronization.

Most of the design is described down to the level of logic gates and related building blocks (e.g. latches and flip-flops), and various logic optimizations are used in the design to reduce calculation delays. The design also avoids active analog circuitry whenever possible – it can be almost entirely implemented with CMOS logic and passive analog components.

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Wayne Pilkington, for providing invaluable guidance during this project, including pointing out its weak points and ensuring it could be justified. I would also like to thank Prof. Tina Smilkstein and Prof. Vladimir Prodanov for their time serving on the thesis committee. And I would like to thank my parents, Wendelin and Mark, for all the support they've given me.

TABLE OF CONTENTS

LIST OF FIGURES

CONVENTIONS

Block Diagram & Schematic Legend

For the diagrams in this paper, analog signals are represented with unfilled arrows and digital signals are represented with filled arrows. The full conventions are shown below:

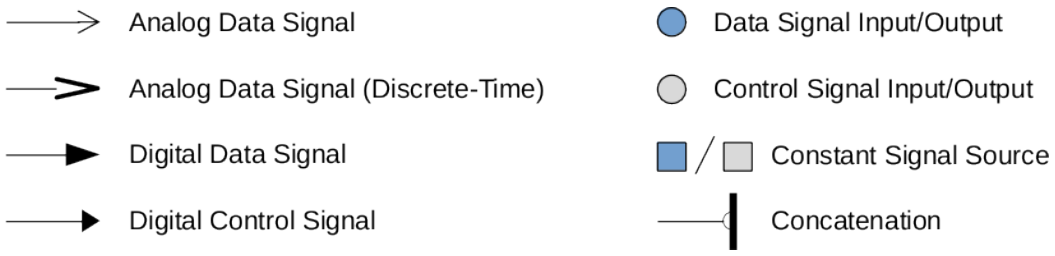

Figure 0.1 – Signal Path Legend

Lines with no arrow signify a wire connection, which can be either analog or digital. For cases where signal direction is ambiguous, signal paths flow out from the bottom and right edges of blocks, and in to the top and left edges.

As for component colors, Level 1 blocks are shown in gray, while most Level 2+ blocks are shown in blue. The exceptions are comparators, which are yellow, counters, which are green, and logic gates, which are white.

Figure Numbers

Figure numbers are in the format *x.y*, where *x* is the chapter number and *y* is the figure number within the chapter. (For figures coming before the main text, *x* is 0.) Figures are numbered independently from sections within chapters.

Chapter 1
Introduction and Background

Analog-to-Digital Converters have found their way into a wide range of applications, and are used in almost every form of digital signal processing. The type of ADC used for any particular application depends on what is required and the resources available. In general there is a three-way tradeoff between speed, precision, and cost in the selection of an ADC. Speed itself has two aspects—latency and conversion rate—although in the case of realtime optimization latency is a much larger issue.

1.1 Speed, Precision, Size, and Accuracy

Two of the most common ADC topologies for high speed, real-time signal processing are Flash and Successive Approximation converters. Flash ADCs are usually used when speed (high conversion rate and low latency) is the most important consideration: their sampling rate is independent of precision and can be in excess of one billion samples per second.[1] However, their precision is typically limited to 8 bits,[2] and their size (and therefore cost) doubles with each bit added. Successive Approximation Register (SAR) ADCs are used when greater precision is needed, as up to 16 bit precision can be achieved at reasonable costs. Unfortunately, Successive Approximation designs require a much longer sampling time, on the order of a few hundred nanoseconds, in order to complete their iterative search for the correct sample value.[3]

Flash and Successive Approximation converters provide different options in the speed-precision-cost tradeoff, as shown in the following examples:

- **4-bit Flash:** very low precision (16 possible values), excellent sampling rate

(~1GSps), moderate size (15 comparators, thermometer-decoding logic).

- **8-bit Flash:** moderate precision (256 possible values), excellent sampling rate (~1GSps), very large size (255 comparators, thermometer-decoding logic).
- **Successive Approximation:** high precision (65,536 possible values), moderate sampling rate (<5MSps), moderate size (counter, clock, comparator, Digital-to-Analog converter).

The speed and precision requirements on the ADC for a particular task depends on what constitutes a "good enough" approximation of the analog input signal for the application. For example, a significant share of signal processing is not time-critical (e.g. audio recording); and in such situations the emphasis is on maximizing the numeric precision to reduce quantization noise. The sampling rate must be above the input signal's Nyquist rate; but beyond that additional speed is theoretically unnecessary and superfluous. For other tasks the situation is reversed – speed is critical and so precision is compromised to a minimally acceptable level. Control systems are closest to the latter case, as they operate best with considerable oversampling, and in most cases giving up a little precision in order to increase sampling rate is beneficial overall, especially when handling situations where high slew rates in control or error signals from sudden changes in command voltages or oscillation prevention effectively raise the Nyquist rate temporarily for the system beyond the normal needs of the closed-loop control system elements. This can happen when a step input is presented to a control system, or when some outside factor causes a large jarring disturbance.

This paper will focus on the second of those two cases, where oversampling is

desired and speed is often valued (at least temporarily) over precision at critical times. The goals of this project are to explore an ADC architecture that will minimize the mean-square error between the analog input signal and its digital counterpart, both overall and at any given short time interval, and provide as fast of a response as possible to large input transient spikes. As such, this paper will discuss tradeoffs between precision and speed, and how they affect accuracy. Since the terms "accuracy" and "precision" can vary in meaning depending on context, for this paper they will carry the following definitions:

- **Accuracy:** how close an estimation is to reality. In this case, how close a digitized signal is to the analog original. Measured in terms of mean-square error.
- **Precision:** the amount of useful information a value has. In this case, the number of significant bits carried by a digital signal.

In addition, the term "speed" is generally used to refer to latency (or rather the inverse) in this paper rather than to sampling rate, although for the ADC topologies explored the two will have a more-or-less direct relationship (e.g. there will be little or no signal path pipelining).

1.2 Flash, Successive Approximation, and Hybrids

Flash is often considered to be the simplest[4] and most direct[5] conversion topology for Analog-to-Digital converters. It works with a simple resistor ladder (2^n resistors, for an n-bit converter) paired with a comparator array (2^{n-1} comparators), followed by a thermometer decoder.[4,5] This is shown below in Figure 1.1.

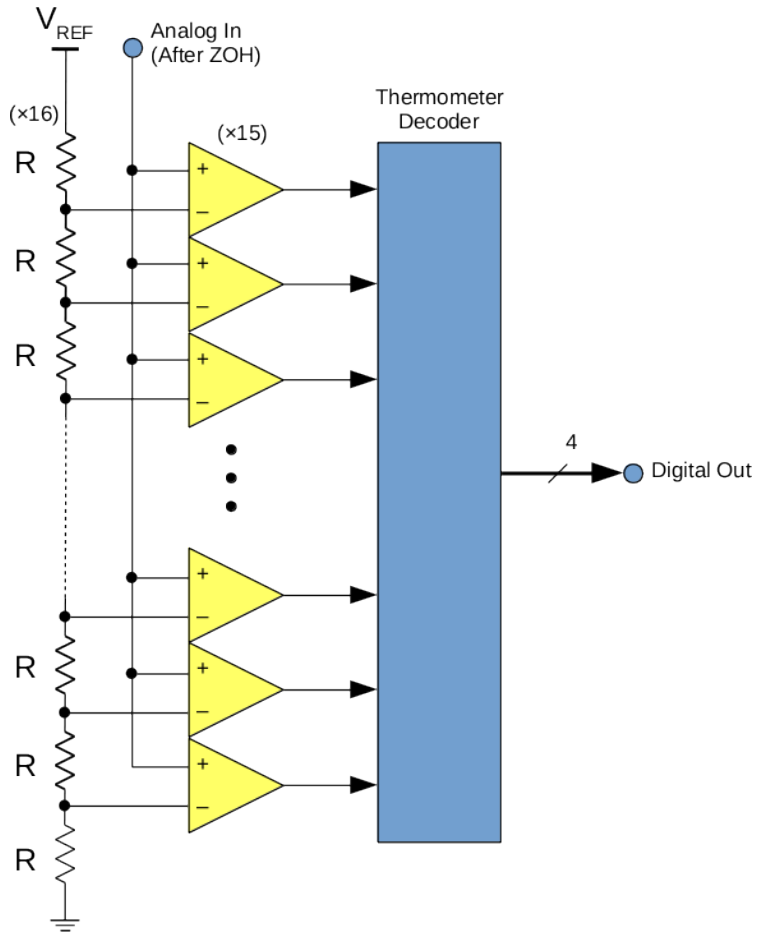

Figure 1.1 – 4-bit Flash ADC

In exchange for this directness, the Flash ADC has an open-loop nature which, coupled with small nonidealities such as component mismatches and noise, limits resolution to roughly 8 bits.[2] (This is elaborated on in Section 5.1.) In contrast, the Successive Approximation Register (SAR) topology is a closed-loop form,[12] enabling much higher precision levels, typically up to 18 bits.[3] The general diagram is shown below in Figure 1.2.[13]

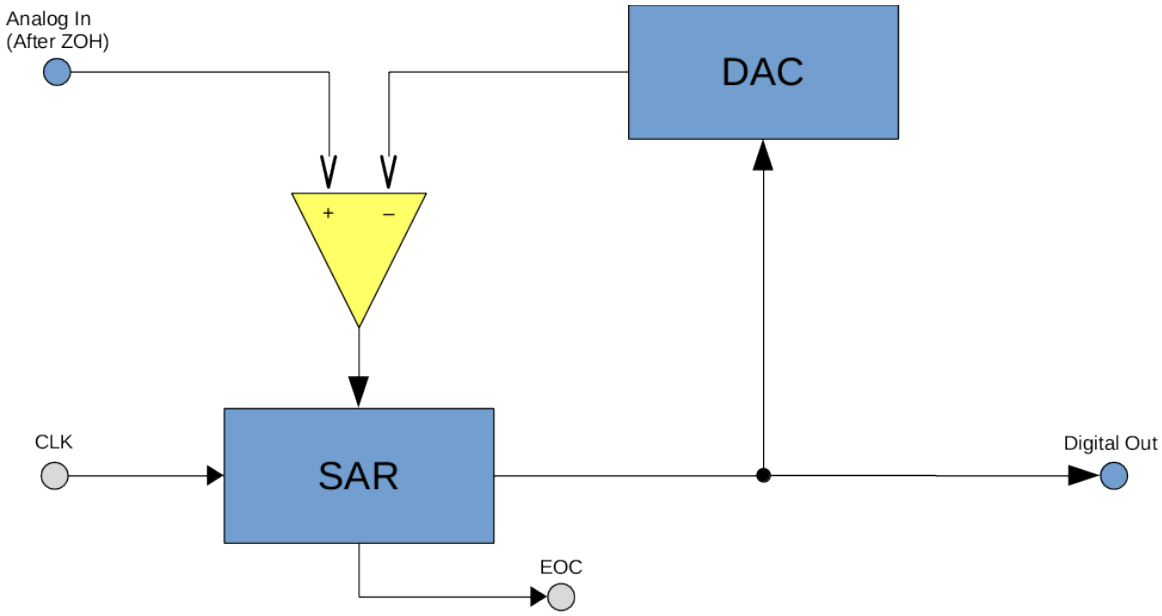

Figure 1.2 – Successive Approximation ADC

Successive Approximation ADCs work by an iterative process in which one bit is set high at a time, starting with the most significant bit. For each bit, if setting it high causes the output (when converted back to analog) to exceed the analog input value, that bit is set low again. If not, the bit remains high. This process synthesizes a digital output which is as close as possible to the analog input, rounded down.

Unlike the Flash ADC, a Successive Approximation ADC requires only minor size growth if precision is increased. A 16-bit SA converter doesn't require much more hardware than a 12-bit converter, for example.[3] The downside is that sampling time increases proportionally with precision rather than being constant, e.g. a 16-bit SA will take an extra third longer than a 12-bit SA, all else being equal.

These two conversion topologies can be combined in numerous ways. One popular combination is the Subranging ADC, which uses a Flash converter inside of a Successive Approximation converter, more-or-less replacing the function of the

comparator. This allows multiple bits to be found each clock cycle, and the end result is a converter which is faster than a Successive Approximation and more precise than a Flash.[6] A Pipelined ADC has similar behavior, using multiple Flash ADCs.[7] Each Flash ADC's output is converted back to analog and subtracted from the analog input, the difference being passed to the next Flash ADC. This is shown below in Figure 1.3:

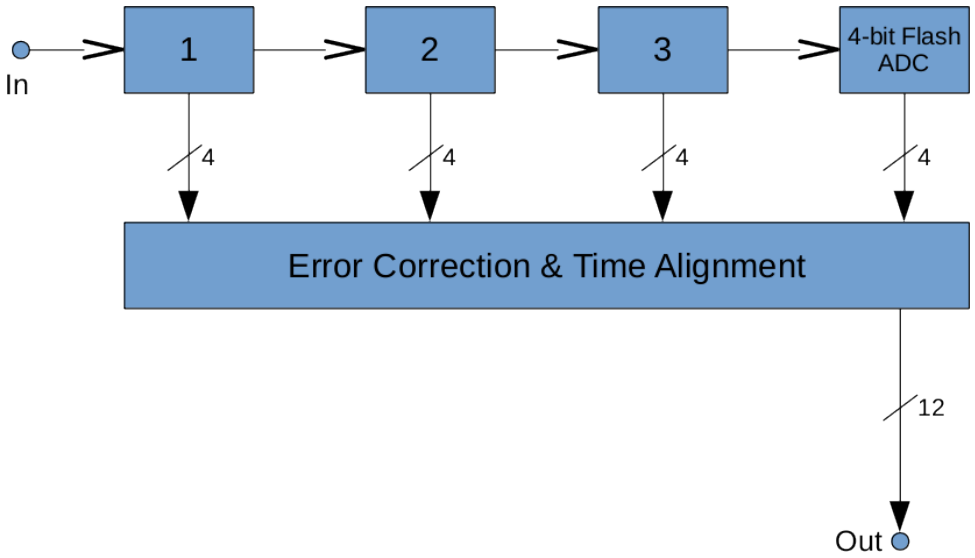

Figure 1.3 – Pipelined ADC

Each block labeled 1-4 contains the interns shown in Figure 1.4:

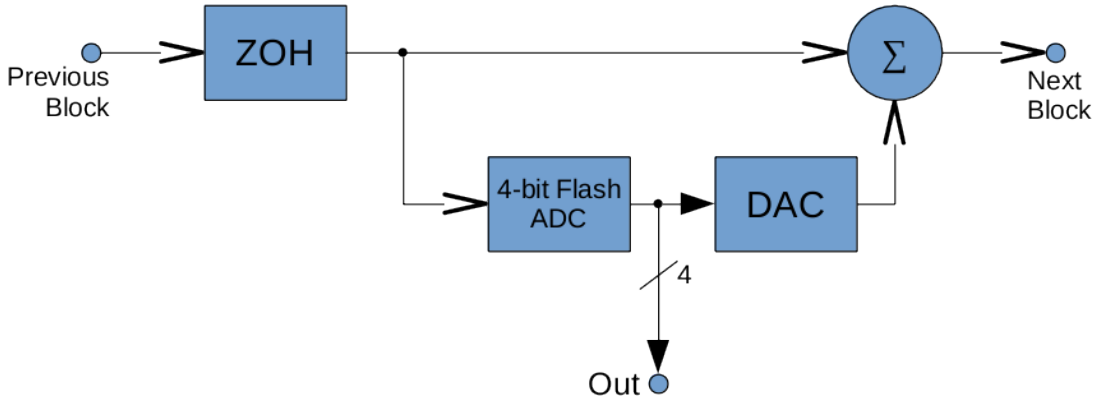

Figure 1.4 – Pipelined ADC Sub-Block

Another possible combination is the so-called Tandem ADC, which is what this

paper proposes. A Flash converter finds the first few bits (4, in this case) and then the conversion task is handed off to a modified Successive Approximation converter, which finds the remaining bits or precision. This creates a system which can do the same job as a Successive Approximation ADC in fewer clock cycles, and does not inherit as much potential for Flash-related errors as does a Pipelined or Subranging ADC. In this proposal the Successive Approximation precision is variable, and in the lowest precision mode the system essentially behaves as an error-corrected Flash ADC.

The general diagram of such a topology is shown below. A more detailed block diagram is shown in Figure 3.1.

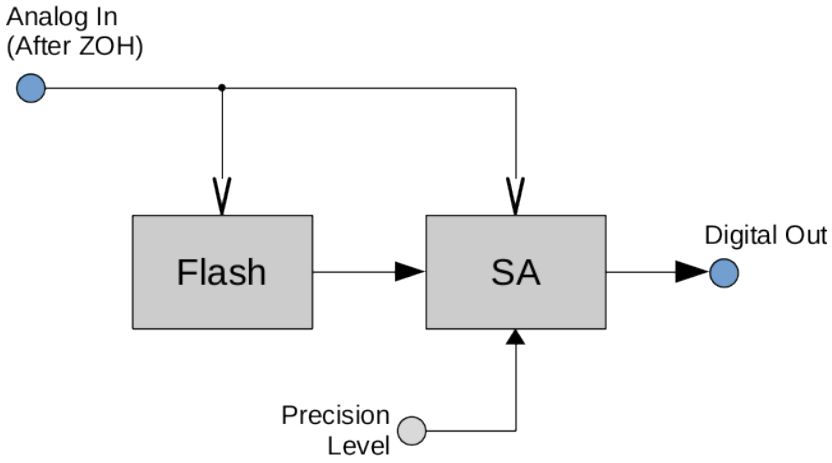

Figure 1.5 – Simple Tandem ADC Architecture

Chapter 2
Justification and Application of Design Concept

The underlying principle is for the Tandem ADC to vary its level of resolution as a function of the input signal characteristics, trading precision for speed as is needed. (Resolution will only change occasionally to keep the system from constantly switching from one mode of operation to another.) While simple in concept, this method presents a few unique design challenges that must be dealt with.

2.1 Of Aliasing and Oversampling

One issue with a variable-speed ADC is that each mode of operation (each sampling rate) has a different maximum input Nyquist frequency. This raises the question of how to design an appropriate anti-aliasing filter. One possibility is to have a different anti-aliasing filter for each mode, and switch between filter outputs with a transmission gate system. However this is overly complicated, and would introduce issues of its own because of transmission gate feedthrough impedance and parasitic capacitances.

Fortunately, in most cases a single anti-aliasing filter is sufficient. This is because the Tandem ADC is meant to work with control systems, most of which are heavily oversampled.[8] The mode switching of this ADC prevents aliasing from occurring by switching over to a faster sampling mode if the Nyquist rate of the input signal approaches the sampling rate. Therefore, a single anti-aliasing filter designed for the fastest mode can be used for the whole system, provided that the noise sources in the system are reasonably constrained.

The Tandem ADC's mode selection is a function of input signal slew rate (see

Section 6.4.1). For any given harmonic input signal, the slew rate is directly proportional to both the waveform's frequency and its amplitude (SR = ω×A). It *is* possible for low amplitude spectral content to show up above the Nyquist rate of any given sampling mode without triggering a switch to a higher mode. However, for any given harmonic pattern and slew rate, the frequency and amplitude are inversely proportional; therefore any input which has high enough frequency to cause aliasing and low enough slew rate to avoid tripping the mode selector will have a small enough amplitude to be negligible.

Note also that the switching of modes is not only done to prevent aliasing. The resulting increase in oversampling improves system performance in general for a control system, which in the right situations outweighs the detriments of increased quantization noise. Some specifics related to group delay are covered in the following section.

2.2 Minimizing Group Delay

In a mixed system (DSP plus ADC/DAC), each component involved has its own complex frequency response, although in many cases certain signal handling components (anti-aliasing filter, zero-order hold, ADC, anti-imaging filter if there's an analog output, etc) are often thought of primarily in terms of their magnitude responses, with phase only being considered for the purposes of phase margin and related stability. This often makes little difference since their phase responses are small-valued compared to that of a digital filter or that of the plant which said filter feeds in the case of a control system. However when it comes to getting as fast of a response to the input as possible (especially when there is the potential for large unexpected transients), and in order to maximize the stability of the control system, every delay in the system matters.

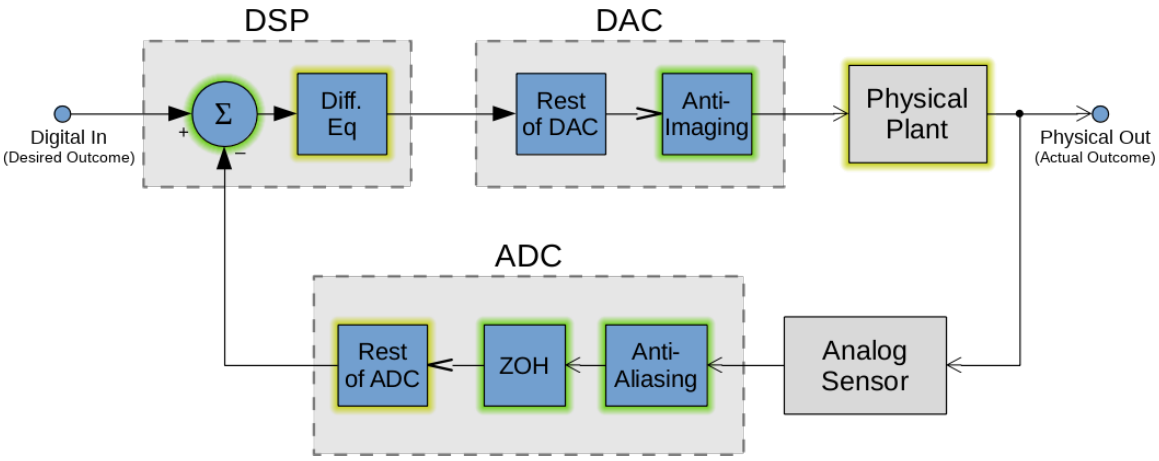

Figure 2.1 – Control System Block Diagram

Shown in Figure 2.1 above is a typical control system, with elements containing significant delays highlighted. (Delay can either take the form of digital logic propagation/computation delay, or group delay, which is defined as the negative derivative of the phase response.) Those with a typical delay of roughly one sample period or more are highlighted in yellow, those with a delay of one sample period or less are highlighted in green. The former are the ones that are almost always modeled, while the latter can sometimes be neglected in terms of phase response and delay characteristics, or have those characteristics only considered for stability purposes.

The primary delays seen above are from the ADC (excluding zero-order hold and anti-aliasing), the difference equation calculation of the DSP, and the behavior of the physical plant itself. Of these the ADC usually has the smallest delay (roughly one sampling cycle) while the difference equation and physical plant have larger delays.

In certain control systems, however, the difference equation can have a very small group delay. In the case of a PID controller (see Chapter 4 for an example) the effective delay can be on the order of one sampling cycle, putting it on par with the ADC.

In these situations that ADC delay becomes important, as does that of the blocks highlighted in green. Also, depending on the implementation, shrinking sampling time can shrink difference equation delay proportionally. (An implementation of this is shown in Chapter 4.) If the difference equation has a fixed continuous-time delay (i.e. filter length scales with sampling rate) faster sampling still has its advantages, but they are less pronounced.

Of the green-highlighted blocks, perhaps the most noticeable is the Zero-Order Hold. It can be modeled as a multiplication of the input signal with an impulse comb function followed by a convolution with a (causal) rectangle function of width equal to the sample time, and so its group delay is half the sample time. This is smaller than the group delay of a PID controller, but it's still comparable. This is another reason why a shorter sampling time is advantageous—the ZOH delay, like the difference equation delay, can shrink proportionally.

There are also the anti-aliasing and anti-imaging filters. Their contribution to delay depends mainly on what type of filter is used; Butterworth and Bessel filters were considered as the best choices because of their flat passband response and relatively constant low group delay. (See Section 6.1 for more.)

And finally, there's the summing junction, labeled Σ. If treated as its own block within the Digital Signal Processor, it has a delay of one clock cycle. (In reality a single subtraction operation will almost certainly take less time than that, but synchronicity must be preserved.) However this delay can potentially be eliminated if the summation is incorporated into the difference equation, which then becomes a multi-input-single-

output (MISO) system.

2.3 Quantization Error and Delay Error

In many signal processing systems (e.g. audio recording) signal delay is not an issue. For control systems and other real-time systems, on the other hand, signal delay can be a huge problem. In keeping with one of the main ideas of this thesis, which is that a fast, imprecise response is in certain situations preferable to a slow but precise response, the error of a system can be measured as the difference between the desired output and the realized output at any given time. (In the case of an ADC these are the analog input and the analog value to which the digital output corresponds.) Such a real-time optimized metric incorporates both the quantization error and delay error of an ADC, treating them as effectively the same.

Minimizing this real-time error is the basis for calibrating the points at which the conversion rate switches, covered in Sections 3.2 and 6.4.1. It is possible to calibrate the switching differently if desired; the best calibration depends on the expected nature of the input signal and the desired behavior of the output. Treating quantization error and delay-induced error as one and the same is only useful for strict realtime optimization as is explored in this paper; for other optimization schemes different metrics may be used.

Chapter 3
Design – Concept and Theory

3.1 Implementation Concept

The fundamental concept for the implementation is to use a Flash ADC and Successive Approximation ADC in tandem (as shown in Figure 1.5), with variable Successive Approximation precision. This can be accomplished by modifying the usual closed-loop architecture used for Successive Approximation converters to include the output bits of the Flash ADC as the highest order conversion result bits. This allows the Successive Approximation logic to essentially pick up where the Flash conversion left off, adding precision to the Flash results rather than starting at the most significant bit and determining all bits as it usually does.

The mode of operation (i.e. the number of bits of precision to complete) should be determined by whichever mode provides the best precision/speed trade-off for a control system; the goal being to minimize the real-time mean-square error between the input signal and the ADC output's equivalent output analog value. As such, the derivative of the input signal is measured in analog form; and serves as a slew rate metric for selecting the optimum conversion rate. The higher the derivative value, the lower the precision and the faster the sampling time setting chosen for analog-to-digital conversion.

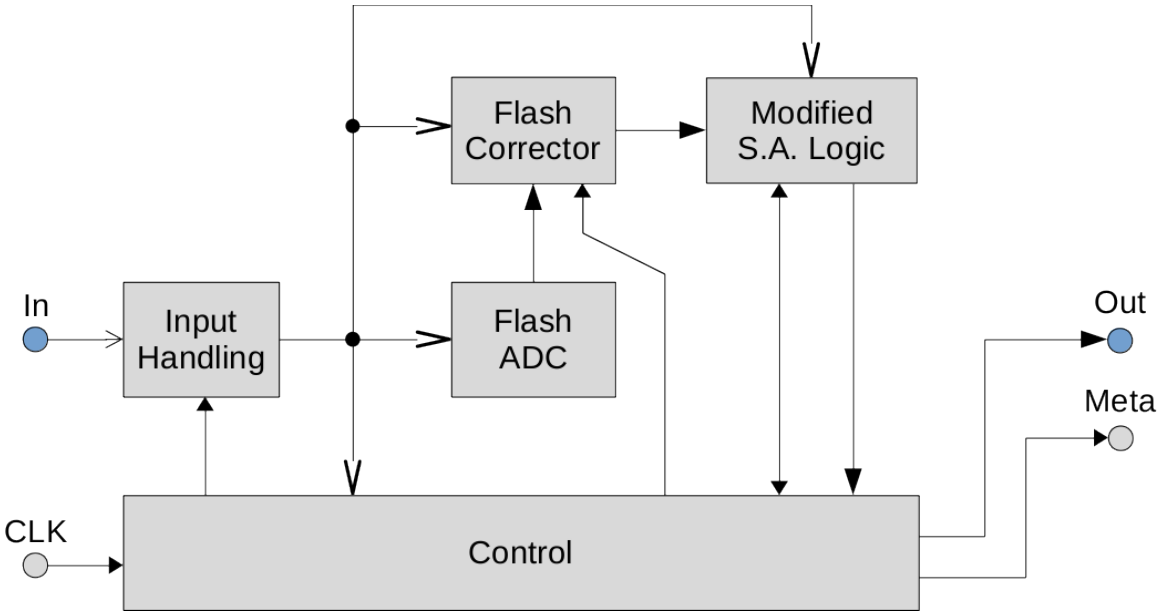

Figure 3.1 – Tandem ADC Overall Design
(Level 1 Block Diagram)

To implement the Tandem A/D Converter, the analog input signal goes to four different subsystem blocks: the standard 4-bit Flash ADC, the flash corrector, the modified successive approximation logic, and the control block. The output signal path begins at the Flash ADC (where the first four bits of the digital signal are found) before moving to the corrector (where most Flash errors are fixed), and then it passes to the Successive Approximation logic (where the rest of the signal is calculated), with the analog signal present each step of the way as the digital signal representation is assembled.

The digital output (Out) is accompanied by a metadata output (Meta) to deliver information about the conversion length and where the ADC is in the conversion cycle.

3.2 Optimal Precisions for Different Analog Slew Rates

The implemented design uses four modes of precision. Each one appends a so-

called insignificant bit (binary 1) to the output to provide proper rounding (since by nature the system rounds down – see Section 6.4.3 for more details).

- **Mode 0**: 4 Flash bits + 11 SA bits + insignificant bit (16 bits, 12 clock cycles)
- **Mode 1**: 4 Flash bits + 7 SA bits + insignificant bit (12 bits, 8 clock cycles)
- **Mode 2**: 4 Flash bits + 3 SA bits + insignificant bit (8 bits, 4 clock cycles)
- **Mode 3**: 4 Flash bits + insignificant bit (5 bits, 1 clock cycle)

To get objective accuracy results for mode switching calibration, each precision mode was tested with sine waves of different frequencies, and the real-time mean square output error was measured as a function of peak slew rate in the sinusoidal signal.

Below in Figure 3.2 is the graph of mean-square error (MSE) for conversion mode 0.

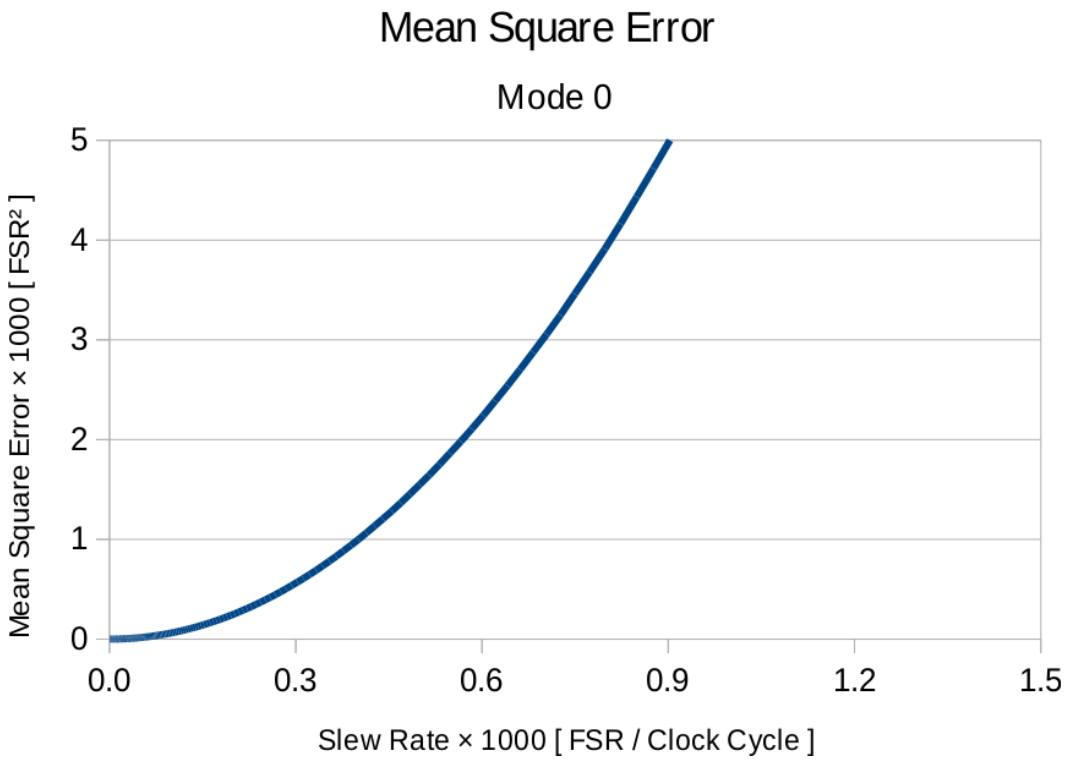

Figure 3.2 – MSE of Tandem ADC in Mode 0

This mode has 15 significant bits, providing 32,768 possible values. The conversion time is 12 clock cycles, but the latency for a step function occurring at any given point in time ranges from 12 to 24 clock cycles, as it takes 12 clock cycles for the previous conversion to finish before the next one starts.

This mode works well for dealing with very small error signals when there's already plenty of oversampling (i.e. when speed is not the bottleneck for overall accuracy).

Below in Figure 3.3 is the MSE for Mode 1.

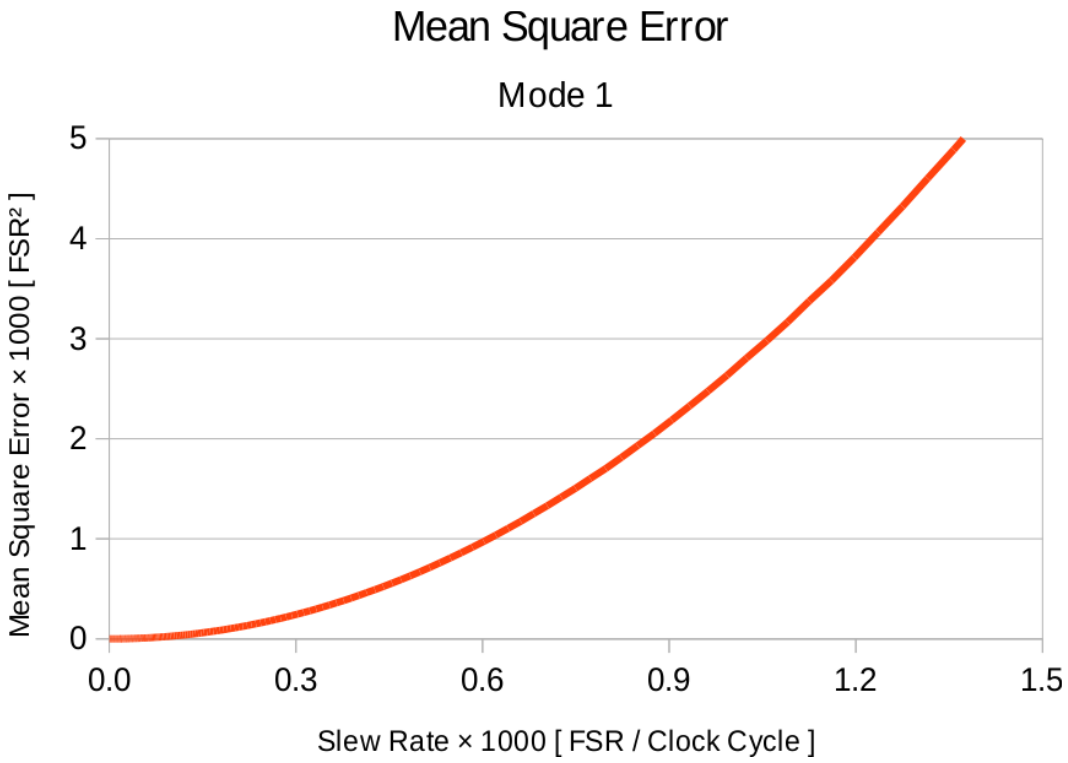

Figure 3.3 – MSE of Tandem ADC in Mode 1

This mode only has 11 bits, giving 2048 possible output values. Quantization error is therefore larger, but at higher frequencies (and higher slew rates, given constant amplitude) the MSE is noticeably lower than when in Mode 0. (See Figure 3.6 for a

comparison). This is because the conversion time is reduced from 12 to 8 clock cycles.

A similar effect happens again when comparing Mode 1 with Mode 2, the MSE of which is shown below in Figure 3.4.

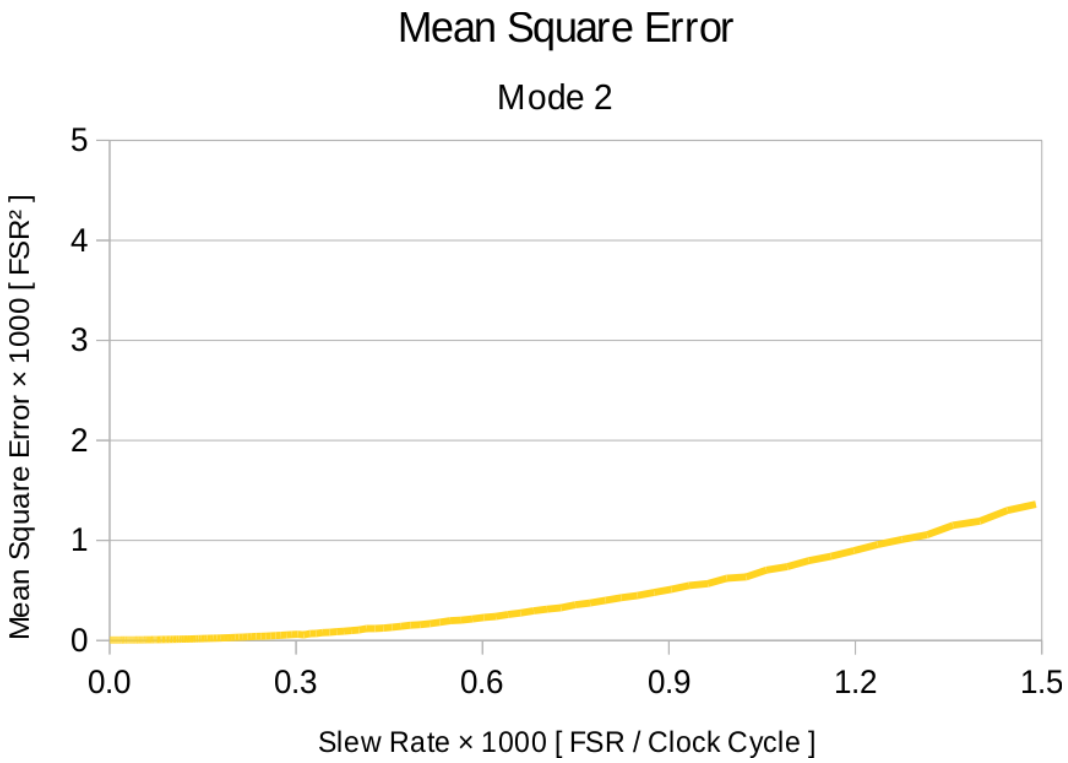

Figure 3.4 – MSE of Tandem ADC in Mode 2

This mode provides a meager 7 significant bits, yielding only 128 possible output values. However conversion time is only 4 clock cycles, so it becomes very useful when there are large slew rates. In general, this mode provides a low error for a wide range of input frequencies and amplitudes.

Finally, there is Mode 3. Its MSE is more-or-less constant, as can be seen in Figure 3.5.

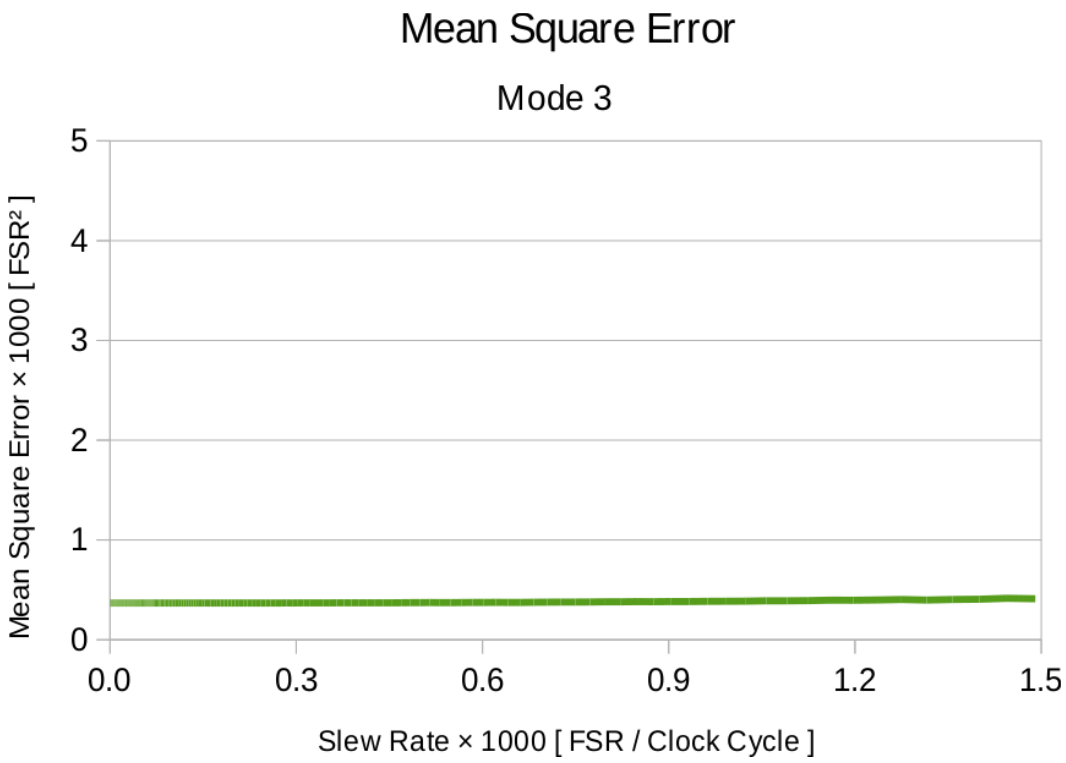

Figure 3.5 – MSE of Tandem ADC in Mode 3

This mode provides only 16 possible values, so it has significant quantization error. On the upside, it provides a new value each clock cycle, meaning it works well with high-frequency inputs and sudden transients. In general it is inferior to the other configurations unless the input signal is of such a high slew rate that the others "can't keep up".

These four error graphs are shown together in Figure 3.6.

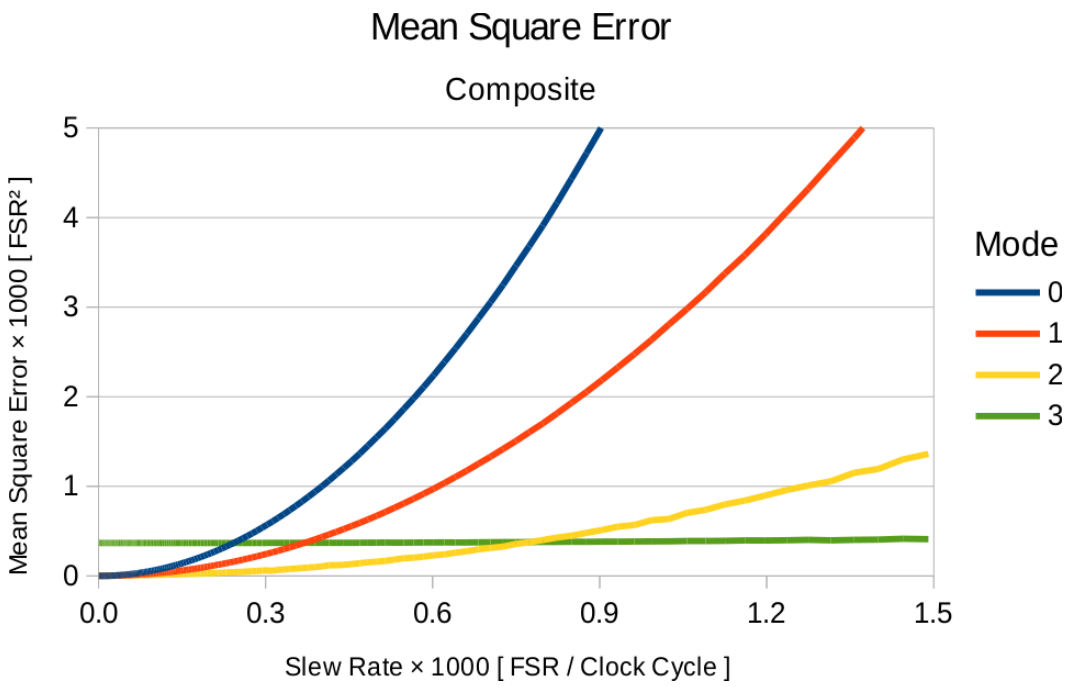

Figure 3.6 – MSE composite for all modes

While the above graph does show the four modes together it makes comparisons difficult, since most of the trade-off points are at lower slew rates and small MSE values. The comparison can be seen much better with a log-log plot, as shown in Figure 3.7.

Figure 3.7 – log-log plot of Figure 3.6

In this log plot, the behavior of the four modes can be seen more clearly. Each has a section where quantization error dominates, and the MSE remains relatively constant as slew rate changes; and another operating region where error from delay dominates, and MSE increases at the square of the slew rate – by 20dB per decade.

Using this data, the optimal points for mode switching can be determined, based on the intersections of the MSE log plots. For any given slew rate, the Mode with the lowest MSE is selected.

Chapter 4
Example Usage
Modified PID Controller

The proposed ADC is best utilized when it is connected to a device which understands and is programmed to work with a variable sampling rate. One possible implementation would make use of a modified PID controller, such as one with the following transfer function:

$$H(z) = \alpha[\tau] + \text{clamp}\left[\beta[\tau] \cdot \left(\frac{1}{z-1}\right)\right] + \gamma[\tau] \cdot (1 - z^{-1})$$

Here α is the proportional gain, β is the integral gain, γ is the derivative gain, and τ is the sampling time in number of clock cycles. (clamp[] is a clamping function.) Each gain coefficient is an indexed function of τ, making this a relatively simple multimodal system.

4.1 General Behavior

$\beta[\tau]$ can be set to 0 for smaller values of τ, rendering the system a PD controller when in a lower-precision mode. For higher precision modes, $\beta[\tau]$ can vary proportionally with τ (yielding constant continuous-time integral gain), or it can exhibit more complicated behavior.

The simplest configuration for the other two Greek-letter functions is one with derivative gain $\gamma(\tau)$ having an inverse relationship to τ, and with constant proportional gain $\alpha(\tau)$. This will yield constant continuous-time proportional and derivative gain, and in general emulate a constant-rate single-mode PD/PID controller. In addition, the multimodal system also gives freedom to increase effective proportional and derivative

gain (i.e. α and the γ/τ ratio) when there is a short sampling time, allowing the system itself to trade precision for speed along with the attached ADC.

A more thorough adaptation would have the controller make use of simpler logic operations when τ is a small value, as sampling time and word length are both shorter. One possibility is to restrict word length on gain values, speeding up multiplication (though this may require specialized hardware). In addition, if $\beta[\tau] = 0$ for small values of τ, the associated integral calculations can be skipped entirely.

4.2 Specific Implementation

Suppose the PID works with three different sampling rates: 4, 8, and 12 clock cycles per sample. The way the ADC is currently designed, these will allow for 8-bit, 12-bit, and 16-bit inputs, respectively.

System is programmed with the following values:

$\alpha[\,4, 8, 12\,] = [\,\alpha_0, \alpha_0, \alpha_0\,]$

$\beta[\,4, 8, 12\,] = [\,0, 0, \beta_0\,]$

$\gamma[\,4, 8, 12\,] = [\,3\gamma_0, 2\gamma_0, \gamma_0\,]$

To save time, all values are pre-calculated and indexed.

Mode 0: $\tau = 12$

$$H(z) = \alpha[12] + \text{clamp}\left[\beta[12]\cdot\left(\frac{1}{z-1}\right)\right] + \gamma[12]\cdot(1-z^{-1})$$

becomes

$$H(z) = \alpha_0 + \text{clamp}\left[\beta_0\cdot\left(\frac{1}{z-1}\right)\right] + \gamma_0\cdot(1-z^{-1})$$

which can be described in the following difference equations:

$$y[n]=y_p[n]+y_i[n]+y_d[n]$$

$$y_p[n]+y_d[n]=(\alpha_0+3\gamma_0)x[n]-3\gamma_0 x[n-1]$$

$$y_i[n]=\text{clamp}\left[\beta_0 x[n]+y_i[n-1]\right]$$

This is a full-fledged PID control system.

Mode 1: $\tau = 8$

$$H(z)=\alpha[8]+\text{clamp}\left[\beta[8]\cdot\left(\frac{1}{z-1}\right)\right]+\gamma[8]\cdot(1-z^{-1})$$

becomes

$$H(z)=\alpha_0+2\gamma_0\cdot(1-z^{-1})$$

which has the difference equation

$$y[n]=(\alpha_0+2\gamma_0)x[n]-2\gamma_0 x[n-1]$$

Since $\beta[8] = 0$, the integrator has been done away with. This simplifies the difference equation, makes it FIR, and removing the clamp function makes it linear. If $\alpha_0+2\gamma_0$ is pre-calculated, each sampling cycle requires only two multiplications and one addition to be performed.

Mode 2: $\tau = 4$

$$H(z)=\alpha[4]+\text{clamp}\left[\beta[4]\cdot\left(\frac{1}{z-1}\right)\right]+\gamma[4]\cdot(1-z^{-1})$$

becomes

$$H(z) = \alpha_0 + 3\gamma_0 \cdot (1 - z^{-1})$$

which, if rounded, has the difference equation

$$y[n] = (\text{round}[\alpha_0 + 3\gamma_0])\,x[n] - \text{round}[3\gamma_0]\,x[n-1]$$

This is similar to Mode 1, the main difference being the rounding done on the coefficients. This can be best utilized by 8-bit multipliers.

4.3 Time-Domain Analysis

For the above-described system, the output y[n] is only a function of x[n], x[n-1], and y[n-1]. The integrator is only for settling steady-state errors, so with regards to dealing with transients it can be ignored, and the remainder of the system relies only on x[n] and x[n-1]. This means there will never be a delay of any sort longer than one sampling cycle.

For this entire system, the total delay from analog input(s) to processed (post-PID) digital output is directly proportional to the sampling time, so this system is especially sensitive to ADC speed. (Total delay is actually proportional to the sampling time plus the ADC delay, but for most ADCs—especially those suited to control systems, including the Tandem ADC—these are very similar, if not one and the same.) Even for other systems—say, if there were a PID system which had identical continuous-time behavior regardless of sampling speed—the ADC delay forms a large part of the total system delay, and so bringing this delay down is still beneficial. For systems with longer filter lengths this is not so much the case. (See Section 2.2 for more details.)

Chapter 5
Idiosyncrasies of Combining Flash and Successive Approximation ADCs

Flash and Successive Approximation ADCs are both well-suited to perform their usual tasks, but substantial adaption or modification is often needed for them to work optimally together as part of a larger system.

5.1 The Open-Loop Nature of Flash

Flash ADCs are purely static (i.e. memoryless) systems. They function as open-loop devices and have no means of error detection or correction. As a result off-by-one errors can be caused by resistors in the resistor chain being off from their nominal values, since this changes the points at which the output will switch from one digital value to another. This limits pure Flash ADCs to around 8 bits of precision.

A 4-bit Flash will have errors much less frequently than an 8-bit Flash, but they will still happen. When using a 4-bit Flash ADC by itself this would not present much of a problem, but in an architecture like that of the Tandem ADC, every off-by-one Flash error ruins the precision for that conversion cycle. The Successive Application logic will not be able to function properly, outputting either all ones or all zeros in an attempt to correct for the Flash error.

For Flash ADCs there is also the issue of sparkle codes. Occasionally a bit in the thermometer coding will be flipped, causing an erroneous sequence. This usually happens due to a comparator timing mismatch, and is much less likely to happen if there is a zero-order hold on the input (which there is in the proposed ADC). If they happen, sparkle codes can cause large conversion errors.[9]

To address these issues, an error corrector is needed. The error corrector can be static in nature as well (i.e. it doesn't need a clock) but should cause the ADC+corrector system to have some sort of closed-loop behavior. For the purposes of speed and simplicity, and because 4-bit Flash ADCs are rarely off by more than one, an off-by-one corrector is used in the Tandem ADC design. The logic is made as fast as possible so that the entire process (flash conversion and correction) can happen in one clock cycle. This system does not address large sparkle code errors, as correction circuitry would slow the system down disproportionately.

5.2 Controlling the Successive Approximation Logic

For the Flash converter, a standard off-the-shelf Flash ADC can be used; it just needs an off-by-one error corrector attached to its output signal. The same is not the case for the Successive Approximation logic – the iterative SA loop needs to be "taken apart" and modified. One change is that the first four bits of the Successive Approximation register are supplied directly from the Flash (after error correction). The other significant change is that the Successive Approximation logic must have programmable precision (i.e. it needs to know when to stop each cycle, depending on the mode). This can be accomplished by giving the Successive Approximation logic and registers externally triggerable start and reset controls.

5.3 Making the Two ADCs Work in Tandem

In addition to the issues discussed in the previous two sections (5.1 and 5.2), the Flash and Successive Approximation subsystems must be properly synchronized with

each other, and must finish the conversion reliably and on-time. This requires external control and signal handling logic, which is carried out by a separate control block within the system.

The system's control block handles the determination of which precision mode to use, and communication with the recipient of the digital signal about this selection. The control block also effects the zero-order hold, commands the Successive Approximation block to start and stop its conversion process at the proper times, and routs the digital signal to the output with an insignificant bit appended. The Flash ADC and its corrector requires minimal intervention from the control block; the zero-order hold on its input is the extent of their interaction.

5.3.1 Resulting Signal Path

From beginning to end, the signal path travels through the following blocks:

- Input Handling
- Flash ADC
- Flash Corrector
- Successive Approximation Logic
- Output Handling (Control Block)

The Input Handling block conditions the input (antialiasing and a zero-order hold) for the rest of the system. The next three blocks each work to put together the digital output signal, using the analog input (after input handling) for reference each step of the way. Finally, the control block appends an insignificant bit and holds the digital output until the next conversion cycle is complete.

Chapter 6
Design – Details and Practical Implementations

As the previous chapter shows, significant handling and modification is required for Flash and Successive Approximation systems to be used in tandem. This consists primarily of a corrector for the Flash ADC, and modifying the typical Successive Approximation architecture to work with external controls and predetermined bits.

These systems are described below, along with the other blocks in the Tandem ADC. (See Figure 3.1 for reference.)

6.1 Anti-Aliasing Filter

Despite having multiple sampling rates, the Tandem ADC can be protected from aliasing with a single anti-aliasing filter which attenuates frequencies above the Nyquist rate of the fastest mode. (See Section 2.1 for details.) Desired qualities in the filter are a flat passband, constant, low group delay, and decent (>30 dB) stopband attenuation. 4^{Th}-order and 6^{th}-order Butterworth and Bessel filters were analyzed, and the results are shown in Figure 6.1 and Figure 6.2 below:

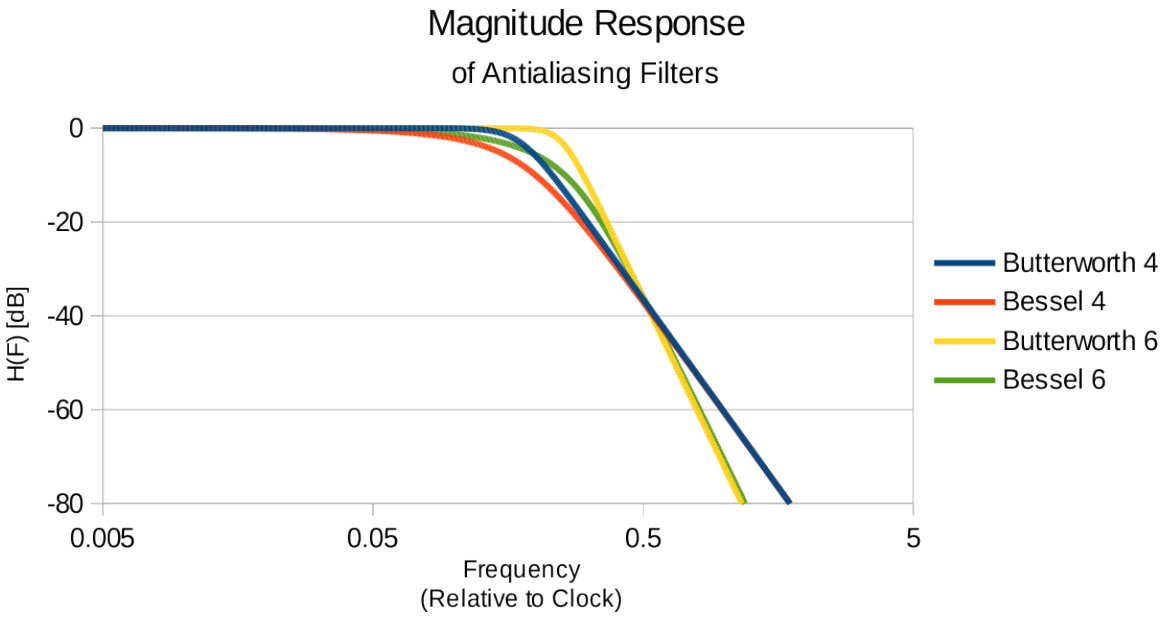

Figure 6.1 – Behavior of Anti-aliasing Filters (Magnitude Response)

The four filters had their nominal cutoff frequencies set so that they all have the same attenuation (36dB) at the Nyquist frequency (0.5×Clock Freq). As shown in the figure above, the 6th-order Butterworth filter has the best magnitude response: it stays near 0dB for longer than the other three, and then drops down at 120dB/decade. Group delay is shown below:

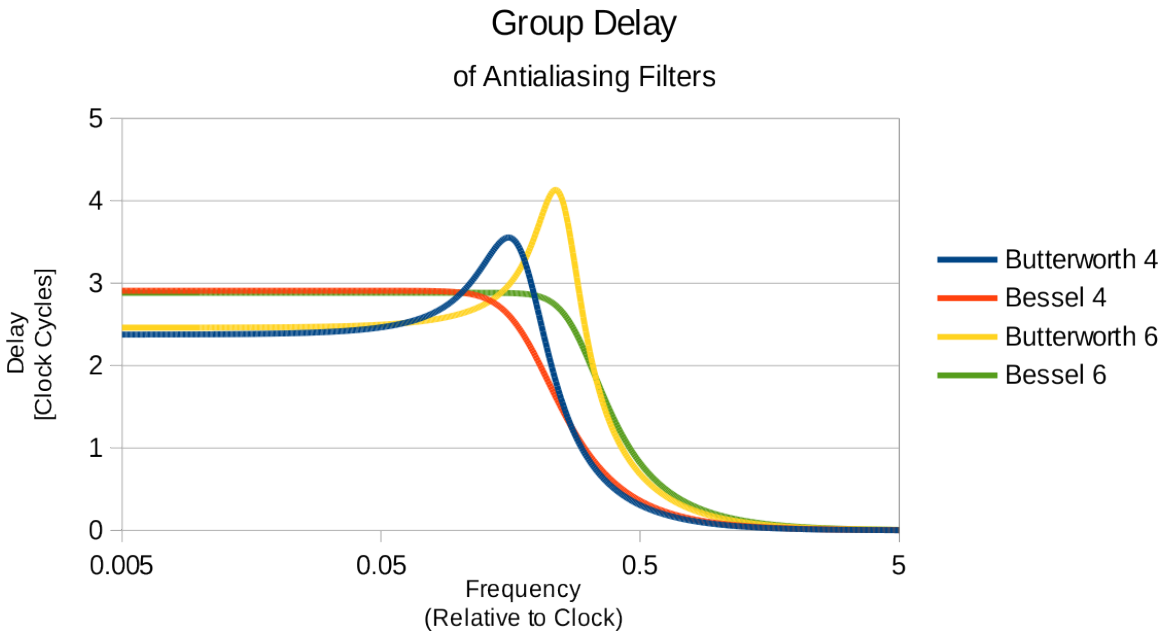

Figure 6.2 – Behavior of Anti-aliasing Filters (Group Delay)

For any given order and cutoff frequency, a Bessel filter will have a smaller group delay than a Butterworth filter.[10] (It also has a nicer step response—faster and less overshoot.[11]) However, since Bessel filters have such a gradual roll-off, their nominal cutoff frequency must be lowered in order for them to have the same asymptotic behavior as a Butterworth filter of the same order. This raises group delay proportionally, and the results can be seen in Figure 6.2 above: Bessel filters have *higher* group delay than Butterworth filters, for a given order and logarithmic attenuation asymptote. A similar effect happens for the step response, which slows down proportionally, resulting in a longer rise time.

Something similar happens when filter order is increased – group delay increases, but nominal cutoff frequency can also be increased if the goal is to have a certain level of attenuation at a given frequency, as is the case here. This in turn decreases

group delay, and the two effects roughly cancel each other out, yielding approximately the same group delay as before.

As a result of these two effects, the 6th-order Butterworth filter has good group delay for most of the passband (narrowly behind the 4th-order Butterworth). The only issue is its rather large delay spike in the transition band, but that is of little importance since very few signals will actually approach the Nyquist frequency. The 6th-order Butterworth filter, therefore, is the best anti-aliasing filter out of the four examined.

6.2 Flash Corrector

The Flash corrector fixes the vast majority of Flash ADC errors, and effects a hold on the output. It compares the analog input with the digital Flash output, and with an incremented version of that Flash output. Its block diagram is shown below in Figure 6.3.

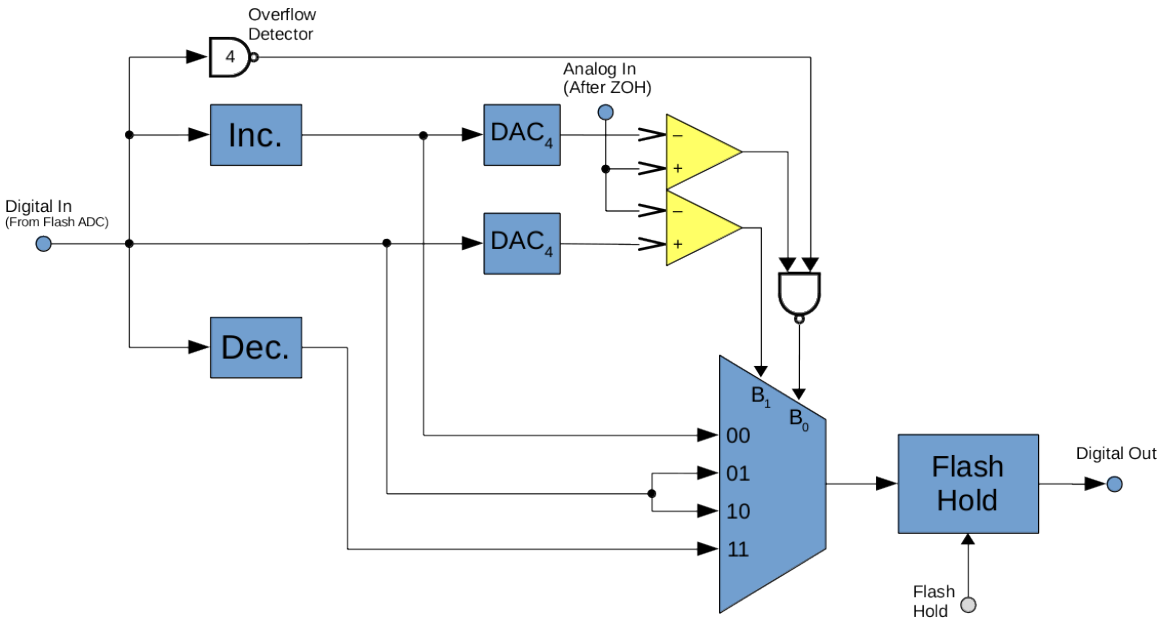

Figure 6.3 – Flash Corrector
(L2 Block Diagram)

The Flash output should (after conversion back to analog) have a lower voltage

than the analog input, while the incremented output should have a higher voltage. If this is not the case the output of the Flash is incorrect, and the Successive Approximation block will not be able to function properly. The SA's iterative search relies on the already-determined output bits producing an equivalent signal that does not exceed the analog input, and that is within twice the bit value of the next conversion bit to be determined. Otherwise it will either a string of either zeros or ones, depending on whether the Flash output is too large or small respectively.

The Flash output, along with an incremented version and a decremented version, are sent to a multiplexer. The correct signal out of these three will be chosen based on how the unchanged and incremented signals relate to the analog input. (The input selected by the multiplexer is the one labeled, in binary, with the values of selecting inputs B_1 and B_0.) Under nominal conditions, the unchanged Flash signal is selected. If the unchanged Flash output is larger than the analog input, the decremented signal is selected. If the incremented Flash output is smaller than the analog input, (with the exception of incrementor overflow) the incremented signal is selected.

Once the correct signal is found, the output is kept constant with the Flash Hold block. This is not strictly necessary, since the rest of the circuitry is static and all inputs are externally held constant, but it helps prevent certain problems. One potential problem is the Flash ADC spontaneously having an off-by-one error, which is fixed by the Flash Corrector in a nonzero amount of time. Without the Flash Hold, this would cause an off-by-one transient in the output of the Flash System. (See Section 8.1 for more details.)

6.2.1 Dealing with overflow / underflow

Overflow and underflow conditions must also be handled in the Flash Corrector. The incrementor itself has no overflow protection; if an overflow condition is detected, the multiplexer is instructed to not use the incremented signal (which has looped around to zero). On the other hand, the decrementor does have underflow protection, and so no change to the multiplexer is needed in underflow cases.

This difference in the handling of overflow and underflow is done in order to make the critical path for signal propagation as short as possible. The incrementor is on the critical path, and modifying multiplexer input logic adds less to that critical path than "normal" overflow protection circuitry. The decrementor on the other hand is not on the critical path, and modifying multiplexer logic to deal with underflow is significantly more difficult than for overflow.

The incrementor layout is shown in Figure 6.4 below. The input bits are labeled $A_{0\text{-}3}$, and the output bits are $B_{0\text{-}3}$. The bits are arranged with the LSB on top and the MSB on bottom.

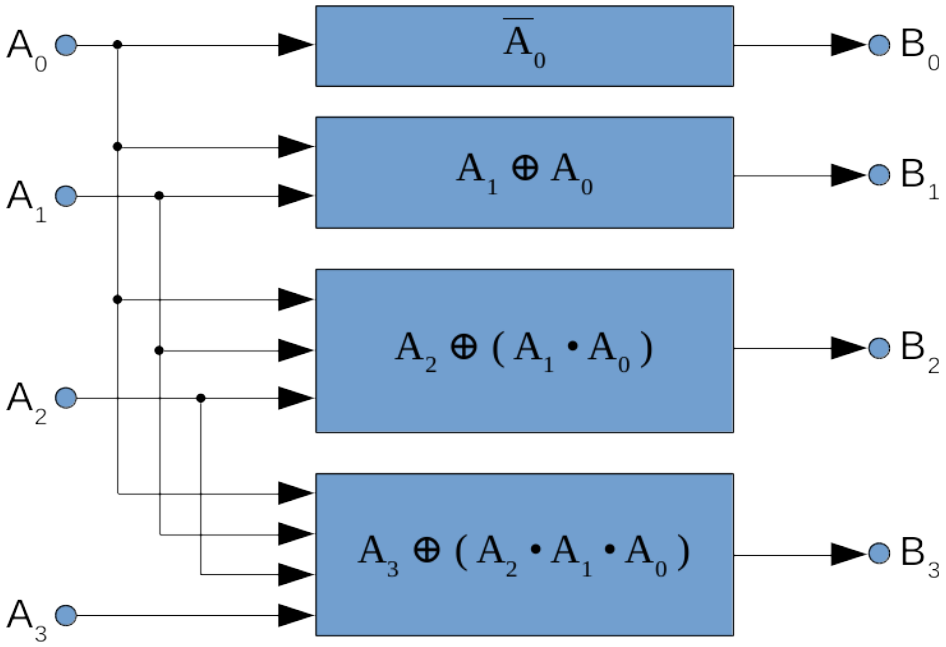

Figure 6.4 – 4-bit incrementor, no overflow protection
(L3 Block Diagram)

Incrementor logic is purely combinational, and as direct as possible, and so it runs much faster than a standard adder. Its critical path is only three gates long, although its fan-in is rather large (around 7 gates for each input bit). Gate-level architecture is described in Appendix A.1.

The decrementor makes direct use of incrementor logic as well. Because of its underflow protection, the decrementor is double-inverting – it inverts the signal, increments it, and corrects for underflow with four NOR gates, inverting the signal again in the process. This is shown below in Figure 6.5.

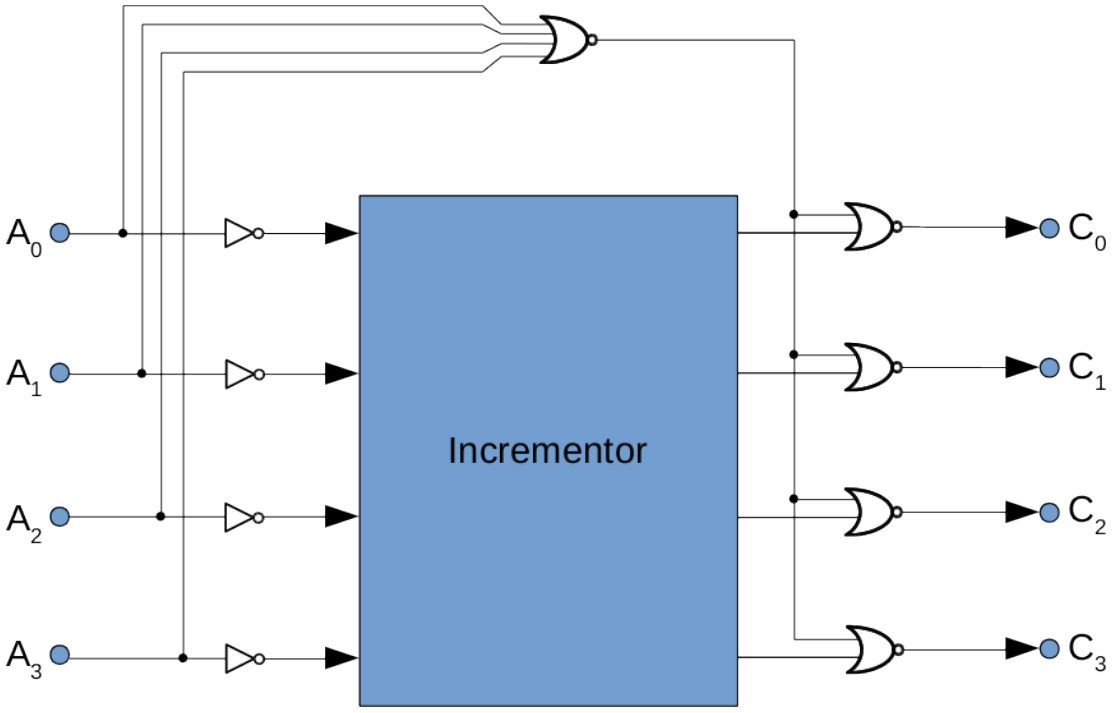

Figure 6.5 – Incrementor adapted to an underflow-protected decrementor

If the four inputs ($A_{0\text{-}3}$) are all zero, the attached NOR gate (top of diagram) goes high, sending the other four NOR gates low. This causes a digital output of zero when the input is zero, overriding the decrementation that underflows and causes an output value of 15 (binary `1111`). The structure also causes the inputs and outputs of the internal incrementor to be inverted, which is what makes it function as a decrementor.

6.3 Successive Approximation Block

The Successive Approximation block handles the signal after the Flash ADC has found the first four bits, and any Flash errors have been corrected. It works much like a normal Successive Approximation ADC, except for two key differences:

- The first four bits are taken from an external source. The Successive Approximation block starts on the fifth bit, and must handle this transition

smoothly.

- The precision level is externally controlled. In addition to a Mode signal telling the Successive Approximation block how far to go, explicit start and reset commands are given externally.

A block diagram for the modified Successive Approximation block is shown below in Figure 6.6. (See Figure 1.2 for an unmodified version.)

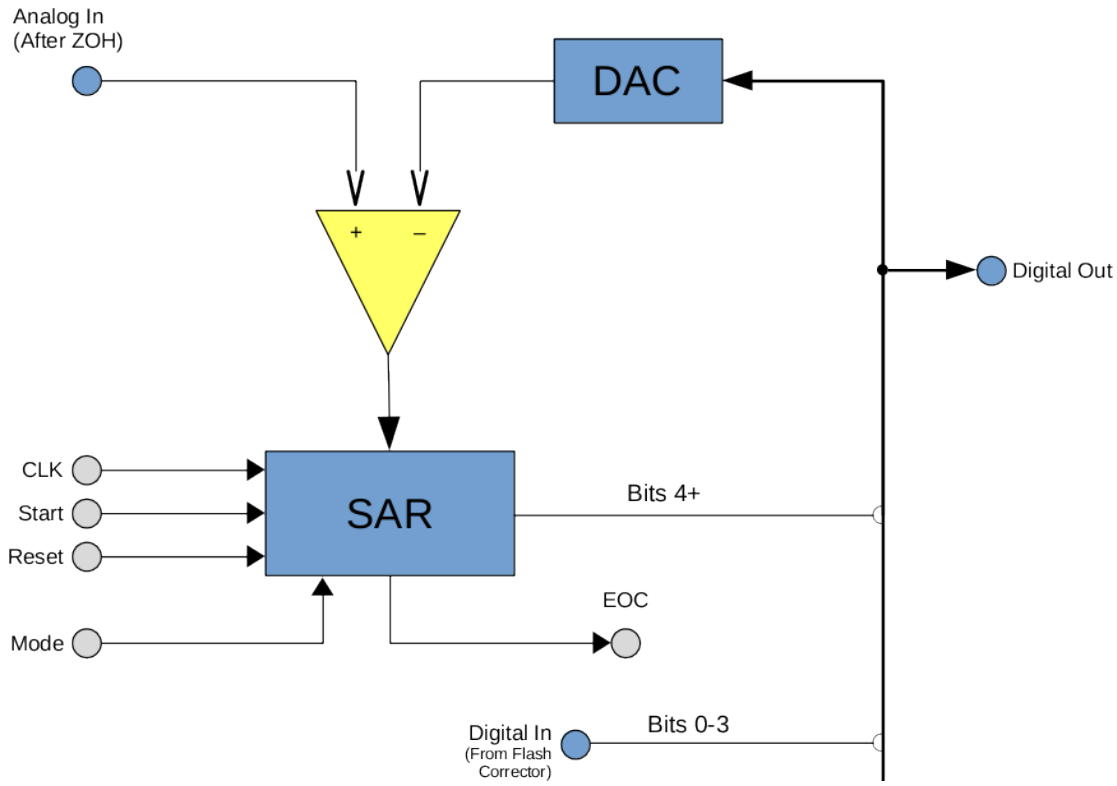

Figure 6.6 – Modified Successive Approximation Logic (L2 Block Diagram)

As shown in the above figure, the digital output is a concatenation of the Flash ADC (after correction) with the bits stored in the Successive Approximation Register (SAR). With this arrangement, each bit is found in the same general manner as with a standard Successive Approximation ADC. The “Mode” input does not directly control

this block's conversion; it only serves to determine when the EOC signal should be sent out, which in turn causes the control block to send a reset signal, followed by a start signal.

With the above setup, one bit of the output is calculated for each clock cycle.

The layout of the internal Successive Approximation Register (SAR block in Figure 6.7) is shown below:

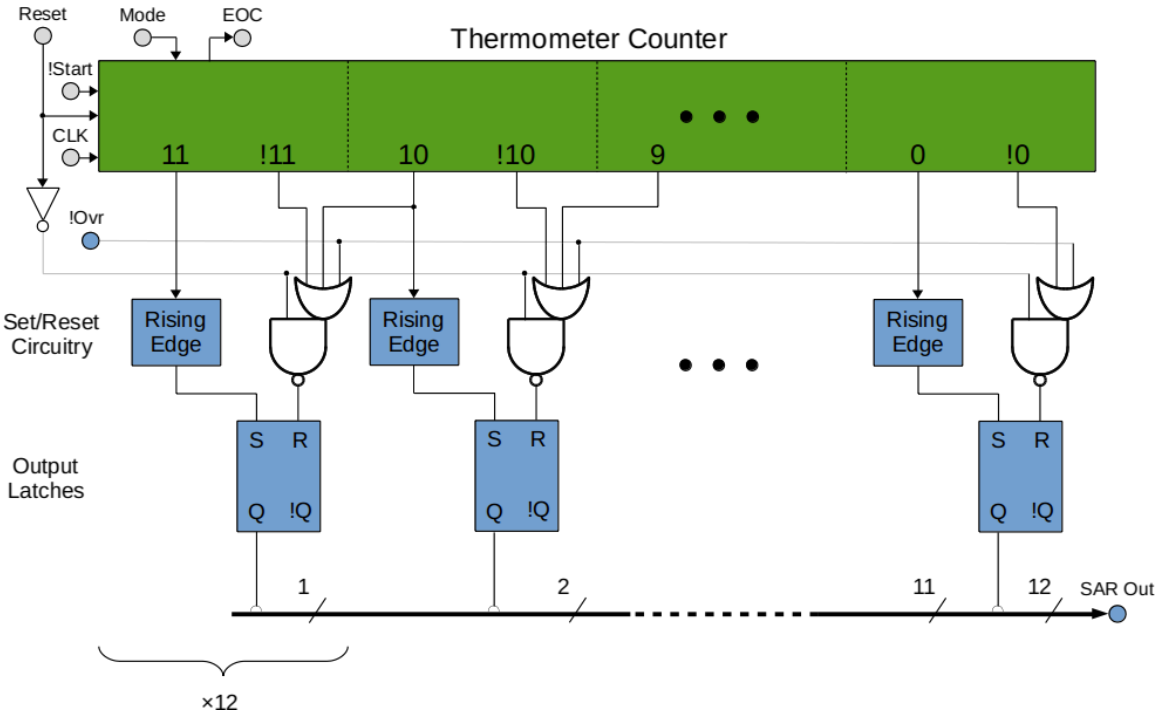

Figure 6.7 – Successive Approximation Register (L3 Block Diagram)

The above diagram details what is primarily a standard Successive Approximation register. The primary difference is the external control – the Start, Reset, and Mode inputs. The EOC output by itself does not have any internal effect on the system; the system must be reset externally. The "!Reset" and "!Ovr" wires are grayed out to reduce clutter.

"Off-cycle" outputs (which go high on the falling clock edge) are available from the counter as well, but are not shown nor heavily used. Their main purpose – aside from propagating the counter along – is to trigger EOC, which goes high on the falling edge while the last bit is being found. (See Section 7.2.1 for details.)

Below the counter, each register bit makes use of a rising edge detector to set its SR latch high when the bit is selected by the counter. An OAI (or-and-invert) gate is then used to reset the latch low if the following conditions are met:

- The output of the Successive Approximation block (taken from the SAR, concatenated with the corrected Flash output) is, when converted to analog, greater than the analog input (after Zero-Order Hold).
- The thermometer-coded counter has a high output for the register bit in question.
- The thermometer-coded counter has a low output for the next register bit.
- All three of the above at once, OR a reset signal is given.

This combination causes the output of the SR latch that was just set high to go back down if it creates an analog value higher than that of the input, while leaving the other latches alone. In addition, a blip on the Reset input will cause all SR latches to be reset, and start the thermometer counter over. (Setting/resetting is discussed in more detail in Section 5.2.)

6.4 Control Block

The control block has three primary functions: it determines the mode for conversion resolution, synchronizes the rest of the system, and handles the output. Each function corresponds to a different sub-block, as shown below in Figure 6.8:

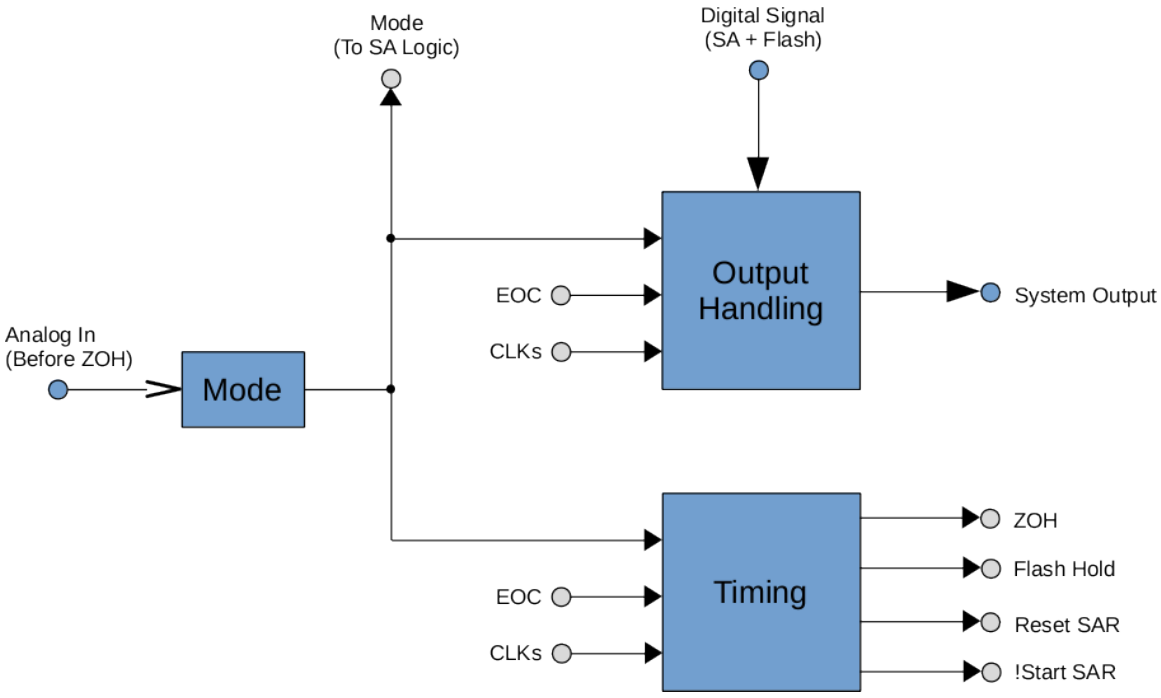

Figure 6.8 – Control Block
(L2 Block Diagram)

The Timing block uses a counter along with other internal circuitry to control the rest of the system. The counter does not run through the entire conversion cycle; its job is complete when the !Start signal is sent to the Successive Approximation block, and the process starts over once it gets an EOC signal back.

The Output Handling block performs a hold on the digital output of the system (Flash and Successive Approximation together). It also appends an insignificant bit to provide proper biasing, and zeroes out all subsequent bits. (Said bits should be zero anyway, but this corrects for any issues if the Successive Approximation block does not stop when it should.)

The Mode block uses a differentiator (approximated by a highpass filter) to measure the rate of change of the input (before ZOH) and determine which mode would be optimal. This mode signal is used both internally by the control block, and sent along

to the Successive Approximation logic. (See Section 7.3 for details on Control block time domain operation.)

6.4.1 Mode Block

The Mode block determines which precision mode is best to use for the input signal, based on input slew rate. Mode number can be increased (precision can be decreased) instantly, and it falls back down after 256 conversion cycles.

The mode is thermometer-coded by three slew rate detectors, as shown below in Figure 6.9.

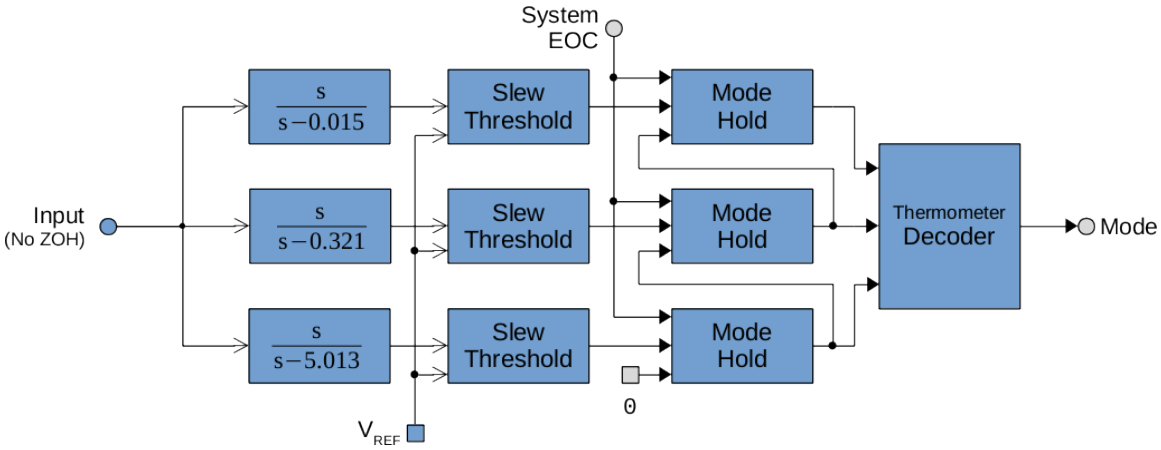

Figure 6.9 – Mode Block
(L3 Block Diagram)

Each detector consists of a analog filter, a threshold detector, and a digital hold to make the mode output drop down gradually rather than jittering between levels. This is implemented because most signals do not have a consistently high slew rate (triangle waves are the main exception), and as a result the maximum slew rate over a finite period of time is a better mode determinant of the required conversion speed than the instantaneous slew rate.

The analog filter and slew threshold detector are shown below in Figure 6.10. The resistor and capacitor values are set to meet the s-domain transfer functions shown in Figure 6.9 above. Note that for those transfer functions the frequency unit is radians per clock cycle, *not* the usual radians per second.

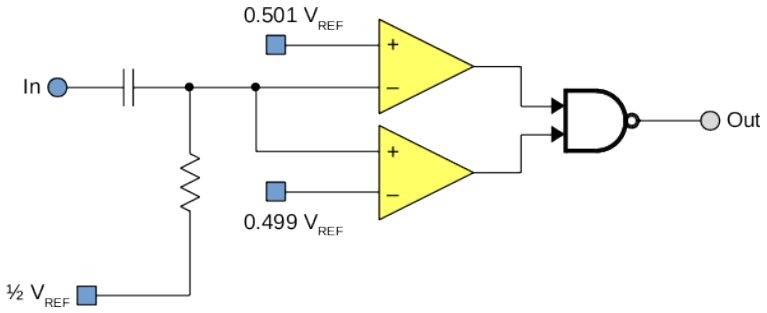

Figure 6.10 – Slew Threshold Detector with analog HPF
(L4 Block Diagram)

The output is then sent to a mode hold, and then to a thermometer decoder. The Mode Hold block is shown below in Figure 6.11.

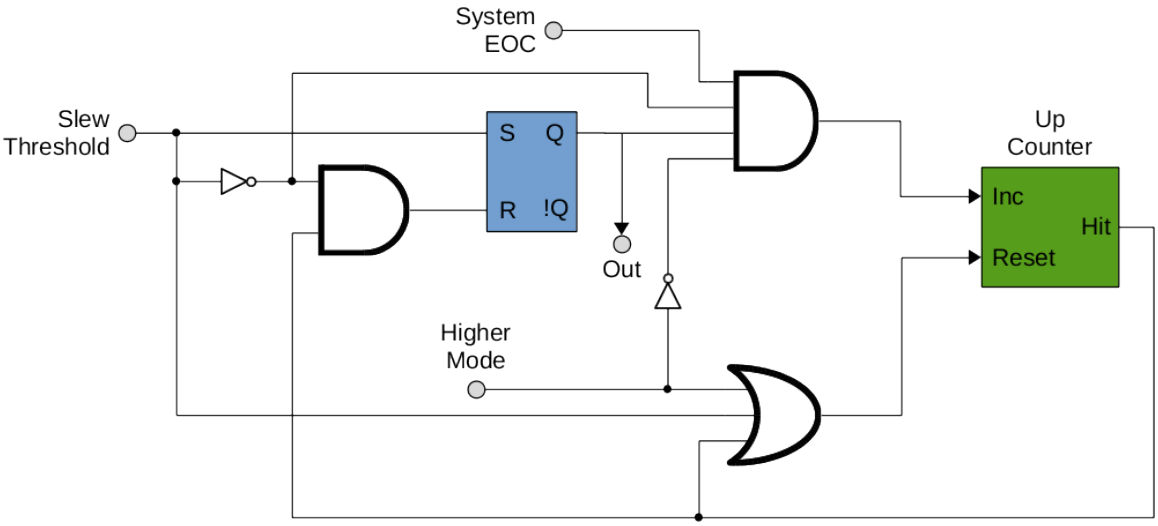

Figure 6.11 – Mode Hold
(L4 Block Diagram)

The Mode Hold block keeps the Mode output from dropping immediately whenever the slew rate drops below the threshold. Whenever the mode signal increases in

value, one of the three Slew Threshold bits goes high. (Thermometer encoding, as shown on Figure 6.9.) This sets the attached SR latch, which is only reset once the Up Counter sends a signal from its Hit output. This happens after 256 conversion cycles, as counted by System EOC. The count only happens for the current mode (otherwise it is reset and suppressed by Higher Mode), and it is reset every time the Slew Threshold goes high. (For more details see Section 7.3.1.)

The Q output of the SR latch is the output of this block; three of these feed the Thermometer Decoder as shown in Figure 6.9. The Mode Hold block has *not* been optimized for propagation speed, as some delay is actually desirable. (This is to keep the Bit Appendor from changing prematurely; see Section 8.3 for details.)

6.4.2 Timing

The Timing sub-block is centered around a thermometer-coded counter, similar to the one used in the Successive Approximation Register. (See Figure Figure 6.7.) The largest difference is that the timer is much shorter, and its outputs are labeled the other way around (i.e. the outputs progress 0, 1, 2 rather than 11, 10, 9, etc). The timer is used to effect the Zero-Order Hold, the flash hold, and to reset and start up the Successive Approximation Register. It gets reset by the EOC input.

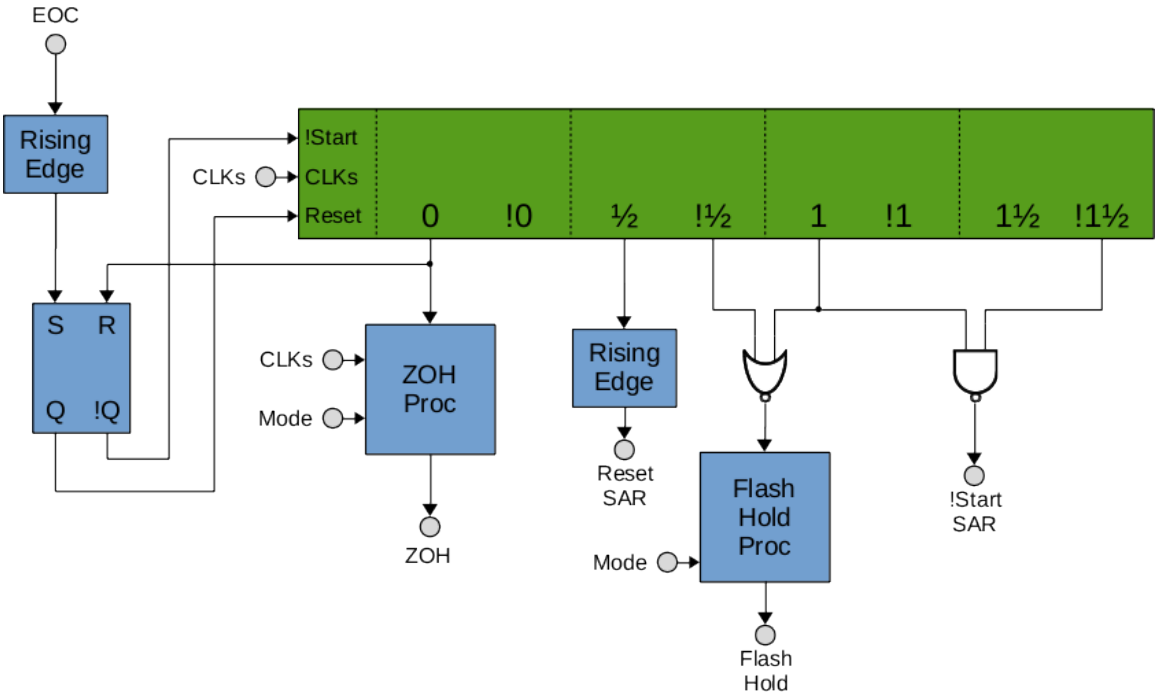

Figure 6.12 – Timing Block (L3 Block Diagram)

Figure 6.12 above shows the architecture of this block. The thermometer counter provides a compliment to each output in the same manner as the SAR counter, but also directly provides all of the "off-cycle" outputs which go high on a falling edge. (These are shown ending in ½.)

The two holds (analog input ZOH and flash hold) can have their usual behavior be overridden by the Mode input. If in Mode 3 (flash-only mode, binary 11) the ZOH happens once per clock cycle and the flash hold is "removed from the circuit" by holding the latches' C inputs high. The end result is that the entire Timing block is essentially bypassed until the Mode signal is set to something other than 3. This is explained further in Section 7.3.2.

6.4.3 Output Handling

The Output Handling sub-block has two main components: the appendor and the output hold. These are shown below in Figure 6.13.

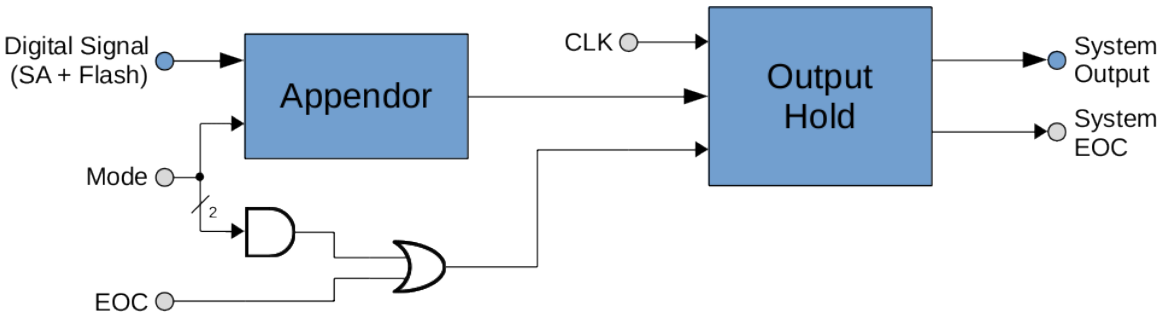

Figure 6.13 – Output Handling
(L3 Block Diagram)

The appendor's job is to attach an insignificant bit to the right point for biasing, and zero-pad up to the end afterwards. This is done using transmission gates rather than logic gates, since the signal is time-critical while the mode is usually determined well in advance. (See Section 7.3.3 for more.) The mode determines which bits are kept as-is, and which are replaced with padding. The padding itself is somewhat determined by the mode as well, since the insignificant '1' bit must be placed in the right position.

This is also the point where the Flash and Successive Approximation bits are brought together for the output. (Previously they were brought together in the Successive Approximation block, but that block doesn't pass the Flash bits along.) The Flash bits are always left untouched by this component.

Output of the Bit Appendor (`F` = Flash, `S` = Successive Approximation):

Mode 0: `FFFFSSSSSSSSSSS1`
Mode 1: `FFFFSSSSSSS10000`
Mode 2: `FFFFSSS100000000`
Mode 3: `FFFF100000000000`

The output hold's job is to hold the output steady once it has been calculated, and keep it in place for the entirety of the following conversion cycle, only to update once the next cycle's output has been fully calculated. Normally the holding is controlled by both the EOC signal and the clock—the hold is effected on the clock rising edge when EOC is high—but in Flash-only mode (mode 3) the EOC is ignored and the corresponding input is continuously held high; this causes the output to update every clock cycle. This bypass is effected by the two logic gates shown in Figure 6.13.

Gate-level architecture of the Output Hold is described in Appendix A, Section A.3.3, particularly Figures A.26 and A.27.

Chapter 7
Time Domain Behavior

Some portions of the Tandem ADC are more clock-indifferent than others (for example the Flash ADC, Flash Corrector and Input Handling have little or no direct clock reliance) but most are affected by the system's timing in some way or another. A network of control signals is used to keep everything synchronized.

7.1 Overview

Below in Figure 7.1 is the block diagram of the Tandem ADC, with an emphasis on clock-dependent blocks and their related control signals. (Compare with Figure 3.1.)

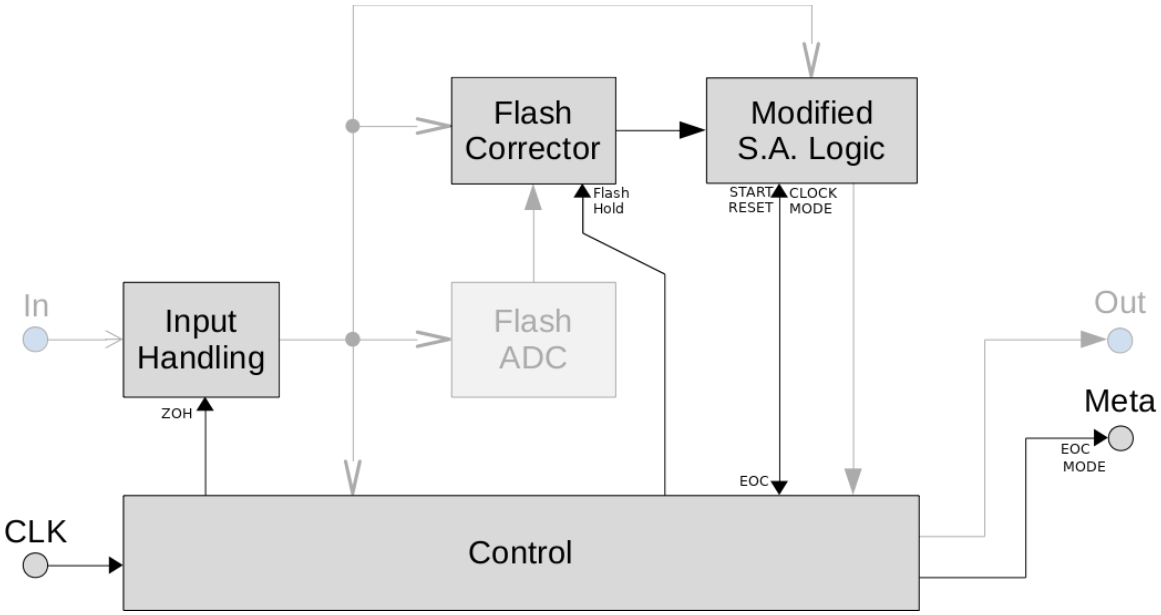

Figure 7.1 – Tandem ADC, Clock-dependent Blocks
(L1 Block Diagram)

Each block has the following clock-related behavior:

Input Handling: Contains an anti-aliasing filter and a zero-order-hold circuit. ZOH is performed at the beginning of each conversion cycle, controlled indirectly by the clock.

Flash ADC: Standard Flash ADC, not influenced by the clock aside from the ZOH imposed on the input.

Flash Corrector: Primarily combinational. Input signals are held by ZOH (directly or indirectly) and settling of output should occur well within the span of one clock cycle. There is a hold on the output once it's settled, but that is the only clock-related effect on this block.

Modified S.A. Logic: Relies on a clock in the same ways a standard Successive Approximation ADC would, although extra signals are needed for mode control and synchronization (set/reset) purposes.

Control: Receives clock signal from outside (or generates it, potentially), uses this to keep its internal control circuits (and the rest of the system) in sync.

Out & Meta: Signal is given to OUT at the end of each conversion cycle. EOC and mode information are given to the Meta output.

7.1.1 General Time-Domain Description

At the start of conversion, a zero-order hold is effected on the input signal. The held input is fed into the Flash ADC and its corrector. Within one clock cycle the output of the corrector settles, is held by a latch array, and is fed to the modified successive approximation logic (and the control block for the final output).

From that point (unless in a flash-only mode) the successive approximation block is given a "start" signal and proceeds to find proper values for the remaining bits. (The number of bits needed has been previously determined by the control block.) One bit is calculated each clock cycle.

When the successive approximation logic is finished, it sends an end-of-conversion signal to the control block, which then appends an "insignificant bit" to the end of the signal to provide proper biasing, and sends it along to the output, which is then held by a latch array until the end of the next conversion cycle. It then resets the successive approximation block to prepare it for the next conversion. (If in flash-only mode, successive approximation output is ignored and only the corrected flash signal is used.)

While this is going on, an analog slew-rate detector within the control block determines the best resolution (mode) for the next conversion cycle. When the desired resolution drops it has an immediate effect on the rest of the system, and the successive approximation logic will either stop once the new, lower resolution has been met, (rather than waiting until the old resolution has been completed) or stop immediately if said resolution has already been met or exceeded. It is also possible (with minimal modifications) for the resolution to be requested from the receiving device.

7.2 Successive Approximation Block Behavior

While most devices in the Tandem ADC have some involvement with the clock, by far the most involved is the Successive Approximation block. Figure 7.2 below shows what happens each clock cycle, with one bit after another. (Compare with Figure 6.7.)

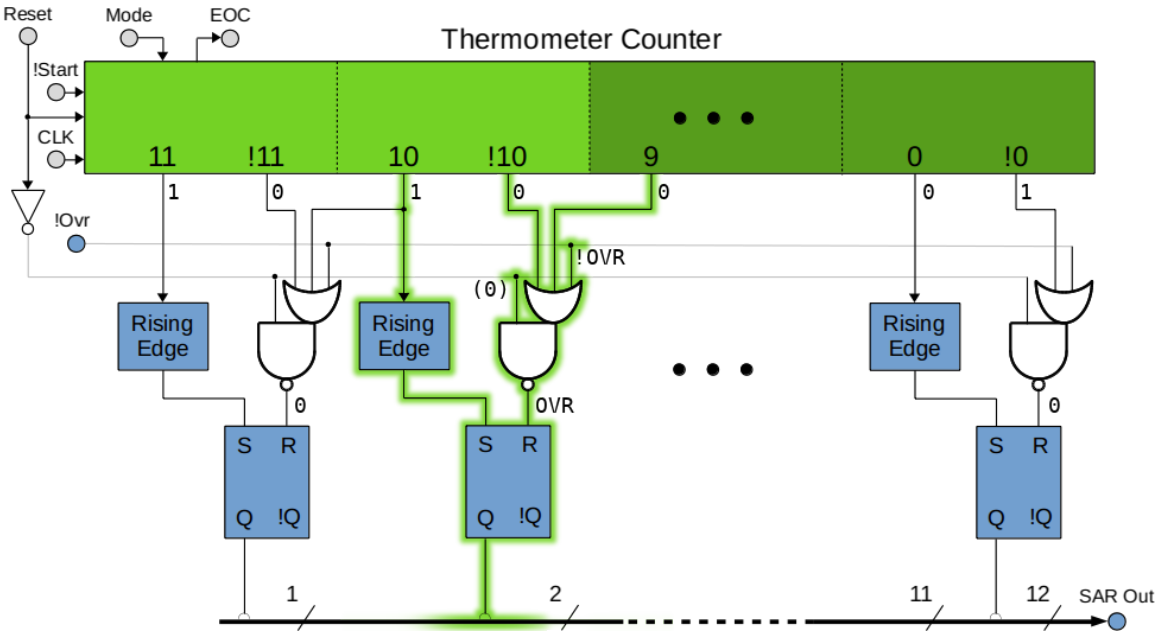

Figure 7.2 – Successive Approximation Register, Time Snapshot (L3 Block Diagram;)

The Successive Approximation block uses its thermometer counter to keep track of which bit is the "active bit", i.e. which bit is currently being operated on in the conversion process. In the above figure, the "on" blocks on the thermometer counter are a brighter green, and the circuitry for the active bit is highlighted with the same color. Below the thermometer counter are the rising edge detectors and OAI gates, which respectively set and reset the SR flip-flops.

For the above figure the active bit is Bit #10. In terms of setting the bit, the active bit is determined by the rising edge detector, which sends a "set" command when Block #10 on the Thermometer Counter turns on. In terms resetting the bit, the active bit is determined by detecting that Block #10 is on while Block #9 is off. This is done with the OAI gate, which effects a logical AND between the current block (#10) being on, the next block (#9) being off, and the "OVR" signal being high. (OVR goes high if setting

the active bit creates too large of a digital value; this functions the same as standard Successive Approximation topology except the Flash's output is involved as well.) The result is that OVR only affects the circuitry of the active bit, leaving the rest alone.

The OAI gate also effects a logical OR between the above conditions and the Reset signal. As a result, the reset signal sets all SR flip-flops low, regardless of which corresponds to the active bit. (Reset signal also resets the Thermometer Counter.)

The above is implemented with inverted inputs, keeping the path from the counter to the "R" input on the flip-flop as short as possible.

7.2.1 Starting, Resetting, EOC

The thermometer counter starts out empty (all outputs low). If !Start is set low, the first counter block (11) goes high at the first rising edge. From there the signal propagates through the counter as the bits of the signal are found. When it reaches a certain point (determined by the Mode input) it sends out an EOC signal. EOC goes high on the falling edge while the last bit is being converted; this does not cause premature termination issues (even though EOC is technically sent before the conversion is actually finished) because all parts of the system that take EOC as an input wait for a rising clock edge before doing anything that would affect the conversion process.

The Reset signal is sent in by the Timing block on the next falling edge, half a clock cycle after the conversion is finished. This gives the output hold enough time to grab the signal while still being early enough to keep Reset and Start commands from being sent to the SAR counter at the same time.

7.3 Control Block

The control block is an amalgamation of various related functions which deal primarily with the details of running the system smoothly and keeping it synchronized. The Mode block determines the system mode, the timing block handles most of the signal holds and starting/stopping, and the output handling block both holds the output in place and effects insignificant bits (both the '1' and trailing zeros).

7.3.1 Mode Hold

Most of the Mode block is clock-independent, with the exception of the Mode Hold blocks. These do not run on the clock directly, but instead are triggered by the System EOC signal.

The three Mode Hold blocks (all identical, see Figure 6.9 and Figure 6.11) keep the Mode output from changing too frequently. There is a Mode Hold block for each bit of the thermometer-coded mode signal (which is decoded into an unsigned 2-bit signal before being sent to the rest of the system), although only one is active at a time. This one corresponds to the current mode of the system, and will drop low once 256 conversion cycles have passed after the last stimulus from the Slew Threshold input. The ones corresponding to higher modes are inactive since their Slew Threshold inputs haven't been triggered, and the ones corresponding to lower modes are suppressed by the one corresponding to the next mode up. This is shown in Figure 7.3 below; the blocks for Mode 1 are on top, those for Mode 2 are in the middle, and those for Mode 3 are on the bottom. (Compare with Figure 6.9.)

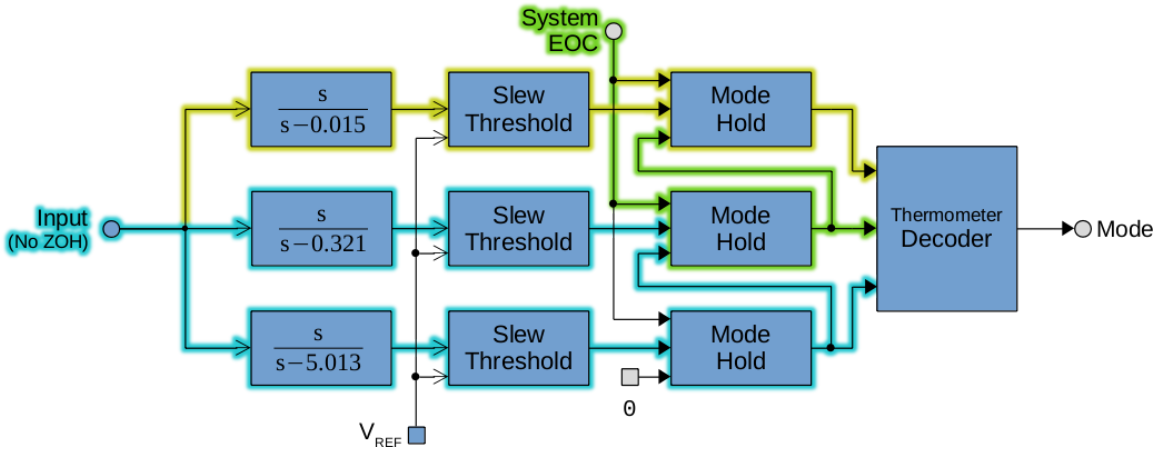

Figure 7.3 – Mode Block, time-domain description
(L3 Block Diagram)

The above figure shows the block in Mode 2. Signals and blocks involved with holding the current mode are highlighted in green, elements that have the potential to reset the hold count are highlighted in blue (those in the middle row will merely reset the count, while those on the bottom will cause the mode to jump up to Mode 3), and "suppressed" elements (either held static, or have no effect on the system) are highlighted in yellow.

This configuration ensures that the Mode signal will only drop after 256 cycles of non-activity for that mode's threshold, as this is how long it takes the Mode Hold block will hold the mode without being reset; and ensures the Mode will only drop by a value of one each time, since the counter of the next mode down is suppressed.

7.3.2 Timing

The Timing block works in the following manner, for all modes except for Mode 3. It is shown in Figure 7.4 below. (Compare with Figure 6.12.)

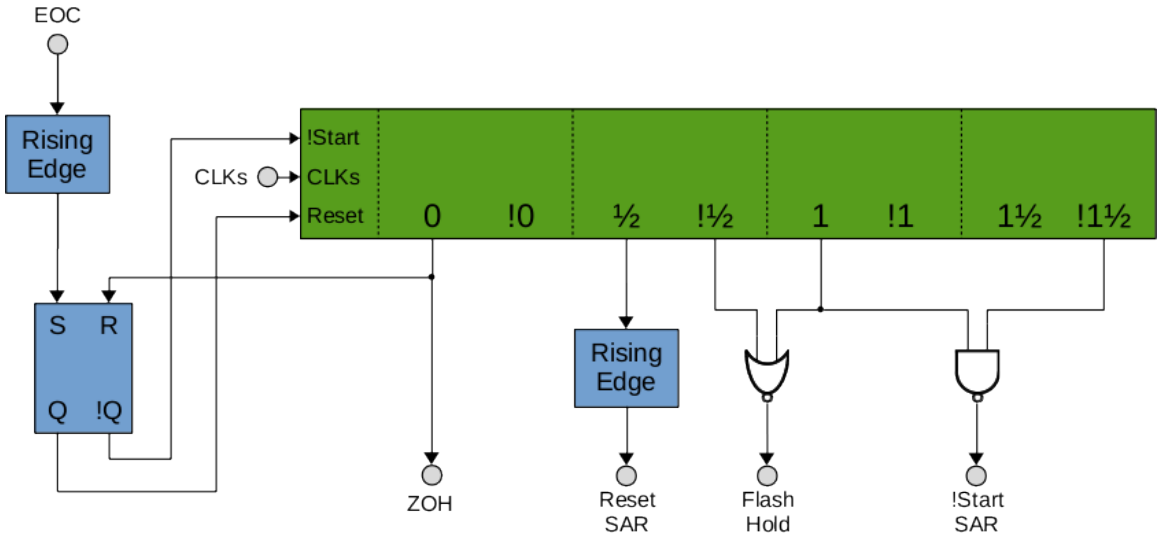

Figure 7.4 – Timing Block, modes 0-2
(L3 Block Diagram)

The process begins when the Timing block gets an EOC signal from the Successive Approximation block. After that, the following steps happen (whole numbers take place on clock rising edges, half numbers take place on falling edges):

- Immediately: Reset and !Start signals are given to the counter. The counter is reset immediately and will being counting on the next rising edge, starting with signal 0.
- **Signal 0**: ZOH set high. It will remain high until the next reset point, since the Zero-Order Hold circuitry is only sensitive to rising edges.
- **Signal ½**: Impulse sent to Reset SAR, resetting the Successive Approximation block to prepare it for its part in the upcoming conversion cycle. Also, Flash Hold is set high. This output signal connects to an array of D latches, so the hold on the flash signal will not be effected until the signal drops low.
- **Signal 1**: !Start SAR is set low, signaling the Successive Approximation block to start its counter. This is where a timing hand-off of sorts is performed, since the

system timing is now in the hands of the SA counter.

- **Signal 1½**: !Start SAR is set high, where it will remain until it is set low by Signal 1 the next time around. From this point on the Timing block remains inert until the next EOC signal is sent.

For Flash-only mode, the behavior is different, as shown in Figure 7.5 below. (Compare with Figure 6.12.)

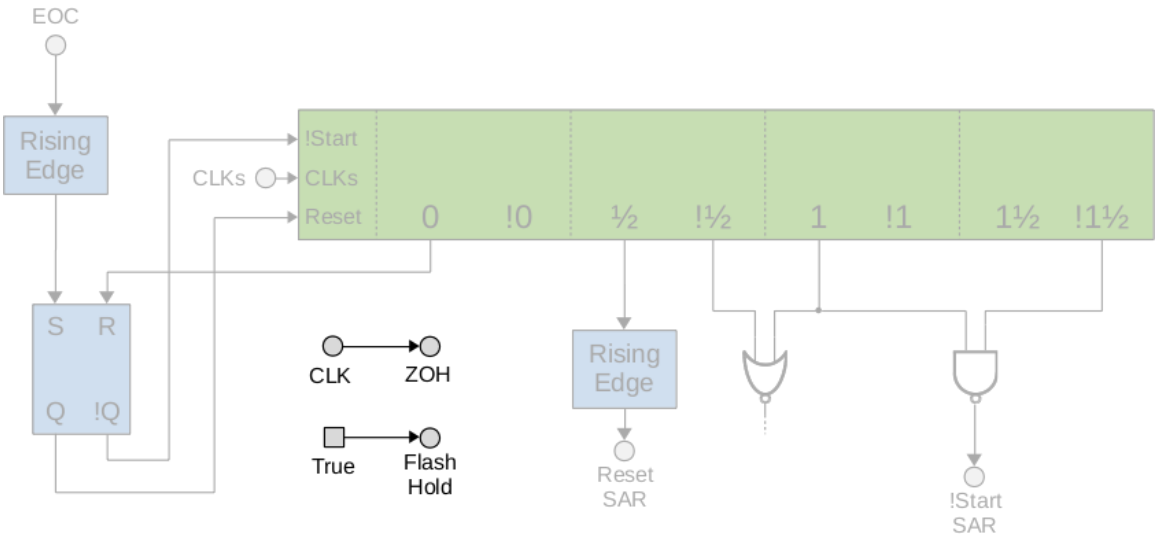

Figure 7.5 – Timing block behavior during Mode 3 (L3 Block Diagram)

In this mode the Timing block is essentially in a bypass configuration. ZOH is connected directly to the clock and Flash Hold is connected to a binary '1' constant; both are done with transmission gates.

It should be noted that in this configuration the Timing block is still running, as is the Successive Approximation block. (They behave as though the system is in Mode 2.) This is negligible since their behavior has no effect on the rest of the system while in Mode 3.

7.3.3 Output Handling – Bit Appendor

The Bit Appendor of the Output Handling block is purely combinational—it stores no memory and has no connection to the clock. However its time-domain behavior is a little fascinating because it makes use of transmission gates on the signal path instead of standard logic gates.

In general, logic gates tend to be preferred over transmission gates because of their speed. However, in a multiplexer-like system where the selection of a signal is less time critical than the content of the signal, transmission gates can be advantageous.[14] The Bit Appendor fits this criterion, since the Mode signal (the selector) changes much less often than the data signal coming in from the Flash Corrector and Successive Approximation blocks, which changes frequently and is time critical—especially the LSB. (Although the Mode does sometimes change in the middle of a conversion cycle, see Section 8.3 for details.)

The Bit Appendor handles the data exclusively with transmission gates; if these gates are made from significantly wider transistors than the rest of the system (~2-4x the width), the drain-source conductance will grow proportionally, as will the gate capacitance.[15] The former effect reduces the delay induced by the transmission gates, while the latter effect slows down Mode switching, which is non-critical. (In fact in many cases a little mode delay is a good thing, as explained in the last paragraph of Section 8.3.)

7.3.4 Output Handling – Output Hold

The Output Hold is an array of 16 D-latches, and its time-domain operation is rather simple. A hold is effected on the output on the rising clock edge if (and only if) the

EOC input is high at the time. (Note that this is slightly different from the SA block's EOC output—it's the same for modes 0..2, but is held high constantly when in mode 3.)

Chapter 8
Potential Problems and Implementation Issues

The proposed Tandem ADC has several potential pitfalls which much be avoided, designed around, or simply observed carefully to ensure problems will not arise. Some of these issues—those involving the behavior of off-the-shelf Flash and Successive Approximation converters—have already been covered in Chapter 3, and issues with the black-box operation of the system have been covered in Chapter 2. This chapter will instead focus on problems pertaining to the system as a whole, and on the internal details of the implementation rather than black-box operation.

8.1 Flash System – Timing and Transients

The Flash system (ADC and corrector) is designed to have its output settle within one clock cycle. Ideally corrector output, once settled, would not change until the next conversion cycle, since everything is held steady (directly or indirectly) by the input ZOH. This is illustrated in Figure 8.1 below. Signals directly held are shown in green, those indirectly held are shown in yellow.

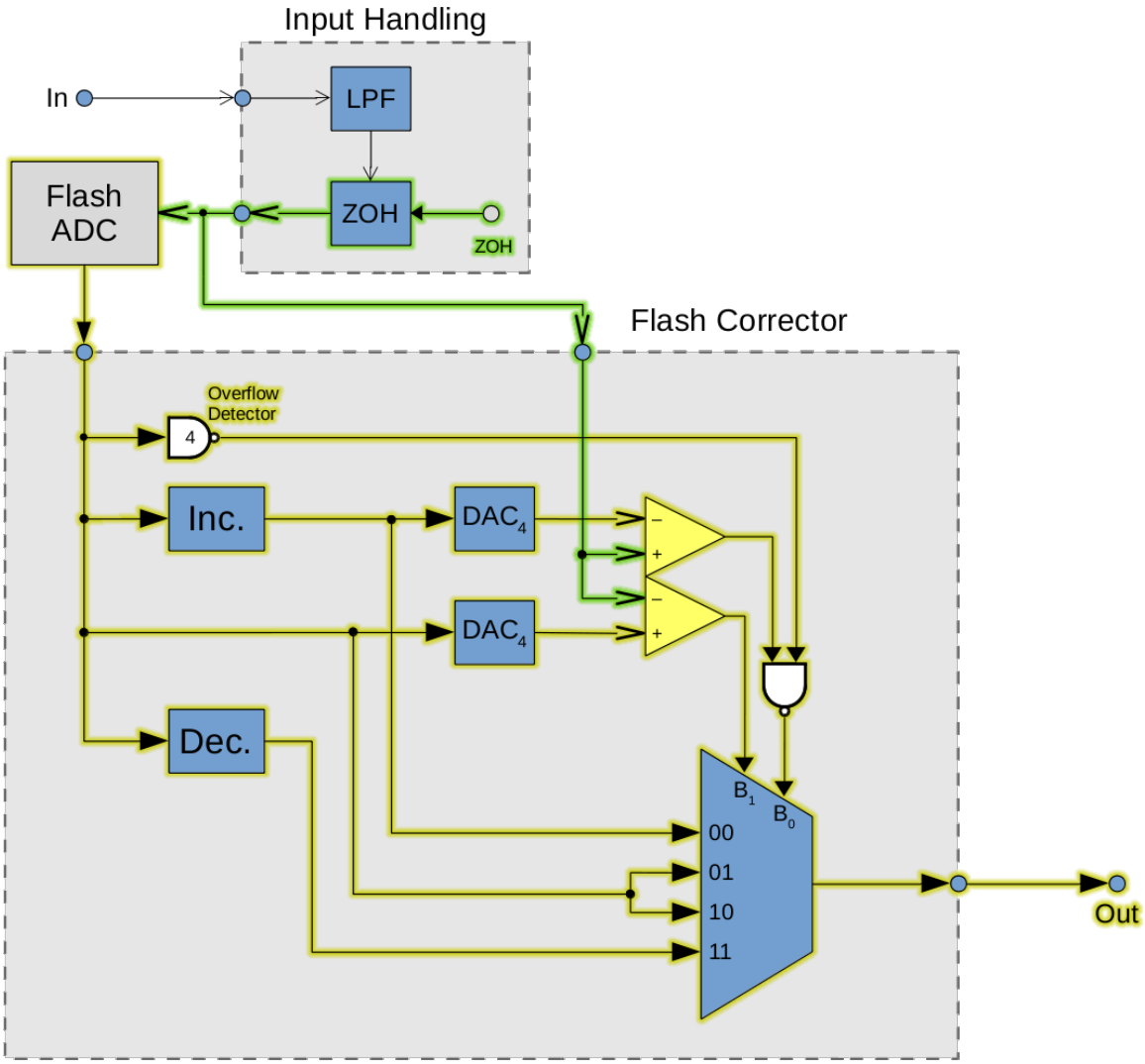

Figure 8.1 – Diagram of Input Handling & Flash System

However, it is possible for the Flash ADC to have an off-by-one error in the middle of a conversion cycle, for example if the analog input is right on the border of two digital values and the zero order hold is not perfect. The corrector can fix errors of this nature, but it takes up to a full clock cycle to do so. A Flash error at the wrong point in time would cause a transient lasting the time it takes for the corrector to settle. This means the Flash system would be providing the wrong output to the Successive Approximation register in the middle of a conversion, and the active bit would attempt to

correct for this error, giving it the wrong value. The solution to this problem is to place a digital hold (using D latches) on the output of the Flash corrector. This way, once the value is found it stays put.

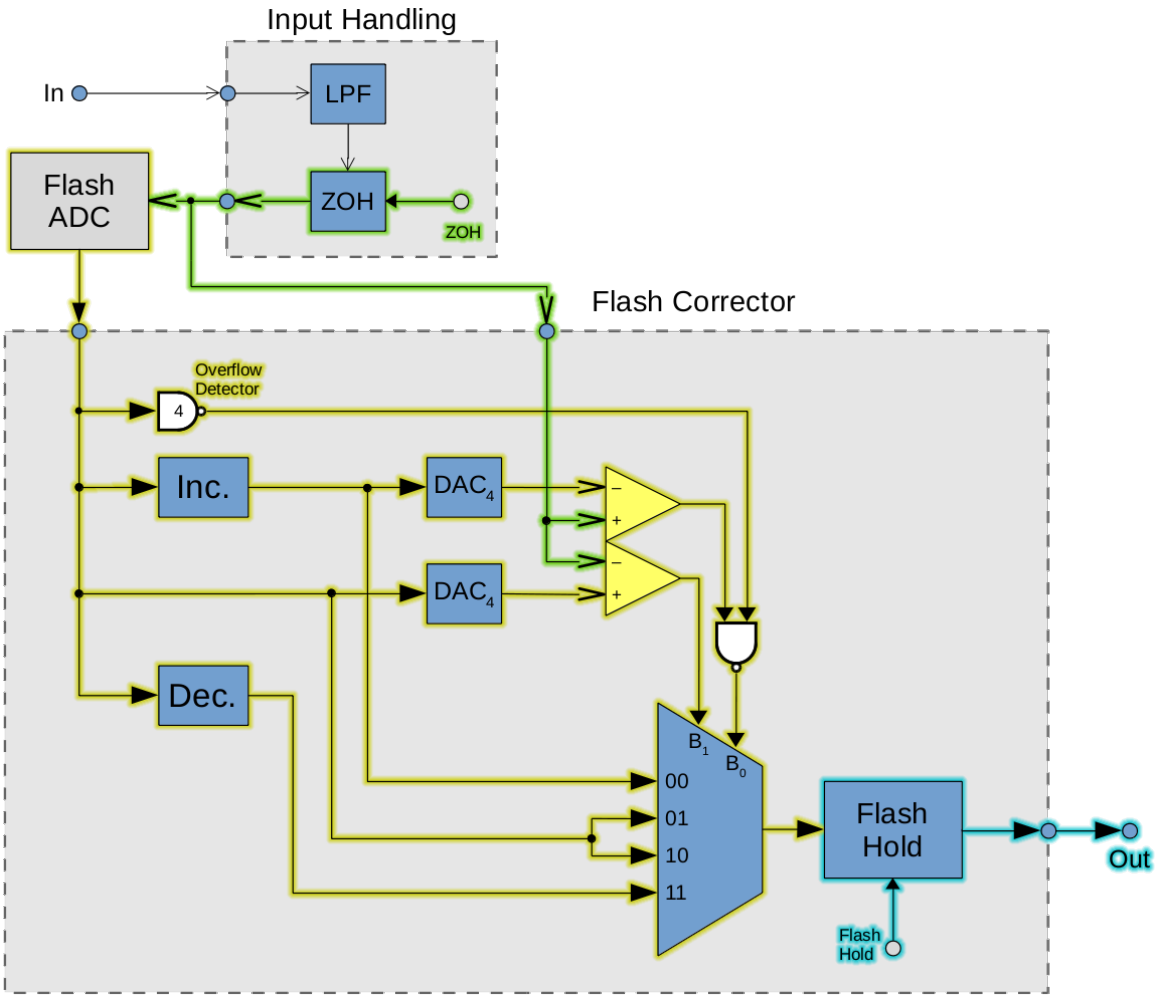

Figure 8.2 – Figure 8.1 with Flash Hold on the output

The implementation is shown in Figure 8.2 above, with the re-held signal shown in blue. (The Flash Hold control signal is managed by the Control block [Timing sub-block], as shown in Sections 6.4.2 and 7.3.2.)

Another potential issue with the Flash system is that it, depending on propagation delay, it might not be able to settle within one clock cycle (for a given clock

speed which is the optimum for the rest of the system). One solution is to simply slow down the clock, but this will slow down the entire conversion, Successive Approximation included, and decrease throughput proportionally. This is not an appropriate solution for the applications of interest here. Instead, the way to run the system the fastest is to have the bottleneck for clock speed be determined by some part of the system which is used multiple times during each conversion cycle, most likely the Successive Approximation block. This can be done—the time bottleneck of the Flash system can be bypassed—by effecting the input ZOH earlier than usual, giving the Flash system extra time to settle. If done correctly this would not decrease throughput, and would only increase conversion delay by the time-delta of the displaced ZOH. (This is because the Flash Hold allows for a small degree of pipelining.)

The above-described ZOH displacement is not implemented, but could be accomplished with minimal modification to the Timing sub-block (see Figure 6.12).

8.2 Successive Approximation

The Successive Approximation block is almost certainly the bottleneck of the system with regards to clock speed, and for most modes, the largest consumer of time. Therefore, to avoid limiting the system's maximum clock speed unnecessarily and in general slowing everything down, the SA block should be made as fast as possible.

Within the Successive Approximation block, one way to remove throughput bottlenecks is to ensure that no single bit takes longer to find than any other. The two bits most susceptible to time variations are the first bit converted (bit 11) and the last bit (bit 9, 5, or 1; for modes 0, 1, and 2 respectively).

The first converted bit runs the risk of taking longer than the others because the Flash Corrector has a hold on its output, which can cause propagation delay, which in turn can cause the Flash bits to not be immediately available when the Successive Approximation block is started. This would then necessitate a small delay on the part of the SA block before converting the first bit, otherwise the conversion might not be performed correctly. This is a significant issue since a "small delay" is nontrivial to implement, and so the maximum clock speed must be reduced, slowing down the entire system.

A better solution (and the one implemented here) is to ensure the Flash Hold causes no real propagation delay; meaning the Flash Hold provides the correct value at or before the clock edge that starts the SA block happens. In general, D latches are often controlled by a rising-edge detector, sending a pulse to the C input right when the clock (or other signal) goes high. This is not instantaneous, however, and the delay caused can be eliminated if the latch is set to the correct signal at least slightly *before* the rising clock edge. This is accomplished by setting the C input of the D latches high at the preceding falling edge, and then dropping them low at the rising edge. This means the latch leaves the signal alone for that half clock cycle, letting it pass with minimal propagation delay, and then clamping down in it right on the next rising edge.

Another potential issue is that of the LSB being properly captured by the output hold. (See Section 6.4.3 for details on the Control → Output Handling block.) This is because the SA block is reset shortly after the LSB is found. This can be addressed by proper timing, and normally would not be much more of an issue than it would be with

standard Successive Approximation ADCs. However, the Appendor adds length to the signal path. If left unaddressed, this would cause an increase in conversion time of the LSB, which would slow down the whole system via the clock. Fortunately the Appendor can be made to add little or no time to the conversion process, using transmission gates. (This is described in Section 7.3.3.)

A third possible problem with the Successive Approximation block is that it might convert too many bits for a given mode. In most situations this is not at all likely, but in Mode 3 the Successive Approximation block is running even though that's the Flash-only mode. Also, in most situations this would not be a problem at all (more bits is better) but the insertion of an insignificant bit means that everything that follows is essentially junk data. The solution is simply to have the Appendor (see Section 6.4.3) drop in zeros following the insignificant bit, rather than allowing the Successive Approximation block to have free reign over those bits and hoping that it doesn't have the time to make them nonzero. This increases the Appendor's complexity but does not affect signal propagation time.

8.3 Synchronicity and Mode Switching

When designing any variable-rate ADC, it is important to make sure that the ADC and the device to which it is connected agree on which conversion rate is being used at any given time. This is true to some extent for ADCs in general – most have an EOC output to notify the connected device when each conversion is finished. For a variable-rate ADC, extra information is very beneficial, if not necessary. In the case of the Tandem ADC, sending the Mode signal downstream (along with the output and the

standard EOC signal) is an easy way of doing this, as it helps the downstream device predict when the next sample will be available, as well as selecting the proper processing procedure associated with that particular sampling rate. (See Chapter 4 for an example of how different processes can be used based on sampling rate.)

Even if the system acts properly for any given mode, there is still the possibility that switching from one mode to another will cause undesired behavior. The way the Tandem ADC is set up either avoids this or mitigates its effects, depending on the scenario. When switching to a faster mode – which the Mode circuit will do immediately if the input's slew rate crosses the needed threshold – the whole of the system is notified immediately, since all relevant blocks have immediate access to the Mode signal. The Successive Approximation block handles this sudden change gracefully, either ending the current conversion after the new precision level has been reached, or ending the current conversion immediately if said precision level has already been surpassed.

The Bit Appendor will also change as quickly as propagation delay allows, but this change may or may not happen in time for the current conversion. If the change happens in time, the Appendor will provide the insignificant bit in the proper place and suppress any extra and unneeded bits found by the Successive Approximation block. (The SA block can sometimes wind up with extra bits if the mode changes in the middle of a conversion. For example, it may have already found seven bits when the system suddenly switches to Mode 3 which only calls for three bits from that block.) If the change does not happen in time the Bit Appendor will not perform its job correctly, but this is only a small error for a single conversion cycle. (Either scenario is possible, since the Mode

block is clock-independent [at least when increasing the system's mode]; therefore the length of time the Bit Appendor has to update itself can range from no time at all to several clock cycles.)

When switching to a slower mode, the Mode block has different behavior. The Mode Hold circuit will drop the mode value down, one at a time, after 256 conversion cycles have passed without the hold being reset. (This is done to keep the mode value from switching too often – see Section 7.3.1 for details.) The counter for conversion cycles is triggered by the EOC signal for the entire Tandem ADC system (System EOC). This is done to assure the mode switching is done at the right point in time. If switching were done at a different point – if the Successive Approximation EOC triggered the Mode Hold counter instead of System EOC – the switching to a lower mode would happen early enough to affect the Bit Appendor for that conversion, but too late to affect the Successive Approximation block. This would cause an error similar to that which can happen when the mode switches to a higher value, where the bit appendor is working with a different mode value than the SA block and provides the insignificant bit in the wrong location. (Though this is not a major issue for the same reason – such an error with the bit appendor only causes a small error for one conversion cycle.)

Chapter 9
Testing & Results

In order to investigate and verify the operation and performance of the proposed Tandem ADC, two system models were developed and tested: a full system implementation model rendered in Simulink (see Appendix A), and a black-box model coded in Matlab (see Appendix B). The Simulink model was used primarily for verification of desired behavior (Section 9.1), while the Matlab model was used for more extensive testing (Sections 9.2.1 and 9.2.2) since these tests could not be run on the Simulink model in a reasonable amount of time. It was assumed that inputs to these models had already gone through an anti-aliasing filter.

9.1 Verification of Expected Operation (Simulink)

The system modeled in Simulink was tested with a large-amplitude sinusoidal waveform. The Mode was controlled manually; the first period of the input wave was converted at Mode 0, the next at Mode 1, followed by Modes 2 and 3. The results are shown below in Figure 9.1.

It's worth noting that the units in the model are not indicative of reality. The converter ran with the range of 0 to 65,535, so one unit in the Simulink model (one "volt") corresponds to the LSB of a 16-bit number. The unit of time (one "second") is one clock cycle.

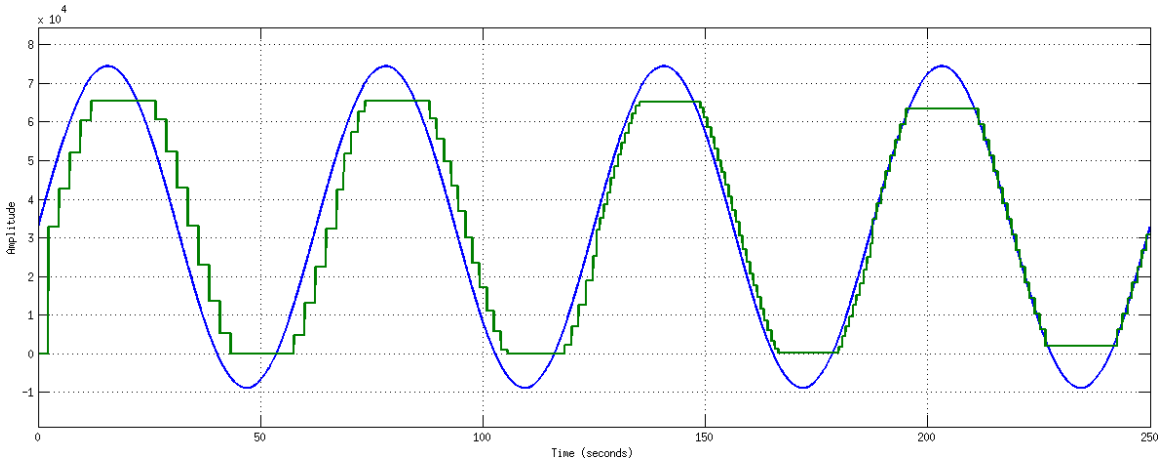

Figure 9.1 – Sampling Waveform from Simulink Model

As shown above, the ADC followed the signal correctly, and handled overflow gracefully, clipping instead of looping around. Since the waveform had such a large amplitude, Modes 2 and 3 performed the best even though all modes oversampled the signal several times over. A signal of the same frequency with much lower amplitude would be best dealt with by Modes 0 and 1.

9.2 Black Box Testing (Matlab)

The Simulink model describes the entire system in detail (see Appendix A), but simulations take significant time to run, sometimes up to an hour. In addition, only one instance of the Tandem ADC can be simulated at once, so comparing behavior of different modes on the same input takes several times as long, and compiling the data from several different Simulink simulations is nontrivial.

For these reasons, a black box model was coded in Matlab (see Appendix B). This model describes the outward operation of the Tandem ADC (delay and quantization) but does not simulate its underlying mechanisms. This allows behavioral modeling to occur much faster (on the order of seconds rather than minutes or hours), and several

mode-locked conversions can be run simultaneously and compared easily.

This model was used for chirp input and step response testing, as described in the next two subsections. It should be noted that this model's full-scale range is 0 to 1, rather than 0 to 65,536. The time axis is the same—one unit of time corresponds to one clock cycle.

9.2.1 Mode Switching and Mean-Square Error

A chirp input was tested on five different converter setups: one with the usual mode circuit in place (orange), and the others with the mode set constant to 0, 1, 2, and 3. The MSEs were graphed as a moving average (correlation with a Hanning window), and the result is shown below in Figure 9.2. The ADC under normal operation is shown in orange, while those locked in modes 0, 1, 2, and 3 are shown in blue, teal, green, and purple respectively. (For error graphs, the y axis measures mean-square error as a proportion of the full scale range.)

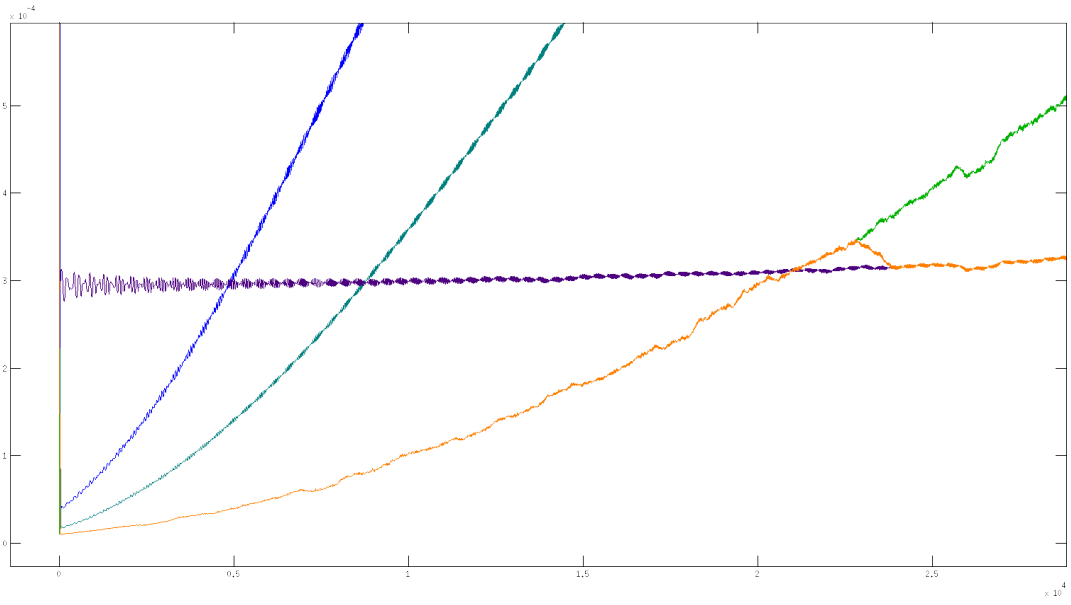

Figure 9.2 – Mean-Square Error Measurements

This bears resemblance to Figure 3.6, although that figure was the result of numerous tests with constant-frequency sine waves rather than a single chirp signal. The rolling MSE measurement adds a bit of noise to the graph, but this technique shows the mode of the converter switching in real time. (The only switching that can be seen is from Mode 2 to Mode 3, since the chirp started out at too high of a frequency for Modes 0 and 1 to be used.)

Running a lower-frequency chirp, the transition from Mode 1 to Mode 2 can be seen. The colors are the same as before, although Mode 3 is not shown.

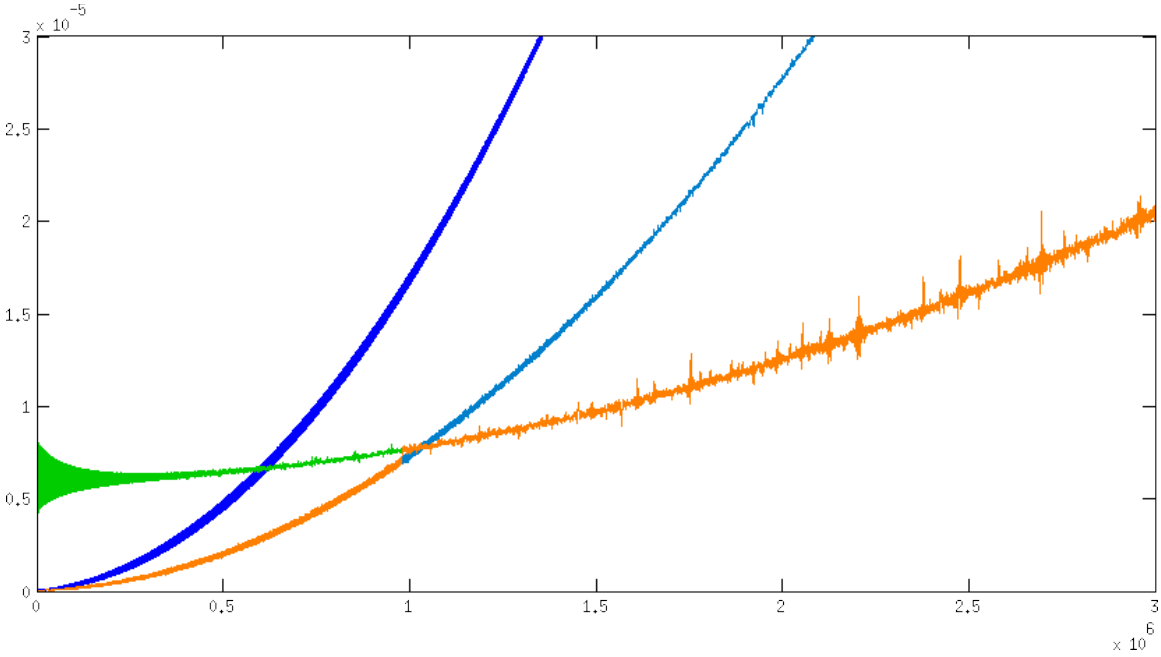

Figure 9.3 – MSE Measurements, showing transition from Mode 1 to Mode 2

In the above graph noise becomes a little more of a problem, but the transition is still clear. Lowering the average slew rate further allows the transition from Mode 0 to Mode 1 to be seen, as shown below.

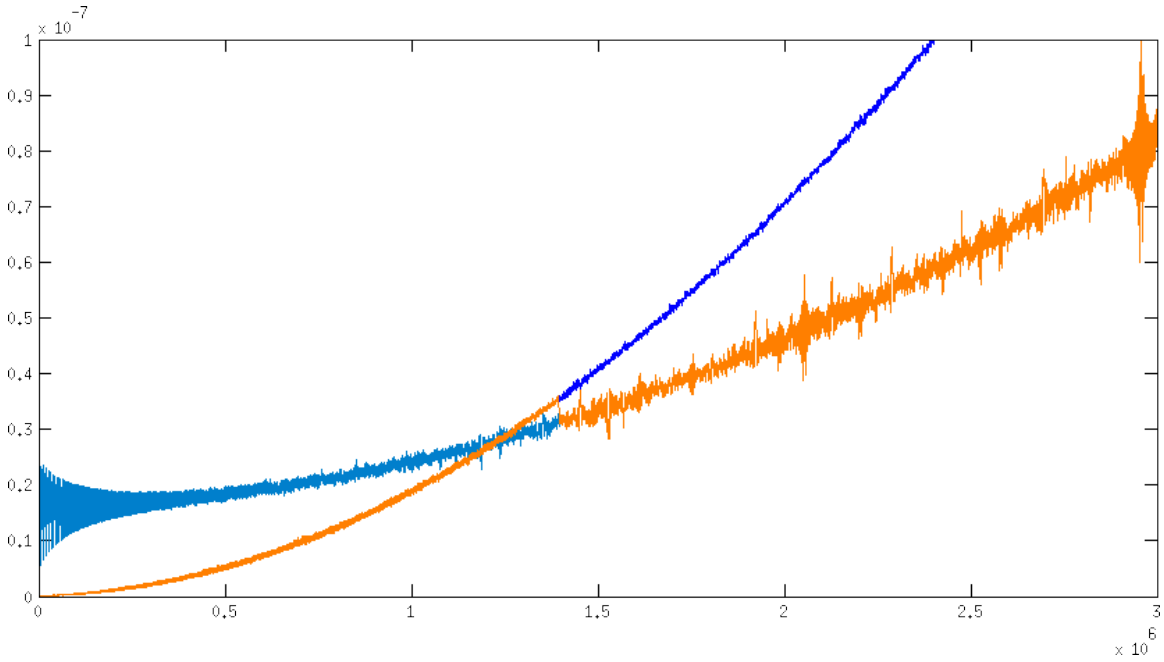

Figure 9.4 – MSE Measurements, showing transition from Mode 0 to Mode 1

Mode down-switching uses a different mechanism than up-switching. (This is to prevent the Mode from shifting too often – see sections 7.3.1 and 8.3.) There is a delay of 256 conversion cycles, and the mode only moves down in value one unit at a time. However the behavior is largely the same when observing a large time span, as is the case in these tests. The behavior of mode down-switching was tested by running the chirp signal backwards, as shown below. Note that it is nearly a mirror image of Figure 9.3.

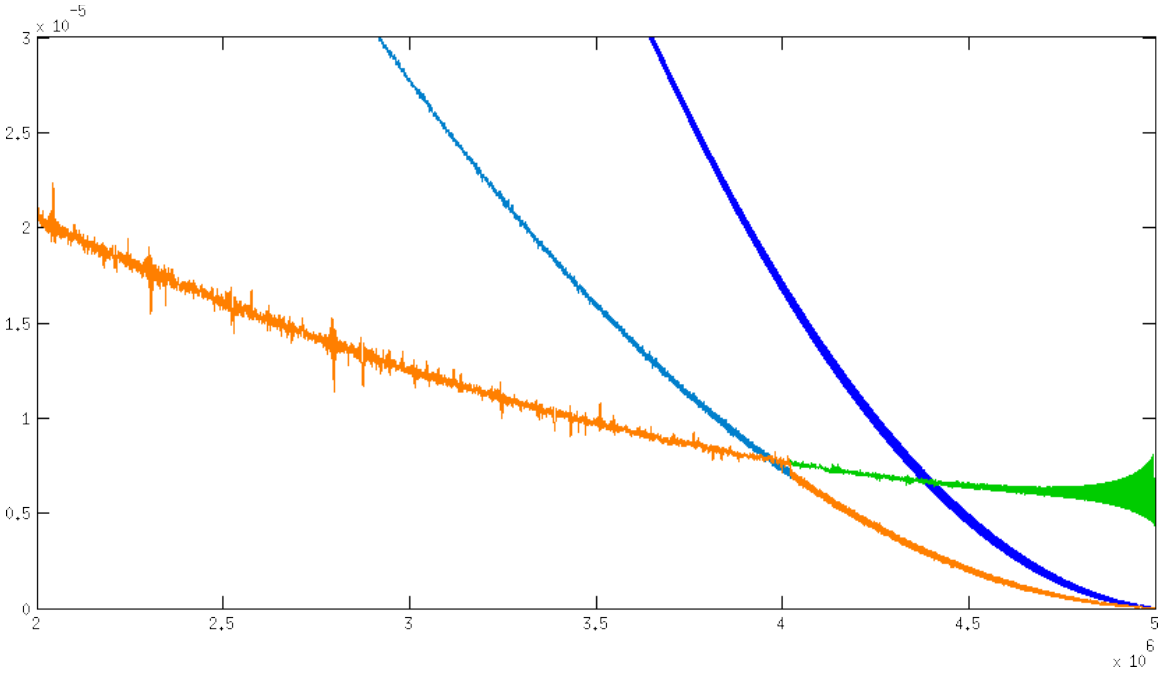

Figure 9.5 – MSE Measurements, showing transition from Mode 2 down to Mode 1

The delay induced by the mode hold is unnoticeable at this scale, but is important to keep the mode from rapidly shifting. Mode switching behavior over shorter time spans is covered in the following section.

9.2.2 Step Response

The step response of the Tandem ADC was tested, and results were mixed. A step input represents the upper extreme of a high slew rate input signal, but only for an instant. A graph of this response is shown below – the step function is black, the step response is orange, and the response of Mode 0 by itself is blue.

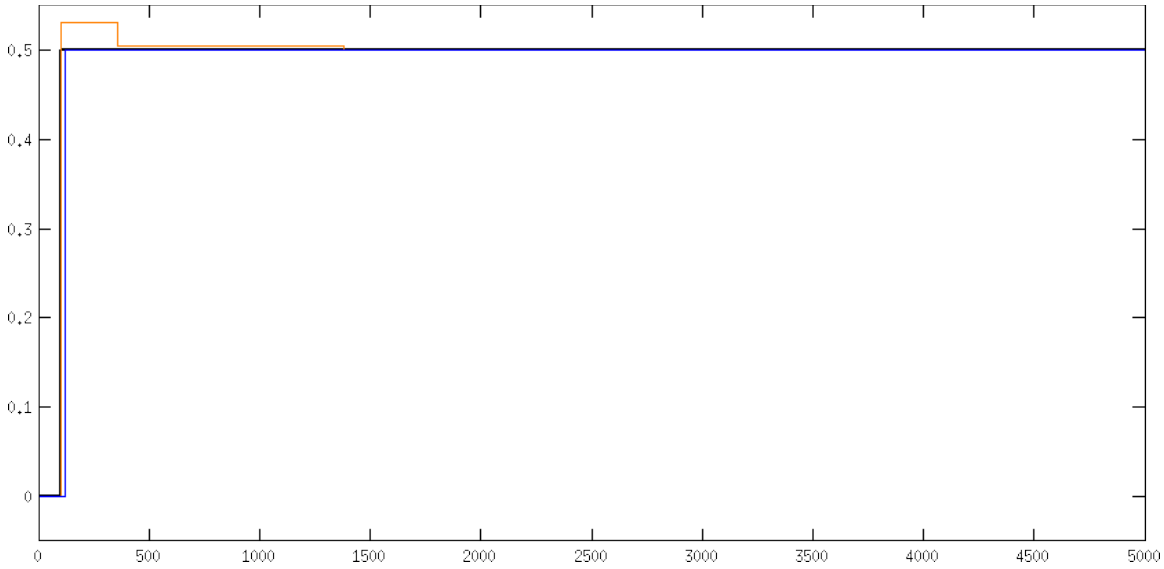

Figure 9.6 – Tandem ADC Step Response

The step response has noticeable overshoot because the sharp transition triggers Mode 3, which has significant quantization error. Eventually the mode drops back down, and the output settles. The mode changing is shown in Figure 9.7 below, along the same time scale. (X axis is still time; Y axis is mode for this figure.)

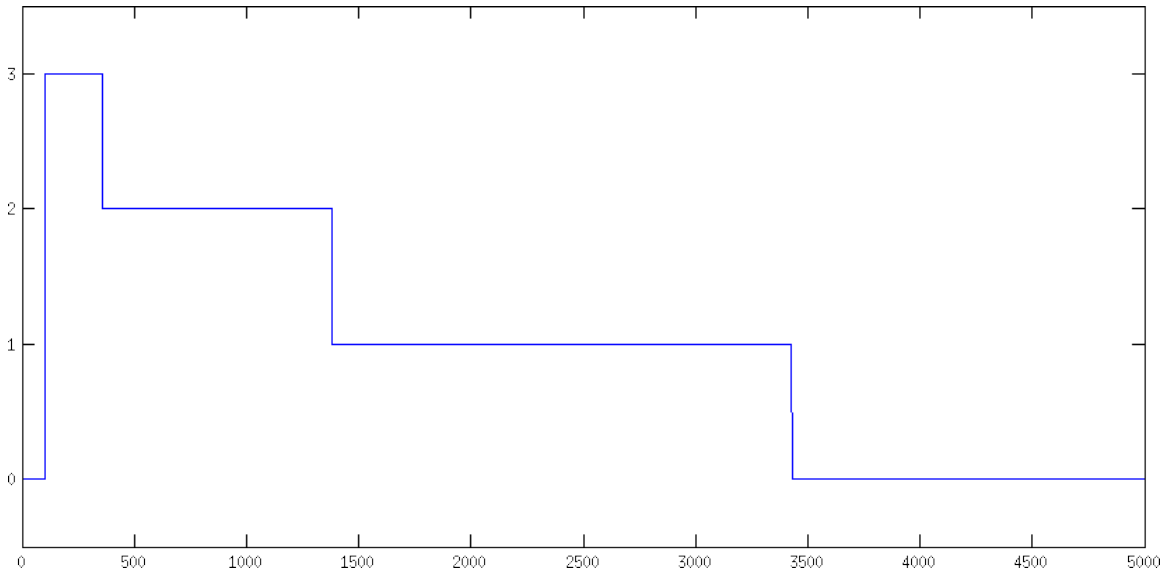

Figure 9.7 – Tandem ADC Mode Step Response

This is a non-optimal step response, but it does have its advantages. Shown

below is Figure 9.6 on a shorter timescale.

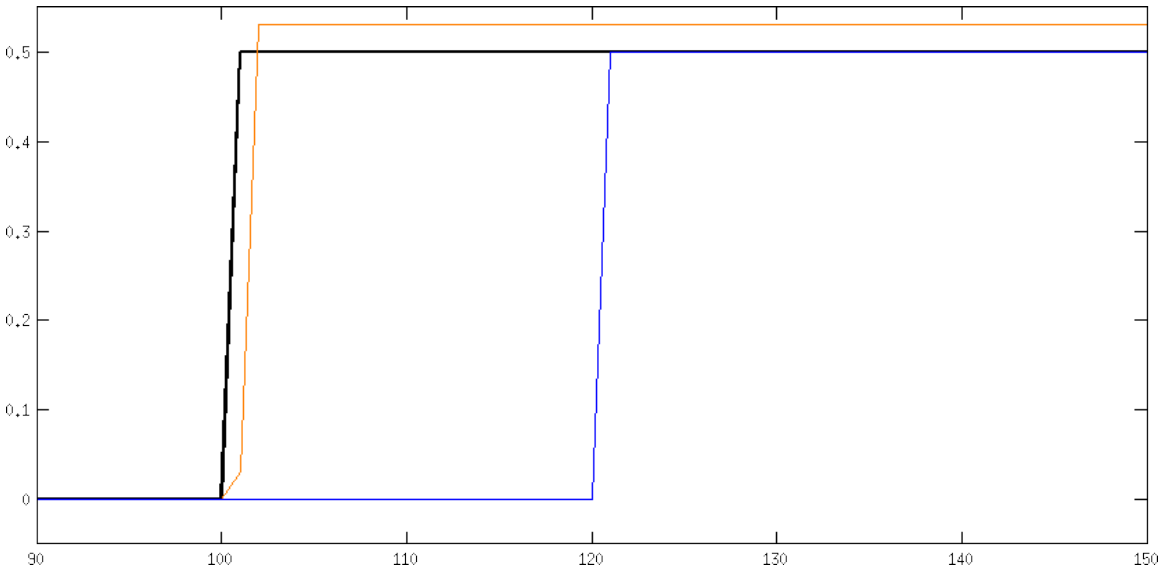

Figure 9.8 – Tandem ADC Step Response, shorter timescale

The step response of the normal Tandem ADC (orange) is imprecise, but it responds much quicker than if it were held at Mode 0 (blue). That being said, a pure step response does not appear to be a strong point of the Tandem ADC, at least in its current configuration for mode switching and mode hold parameters. The ADC therefore may not work well with inputs that have "clean" transients, such as when there's a step function followed by a calm, unchanging signal. It is better suited for situations where a step function at one point in the system causes a messy analog signal at another point—it could be applied to the messy signal. This scenario often occurs in control systems, like the one shown in Figure 2.1. If a step response is given digitally in this system the physical plant will respond in a certain way, and the attached analog sensor would give out a rather busy signal which could then be converted by the Tandem ADC and fed to the DSP.

9.2.3 Step Response with Reduced Mode Hold

In its implemented configuration, the Tandem ADC has a rather poor step response. This is largely due to the mode hold, which is configured to only decrement the mode after 256 conversion cycles. This produces a significant region of unnecessarily high quantization error in response to a step function.

In search for a solution, the step response was tested again with the mode hold reduced to 4 conversion cycles. The response is shown below. (The line colors are the same as those of Section 9.2.2.)

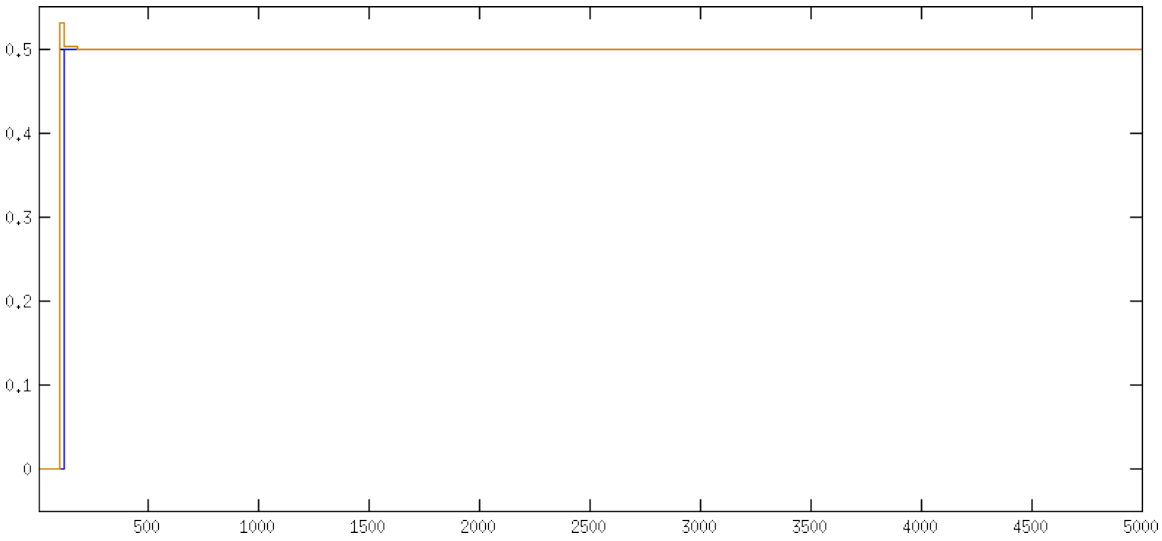

Figure 9.9 – Step Response for Mode Hold = 4

This response still has significant overshoot, but it lasts for much less time (1/64 as long as before). This is because the system mode drops down much more quickly, as can be seen in the following figure.

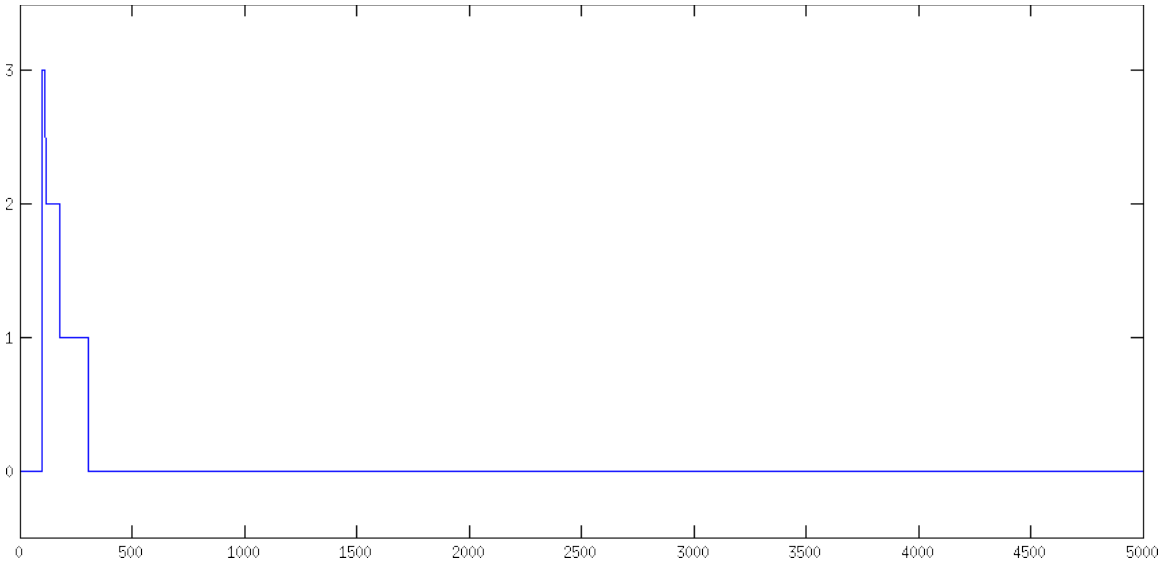

Figure 9.10 – Mode Step Response, Mode Hold = 4

Compared to Figure 9.7, this is a much faster dropoff. The benefits of this can be better seen in Figure 9.11 below, which is comparable to Figure 9.8.

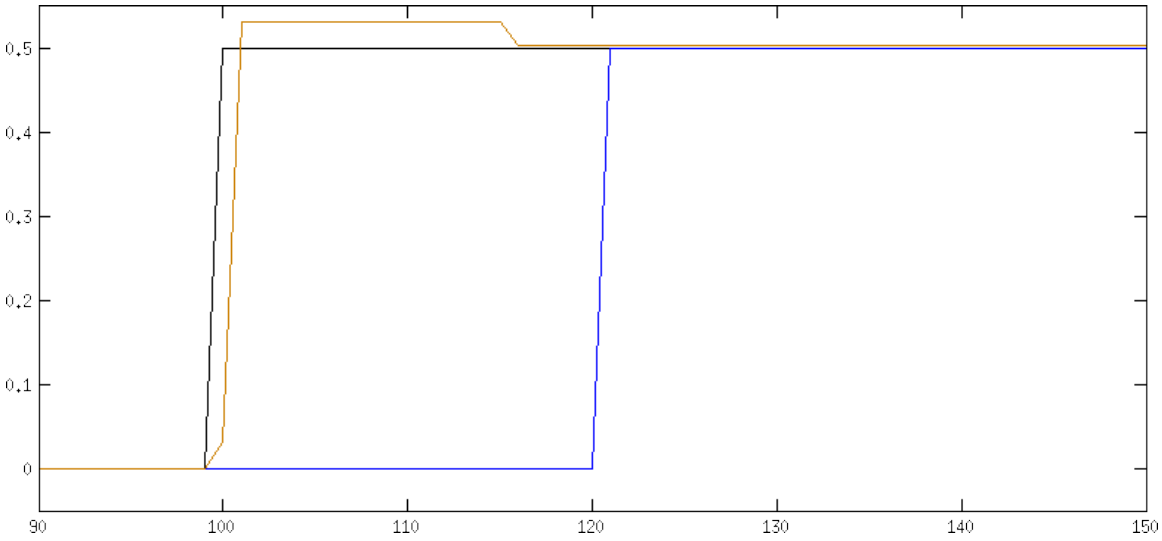

Figure 9.11 – Tandem ADC Step Response,
shorter timescale, Mode Hold = 4

As can be seen, the normal Tandem ADC drops down to Mode 2 (indicated by the reduction in quantization error) before the Mode-0-locked version is even able to respond. This demonstrates that the Tandem ADC can be adapted to respond well to step

inputs. This does increase the probability of mode jitter, however, since a constantly moving input would keep triggering the mode threshold after it dropped back down.

9.2.4 Testing with Random Noise

Tests were done random noise, to determine how the Tandem ADC responds to unpredictable signals. The first was pink noise, generated by integrating uniform white noise. A graph of this noise is shown below.

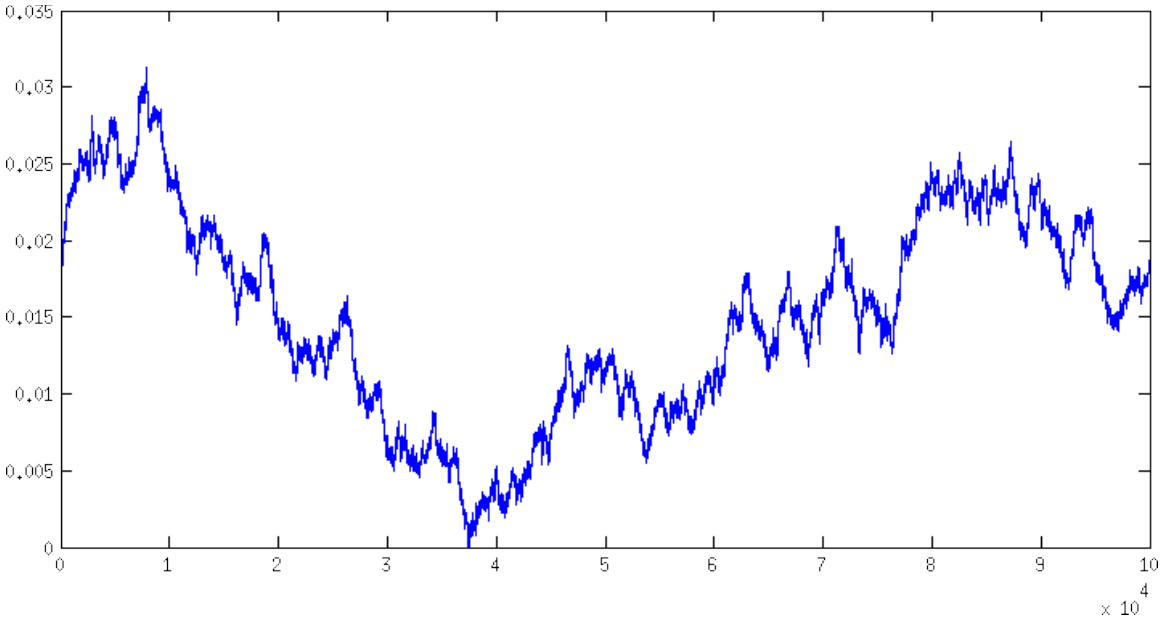

Figure 9.12 – Pink Noise Waveform

This was run through the ADC black box the same way as the chirp waves and step function (Mode Hold was set to 256), and the result is shown below:

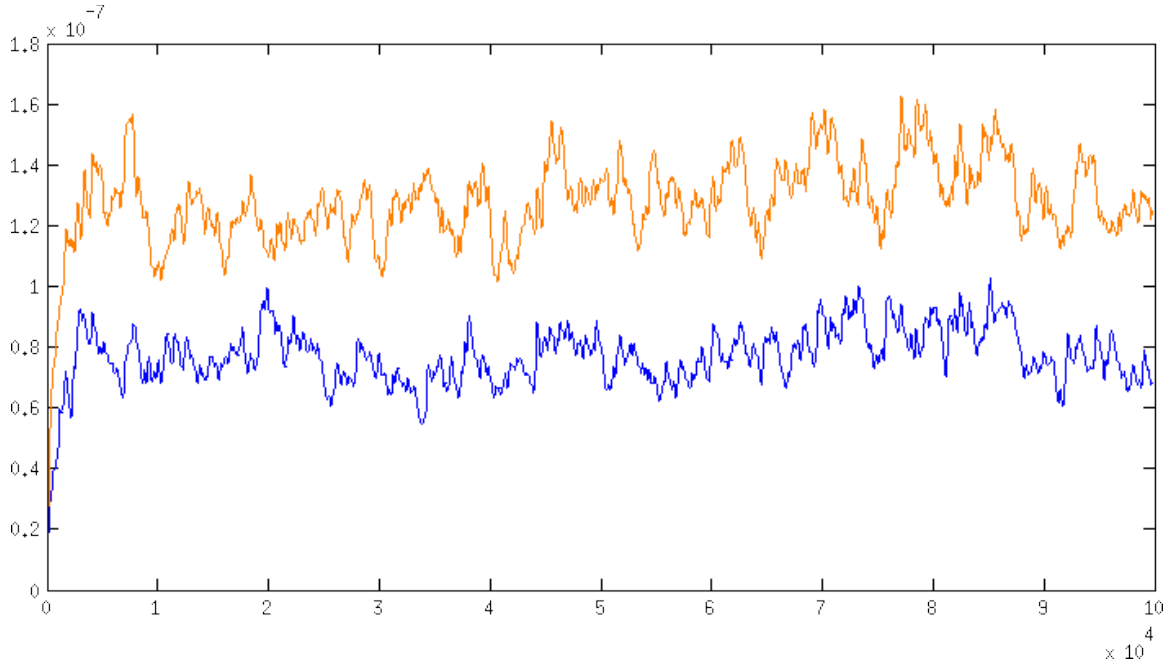

Figure 9.13 – Pink Noise MSE

For this waveform, Modes 2 and 3 had very large error values and are not included in the above graph. The Tandem ADC was on Mode 1 the entire time, exhibiting more error than Mode 0, but much less than Modes 2 and 3. This provides evidence that the mode switching may be calibrated for too low of trade-off point slew rates, and that in general pink noise does not highlight the resolution-switching capabilities of the Tandem ADC. Results could be worse however; the second-best mode was still automatically chosen.

After that test, a random walk was run thorough the ADC, with the average slew rate variance slowly falling. This signal was generated by taking a pink noise signal and fading it into a constant value, as shown below.

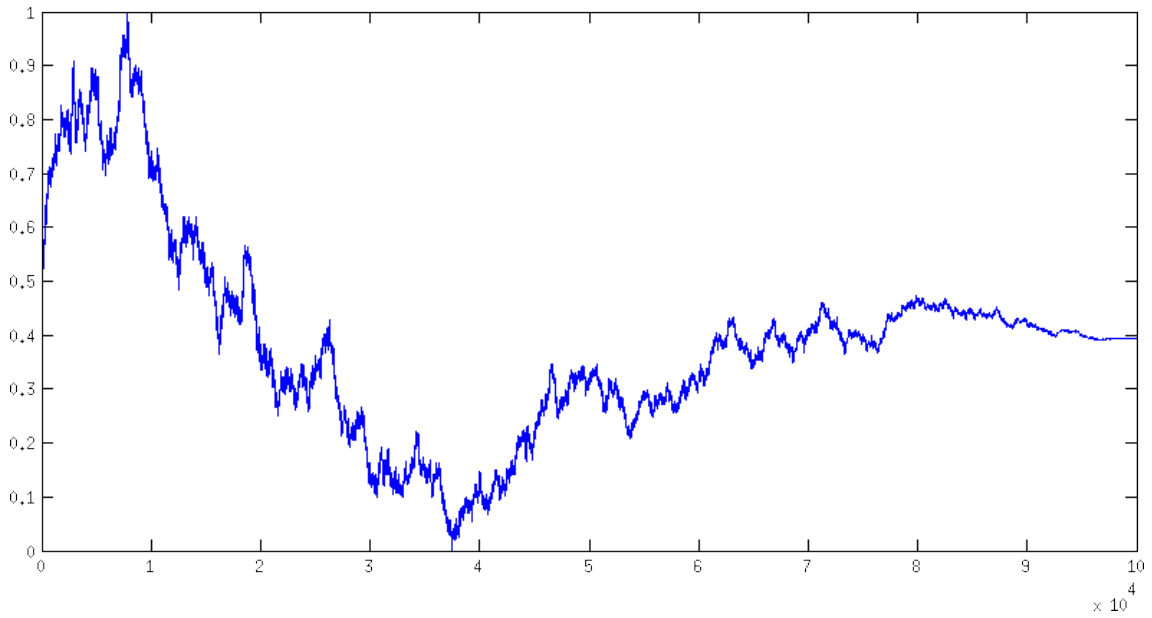

Figure 9.14 – Tapering Pink Noise

This is the same noise used in the pink noise test, but is tapered off to its average at the end. When integrated, this creates a random walk that exhibits less and less movement as time goes on, showing some resemblance to a physical system converging on a correct value. This is shown in Figure 9.15 below.

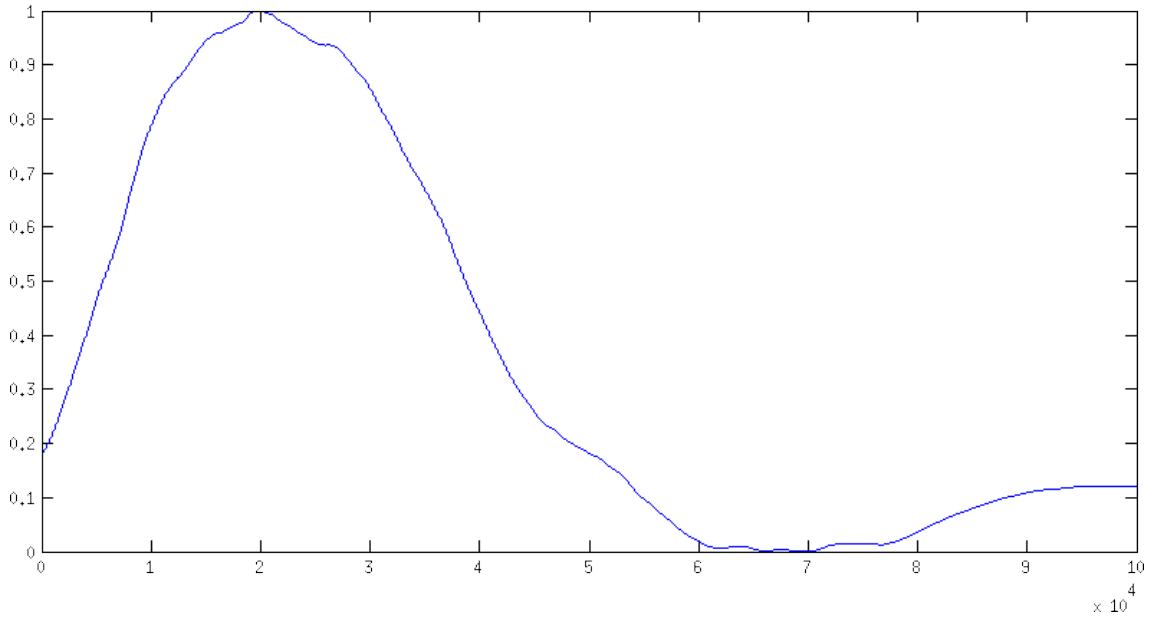

Figure 9.15 – Converging Random Walk

This was tested on the Tandem ADC, with the MSE results shown below in Figure 9.16. (Mode 0 shown in blue, Mode 1 in teal, actual ADC in orange.)

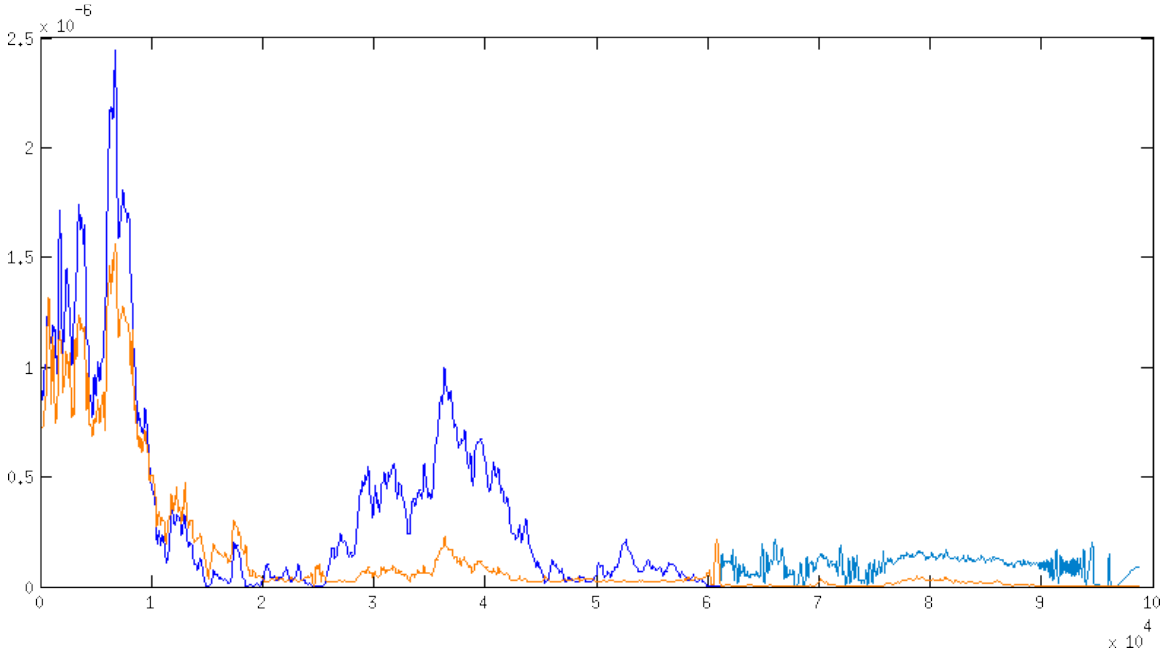

Figure 9.16 – Converging Random Walk MSE

As seen in the above figure, the Tandem ADC exhibits the mode of lowest available error for most of the sequence. Around 60,000 clock cycles into the simulation, Mode 0 becomes more accurate than Mode 1, and the system switches. This is shown implicitly in the graph above, where the Tandem ADC trace (orange) switches from covering up the Mode 1 trace (teal) to the Mode 0 trace (blue); it is also shown explicitly in the mode graph below.

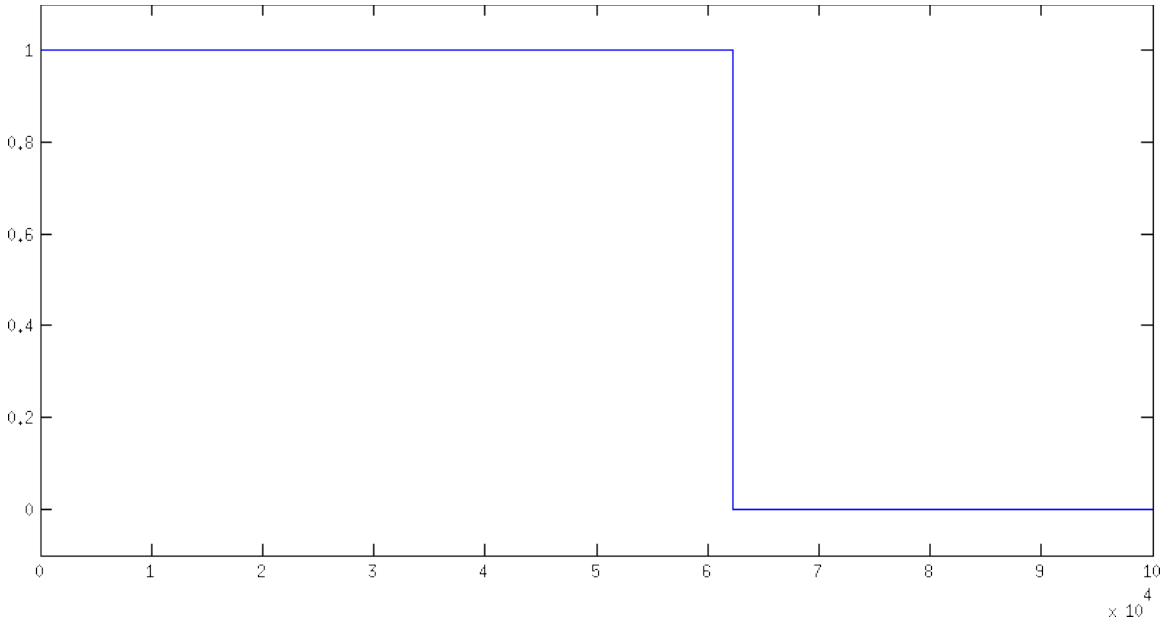

Figure 9.17 – Mode of Tandem ADC during Random Walk

9.3 Underdamped Step Response Input (Simulink)

As another way to test the Tandem ADC with a realistic scenario (the other was the converging random walk in the previous section), a step response was fed through an underdamped second-order filter with the transfer function

$$H(s) = \frac{0.00225}{s^2 + 0.03s + 0.00225}$$

where **s** is complex angular frequency measured in radians per clock cycle. The poles of this filter are s = 0.015 ± j0.045, and Q = 1.58. This creates an oscillating response that dies down after roughly two cycles. This waveform and the associated Tandem ADC output, with Mode Hold reduced to 4 (as was also done in Section 9.2.3) are shown below:

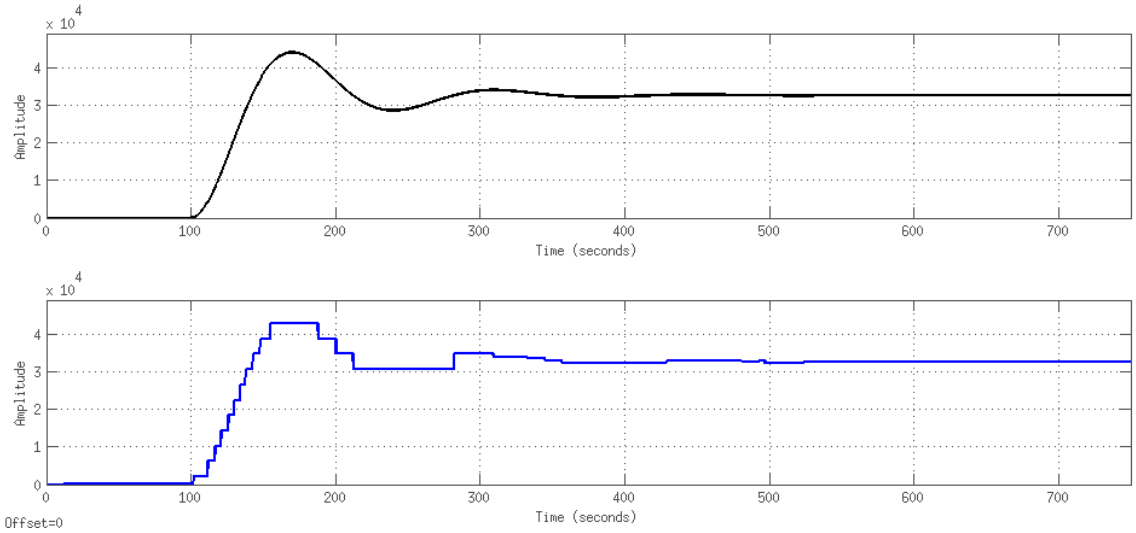

Figure 9.18 – Underdamped Oscillation and Tandem ADC Output

The comparison can be better visualized when the two are superimposed:

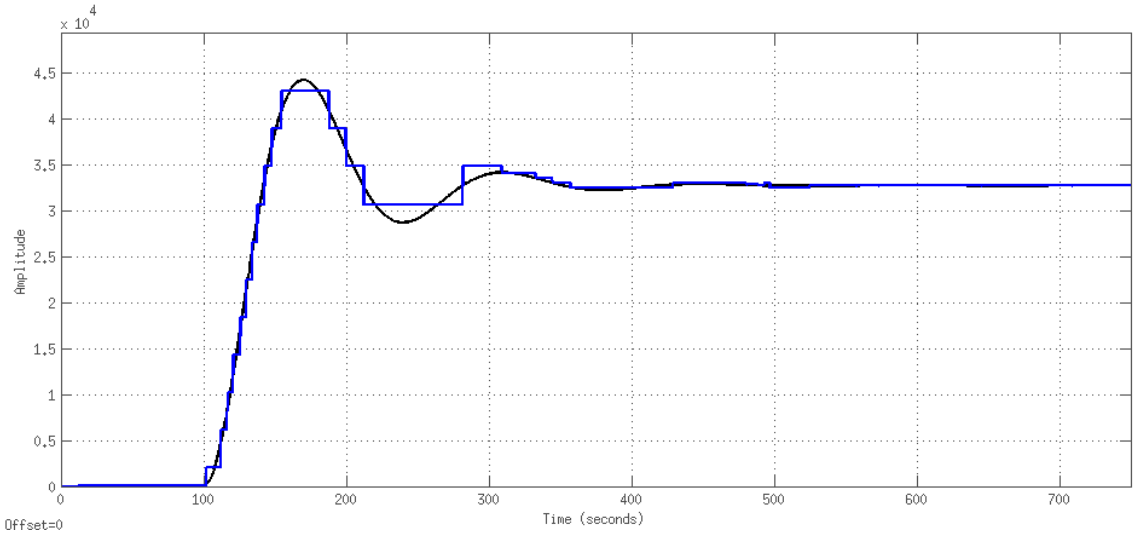

Figure 9.19 – Superimposed version of Figure 9.18

The Tandem Exhibits large amounts of quantization error during the initial rise, but does not "fall behind" the input. This is because the system jumps up to Mode 3, as shown in the mode plot in Figure 9.20:

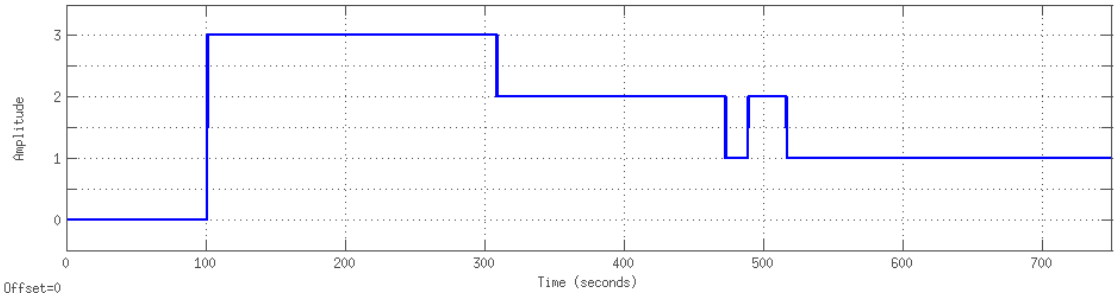

Figure 9.20 – Mode Response of Underdamped Oscillation input

This shows that the system reaches Mode 3 almost immediately after the input start (at time = 100) in order to quickly follow the rising input value, and then drops off as the input waveform begins to settle. This graph also shows that the ADC exhibits very little mode jitter during this conversion, demonstrating that a Mode Hold as low as 4 can be sufficient for underdamped oscillating inputs. The system never drops all the way to Mode 0, but quantization error is low enough in Mode 1 to prevent significant hunting (maximum deviation from hunting in Mode 1 is less than 0.05% of full-scale range).

This underdamped input was also tested with the Tandem ADC while it was locked in Modes 1 and 2. The result of the ADC in Mode 1 is shown below:

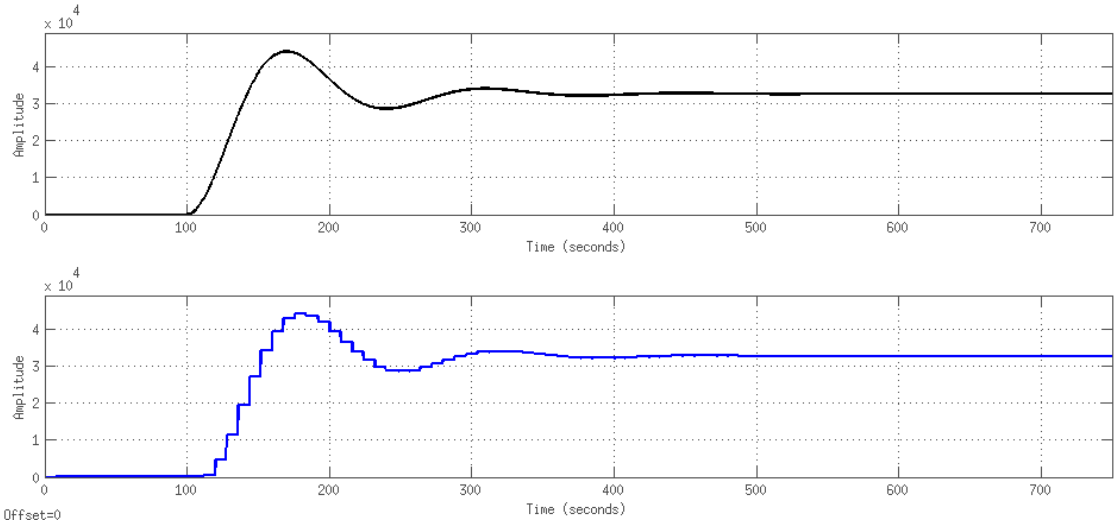

Figure 9.21 – Underdamped Oscillation, Mode 1 Tandem ADC Response

This conversion appears to be comparable to—if not better than—the dynamic-mode conversion at first glance, but the initial rise exhibits a significant amount of delay error, as shown in Figure 9.22:

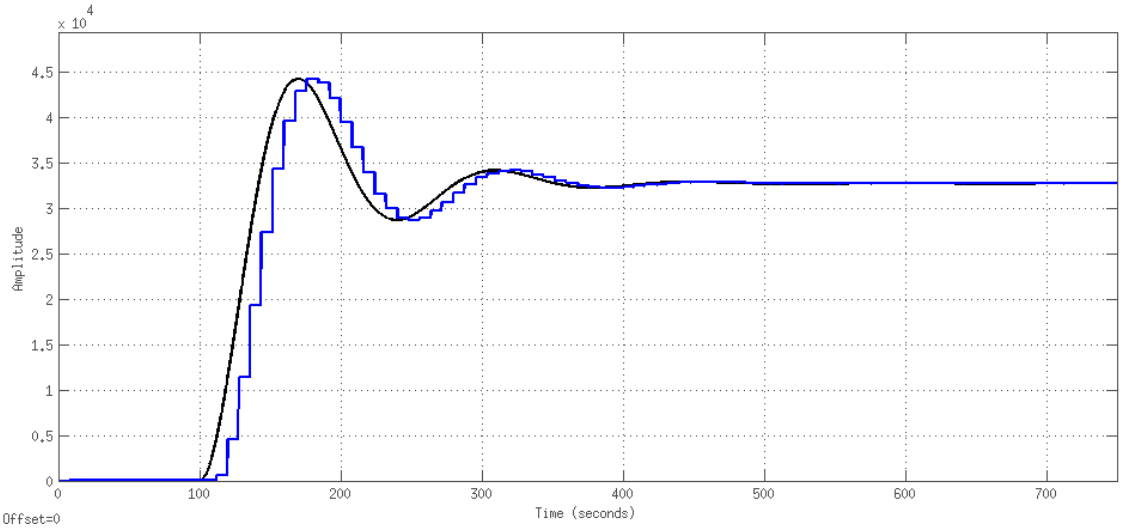

Figure 9.22 – Superimposed version of Figure 9.21

For a realtime-optimized system this is an inferior conversion output.

The result of the ADC in Mode 2 is shown below:

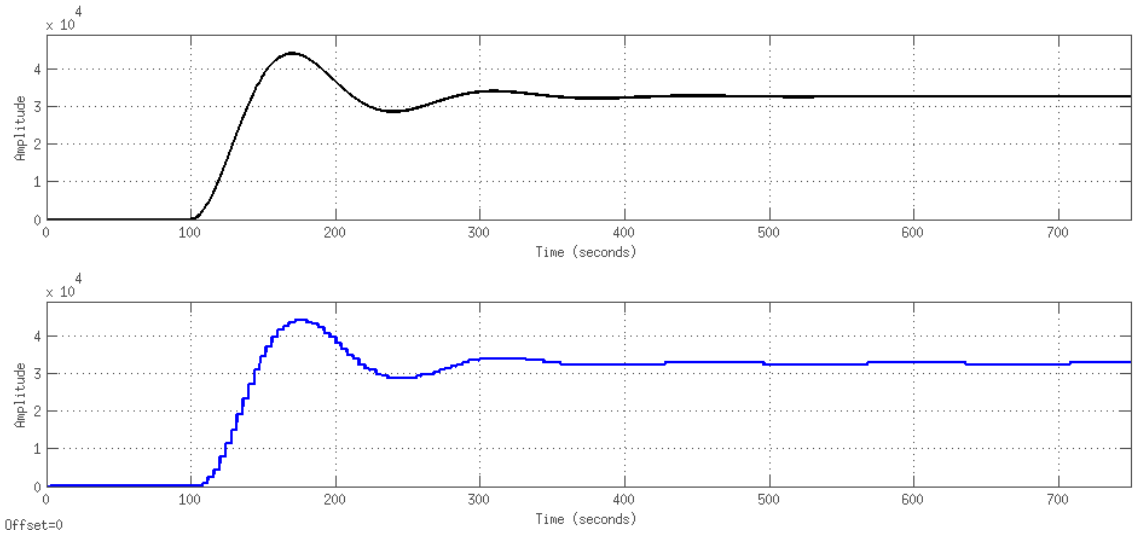

Figure 9.23 – Underdamped Oscillation, Mode 2 Tandem ADC Response

This exhibits better real-time conversion of the initial rise than the Mode 1

response (compare Figure 9.24 to Figure 9.22) but exhibits significant hunting as the input oscillation dies down. This hunting has a magnitude of just under 1% of the Tandem ADC's Full Scale range, and is visible in Figure 9.23 above.

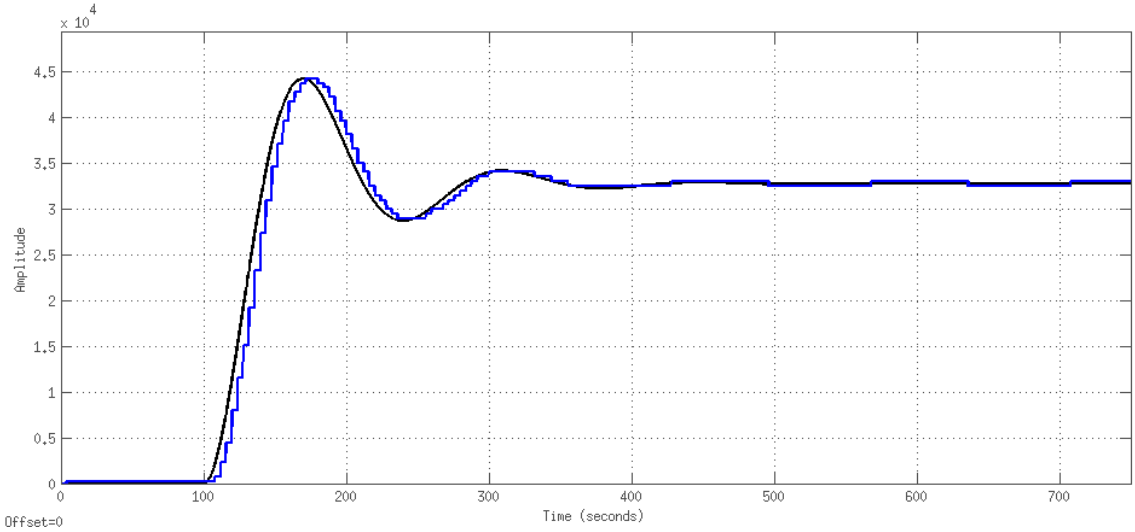

Figure 9.24 – Superimposed version of Figure 9.23

As shown by these tests, no single mode can optimally convert all aspects of an underdamped oscillation input. Mode 1 has too much latency for the initial rise, while Mode 2 exhibits too much hunting as a result of its high quantization error.

Mode 0 was tested as well and found to have the same issues as Mode 1 (but exaggerated), and Mode 3, while not tested, would have undoubtedly exhibited extreme hunting issues, due to its large quantization error (6.25% of full-scale range).

The underdamped oscillation was also tested on the Tandem ADC with its original Mode Hold value of 256. The results are shown below:

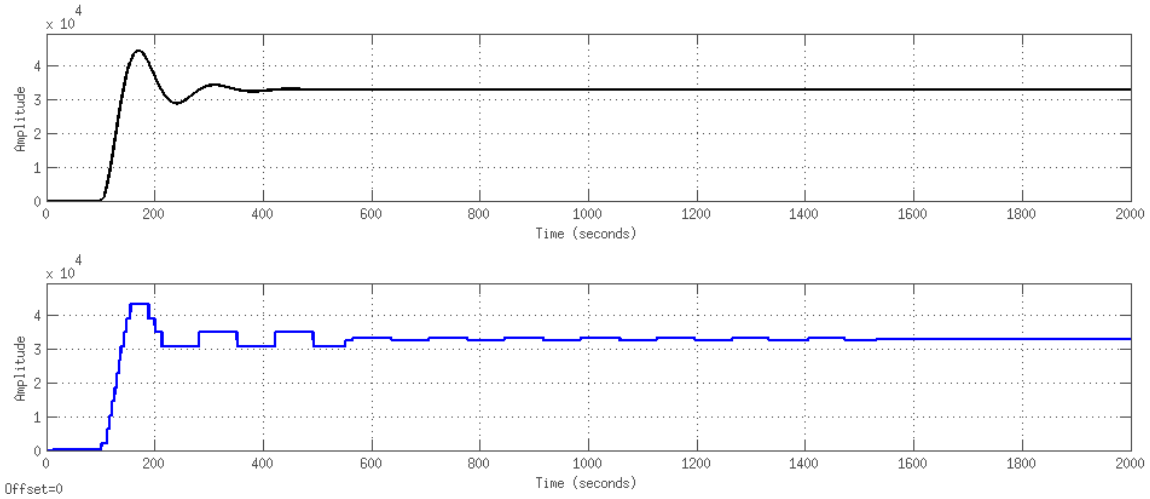

Figure 9.25 – Underdamped Oscillation, Mode Hold = 256

This shows the optimal response to the initial rise, but far too much hunting afterwards. This is explained by the mode response, shown below:

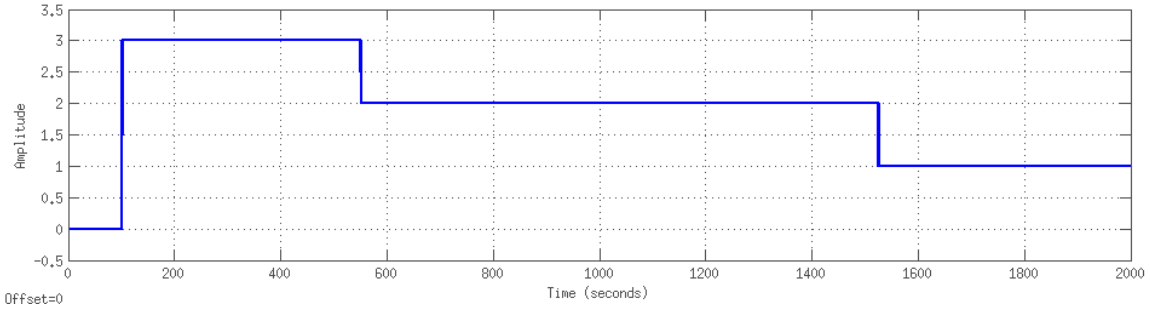

Figure 9.26 – Mode Response of Figure 9.25

This slow descent of the Mode value caused an unnecessary amount of quantization error after the analog input had settled, without much benefit in terms of jitter reduction (compare Figure 9.26 to Figure 9.20). This is strong evidence that 256 conversion cycles is not always the best length of time for the mode hold, as in this case it's far too long.

On the other extreme, the output without any mode hold is shown below:

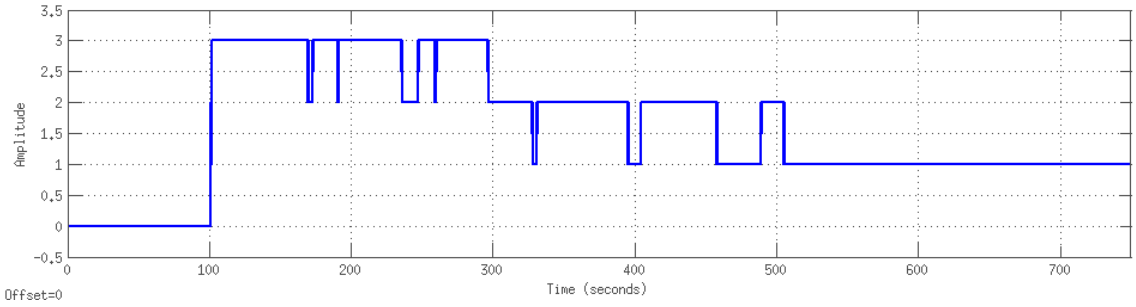

Figure 9.27 – Mode Response to Underdamped Oscillation without Mode Hold

A mode hold value of 4 (Figure 9.20) carries much of the useful information kept in this waveform while eliminating most of the jitter. A hold of 256 (Figure 9.26) has absolutely no jitter in this case, but keeps the mode high for several times as long as it should. (Note that the time scale is expanded from 750 clock cycles to 2000 clock cycles for Figures 9.25 and 9.26.)

9.4 Analysis & Conclusions

It appears the Tandem ADC does possess new utility, and can accomplish some tasks that neither a Flash nor a Successive Approximation ADC could perform on their own. The tandem combination of a Flash and a Successive Approximation converter allows SAR-like operation with three more bits of precision than a standard SAR could find for any given number of clock cycles. The Tandem ADC's variable precision also allows it to select a precision/latency trade-off which provides the best real-time accuracy, and this has been demonstrated to work with a few chirp inputs as well as a converging random walk.

However, the utility of the Tandem ADC is limited to certain applications. As it

has been implemented (i.e. with real-time optimized mode switching points) it works best for control systems where reducing latency is crucial. It does not handle step responses well (as demonstrated by Figure 9.6) and so it may be better suited for reading an error signal, or reading an analog signal from a controlled plant.

Figure 2.1 Shows what is perhaps the best usage of the Tandem ADC: in that control system, the system input is already digital, and the DSP takes two inputs instead of a single error signal. This setup allows the analog sensor of the physical plant to connect directly to the Tandem ADC, which then deals solely with reading plant behavior rather than interpreting a human-controlled input. This gives the ADC an environment which is unpredictable and will likely have some resemblance to a converging random walk, and is also free of step inputs.

With regards to the physical aspects of a control system, the Tandem ADC is likely best suited for a system with external interference, such as a servomotor which must hold itself steady despite being jostled by its environment. The Tandem ADC could respond quickly to such physical interference and allow the system to correct itself in a minimal amount of time.

9.5 Improvements & Future Work

One way to expand the capabilities of the Tandem ADC would be to explore different mode optimization schemes, with real-time optimization on one end of the spectrum, and sampling as close to the Nyquist rate as possible on the other. It may also be a good idea in many cases to disable Mode 3 entirely, since its quantization error is so large (>5% of full-scale range). The mode hold length is another thing which can be

tweaked; the implemented value of 256 conversion cycles avoids mode jitter in almost all situations, but it also keeps the system in higher-than-optimal modes for extended periods of time. When tested, a Mode Hold of 4 conversion cycles caused minimal jittering for oscillatory responses (Sections 9.2.3 and 9.3), but it may still cause jittering for other inputs. (Preliminary testing, which resulted in the Mode Hold value of 256, showed significant jitter for high-amplitude sinusoidal inputs with lower Mode Hold values.)

There may also be ways to improve performance in general. One potential modification would be the replacement of the so-called insignificant bit with a more conventional rounding scheme. This would be difficult to implement on a variable-precision system, since the biasing involved changes depending on precision, but it is almost certainly doable. One possibility is to modify the SAR's internal DAC to subtract half of the LSB value from its output, which would cause round-to-nearest operation. This would enable the Tandem ADC to add another significant bit to each of its modes. Furthermore, if the 4-bit Flash ADC were replaced with a 5-bit Flash ADC, round-to-nearest operation—along with the extra bit of precision—could be implemented without increasing conversion time or latency.

There are likely other possible improvements as well. Two things come to mind are the structures of the Flash Corrector and the Successive Approximation Register. Both have been gate-optimized for minimal latency, but it may be possible to improve their performance further with different logic schemes.

There is also the need for more quantitative testing. Much of the Tandem ADC testing, while informative, was qualitative, and therefore difficult to translate to specs on

a datasheet. Specifications such as INL and DNL are particularly difficult to quantify since they tend to be geared for a fixed-precision converter.

References

[1] C. Sandner, M. Clara, A. Santer, T. Hartig, F. Kuttner, “A 6bit, 1.2GSps Low-Power Flash-ADC in 0.13µm Digital CMOS”, *Infineon Technologies Austria, Development Center Villach*.
Online: **http://arxiv.org/pdf/0710.4838.pdf**

[2] Maxim Integrated, “Understanding Flash ADCs”, APP 810, Sep 16 2010.
Online: **http://www.maximintegrated.com/app-notes/index.mvp/id/810**

[3] Maxim Integrated, “Understanding SAR ADCs: Their Architecture and Comparison with Other ADCs”, APP 1080, Oct 02 2001.
Online: **http://www.maximintegrated.com/app-notes/index.mvp/id/1080**

[4] All About Circuits, “Flash ADC”, Volume IV – Digital > Digital-analog Conversion. Accessed Mar 2014.
Online: **http://www.allaboutcircuits.com/vol_4/chpt_13/4.html**

[5] J. Errington, “Analog to Digital Converters”. *Skillbank, John Errington’s Data Conversion Website*. Accessed Jan 2014.
Online: **http://www.skillbank.co.uk/SignalConversion/adc.htm**

[6] A. G. F Dingwall, V. Zuzzu, “An 8-MHz CMOS Subranging 8-Bit A/D Converter”. *IEEE Journal of Solid-State Circuits*, vol. sc-20, no. 6. December 1985.
Online: **http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1052451**

[7] Maxim Integrated, “Understanding Pipelined ADCs”, APP 2013, Oct 2 2001.
Online: **http://www.maximintegrated.com/app-notes/index.mvp/id/1023**

[8] P. Poshala, “Why Oversample when Undersampling can do the Job?”, *Texas Instruments Application Report*. July 2013.
Online: **http://www.ti.com/lit/an/slaa594a/slaa594a.pdf**

[9] W. Kester, “Find Those Elusive ADC Sparkle Codes and Metastable States”.

Analog Devices, MT-011 Tutorial. Accessed Mar 2014.
Online: **http://www.analog.com/static/imported-files/tutorials/MT-011.pdf**

[10] EEWeb, "Filter Topology - Chebyshev, Butterwroth, and Bessel", *Electronics Quis of the Day*. March 11, 2011.
Online: **http://www.eeweb.com/electronics-quiz/filter-topology-chebyshev-butterworth-and-bessel**

[11] Odelica. "ZerosAndPoles.Design.Filter Information", *Modelica Linear Systems 2.* Accessed March 2014.
Online: **https://build.openmodelica.org/Documentation/Modelica_LinearSystems2.ZerosAndPoles.Design.filter.html**

[12] All About Circuits, "Successive Approximation ADC", Volume IV – Digital > Digital-analog Conversion. Accessed Mar 2014.
Online: **http://www.allaboutcircuits.com/vol_4/chpt_13/6.html**

[13] S. Mortezapour and E.K.F. Lee, "A 1-V, 8-Bit Successive Approximation ADC in Standard CMOS Process", *IEEE Journal of Solid-State Circuits*, vol. 35, no. 4, April 2000. Online: **http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=839925**

[14] G. Palumbo and M. Pennisi, "Design Guidelines for High-Speed Transmission-Gate Latches: Analysis and Comparison", *University of Catania, DIEES; IEEE Journal, Circuits and Systems 2008 p. 145-148.*
Online: **http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=4674812**

[15] Maxim Integrated, "What is a Transmission Gate (Analog Switch)?" APP 4243, Jun 10 2008. Online: **http://www.maximintegrated.com/app-notes/index.mvp/id/4243**

Appendix A
Simulink Model

Entirety of Simulink Model shown below, roughly analogous to Figure 3.1.

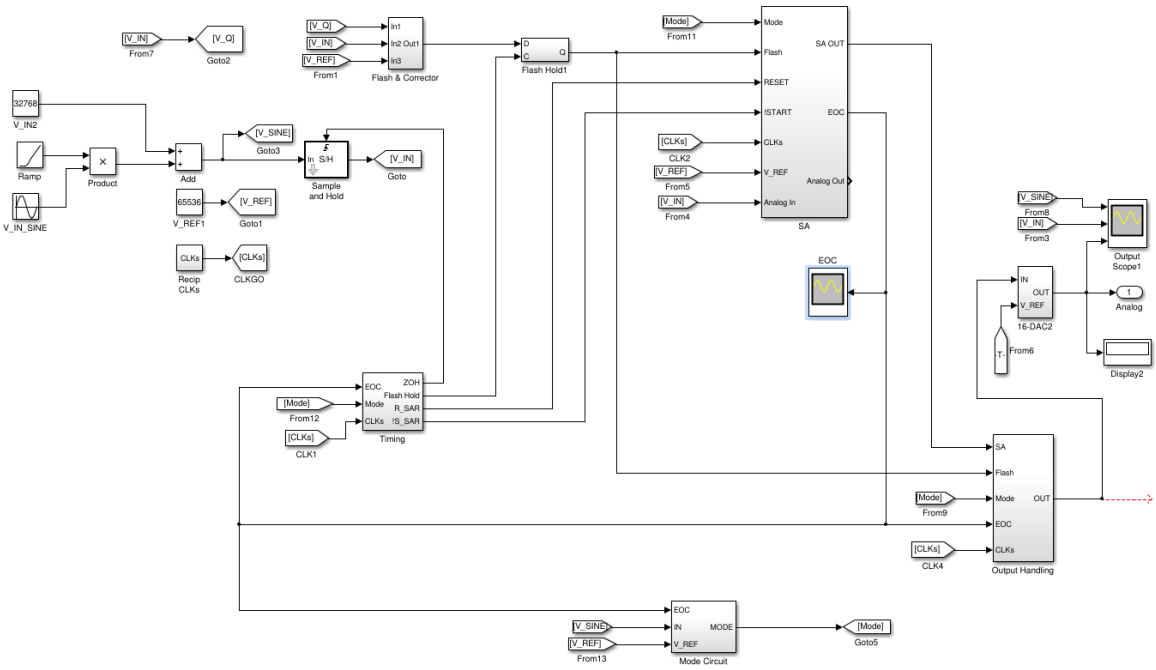

Figure A.1 – Entire Tandem ADC

On the far left is the input synthesizer, creating a sine wave. On the far right is the system data output, shown as a red dashed line. Above that is a diagnostic circuit for aquiring the analog value of the digital output.

A.1 Flash System

This section roughly corresponds to Section 6.2 of the main text. The entire Flash system (except for the Flash Hold) is shown below in Figure A.2:

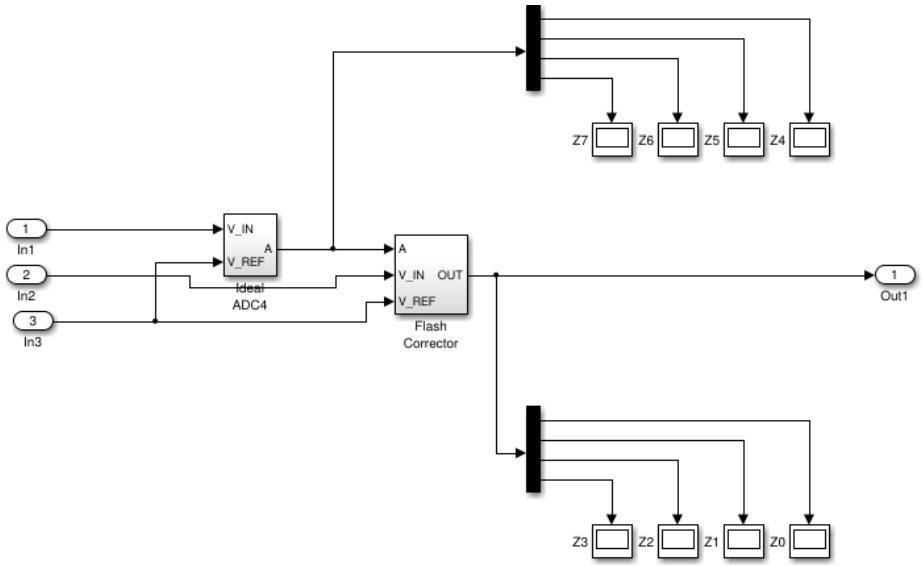

Figure A.2 – Flash System

Input 2 is the system analog input (after ZOH), while Input 1 is a potentially-corrupted version of the analog input to simulate a Flash ADC error. The eight boxes on the right are used to display the Flash values, before and after correction.

Flash Corrector is shown below in Figure A.3. (This corresponds to Figure 6.3.)

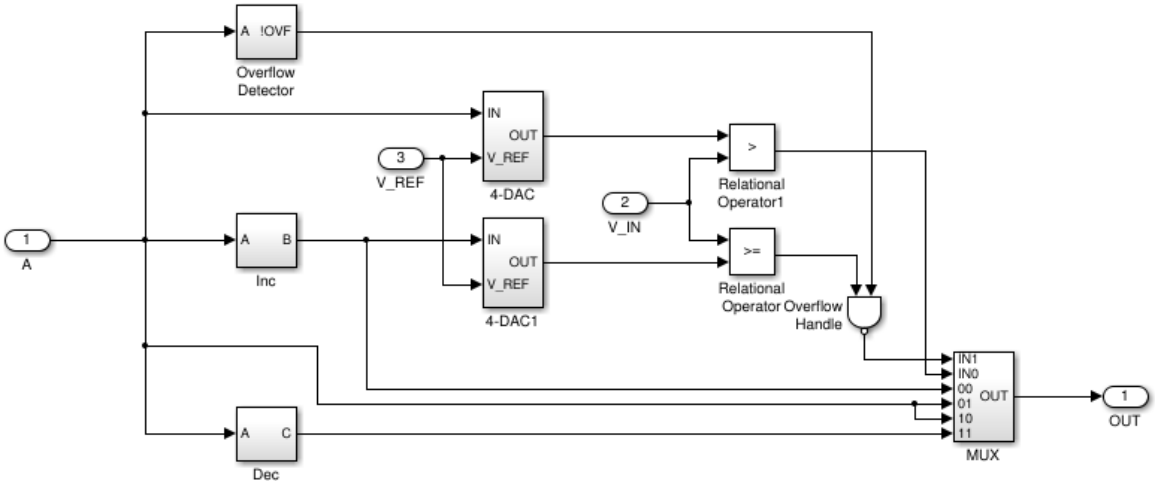

Figure A.3 – Flash Corrector

The two Relational Operators are simple comparator models, and the DACs are ideal models as well.

The Incrementor (Inc) is shown below in Figure A.4. (This corresponds to Figure 6.4.)

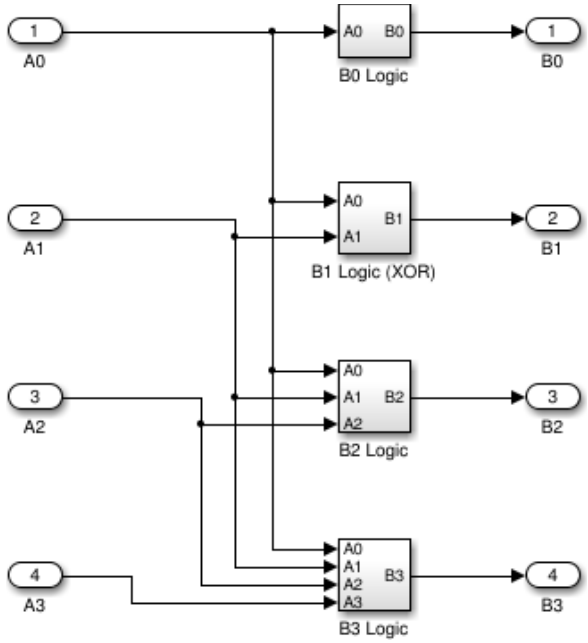

Figure A.4 – Incrementor

The logic blocks (B0-B4) are shown below:

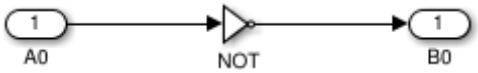

Figure A.5 – Block B0, LSB

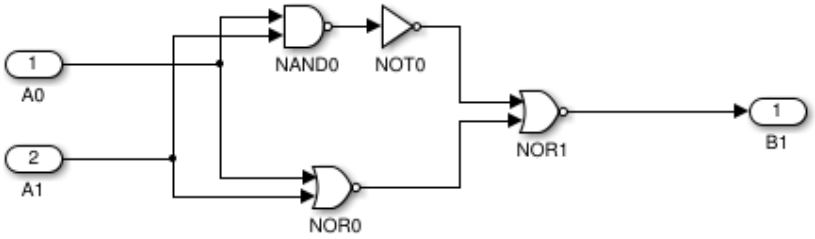

Figure A.6 – Block B1

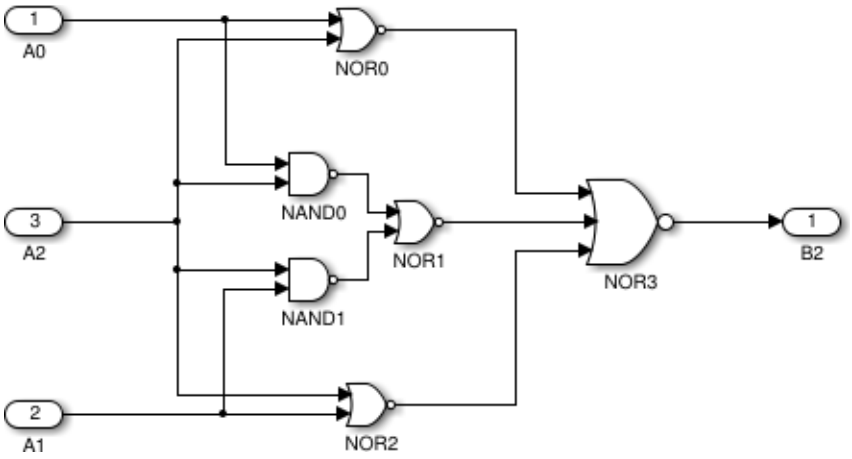

Figure A.7 – Block B2

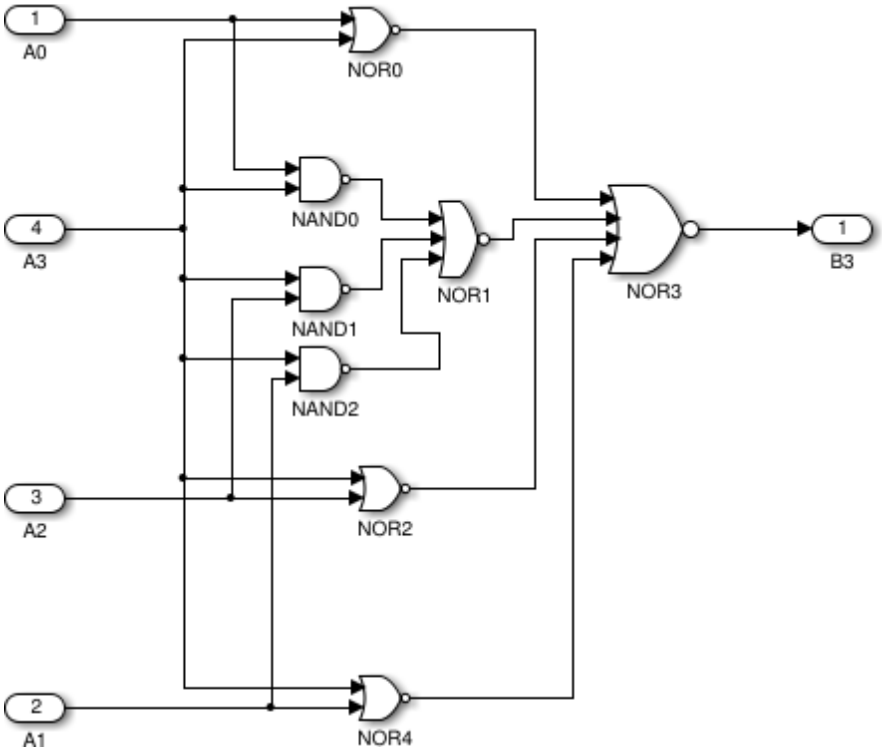

Figure A.8 – Block B3

All of these are some variation on an XOR gate.

The Decrementor (Dec) is shown below in Figure A.9. (This corresponds to Figure 6.5.)

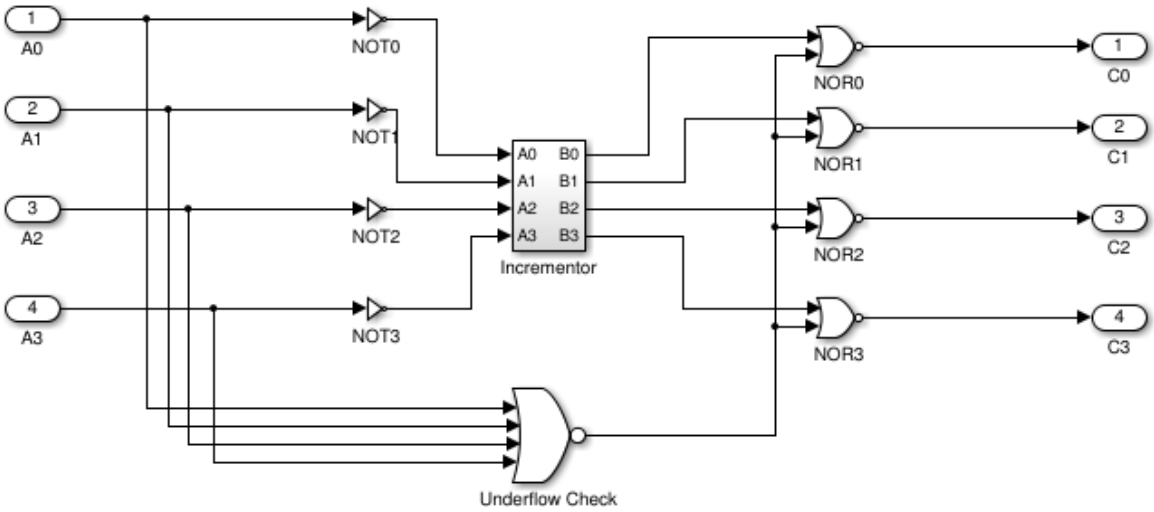

Figure A.9 – Decrementor

The MUX is shown below in Figure A.10.

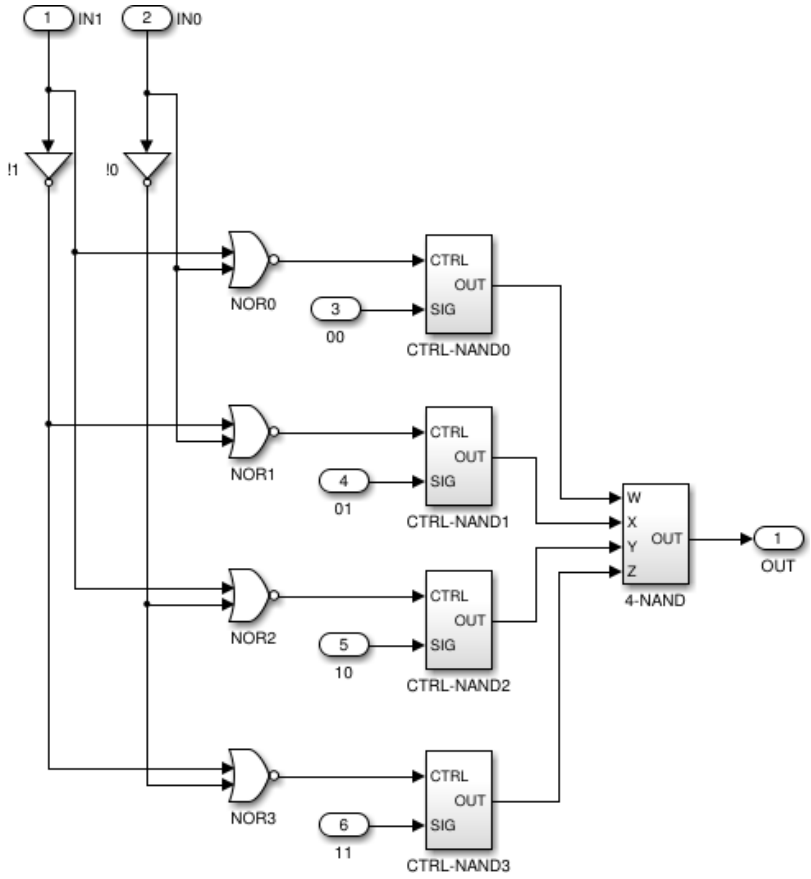

Figure A.10 – Mux

The “CTRL-NAND” blocks are simply NAND gate arrays:

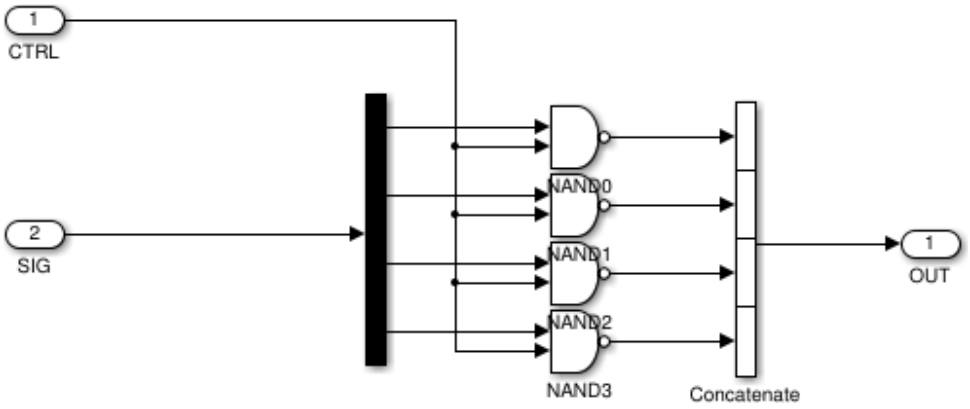

Figure A.11 – CTRL-NAND Array

These are used to blank out (change to `1111`) signals which are not selected. The four signals—the selected signal (inverted and three blanked-out signals, are then sent to the 4-NAND block, shown below:

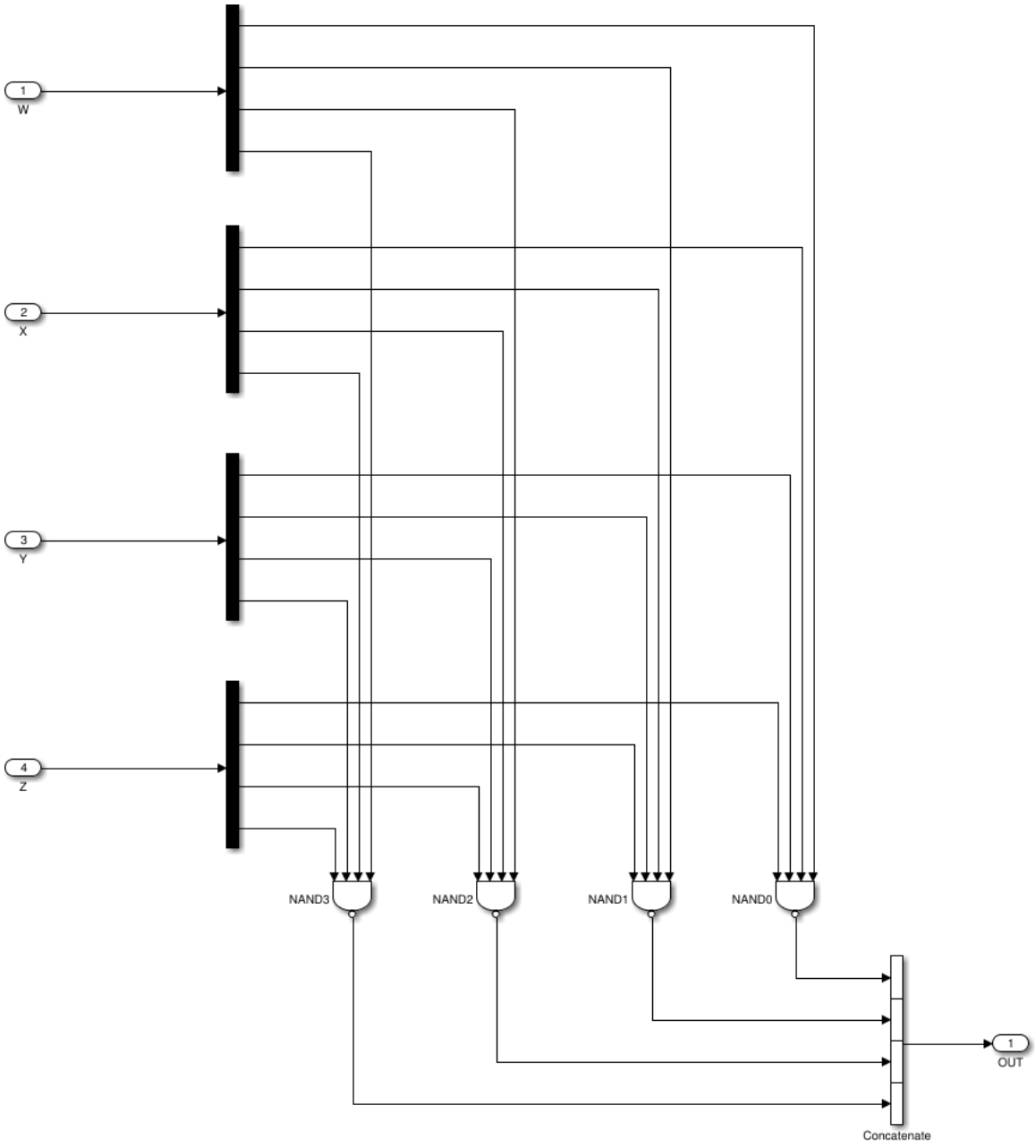

Figure A.12 – 4-NAND Array

This block NANDs the four input signals, yielding the selected signal, non-inverted. (In a non-optimized topology, CTRL-NAND would use AND gates instead, and 4-NAND would use OR gates. In that situation CTRL-NAND zeros out non-selected signals, and 4-NAND performs a logical OR on each bit. This propagates the selected

signal along the same way, but without the double inversion.)

Below is the Flash Hold block. It exists outside the Flash System block in the Simulink model.

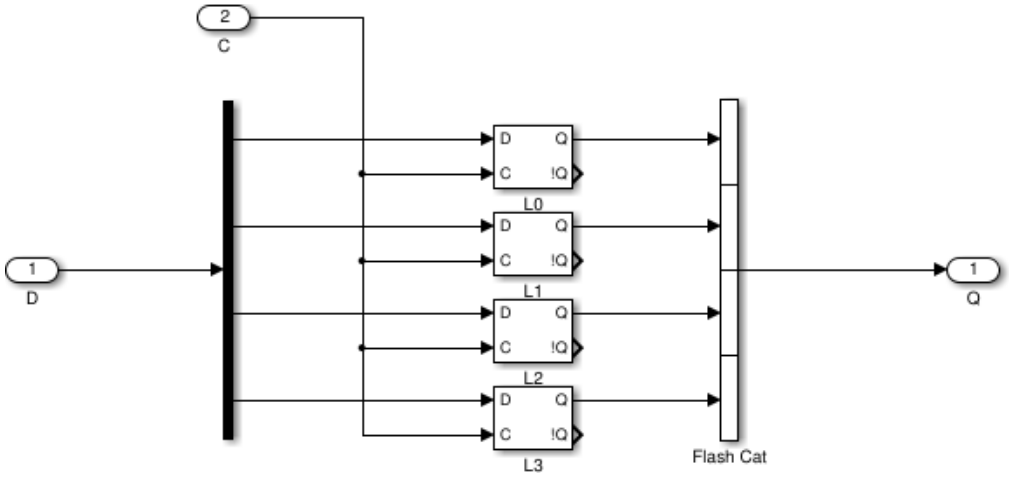

Figure A.13 – Flash Hold

It is just an array of D latches, controlled by the Flash Hold signal.

A.2 Successive Approximation

This section corresponds to Section 6.3 of the main text. The Successive Approximation logic is shown below in Figure A.14:

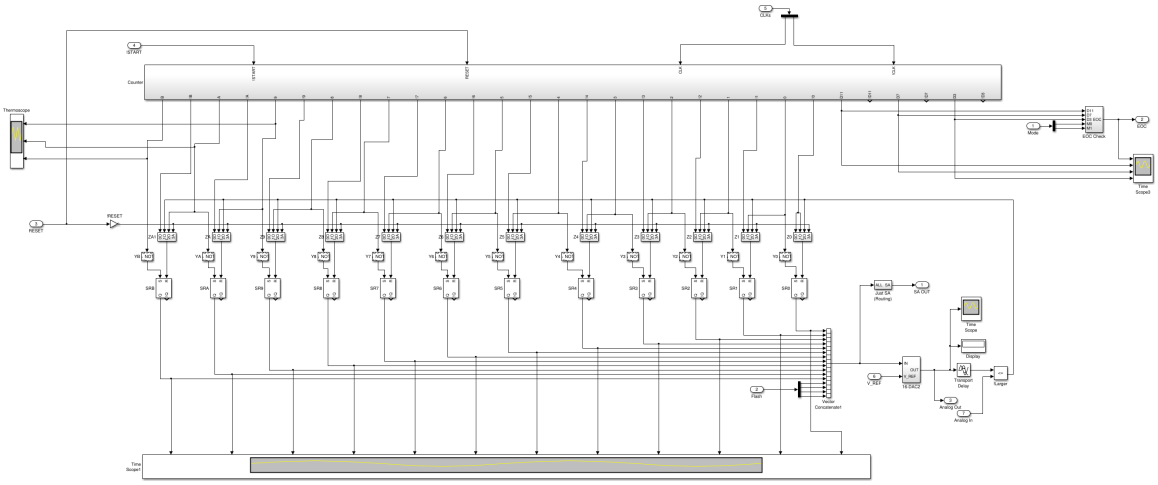

Figure A.14 – Successive Approximation Block

This subsystem contains everything shown in Figure 6.6, also showing the detail of Figure 6.7. The blocks labeled Z9, Z8, Z7, etc. are or-and-invert gates, and the blocks labeled Y9, Y8, Y7, etc. are rising-edge detectors. The large block on top is a thermometer counter, and the small block in the upper-right is the EOC output circuit. The former is shown in Figure A.15 below:

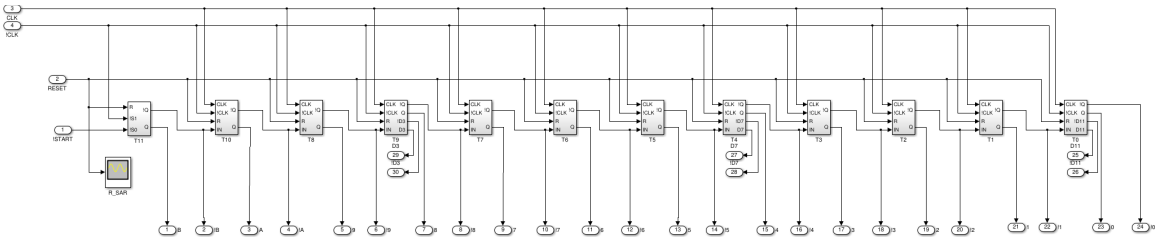

Figure A.15 – SA Thermometer Counter

After the start signal is given, a new output goes high on each rising edge. There is sometimes such an output on a falling edge as well, used for EOC triggering. Either way, each block in the above figure corresponds to one full clock cycle, the ones with falling edge functionality having an extra pair of complimentary outputs. Either way, both blocks are a variation of Figure A.16.

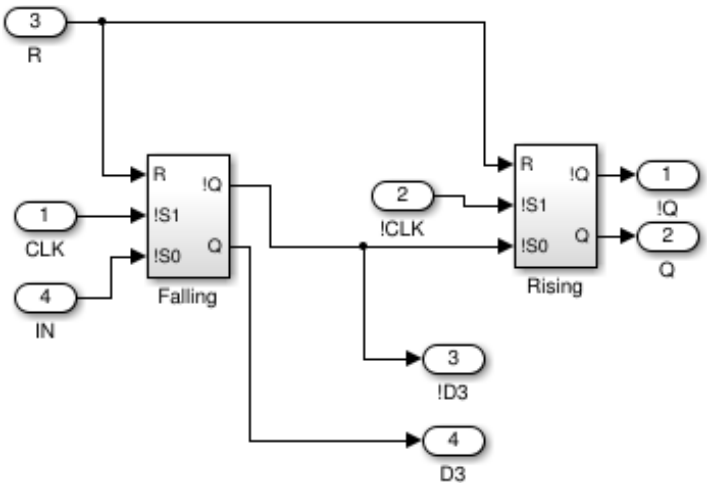

Figure A.16 – Thermometer Counter "T" Block

The blocks with rising-edge and falling-edge functionality look exactly like this figure, those with only rising-edge outputs lack outputs D3 and !D3. The two blocks in the above figure are identical, and have the topology shown in Figure A.17.

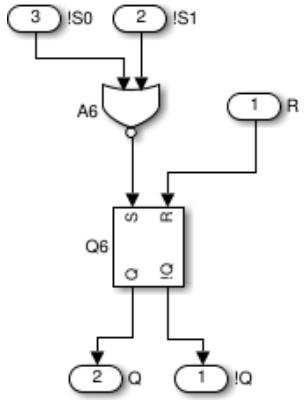

Figure A.17 – Rising/Falling Block

This is nothing more than an SR latch with a NOR gate on the S input. One of the inputs is the clock (or its reciprocal) which causes the signal to propagate only when the clock ticks. Falling's output goes high on the downtick, Rising's output goes high on the uptick.

The only other major sub-block the Successive Approximation block has is EOC Check, shown below in Figure A.18:

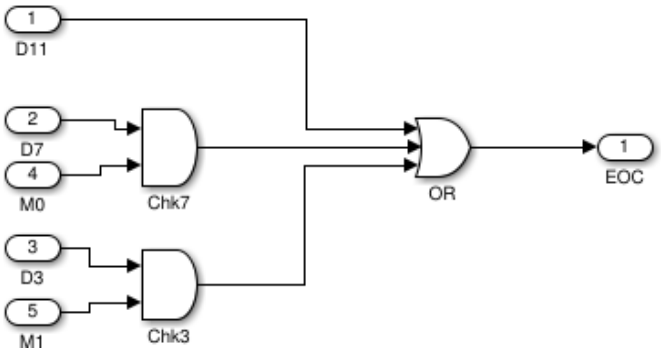

Figure A.18 – EOC Check

This sends an EOC signal after three conversions for Mode 2 (Mode ≥ 2,

bypassed during Mode 3) or after seven conversions for Mode 1 (Mode is odd, bypassed during Mode 3) or after eleven conversions for Mode 0 (Mode is not actually checked; instead this only gets a chance to happen if EOC is not triggered by one of the two previous conditions).

A.3 Control Block

The Control Block, described in Section 6.4, consists of the Timing Block, the Mode Circuit Block, and the Output Handling Block, which are all found on the top level of the Simulink model (Figure A.1).

A.3.1 Mode Block

The mode block is described in Section 6.4.1. Its Simulink model is shown below:

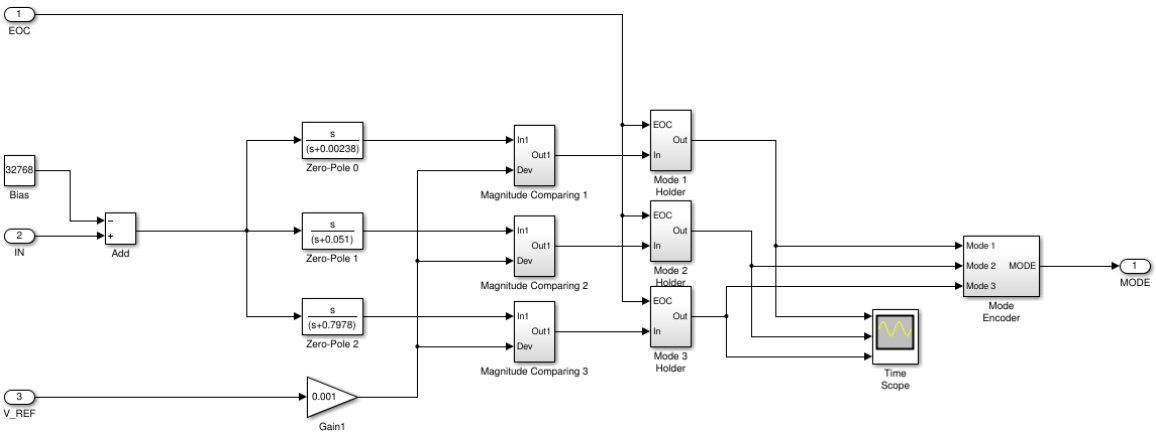

Figure A.19 – Mode Block

This closely follows Figure 6.9, although there are a few differences to smooth out the simulation, such as the bias added to the signal to allow steady-state operation for any given input to be reached sooner.

The threshold detectors (all identical) are shown below:

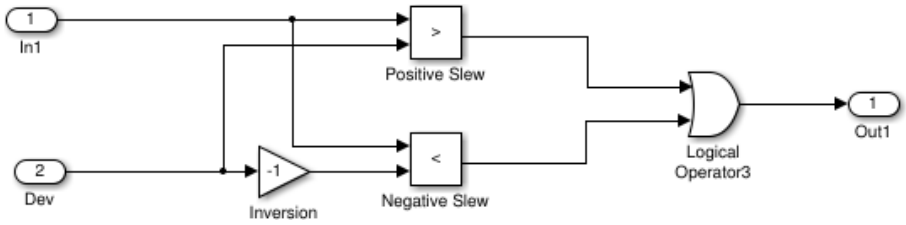

Figure A.20 – Slew Threshold

These detect both positive and negative slew rates, based on the input threshold value.

The mode hold models (all identical as well) are shown below:

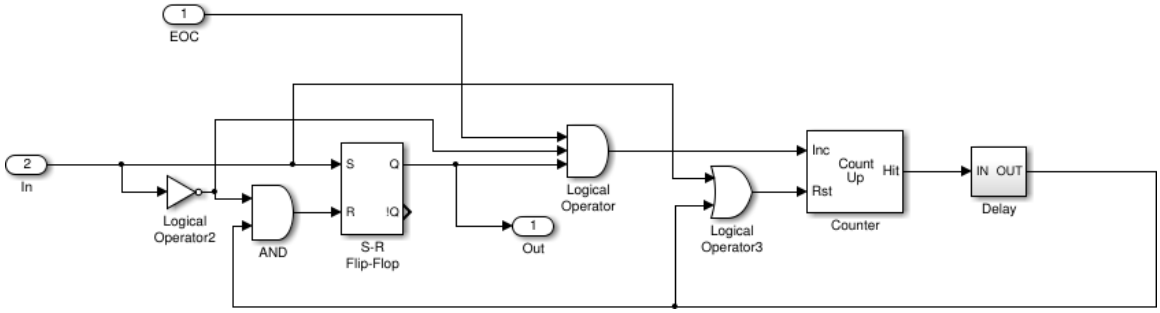

Figure A.21 – Mode Hold

This uses a counter to decrement the Mode after a certain number of conversion cycles, with various inputs to disable/reset the count.

The Mode Encoder is a simple thermometer-to-standard binary converter:

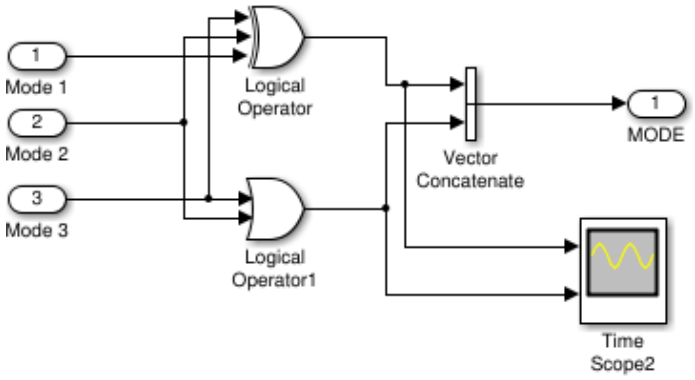

Figure A.22 – Mode Encoder

A.3.2 Timing Block

The Timing Block controls the overall system timing, and is described in Section 6.4.2. The Simulink model is shown below:

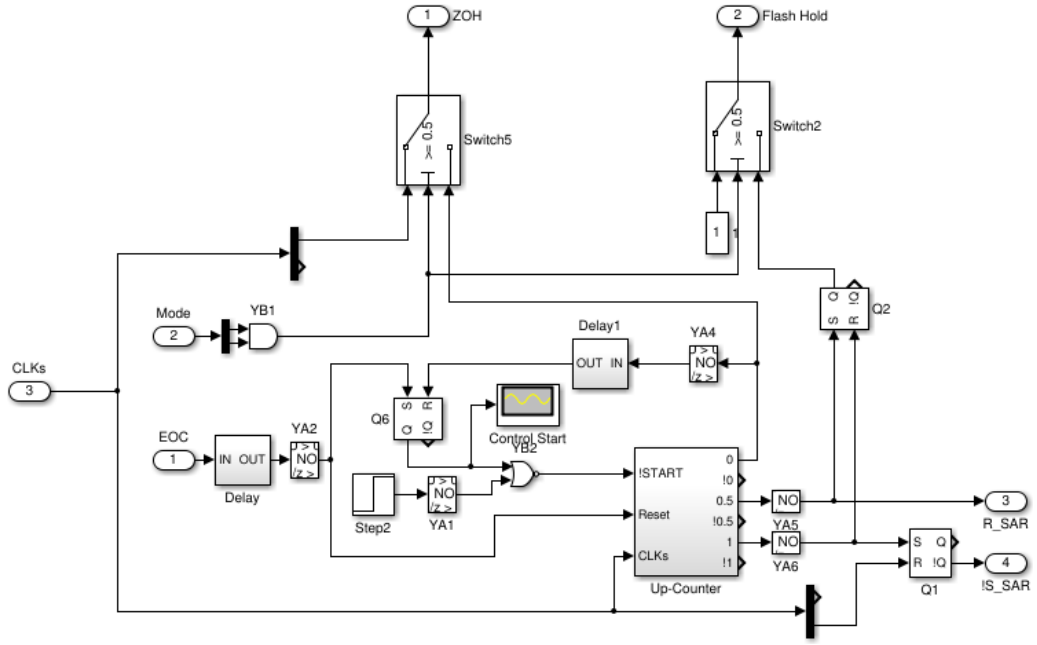

Figure A.23 – Timing Block

This corresponds to Figure 6.12, although it shows a bit more detail in certain areas. The thermometer-coded Up-Counter uses the same topology as the thermometer counter in the Successive Approximation block (see Section A.2), although the output numbers are in reverse order.

A.3.3 Output Handling

The Output Handling block is described in Section 6.4.3; its Simulink model is shown below:

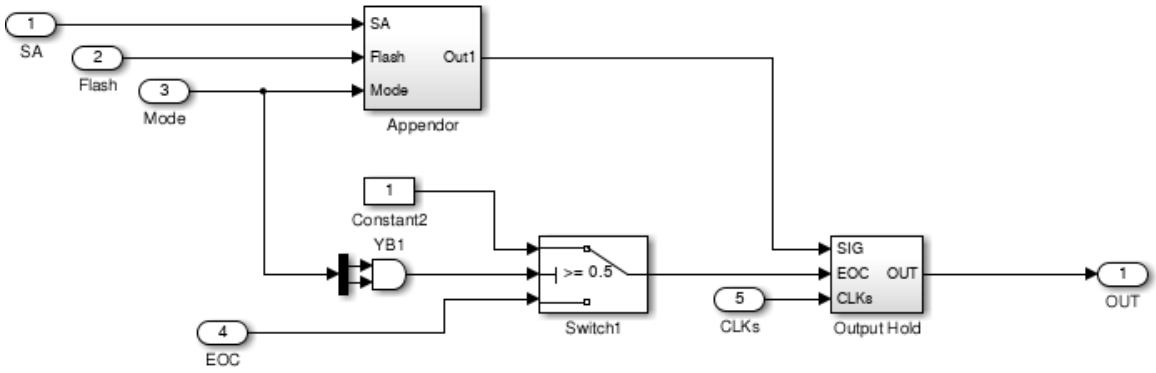

Figure A.24 – Output Handling

This corresponds to Figure 6.13 in the main text, although the OR gate is implemented with a transmission gate switch. This reduces propagation delay of the EOC signal when it is used (i.e. when not in Mode 3).

The Appendor is shown on the next page:

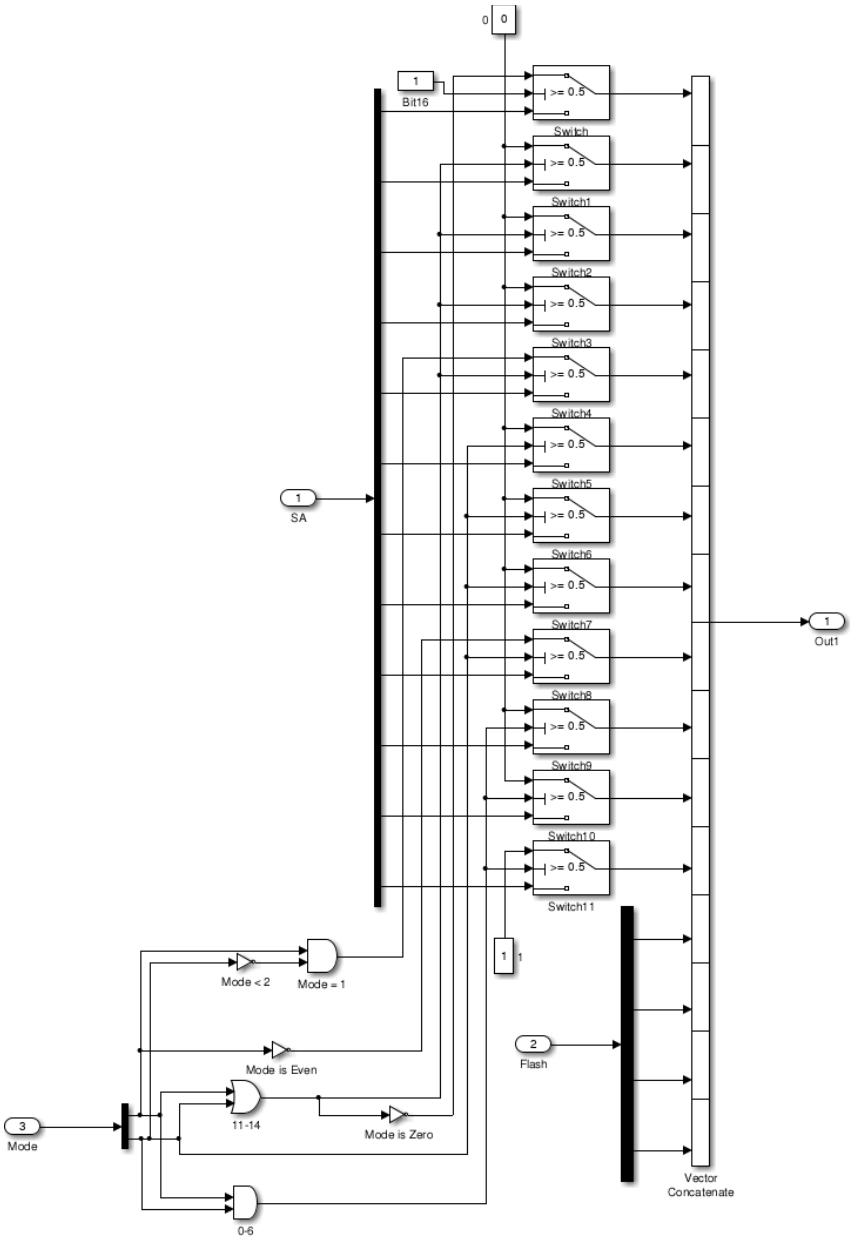

Figure A.25 – Bit Appendor

This appends the insignificant bit using transmission gate switches.

The only other part of the Output Handling block is the Output Hold, shown in Figure A.26.

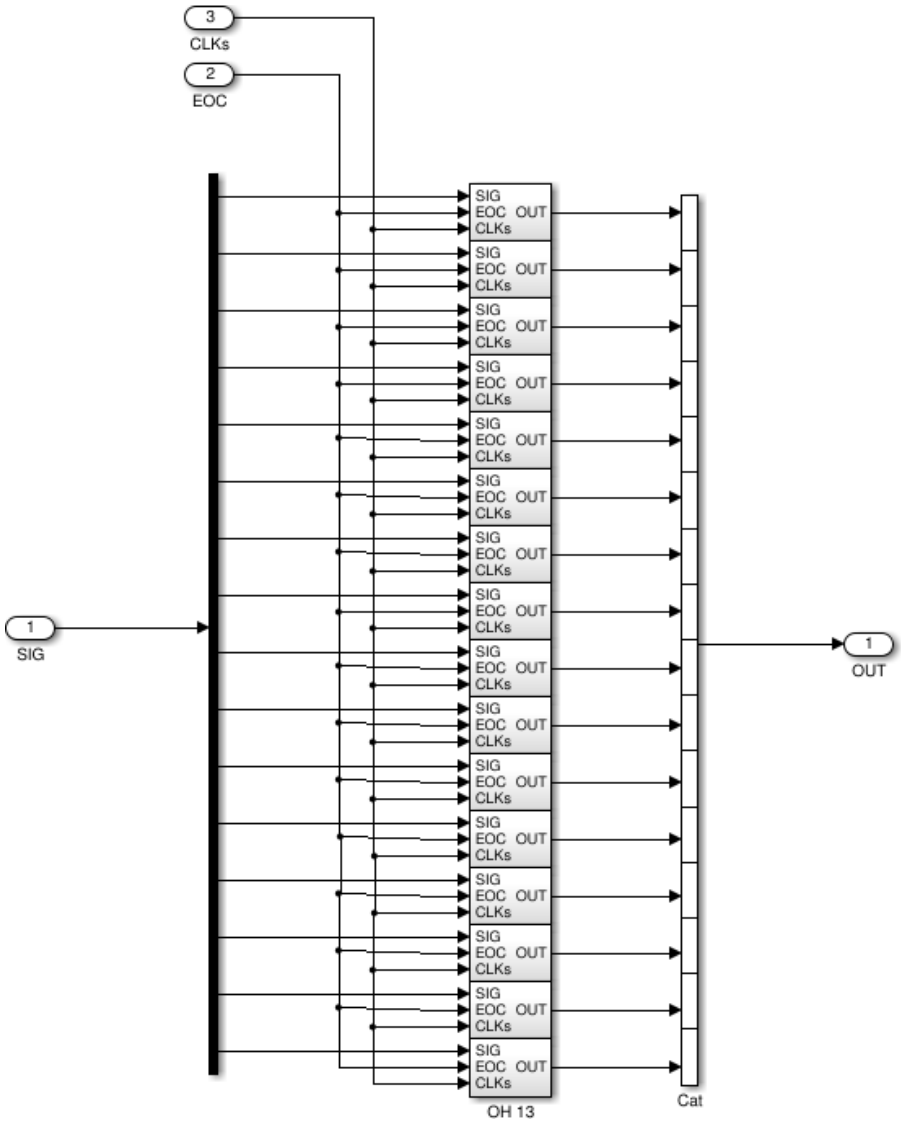

Figure A.26 – Output Hold

This holds the output steady at the end of each conversion cycle. Each of the blocks shown above are identical, and contain the following:

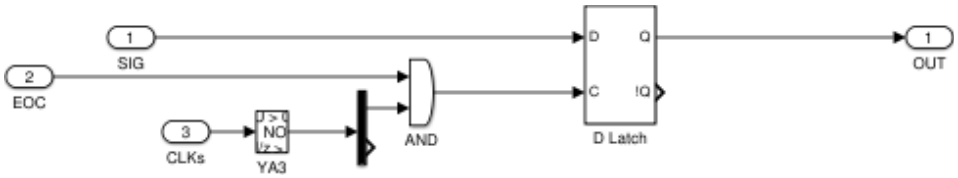

Figure A.27 – Output Hold Sub-Block

These update the output signal based on the clock and the EOC signal.

Appendix B
Matlab Code

B.1 Black Box Code & Error Calculations

Due to the lengthy computation times of the Simulink model, a black box model was coded in Matlab. Its Full Scale Range goes from 0 to 1, rather than the Simulink model's 0 to 65,536. (See Section 9.2 for more on the motivation and details of the black box model and how it differs from the Simulink model.)

ADC.m: Generates Tandem ADC result at constant argument-specified precision level.

```
function [ B ] = ADC(A,r)
% Tandem ADC Black Box
% A: Input
% B: Output
% r: Bits of presicion
% For modes 0-3, r = 15, 11, 7, 4 respectively

r = max(min(r,15),4); % Range of precision values is 4-16 bits

z = length(A);

t = r-3; % Conversion time (clock cycles)

d = 0.5^r; % Analog equivalent Value of LSB

k2 = 0; % Counter for input (k is counter for output)

B = zeros(1,z); % Initialize Output

for k=r:z-1 % Zero-referencing.
    k2 = (floor(k/t)-1)*t; % k2 = round down of k
    B(k+1) = A(k2+1) + 1; % k, k2 effect ZOH. Adds 1 to allow mod() to work
    B(k+1) = B(k+1) - mod(B(k+1),d) - 1; % Round down, subtract 1 again
    if B(k+1) < 0 % Underflow compensation: 0 is minimum
        B(k+1) = 0; % Set to minimum
    end
    if B(k+1) >= 1 % Overflow compensation: 1-d is maximum
        B(k+1) = 1 - d; % Set to maximum
    end
    B(k+1) = B(k+1) + d/2; % Add insignificant bit
end

end
```

This can generate any resolution-locked result; it is only used for the four modes actually implemented, but is not constrained to them. Provides a black box for the functionality of the Tandem ADC excluding mode selection.

Mode.m: Finds system mode based on slew rate, performs mode hold as well.

```
function [ M, L ] = Mode( A )
% Mode Selector %

z = length(A);

% Mode cutoff points
S = [0.000002385367*2*pi 0.0000510013865*2*pi 0.0007977959152*2*pi];

% Mode hold length (in conversion cycles)
C = 256;

% Mode hold length (in clock cycles) for modes 1, 2, 3
W = [C*8 C*4 C];

% Mode without hold initialization
L = zeros(1,z);

for k = 2:z % One-referencing
    d = abs(A(k)-A(k-1)); % Derivative (ideal)

    % Mode 1 check
    if(d > S(1))  % threshold
        L(k) = 1; % set mode to 1
    end

    % Mode 2 check
    if(d > S(2))  % threshold
        L(k) = 2; % set mode to 2
    end

    % Mode 3 check
    if(d > S(3))  % threshold
        L(k) = 3; % set mode to 3
    end
end

M = L; % Held mode starts out as unheld mode

hold = 0;

for k = 2:z % iterate through unheld mode to effect hold

    m = zeros(1,3); % reset Mode candidates for each iteration

    % Look back for each mode, see if was reached in un-held Mode array
    for n = 1:3
        w = sum(W(n:3)); % w = total lookback time for given mode
        % look back for max mode in last k-w, limit to value of current mode
        m(n) = min(max(L(max(1,k-w+1):k)),n);
    end

    M(k) = max(m); % Held mode is max of three mode values triggered

end

end
```

This function has variable internal parameters which allow mode switching metrics to be changed, although they were kept constant for most of system testing.

AllModes.m: Script for finding Tandem ADC output for the four implemented modes, and combining them based on the mode selector.

```
% Script for creating outputs for Tandem ADC in all four modes,
% and composite (the actual output) based on these values.

l = length(A);

ADC0 = ADC(A, 15); % Mode 0
ADC1 = ADC(A, 11); % Mode 1
ADC2 = ADC(A,  7); % Mode 2
ADC3 = ADC(A,  4); % Mode 3

ADC_ALL = [ADC0; ADC1; ADC2; ADC3]; % Matrix of ADCs

% MODE is mode sequence with mode hold, L is without hold
[MODE, L] = Mode(A);

ADC_OUT = zeros(1,l); % Tandem ADC output initialization

% Iteration through time values to select output based on mode
for k = 1:l

% Selects mode at each point in time
% '+1' compensates for one-referencing
    ADC_OUT(k) = ADC_ALL(MODE(k)+1,k);

end
```

This function calls **ADC.m** four times with input r=15, r=11, r=7, and r=4, once each. Calls **Mode.m** to determine which mode to use at each point in time, then combines mode-locked outputs to produce a dynamic-mode output (the actual Tandem ADC output). Although mode switching is applied after the fact, the mode-determining algorithm works in a causal manner based only on the input signal, and so the dynamic-mode output is achievable with realtime mode switching. Tandem ADC output is `ADC_OUT`, or `ADC_ALL(4,:)`.

RollingMSE.m: Calculates ADC square error as a rolling average. Uses a Von Hann windowing function for weighted averaging.

```
function [ E ] = RollingMSE( A, B, width )
% Find Mean Square Error as a moving average.
% A: input 1
% B: input 2
% width: rolling average length
% E: error output

l = length(A);
```

```
q = length(B);

if(l ~= q) % Ensure A and B are the same length
    E = -1;
    return;
end

w = hann(width+2)'; % Hann windowing function
w = w(2:width+1);   % trim zero-values off of Hann

E = zeros(1,l-width+1); % Initialize error output

F = ((A-B).*(A-B))/(sum(w)); % Instantaneous Square Error, divided by window
    sum

for k = 1:(l-width+1) % Iterate through values of F, stop short of end
    E(k) = sum(w.*(F(k:k+width-1))); % Perform rolling average via window
    correlation

end

end
```

This performs a corellation between the square error of each input and the window function.
Input A is the original analog signal and input B is the ADC output, although the variables are actually interchangeable.

AllError.m: Script for finding rolling average values by calling **RollingMSE.m** once for each output of **AllModes.m**.

```
% Script to find Rolling MSE for all modes (including composite)

E0 = RollingMSE(ADC0,A,Width);
E1 = RollingMSE(ADC1,A,Width);
E2 = RollingMSE(ADC2,A,Width);
E3 = RollingMSE(ADC3,A,Width);
E4 = RollingMSE(ADC_OUT,A,Width);
```

B.2 Input Generation

AmplitudeError.m: Script used for finding mode switching points for system calibration. Generated a swath of constant-frequency sine waves to be fed to the Tandem ADC Black Box in various locked modes.

```
q = 32; % down-scaling for sine wave frequency

a = floor(log(start))*q; % start frequency
b = ceil(log(stop))*q;   % stop frequency

% Initialization
E1 = zeros(1,b-a+1);
```

```
E2 = zeros(1,b-a+1);
E3 = zeros(1,b-a+1);
E4 = zeros(1,b-a+1);
Er = zeros(1,b-a+1);

for k=a:b   % Frequency sweep (geometric)
    A1 = Sine(exp(k/q)); % input signal of given frequency
    B1 = ADC(A1,15); % Mode-locked ADC outputs
    B2 = ADC(A1,11);
    B3 = ADC(A1, 7);
    B4 = ADC(A1, 4);

    Er(k-a+1) = exp(k/q); % Frequency (x coordinate)

    E1(k-a+1) = MeanSquare(A1,B1); % Error (y coordinate)
    E2(k-a+1) = MeanSquare(A1,B2);
    E3(k-a+1) = MeanSquare(A1,B3);
    E4(k-a+1) = MeanSquare(A1,B4);
end
```

This code also measured the MSE of each mode as a function of frequency, which could easily be translated into a function of maximum sinusoidal slew rate since input wave amplitude was constant (SR = ω×A). This was used to generate error data for which the crossover points were optimal for mode switching, as well as Figures 3.2 through 3.7.

Chirp Testing: Chirp waveforms for testing were generated using Matlab's `chirp` function, usually in this sort of manner:

```
for z = 1:5000000
A(z) = chirp(z,0.0001,5000000,0.01);
end
```

This creates a chirp with a wide frequency range, but it's not usually sufficient for measuring more than one mode switch. As a result it was slowed down and/or scaled down to observe transitions between lower modes. This general method was used to generate chirp signals used to create Figure 9.2 through 9.5.

Step Function: This signal is the simplest to create.

```
A = zeros(1,5000);
A(100:5000) = 0.5;
```

There is a little time padding before the step to ensure there's an actual transient, rather than "starting" at the high value. Also, the step only goes to 0.5 because that's right in the middle of the system's full-scale range. This function was used to generate Figures 9.6 through 9.11.

NoiseGen.m: Script for generating white noise, pink noise, and Brownian (random walk) noise. (It should be noted that noise color refers to signal spectral density rather than power spectral density.)

```
for x=1:NoiseLength
    WhiteNoise(x) = rand; % White noise generation
end

WhiteNoise = WhiteNoise – mean(WhiteNoise); % Remove any biasing

% PinkNoise is integral of WhiteNoise
PinkNoise(1) = WhiteNoise(1);

for x=2:NoiseLength
    PinkNoise(x) = PinkNoise(x-1) + WhiteNoise(x);
end

% BrownNoise is integral of PinkNoise
BrownNoise(1) = PinkNoise(1);

pm = mean(PinkNoise); % Mean of Pink Noise (for removal of any biasing)

for x=2:NoiseLength % Integrate, minus the biasing
    BrownNoise(x) = BrownNoise(x-1) + PinkNoise(x) – pm;
end

% TaperedPinkNoise fades from PinkNoise to a constant value gradually
TaperedPinkNoise = PinkNoise.*linspace(1,0,NoiseLength) +
pm.*linspace(0,1,NoiseLength);

% ConvergingBrownNoise is integral of TaperedPinkNoise
ConvergingBrownNoise(1) = TaperedPinkNoise(1);

for x=2:NoiseLength % Integrate, minus the biasing
    ConvergingBrownNoise(x) = ConvergingBrownNoise(x-1) + TaperedPinkNoise(x)
- pm;
end

% Normalization
% Min value is 0, max value is 1

PinkNoise = PinkNoise - min(PinkNoise);
PinkNoise = PinkNoise / max(PinkNoise);

BrownNoise = BrownNoise - min(BrownNoise);
BrownNoise = BrownNoise / max(BrownNoise);

TaperedPinkNoise = TaperedPinkNoise - min(TaperedPinkNoise);
TaperedPinkNoise = TaperedPinkNoise / max(TaperedPinkNoise);

ConvergingBrownNoise = ConvergingBrownNoise - min(ConvergingBrownNoise);
ConvergingBrownNoise = ConvergingBrownNoise / max(ConvergingBrownNoise);
```

Resulting signals `PinkNoise` and `ConvergingBrownNoise` were of the most utility for analyzing the behavior of the Tandem ADC. Many of the other signals were created primarily for the purpose of generating those two. These signals were used to help generate Figures 9.12 through 9.17.