ASYNCHRONOUS LEVEL CROSSING ADC FOR BIOMEDICAL RECORDING APPLICATIONS

Kieren Pae

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

August 2021

APPROVED:

Ifana Mahbub, Major Professor
Gayatri Mehta, Committee Member
Xinrong Li, Committee Member
Shengli Fu, Chair of the Department of Electrical Engineering
Hanchen Huang, Dean of the College of Engineering
Victor Prybutok, Dean of the Toulouse Graduate School

Pae, Kieren. *Asynchronous Level Crossing ADC for Biomedical Recording Applications*. Master of Science (Electrical Engineering), August 2021, 78 pp., 6 tables, 31 figures, 3 appendices, 51 numbered references.

This thesis focuses on the recording challenges faced in biomedical systems. More specifically, the challenges in neural signal recording are explored. Instead of the typical synchronous ADC system, a level crossing ADC is detailed as it has gained recent interest for low-power biomedical systems. These systems take advantage of the time-sparse nature of the signals found in this application. A 10-bit design is presented to help capture the lower amplitude action potentials (APs) in neural signals. The design also achieves a full-scale bandwidth of 1.2 kHz, an ENOB of 9.81, a power consumption of 13.5 microwatts, operating at a supply voltage of 1.8 V. This design was simulated in Cadence using 180 nm CMOS technology.

ACKNOWLEDGEMENTS

This work is based upon work supported by the National Science Foundation (NSF) under grant No. ECCS 1943990.

I would like to thank my major professor Dr. Ifana Mahbub for all of her help and guidance through this project. Her insight was invaluable during my work on this thesis. I would also like to thank Dr. Gayatri Mehta and Dr. Xinrong Li for taking the time to serve on this thesis committee.

Lastly, I would like to thank my friends and family for helping make this possible. Their encouragement and support through this entire process has helped me accomplish this goal of attaining a Master's degree in Electrical Engineering.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The amount of information being recorded and processed in today's world has never been higher. That amount of information also continues to grow with each passing year. Computers and other digital devices handle the bulk of these processing requirements. Unfortunately for these devices, we live in an analog world, and that fact has led to the analog-to-digital converter (ADC) being one of the most important building blocks in today's technology. ADCs take continuous-time (CT) signals and convert them into a form that digital devices can take as an input. The basic operating principle behind this is that the ADC samples the CT signal and outputs the data as bits, which is what the digital devices are looking for as an input to be able to process the information.

ADCs are found in a wide range of devices, including wireless sensors, cameras, audio devices, cell phones, and televisions. Another application for these circuits are found in portable and implantable medical devices. These devices help medical personnel have the information they need to properly diagnose and treat various diseases. The ADCs required for these biomedical devices will be the main focus of this thesis. The following section will continue with a more in-depth look at the requirements for these devices.

1.1. Biomedical Sensing Constraints

As the medical science field grows and evolves, it is only natural for the devices used by medical personnel to also grow and evolve. One great asset already mentioned in this thesis can be found with portable and implantable devices. These allow for a level of information available to doctors that was not possible for most of human history. The designs for these systems, however, are required to be extremely power efficient and have a low silicon footprint [1].

One of the areas that has developed over the recent decades is neurological signal acquisition. These signals can be classified into local field potential (LFP) and action potentials (APs) [2]. APs are the result of activity for a single neuron. LFPs, on the other

hand, are generated by superimposition of electrical activity of neurons in the region. The signal characteristics of LFPs are typically in the frequency range of 0.1-250 Hz with a peak amplitude of several mV. For APs, the peak amplitude can be about a few μV and be found in the 0.25-5 kHz bandwidth [3]. A low-power ADC is required to digitize these signals. The typical choice for this process is an SAR ADC with a resolution ranging from 8-10 bits [4]. The SAR architecture is popular because it can meet the demands of biopotential sampling, including low power, low sampling rate, and the resolution criteria. However, due to the time sparse nature of biomedical signals, level crossing ADCs have gained attention as an alternative method [5]. Since this architecture only samples when the signal is active, it has been shown to save power for signals such as this [6].

1.2. Original Contributions

A unique design for a 10-bit level crossing ADC is presented in this thesis. The design utilizes concepts found in literature to reliably process both LFP and APs in neuropotential signals. This architecture uses the concept of level crossings to facilitate a low power design that can still accurately reconstruct the signal. Designs for the digital to analog converter (DAC), comparators, and digital logic are to be presented and discussed.

1.3. Outline of Thesis

This thesis is organized as follows. Chapter 2 starts with an exploration of synchronous ADC principles along with some sample architectures in that space. Then a discussion on how asynchronous ADCs are defined followed by a literature review on the work being done in biomedical asynchronous ADCs. After that Chapter 2 wraps up with a discussion of various performance metrics found in the literature. Chapter 3 begins with the discussion on the design for this thesis. Then simulation results are shown with a comparison with the state-of-the-art. The future work that is planned on this project is found at the end of this chapter. Finally, this thesis wraps up in Chapter 4 with the conclusion.

CHAPTER 2

OVERVIEW OF ADC ARCHITECTURES

As was briefly mentioned in the introduction to this thesis, an ADC takes a CT analog signal and converts it into a sequence of digital values proportional to the input signal's magnitude [7]. This process is generally much more difficult than the reverse process of taking a digital signal and converting it into analog form [8]. Fig. 2.1 shows a very basic block level diagram of an ADC. The input signal, V_{in}, is fed into an anti-aliasing filter, which leads to the ADC block with the N-bit digital output. This anti-aliasing filter is not always necessary, and it's function will be discussed later in this section.

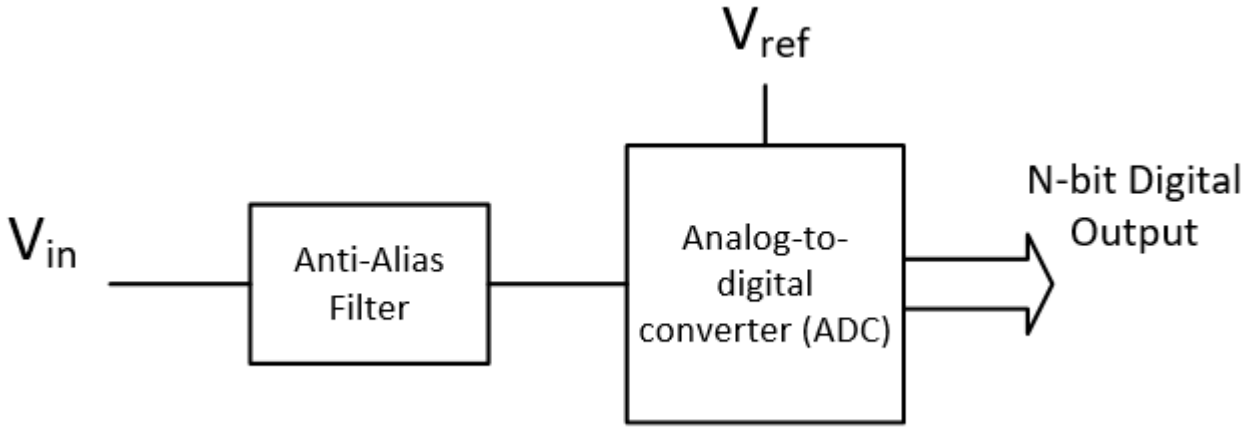

FIGURE 2.1. ADC Block Diagram

The N-bit digital output consists of N-bits labeled as D_0 to D_{N-1}. D_0 corresponds to the least significant bit (LSB) while D_{N-1} represents the most significant bit (MSB). A typical digital output code is shown in Eqn. 2.1.

$$D_{\text{out}} = D_{\text{N-1}}...D_1D_0 \tag{2.1}$$

This digital output that is delivered at the output of the ADC is useless without knowing the code and conversion relationship that is being used in the architecture [7]. The most commonly used code is straight binary (base 2). This results in the LSB having a value

weight of 2^0, the next bit's weight is 2^1, up until the MSB value whose weight is given as 2^{N-1}.

The value of N for the ADC determines the resolution, or in other words how many quantization levels the component will have. The ADC will split the V_{ref} voltage shown in Fig 2.1 into 2^N equal parts. As an example to help better explain this concept as well as the digital code discussed earlier an example of a V_{ref} value of 1.8 V and N being equal to 3 will be used. Now since the reference voltage is split into 2^N equal parts the value of a change in 1 LSB is $V_{ref}/2^N$. Plugging in the values for our example gives us the simple arithmetic of $1.8/8 = 0.225$ V. Now that we have our conversion relationship mentioned in [7] as 1 LSB = 0.225 V the binary output code can now be converted to a voltage value. The code to be used in this example is $D_{out} = 110$.

Now we take the value of D_{out} and multiply it by the value of 1 LSB in volts. This is given by $(110)_2 0.225\text{ V} = (6)_{10} 0.225\text{ V} = 1.35$ V. Now this example is admittedly a little misleading because it doesn't take into account the quantization error that arises from this operation. What is meant by this statement is that the actual value of the analog input signal is in the range of 1.35 V to the next quantization value, which is 1.575 V for this system. 1.575 V corresponds to a digital code of $(111)_2 = (7)_{10}$. To put it more concisely, with an output code of $(110)_2 = (6)_{10}$ the real analog input voltage value will be in the range of $1.35\text{ V} \leq V_{in} < 1.575$ V. Fig. 2.2 will help to further explain this concept.

Fig. 2.2(a) shows D_{out}, digital output, of an ideal 3-bit ADC [8]. The dashed line represents a ramp input from an analog signal while the staircase plot shows how the digital output changes with the aforementioned analog input. The x-axis has been normalized to V_{ref}. This means that at the ADC's maximum output, corresponding to $111_2(2^N\text{-}1)$, will represent a value of $V_{in}/V_{ref} \geq 7/8$.

Fig. 2.2(b) shows how the quantization error changes as the analog input increases [8]. The way to look at quantization error is that it naturally arises due to the differences between analog and digital systems. Digital systems only have a finite number of available outputs, 8 in this case, while the analog input shown theoretically has an infinite number of

points on that line. To better explain the quantization concept the part of Fig. 2.2 where $1/8 \leq V_{in}/V_{ref} \leq 2/8$ will be used as an example. The first points to look at are the edges of this boundary. When V_{in}/V_{ref} are equal to 1/8 and 2/8 there is a discontinuity in the quantization error graph where it goes from 1 to 0 bits. It can be seen that this discontinuity happens at every boundary for the D_{out} values. This is simply representing the fact that at exactly those values of the boundaries the digital and analog signals are in complete agreement. Thus the bit error is 0. However, if the signal is below or above the boundaries respectively, then the bit error is almost 1 bit showing that the system is losing almost that entire bits worth of information due to the digitization process.

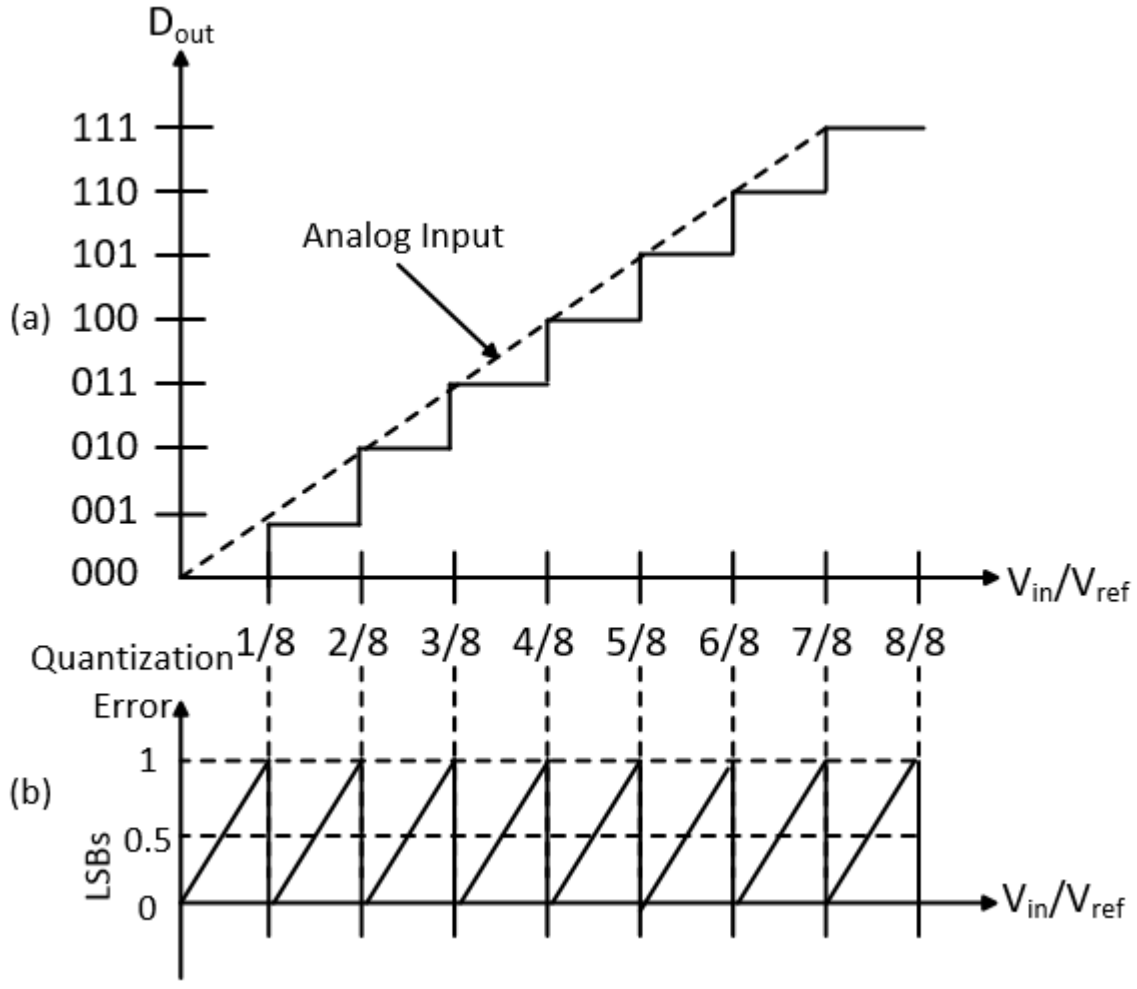

FIGURE 2.2. Ideal ADC example(a) Ideal ADC graph (b) Quantization error

One of the most important details of synchronous ADC's is the sampling rate the ADC will operate at. This determines how often the ADC samples the CT analog signal and transforms it into the corresponding digital code. The sampling rate, f_s, will largely be application dependent, but there is one condition that must be met that is well known in literature. Eqn. 2.2 shows the *Nyquist Criterion*. The equation reads that the sampling frequency, f_s, must be more than two times of the highest frequency found in the analog signal, f_0 [8].

$$f_s > 2f_0 \tag{2.2}$$

If this condition is met than there will be no aliasing in the frequency domain of the sampled signal. There will be no way to restore the signal from digital to analog without distortion if the condition in Eqn. 2.2 is not met without using other techniques [9]. One such way to subvert the *Nyquist Criterion* is by using compressive sensing (CS) algorithms. These methods allow the sampling rate to be determined by the signals sparsity instead of it's highest frequency content [10]. This can facilitate a reduction of data rate and power savings for the circuits processing the information [11].

To better explain the *Nyquist Criterion* Fig. 2.3 will be used to help the reader visualize the aliasing concept. In this figure it can be seen that the signal is represented in both the time and frequency domains, the sampling function consisting of a unit impulse train, as well as two examples of the resulting sampled signal. Fig. 2.3 (a) shows a band-limited signal with center frequency of f_0. This means the frequency range shown in the figure contains all of the frequency content found in the signal. As mentioned earlier the unit impulse train shown in Fig. 2.3 (b) represents the sampling function with a sampling period, $T = 1/f_s$. This part of the figure is displaying the action of sampling at discrete points in time. In the frequency domain it can be seen that the signals are spaced f_s apart.

All of the impulses shown have a value of 1 so that when this signal is multiplied with Fig. 2.3 (a), the ensuing signal is given in Fig. 2.3 (c). It can be seen from this part of the figure that the frequency domain representation for the sampled signal is simply multiple versions of the band-limited signal at multiples of the sampled frequency.

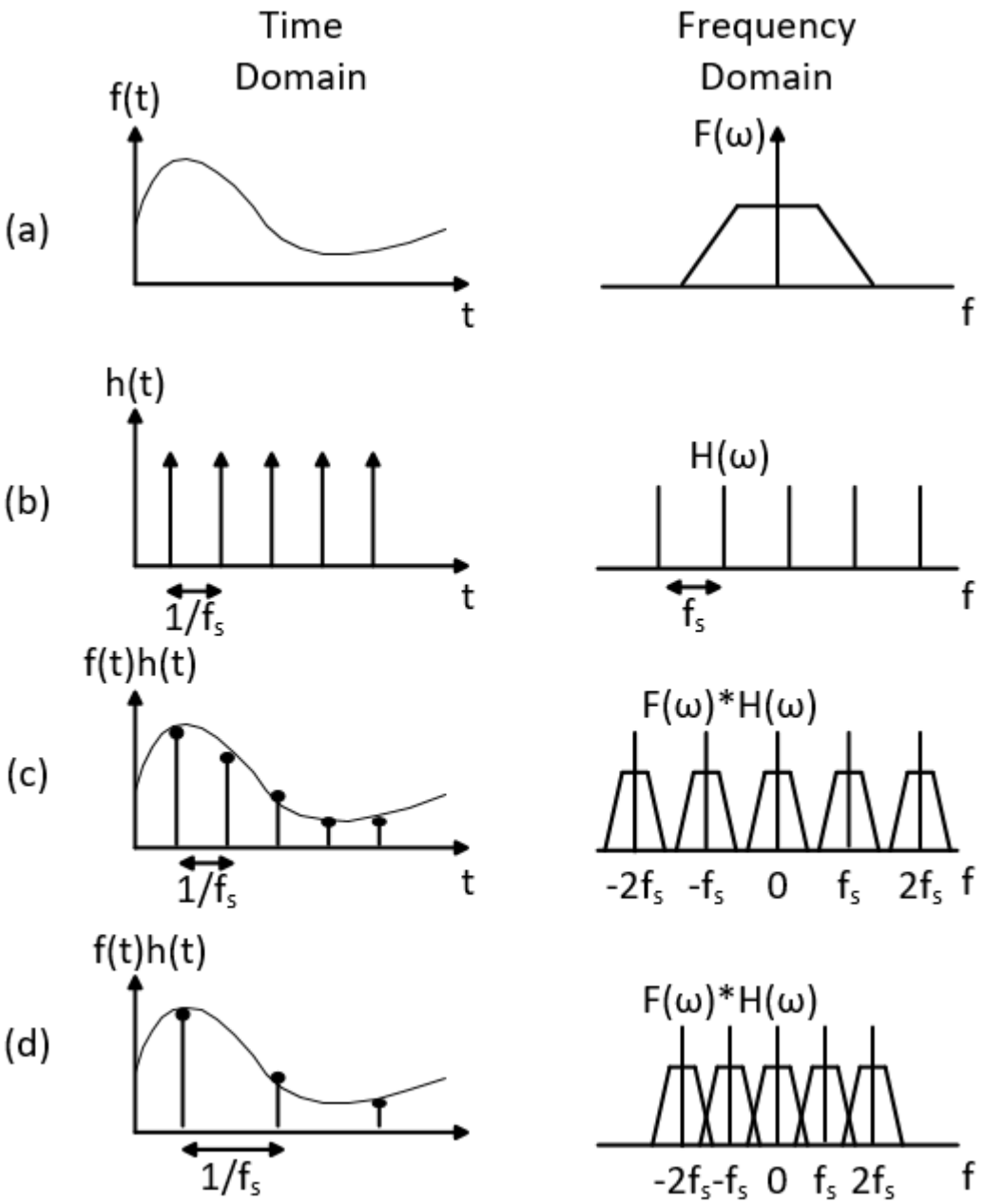

FIGURE 2.3. Explaining the effects of aliasing. (a) Input analog signal (b) Sampling function (c) Overall signal (d) Aliasing example

Fig. 2.3 (d) shows that as f_s is decreased the multiple versions that arise in the frequency domain squeeze closer and closer together. There comes a point where the signals finally overlap and cause the aliasing that been talked about and is shown by the frequency

domain side of Fig. 2.3 (d). The point at which the copies begin to overlap is called the folding frequency [8]. This effect is obviously undesirable as a loss of information occurs with the overlapping. One solution to this problem can be found by taking another look at Eqn. 2.2. The designer simply needs to increase the sampling frequency to avoid the folding effect. This solution may not always be feasible or desirable for an application, which leads to a more elegant solution. That solution comes in the form of an anti-aliasing filter.

An anti-aliasing filter has a simple working principle. The filter cuts off the signal so as to prevent the information from overlapping as shown in Fig. 2.3 (d). To better demonstrate this principle please refer to Fig. 2.4. It can be seen that the input is the same as the one shown in the previous figure. We are now assuming that the frequency f_0 does not meet the Nyquist criterion. This leads to Fig. 2.4 (b) where it is shown what an ideal case for a filtered signal would be. To prevent folding, the designer simply needs to limit the bandwidth to $f_s/2$. This will result in what is shown in Fig. 2.4 (c). When the full spectra in the frequency domain is shown the signals no longer overlap as they would without the filter. While it is clear that there is still a loss of information with this method, the integrity of the signal is still intact. The designer knows that the output of this signal is not tainted by operating under the folding frequency. To meet high performance requirements of today's systems, an anti-aliasing filter must have a strong stop-band suppression, high flatness in the band, as well as a large bandwidth [12].

2.1. Architectures

With the fundamentals of ADCs now explained, we can get into some of the details associated with the different architectures found in the literature. As with all the basic building blocks in engineering and other disciplines, ADCs can be tailored by the designer for their exact specifications. For instance, if a designer values speed over accuracy then the most common solution is to use a Flash ADC [13]. If the system that is being designed for only has signals below 10 MHz, then pipeline ADC is the correct choice [14].

The rest of this section explains these two example architectures in more detail as well as others. While these are not the only architectures found in literature, they were

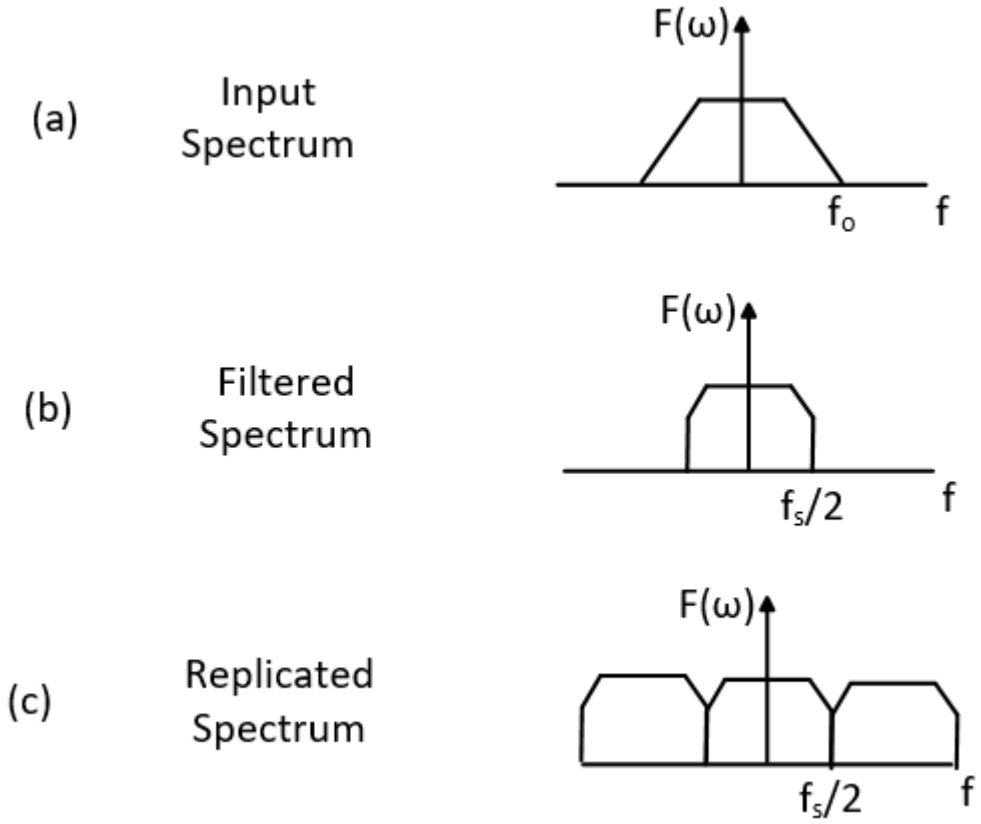

FIGURE 2.4. Demonstrating the effects of an anti-aliasing filter. (a) Input signal (b) Filtered signal (c) The effect of the filter

chosen to be representative of different requirements that ADCs have in practical designs.

2.1.1. SAR

The first architecture that will be discussed is the successive approximation register (SAR). The SAR ADC architecture is widely used when the application requires high precision and medium speed processing of analog signals [15]. The block diagram for this architecture is shown in Fig. 2.5. The analog input, V_{in}, is fed to a sample and hold circuit. The output of this circuit is then fed into a comparator. This comparator is instrumental to the working principle of the SAR ADC, which is the binary search algorithm [16]. This algorithm that is carried out by the SAR logic block works as follows. The initial comparison is between the V_{in} and $V_{ref}/2$. In this instance, V_{ref} is meant to represent a full scale input.

If true, then the comparator outputs a logic '1'. Then the algorithm compares the input signal to $3V_{ref}/4$. If the input signal is still higher, the SAR logic block compares the input to $7V_{ref}/8$. If the input was lower than $3V_{ref}/4$ then it is compared to $5V_{ref}/8$. The binary search algorithm continues in this manner until the resolution of the ADC cannot get any finer.

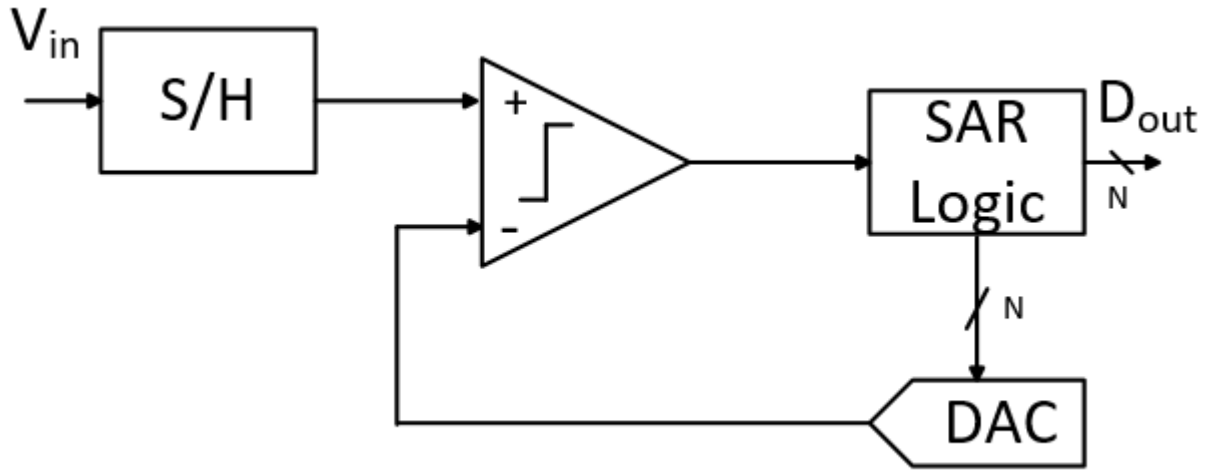

FIGURE 2.5. SAR ADC Architecture

2.1.2. Pipeline

Another common architecture that can be found in literature is the Pipeline ADC. This architecture has grown popular due to its good balance between power dissipation, size, speed, and resolution [17]. The Pipeline ADC got its name because of how the stages are connected. As can be seen from Fig. 2.6, the architecture consists of stages that are connected end to end, resembling how a pipeline is laid out.

To better understand why the layout is as such, the operation of each stage will be discussed. Fig. 2.7 shows a close up of one of the stages. The input signal V_{in} is fed into a sample and hold circuit labeled S/H as well as to a sub-ADC. The output of the ADC is fed into the DAC and the digital error correction or similar stage. The DAC output is connected to the summing block which is followed by a gain block that leads to the output of

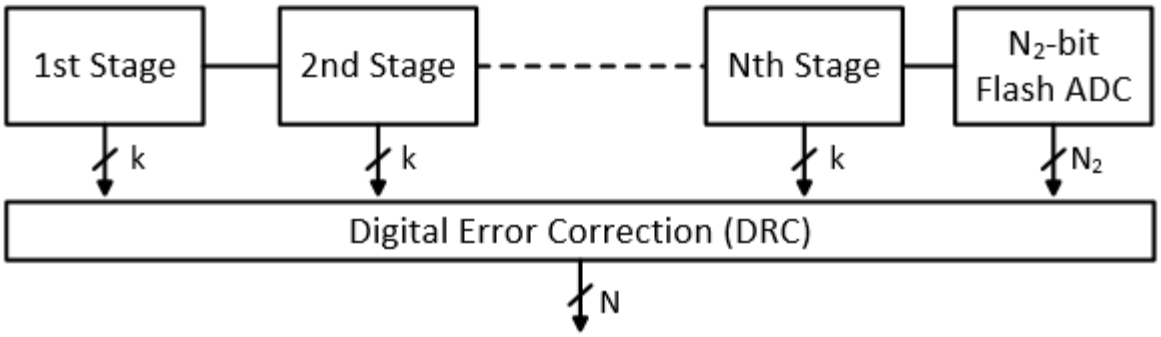

FIGURE 2.6. Pipeline ADC Architecture

the stage. The S/H, DAC, summing block, and the gain block are usually laid out together in one block called a Multiplying-DAC (MDAC) [18]. Now that the basic structure has been shown, it will be easier to explain what the purpose of each stage is. After the output of the sub-ADC is fed into the DAC and converted into an analog signal, it is subtracted from the original V_{in} to create a residue voltage [19]. This residue voltage is then gained up and fed into the next stage. The easiest way to look at this architecture is to think about each stage resolving certain bits in the output of the ADC. For instance, the first stage resolves the most significant bits in the output, while the Flash ADC located in the far right of Fig. 2.6 resolves the last N_2 bits in the output of the ADC. N_2 is used to distinguish between the two different N values found in the figure. Simply put, the number of stages, represented by N, does not necessarily have to equal the resolution of the Flash ADC, N_2.

2.1.3. Flash

As mentioned at the beginning of this section, flash analog-to-digital converters are used when speed is the most important factor in the design. This architecture achieves this speed due to the parallel nature of its operation. Fig. 2.8 shows a typical flash architecture.

As can be seen from Fig. 2.8, the architecture comprises of three stages. The first stage is a resistor ladder that sets the reference voltages. The number of resistors used in this ladder is 2^N, where N is the resolution of the ADC [20]. These reference voltages are then fed to the second stage of comparators. The number of comparators is 2^N-1, which is equal to the 2^N-1 reference voltages generated in the resistor ladder. In terms of the ADC, each

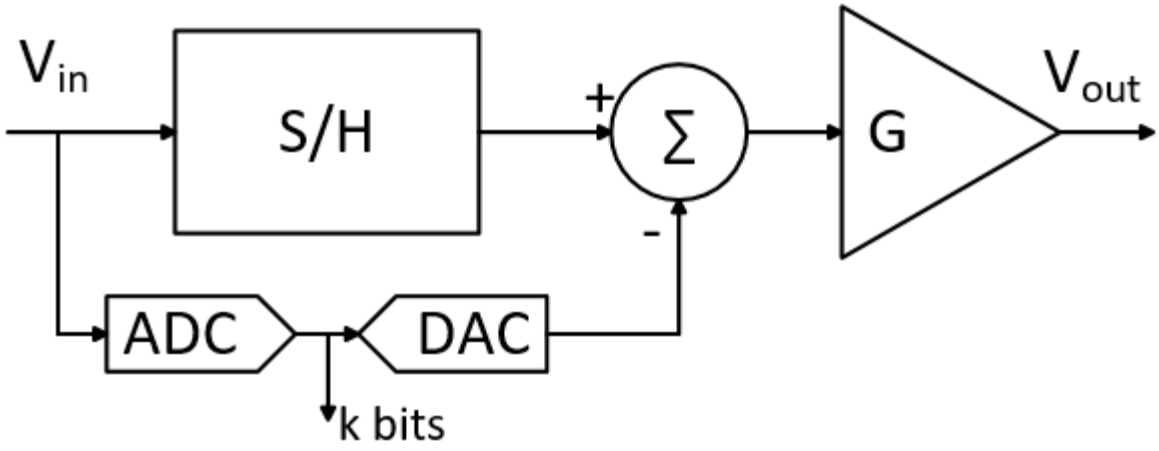

FIGURE 2.7. Pipeline Stage Block Diagram

comparator is 1 LSB away from the comparators on either side of it [13]. These comparators output a logic high when the input voltage is higher than the reference voltage. The outputs are then fed into the third and final stage of the architecture which is the encoder. The encoder's job is to take the outputs of the 2^N-1 comparators, called thermometer code, and transform them into a binary code [20].

2.1.4. Σ-Δ ADC

The last architecture to be discussed for synchronous ADCs is the Σ-Δ type. This architecture is also referred to as an oversampling ADC in the literature. This name came about from the fact that this method samples the analog input at a rate higher than the *Nyquist Criterion* that was discussed earlier in this chapter. The main home for this type are applications wanting low silicon footprint, low power consumption, and a high resolution [21]. Some of the applications this type is useful for are data acquisitions, communications, and instrumentation [22]. To help explain the architecture, Fig. 2.9 shows some of the basic building blocks. V_{in} is processed in the Sigma Delta Modulator block at the sampling rate, f_s. The output of this block is given as a 1-bit stream of data going to the digital filter. Then the digital and decimation filter reduces the data rate to a more usable value, as well as extract useful information from the stream coming from the Sigma Delta Modulator [23].

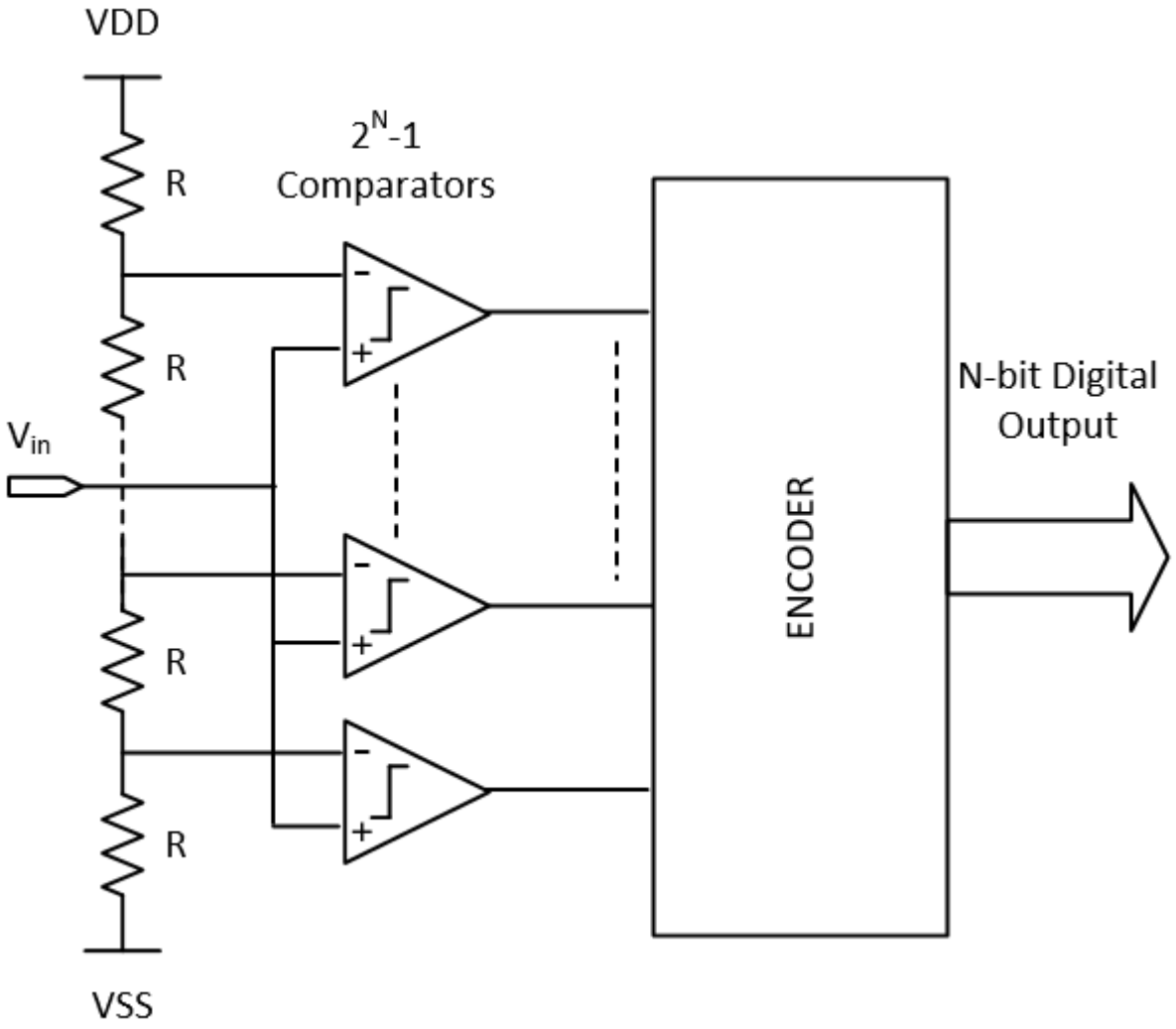

FIGURE 2.8. Flash ADC Architecture

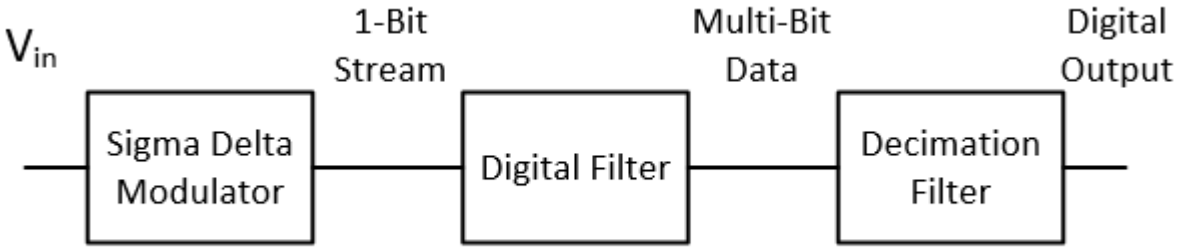

FIGURE 2.9. Σ-Δ ADC Architecture

2.2. Synchronous vs. Asynchronous

Until now, the focus of this chapter has been on synchronous architectures and concepts. This was important to explain in some amount of detail so that the differences between synchronous and asynchronous ADCs, also called level-crossing ADCs, could be more fully understood. The principle of level-crossing ADCs in literature can be traced back to papers as early as 1966 [24, 25]. At the time, the level-crossing architecture was referred to as delta modulation. This scheme works similar to a flash ADC, with the difference of only using two comparators in its design [26]. Fig. 2.10 will be used to help explain the differences in synchronous vs. asynchronous. Fig. 2.10 (a) shows how a uniform sampling scheme works in ADCs. As can be seen no matter how the signal is behaving, the synchronous system takes a data point, the black circles found on the signal, at every dotted line. The dotted line represents the sampling frequency for our example ADC. The level-crossing example shown in Fig. 2.10 demonstrates how this scheme only takes data points when certain voltage thresholds are crossed. Because of the nature of this method, the time between samples, Δt is also required for reconstruction. This value is kept track of using a timer circuit.

With the fundamentals of asynchronous sampling laid out, the applications for this architecture are easily comprehensible. Systems with signals that have varying activity over time and run on low power resources are suitable for this approach. Applications where this is true can be found in intelligent sensor networks and audio signal processing [27, 28]. These systems only require data transmission when their sensors indicate a significant change in level. One can also infer that speech signals would be a good use case for this approach due to their burst like nature with intervals of silence.

As the title of this thesis would suggest, a level-crossing scheme is well suited for biomedical devices that are wearable and implantable [29]. The time sparse nature for the output of these devices make them the perfect candidate for this technology. This approach can also facilitate a design to compress the output data, further increasing the power saving benefits for these systems [30].

To help explain how the system works more in depth a term I am borrowing from

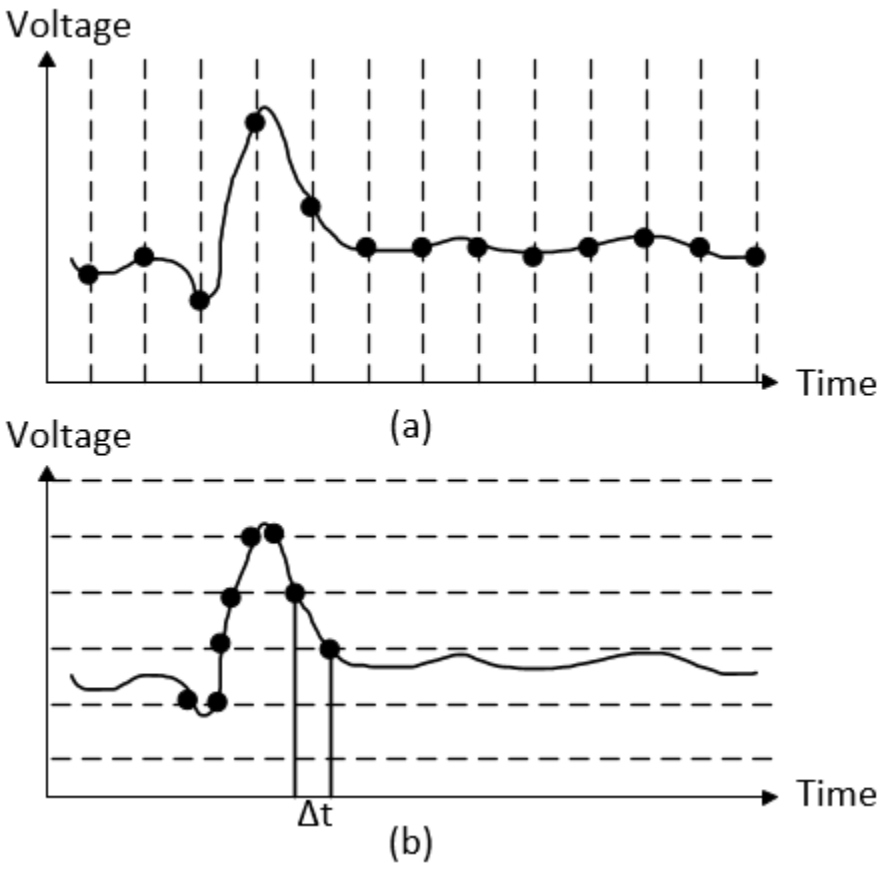

FIGURE 2.10. Synchronous vs. Asynchronous(a) Uniform sampling (b) Asynchronous or level-crossing sampling

a paper by Agarwal, Trakimas, and Sonkusale will be used [31]. The architecture to be presented is a regular asynchronous method. Briefly put, this simply means that the voltage threshold levels the ADC uses to monitor the input signal only move up or down by 1 LSB regardless of how the input is behaving. The block level diagram for my design is shown in Fig. 2.11.

The figure shows that the architecture has three main components. These are the DAC, the comparator, and the digital logic. The inner workings of these blocks will be explored in the subsequent sections, but for now a discussion of the role each of these have in the overall level-crossing ADC will be had. To explore the interconnections of the block diagram the labels on the signal paths will be defined. See Table 2.1 for the explanations.

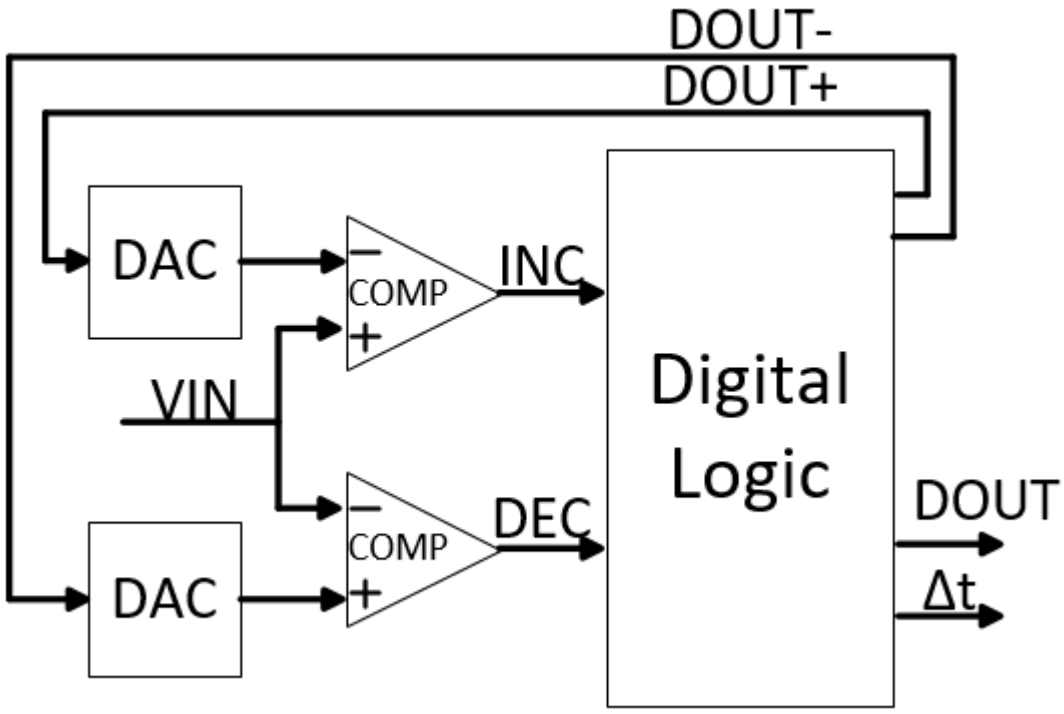

FIGURE 2.11. Asynchronous ADC architecture

TABLE 2.1. Signal Definitions

Signal Label	**Purpose**
VIN	Input signal
INC	Indicates when the upper voltage threshold has been hit
DEC	Indicates when the lower voltage threshold has been hit
DOUT	10-bit output of the ADC
DOUT+	Sets the upper voltage threshold
DOUT-	Sets the lower voltage threshold
Δt	Reports time in between samples

As was mentioned in the table, the design for this thesis was chosen to have a resolution of 10 bits. The operation of the ADC will be described by first assuming that the digital logic block has just taken a sample and has stored the digital output in "DOUT".

One of the first actions taken by the logic is to reset the internal timer that keeps track of the value for "Δt". After this the logic needs to set to new values for "DOUT+" and "DOUT-" so that the voltage thresholds can be accurately set by the two DACs for the next sample. At this point the digital logic simply waits for either the "INC" or "DEC" lines to go high which signal that a voltage threshold has been crossed and the aforementioned process needs to start again.

As was mentioned earlier in this section, it was important to look at different synchronous architectures to help the reader understand the differences between those and asynchronous systems. While the technical details of the differences have just been laid out, there is another difference worth mentioning and exploring here. That is the different types of sub architectures found in asynchronously sampled biomedical systems. What is meant by this is that Section 2.1 laid out several different methods to synchronously take samples. All of these approaches vary in some fundamental way that changes what application the systems are suited for. For level crossing systems, the recent works have been tweaking the approach and not fundamentally changing how the process works. Some of the optimizations that are presented in prior works will now be discussed.

One of the methods found is the concept of an adaptive resolution [30, 32, 33]. Adaptive resolution means that the ADC can actually change the resolution it operates at. This has been proven to not only save power but also perform data compression, which is another desired feature for this application. This can be achieved in many ways but the principle is the same. In times that the input has a high enough slope, the ADC lowers the resolution so that the voltage thresholds used for comparison do not trigger as often. For instance, in one of the papers the authors designed the system to simply look at the time in between samples and decrease the resolution if they were too close together [30].

Another evolution found in the architecture is the advent of fixed-window comparisons [5, 26, 34, 35, 36]. For the conventional architecture discussed earlier in Section 2.2 a floating window structure is discussed. This simply means that the inputs to both of the comparators change with the input signal as it is moving. With a fixed window system the voltages used

for comparison are fixed, usually around the midpoint of the rail-to-rail voltage. Then whenever a threshold crossing has occurred the input voltage V_{in} is either subtracted or added to for a newly defined voltage V_m. This V_m value is halfway between the fixed voltage thresholds. In essence the tracked V_{in} is artificially made to stay within the fixed bounds of comparison by resetting the input voltage to V_m at every level crossing.

One of the biggest power draws for the conventional level crossing architecture are the continuous-time (CT) comparators. One way to improve upon this aspect is the use of dynamic comparators [34, 37]. Dynamic comparators only perform comparison upon receiving a clock signal and therefore only use power during those comparisons, saving power over their CT counterparts. The paper by Ghasemi et. al even goes the extra step by only having 1 comparator in the design to have an even further reduction in power [34].

Another change some designers have made is to replace the DAC circuits with a suitable alternative [33, 37]. A scaler circuit is used in one paper to replace the N-bit DAC [33]. Instead of a DAC circuit to add and subtract to the input voltage V_{in}, the scaler, as its name suggests, scales V_{in} based on a novel gain code generator that controls the ratio of the scaling. In the other paper that was cited, an analog memory cell is used for the replacement of the DAC [37]. This memory cell acts as a DAC in their proposed architecture by holding the last voltage level crossed. The paper mentions that the main drawback for this design is the voltage nature and hold times for the cell.

The main design goal for this thesis is being able to acquire the APs mentioned in Section 1.1. One of the often used methods to capture these are a bandpass filter followed by a simple threshold. However, much of the information about the shape of the spikes as well as sub threshold regions are lost [38]. Because of the low amplitudes found in APs, the design will use a 10-bit scheme to reliably capture these signals and will be discussed more in depth in Chapter 3.

2.3. Performance Metrics

Mean Squared Error (MSE)

Mean squared error (MSE) is a well known measure of how close two signals are. The typical formula has been taken and adapted to fit with the variables in an ADC [39].

$$MSE = \frac{1}{n}\sum_{k=1}^{n}(V_{\text{in}} - V_{\text{out}})^2 \tag{2.3}$$

In Eqn. 2.3 n stands for the number of samples, V_{in} represents the input voltage to the ADC, and V_{out} is the output voltage of the ADC after interpolating it's outputs.

Differential Nonlinearity (DNL)

The differential nonlinearity (DNL) for an ADC is a measure of the difference between non-ideal ADC step size and ideal ADC step size. A succinct view of this concept can be found in Eqn. 2.4 [8].

$$DNL = Actual\ step\ width - Ideal\ step\ width \tag{2.4}$$

DNL can be defined with either the units of volts or LSBs. This is a result of the step widths property of being expressed by either of these units. Fig. 2.12 showcases what an example of a DNL would look like.

This figure may look familiar to the reader as it is a modified version of a previous figure in this chapter. That figure showcased the ideal 3-bit ADC. The original ideal case for an ADC is modified in Fig. 2.12 to help show what DNL looks like. Before continuing with the explanation there will be a brief discussion on the notation to be used. When referring to step widths the form DNL_x will be used, where x is the corresponding D_{out} value in decimal form. For example, DNL_2 refers to the transition for the D_{out} value of 010. Now with that better explained a more detailed look at how to actually calculate the DNL will be presented.

As in Fig. 2.2 (b), Fig. 2.12 (b) shows the quantization error expressed in LSBs. For an ideal transition $DNL_x = 0$. The ideal transition points for this example can be found in DNL_0, DNL_1, DNL_3 DNL_5, DNL_6, and DNL_7. One thing to note for these ideal cases is that DNL_7 sticks out when one is looking at Fig. 2.12. This is simply because the ideal transition at this point is 1.5 LSB since this is the maximum value of our ADC.

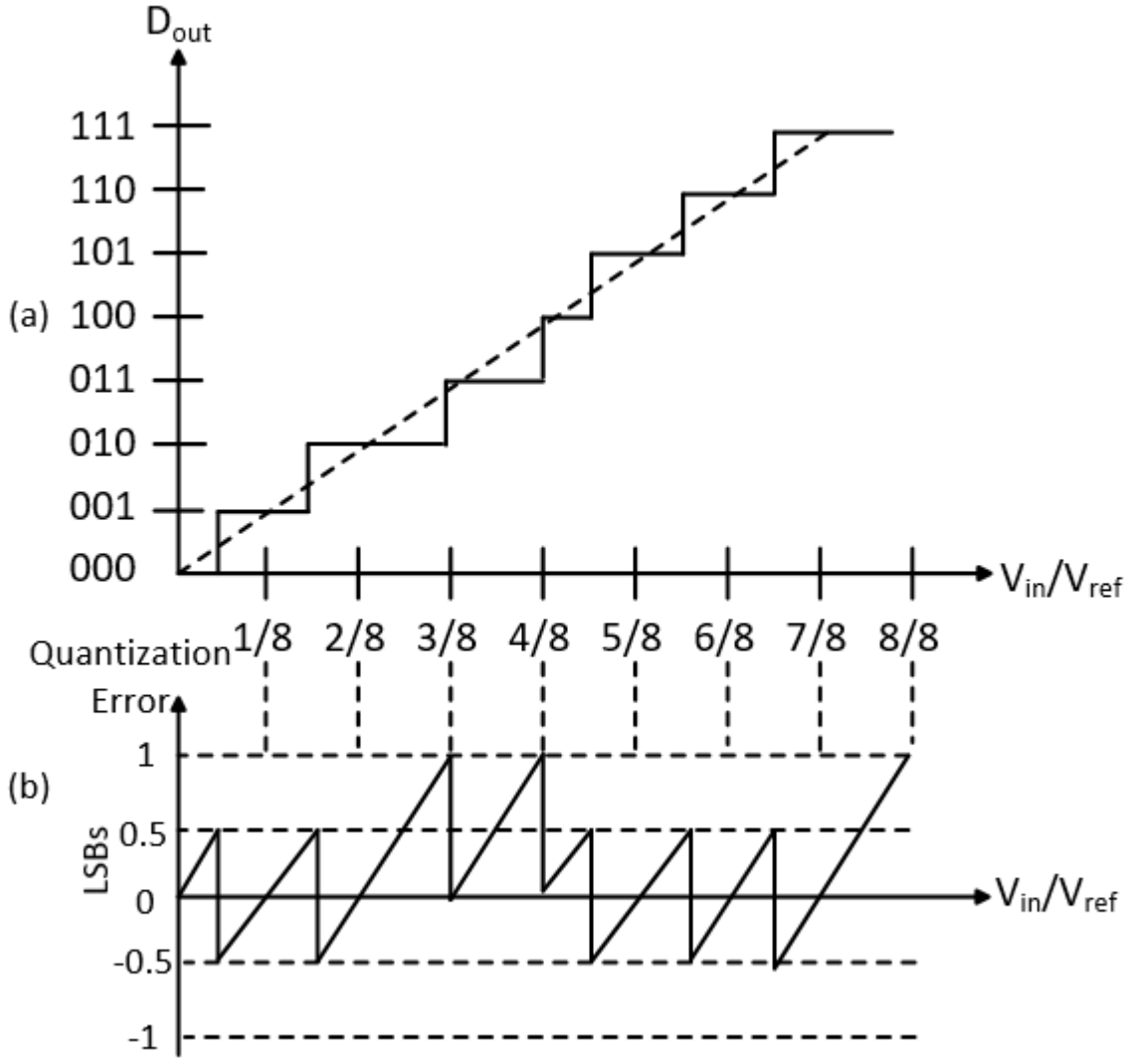

FIGURE 2.12. Demonstrating DNL (a) Nonideal 3-bit ADC (b) DNL illustrated with quantization error

Now the calculation of the DNL for the non-ideal case will be looked at. It will discuss more in depth momentarily, but the non-ideal step width can be found in DNL_2, DNL_4. It can be seen that for $D_{out} = 2$ that the step width shown is 1.5 LSB. This can be calculated by (3/8) - (3/16). This results in 3/16, and knowing that 1 LSB = 1/8 gives us the previously mentioned 1.5 LSB for DNL_2. Through a similar process DNL_4 can be found to be equal to -0.5 LSB.

One thing to observe is that when DNL = -1 then there is a guarantee that the ADC will have a missing code [8]. This means one of the values of D_{out} would not have a representation on a graph similar to 2.12 (a). One final note on DNL is that passive element mismatches are typically the cause of non-ideal DNL values [40].

Integral Nonlinearity (INL)

As with DNL, a modified version of Fig. 2.2 will be used to help explain integral nonlinearity (INL). This can be seen in Fig. 2.13. The first thing to point out in this figure is the location of the dotted line. Unlike in DNL, for the INL graph the dotted line is positioned to connect the first and last transitions by their end points. This allows us to define INL as the difference between D_{out} transition points and the straight line while all the other errors are set to zero [8]. The concise way to communicate this can be found in Eqn. 2.5.

$$INL = Actual\ transition\ point - Ideal\ transition\ point \tag{2.5}$$

To better explain the notation from the discussion on DNL will once again be used where INL_x represents the D_{out} value in decimal form. A quick inspection of Fig. 2.13 reveals that there are only two transitions not on the dotted line. This leaves us with INL_0, INL_1, INL_3, INL_4, INL_5, INL_7 being equal to zero.

We are left with the two points D_{out} = 010 and 110. There are two ways to determine the INL with the graphs given. For INL_2, one can use Eqn. 2.5 along with Fig. 2.13 (a) to come up with (2/8) - (3/16) = 1/16 or 0.5 LSB. To determine INL_6 we will now look at the quantization graph given in Fig. 2.13 (b). Whenever the error lies outside the ± 0.5 LSB band it will show where the non-ideal INL resides. As can be seen INL_6 is at -1 LSB, therefore INL_6 = -0.5. INL is also caused by passive element mismatches as well as active block non-linearities [40].

Signal to Noise Ratio (SNR)

SNR in ADCs is defined as the ratio of the value for the largest RMS input signal and the noise power [8]. The noise sources include the quantization noise and that of the passive and active devices used in the ADC. The SNR for asynchronous ADCs can be calculated by

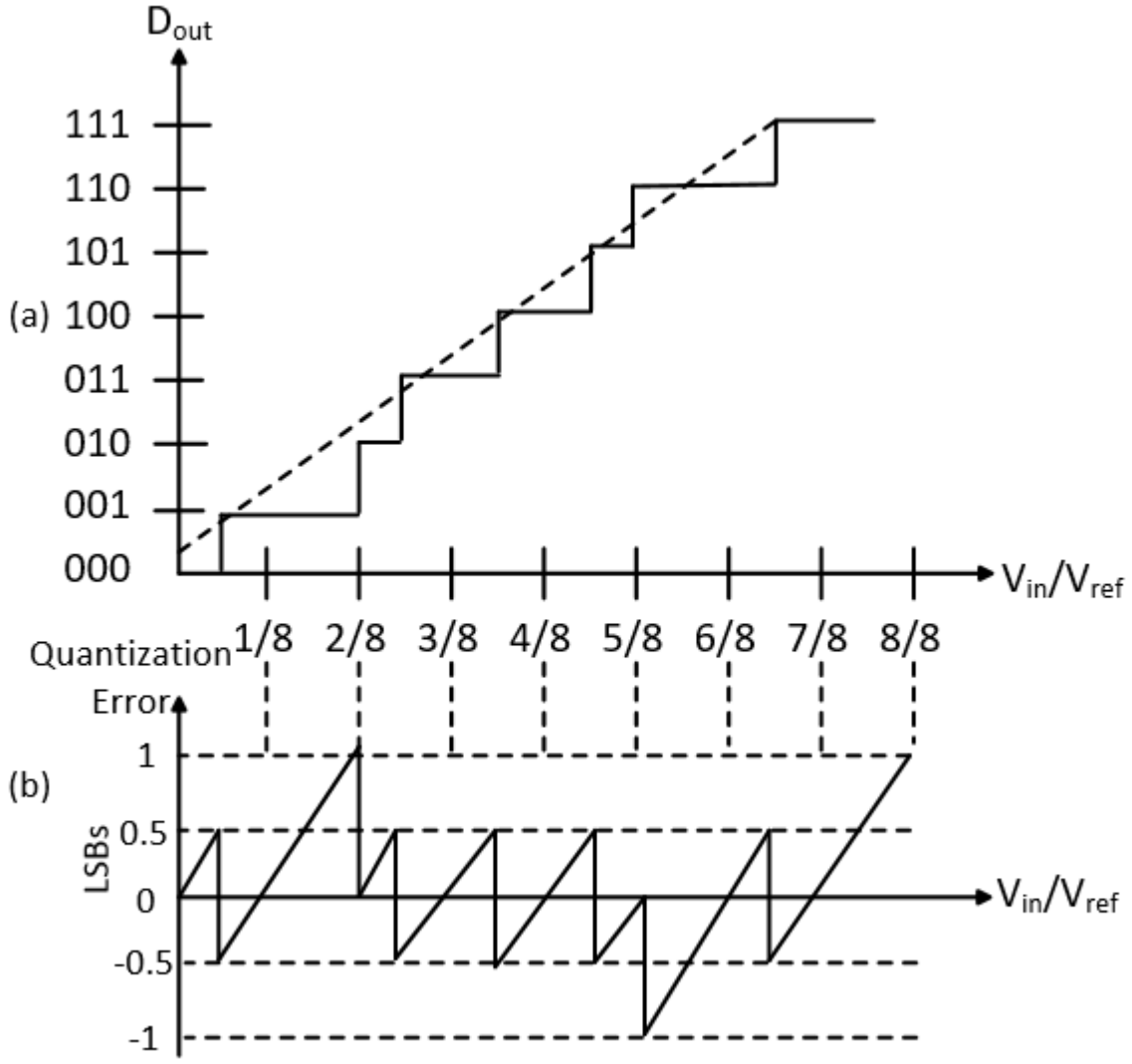

FIGURE 2.13. Demonstrating INL (a) Nonideal 3-bit ADC (b) INL illustrated with quantization error

transforming the quantization error in time to an error in amplitude [41]. Eqn. 2.6 shows the result of such an analysis.

$$SNR = 20\,log\,OSR - 11.2\,dB \tag{2.6}$$

This was first derived in a work by Sayiner et. al [42]. The term OSR in this equation stands for the clock oversampling ratio. This is calculated by taking the ratio of the timer

frequency to the input signal frequency. The formula in Eqn. 2.6 works under the assumption that the value of OSR is high[26].

While this equation can be found in literature, there was an assertion made in [43] that the derivation for Eqn. 2.6 was done incorrectly and produced an erroneous result by a wrong method of calculating average slope and average power in a non uniformly sampled signal. The corrected expression is found in Eqn. 2.7

$$SNR = 20\,log\,OSR - 14.2\,dB \tag{2.7}$$

As can be seen the result claims there is a 3 dB overstatement of the theoretical SNR as previously calculated. For the result shown in Table 3.5 the SNR calculation from Eqn. 2.6 will be used. These equations can be compared to the one found in Eqn. 2.8 that shows the SNR for a conventional ADC.

$$SNR = 6.02\,n + 1.76\,dB \tag{2.8}$$

In this formula n represents the number of bits for the output of the ADC. As you can see this equation only cares about the number of bits in the design. This makes a good amount of logic sense as in the conventional ADC structure a big source of noise comes from the quantization noise of the voltage, which is directly affected by the value for n in Eqn. 2.8

Signal to Noise and Distortion Ratio (SNDR)

This metric is a slightly modified version of the previously shown SNR. The signal to noise and distortion Ratio (SNDR) is defined as the ratio of the value for the largest RMS input signal to the noise and distortion power. This metric includes all distortion and noise sources in the ADC [40].

Effective Number of Bits (ENOB)

The value for effective number of bits (ENOB) helps quantify how well the ADC performs in terms of the number of bits it can produce effectively. Nonlinearity and component noise will affect the value for this metric [40]. Eqn. 2.9 shows how the ENOB is calculated using the SNDR for the system.

$$ENOB = \frac{SNDR - 1.76\,dB}{6.02\,dB} \tag{2.9}$$

This metric is very useful and one that can be seen in much of the literature. It is derived by solving for n in Eqn. 2.8.

Figure of Merit (FOM)

This metric is a popular measurement across disciplines. While the exact meaning for a figure of metric (FOM) changes depending on the topic at hand, the rationale behind it is the same. A FOM is used to characterize the performance of a device, system, or method, relative to its alternatives [44]. For ADCs, a FOM is a quantity that describes how much energy is consumed per conversion step. For asynchronous ADCs the FOM is defined as seen in Eqn 2.10.

$$FOM = \frac{Power}{2^{ENOB} * 2\,BW} \tag{2.10}$$

This equation encompasses most of the information for the ADC. By inspection it is easy to see how to design the ADC to improve the FOM. One option that is always good to consider is lowering the power consumption. Another direction to tackle the improvement of the FOM is to increase the value of either ENOB or the bandwidth, BW.

As with the SNR metric discussed earlier, the FOM also has a different version for uniform sampling systems. The only difference between the two equations is that the sampling frequency f_s replaces the 2BW term for the asynchronous case. This makes a good amount of intuitive sense as f_s is one of the most important parameters for synchronous sampling schemes.

$$FOM = \frac{Power}{2^{ENOB} * f_s} \tag{2.11}$$

One thing that is rather interesting is how the previous two metrics can be calculated. If you notice the FOM depends on ENOB and the ENOB depends on SNDR. As you can recall the SNDR is a measure of how much noise and distortion is in your system relative to your signal. It's a nice sanity check that these last two metrics, which are the most often

cited metrics in literature, can be improved by improving the signal quality which makes a good amount of intuitive sense.

CHAPTER 3

ASYNCHRONOUS ADC DESIGN

In this chapter, we will get into a more detailed look at the design for this thesis. The 10-bit resolution was chosen to help detect the APs in the signal. Assuming the AP can have a μV-level amplitude, as previously stated in this paper, the gain of the amplifier that would precede the ADC block in a typical neural signal recording system would need to be $\sim$60 dB [3]. This puts the amplitude of the AP at a few mV after amplification. Given this requirement, the 10-bit resolution combined with the rail voltage of 1.8 V gives an LSB value of around 1.7 mV and thus is able to capture these events.

3.1. DAC Design

While all of the blocks in this architecture are important, the DAC holds a very important place among the others. This block is responsible for setting up the threshold levels for the comparators based on the "DOUT+" and "DOUT-" 10-bit inputs. The accuracy of the two DACs is a major component of how the overall system performs. This assertion can be thought of intuitively by remembering how the overall architecture works. For conventional level crossing schemes, the error produced in the system is one in the value of "Δt" and not the voltage level. So having an accurate output for the DAC to get as close to the ideal case is important to accomplish.

The topology chosen for this design is the popular charge-scaling DAC. While not ubiquitous in the literature for this application, it can be found in various forms over the years [45, 46]. To explain the architecture please refer to Fig. 3.1. As it can be seen from the figure there is a binary-weighted array of capacitors with a total value of 2^{N}C. A digital code $D_0D_1...D_{N-1}$ is sent to switches connected to the bottom plate of the capacitors. These switches are labeled as Selection Circuit in Fig. 3.1. The selection circuit switches between either V_{ref} or ground depending on the corresponding bit in the digital code. This causes V_{out} to be a function of voltage division between the capacitors.

To better understand this process one can find a formula to relate the input code

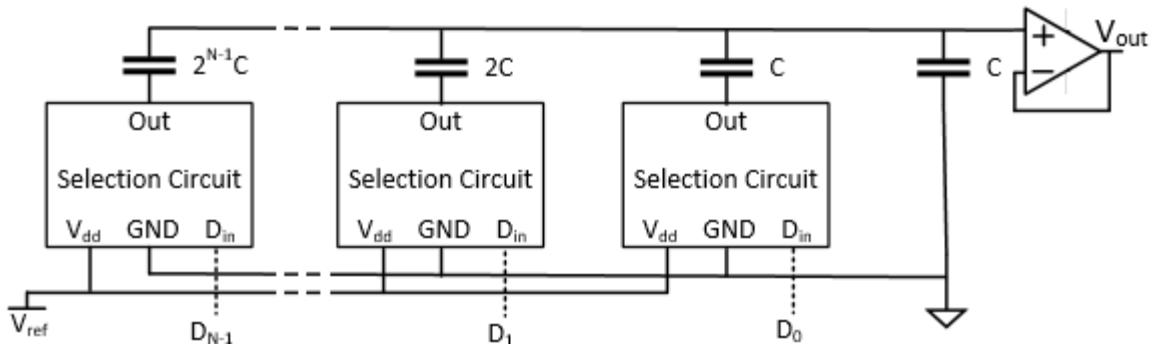

FIGURE 3.1. DAC block diagram

to the output voltage. To derive the equation simply take the impact that each individual capacitor has on the output and use the principle of superposition to get the entire output [8]. The result of such an analysis produces the formula shown in Eqn. 3.1.

$$V_{\text{out}} = \sum_{k=0}^{N-1} D_{\text{k}} 2^{\text{k-N}} * V_{\text{ref}} \tag{3.1}$$

To verify this equation we will take a simple example and analyze. Let us assume that the digital code consists of all zeros except the MSB D_{N-1}. This represents half of the range and as such we should expect D_{out} to be $V_{ref}/2$. Plugging into the equation we know that it will be all zeros except for the case when k = N-1. This leaves us with 2^{k-N} as being equal to $2^{N-1-N} = 2^{-1} = 1/2$. Therefore the output voltage V_{out} is equal to $V_{ref}/2$, confirming our test case agrees logically and mathematically.

The switches in the selection circuit shown in Fig. 3.1 are realized with a pair of transmission gates shown in Fig. 3.2. Transmission gates are a powerful tool in digital logic designs as they have the capability of passing strong 1s and 0s. When D_{in} is high, then V_{ref} is passed to the output. On the other hand, GND is passed to V_{out} when D_{in} goes high. Transmission gates typically have a high W/L ratio. Transistors M_0 and M_2 have W/L ratio of 10/1, while devices M_1 and M_3 have a W/L ratio of 20/1.

3.2. Comparator Design

Due to the asynchronous nature of the level crossing method, a continuous-time (CT) comparator is used in most of the literature. This paper is no different. The comparator

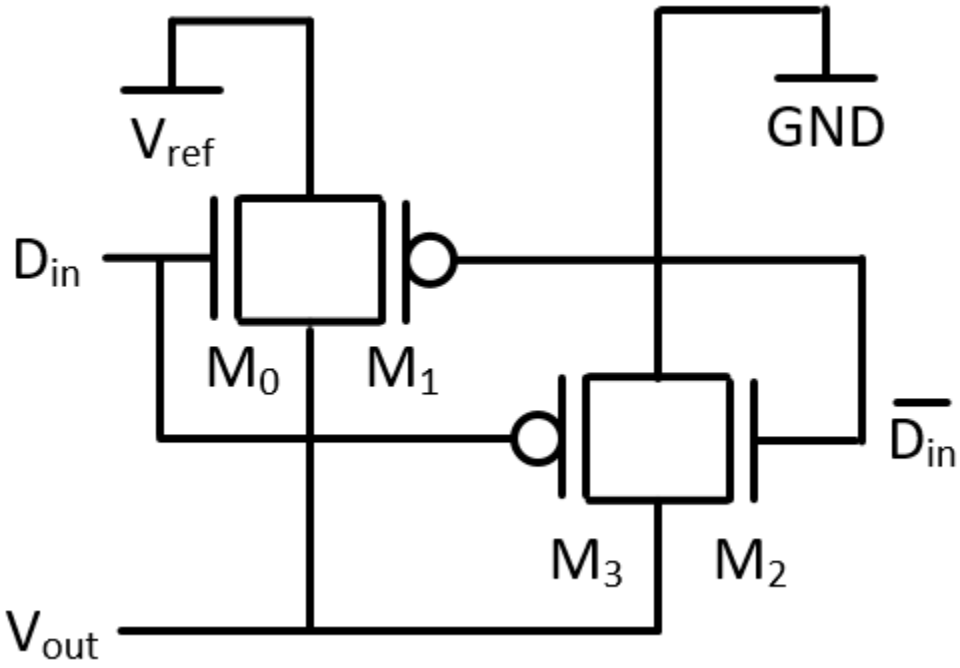

FIGURE 3.2. DAC selection circuit

designed for this equation is a Class-A amplifier that sharpens the transitions between 1s and 0s through a pair of inverters at the output [47]. The V_{bias} voltage is generated through a current mirror using a bias current of 5 μA. Table 3.1 presents a summary of the devices found in the comparator.

3.3. Digital Logic

The digital logic block is responsible for handling all the outputs for the system as well as telling the DACs where to set the voltage thresholds. To accomplish this, the block takes the"INC" and "DEC" lines shown in Fig. 2.11 as inputs. The block diagram for the logic is shown in Fig. 3.4.

The block labeled Output Logic controls the output lines of the system. Whenever a threshold is crossed in the system, as indicated by "INC" or "DEC" going high, an internal signal named "Change" is generated. This signal triggers 10 different D flip flops that take the now previous "DOUT" value and save it as a temporary signal "DOUT_tempX". The X represents the bit of "DOUT" that it has saved. This temp signal is sent to an

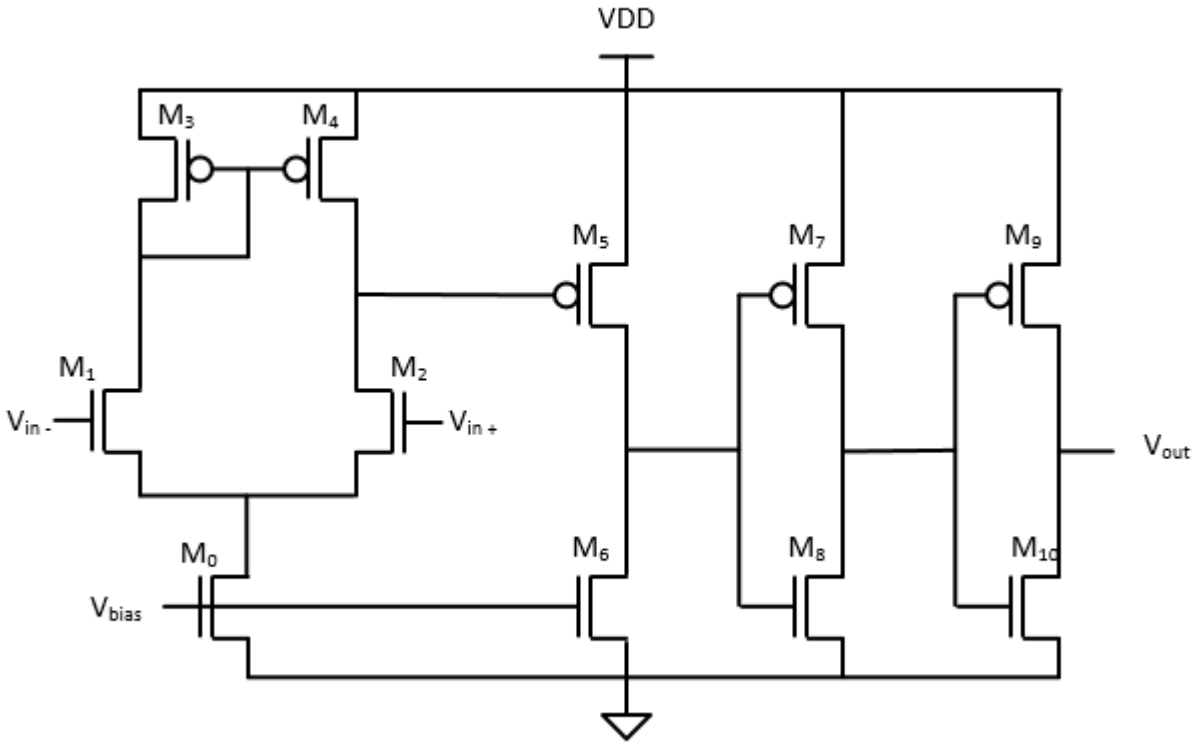

FIGURE 3.3. CT Comparator

TABLE 3.1. Comparator Summary

Device	**W/L**	$\mathbf{g_m/i_d}$
M_0	1.4 μm/500 nm	8.92
M_1, M_2	700 nm/500 nm	17.44
M_3, M_4	1.2 μm/500 nm	15.51
M_5	1.2 μm/500 nm	15.20
M_6	700 nm/500 nm	8.32
M_7	1.2 μm/500 nm	0.47
M_8	700 nm/500 nm	28.60
M_9	1.2 μm/500 nm	30.83
M_{10}	700 nm/500 nm	0.47

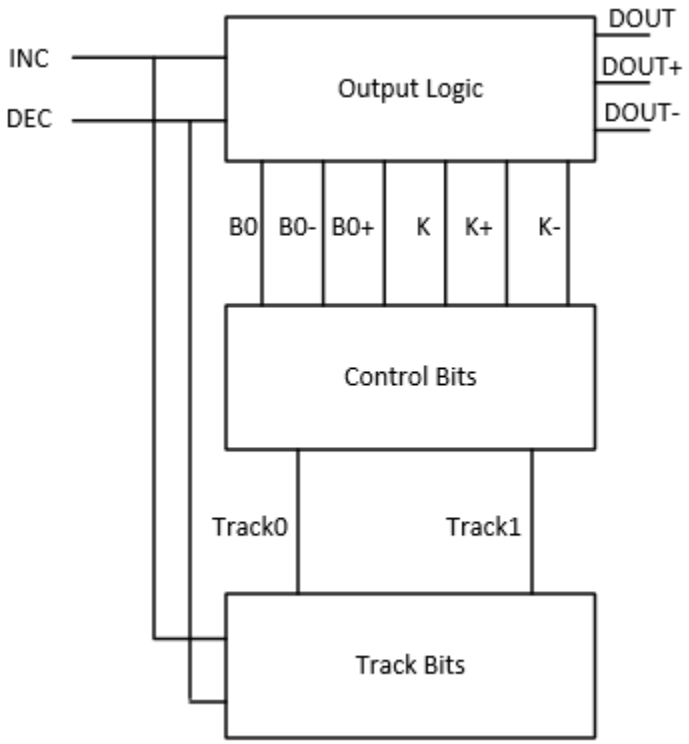

FIGURE 3.4. Digital logic block diagram

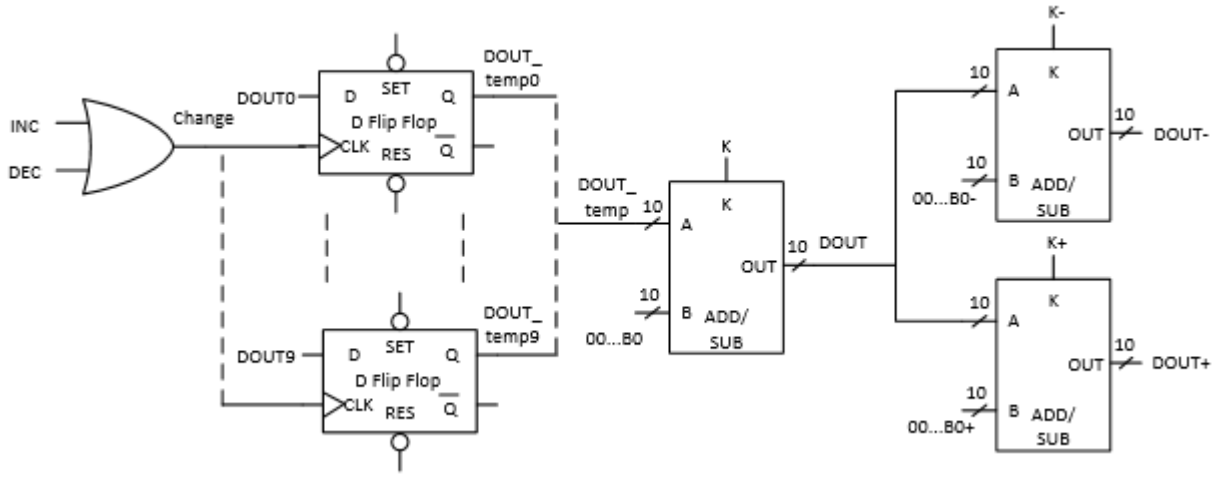

FIGURE 3.5. Output logic block

adder/subtractor circuit that has K and B0 as its other inputs. The new value for "DOUT" is set by these parameters. the value of K controls if the block is adding or subtracting, 1 is subtracting 0 is adding, and B0 is the LSB for the number being added to "DOUT" while

all of the other bits on that line are set to 0. After the new "DOUT" is generated, this signal is sent as an output as well as to two more adder/subtractor circuits that handle the generation of the "DOUT+" and "DOUT-" lines. Similar to "DOUT", the "DOUT+" and "DOUT-" signals are controlled with the variables K+, K-, B0+, and B0-. These variables used for the adder/subtractor circuits are generated inside of the Control Bits block found in Fig. 3.4.

To better explain the logic behind setting the "DOUT+" and "DOUT-" there will be terminology borrowed from a paper by Li et al [26]. For this architecture there are two types of level crossings. These will be coined consecutive level crossings (CLC) and repeated level crossings (RLC).

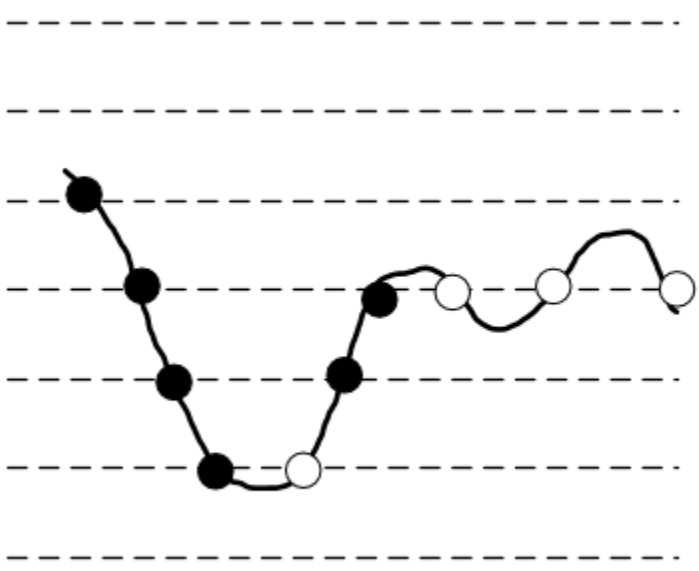

FIGURE 3.6. Level crossings in the ADC. Consecutive level crossing (CLC) shown with black circles. Repeated level crossings (RLC) shown with white circles.

Fig. 3.6 shows the CLC and RLC concepts in action. RLC can be seen as when the signal comes back on itself and crosses over the previous threshold that was triggered. The CLC is when the signal crosses the threshold that it did not previously cross. Keeping track of this is important as it informs the system as to how "DOUT" is moving.

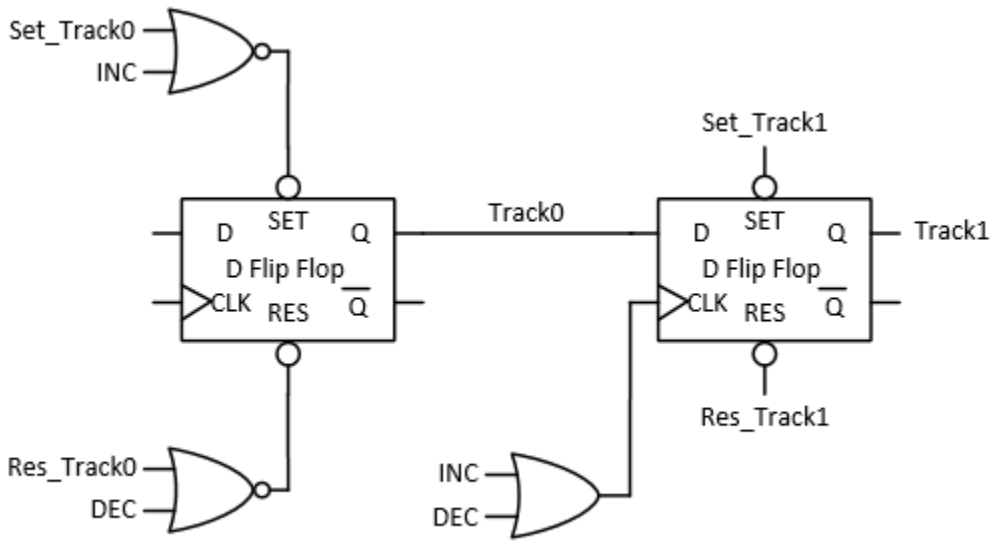

FIGURE 3.7. Track bits block

To accomplish this, the circuit shown in Fig. 3.7 has been employed. This represents the circuits found inside the Track Bits block found in the block diagram. The Track0 and Track1 lines are used for the memory of the level crossings. A pair of D flip flops are used to set these values. As can be seen in the figure the Track1 line is set by the previous level of Track0. At the same time the new Track0 is being set by either the "INC" or "DEC" lines going high. Assuming Set_Track0 and Res_Track0 are both low, the output of the NOR gates attached to the SET and RES have a high output while "INC" and "DEC" are low. As soon as either of these lines goes high, however, the respective NOR gate goes low and updates the value for Track0. An upper threshold crossing, indicated by "INC", corresponds to the Track bits being high, while the lower threshold crossing, indicated by "DEC", is represented by the Track bits being low.

Now that we have a good system to keep track of the RLC and CLC instances, there needs to be a way to convert the Track bits into inputs for the Add/Sub circuits shown in Fig. 3.5. To accomplish this, a flowchart for the logic is shown in Fig. 3.8. DOUTp stands for "DOUT+" and DOUTm stands for "DOUT-". The logic starts off by asking if "INC" or "DEC" has gone high. For either answer, a follow up question of the previous level crossings

value is asked. With these two questions all four of the possibilities are covered. Let us first take the instances where the first and second answers are different, i.e. "INC" to "DEC" and "DEC" to "INC". The actions taken at the end of these branches are very similar. The value for "DOUT" will be the same as the signal crossed the threshold where "DOUT" is already at. Then if the signal was increasing the "DOUT+" line is set to one above the new "DOUT" value while "DOUT-" is set to the new "DOUT" value. A similar process is done if the signal is decreasing but instead the "DOUT-" line is set to one below "DOUT" and "DOUT+" is equal to the new "DOUT".

Now let us look at when the history of level crossings is the same. These branches are found when the sequence is "INC" to "INC" and "DEC" to "DEC". As with the other branches, these two have a very similar function to be performed. As can be seen in Fig. 3.8 when the signal is "INC" to "INC" then we know that the value for "DOUT" needs to increase by one. Then "DOUT+" is set to one above the new "DOUT" and "DOUT-" is the new "DOUT".

Following the flow shown in Fig. 3.8 we now have to create digital logic to accomplish all of these actions. Shown in Table 3.2 is the truth table that was created to help figure out the most efficient logic to achieve this.

TABLE 3.2. Inputs to Add/Sub Circuits

Track1	**Track0**	**K**	**K-**	**K+**	**B0**	**B0-**	**B0+**
0	0	1	1	X	1	1	0
0	1	X	X	0	0	0	1
1	0	X	1	X	0	1	0
1	1	0	X	0	1	0	1

There are many ways to approach designing a digital logic circuit, but in this case using the "don't care" values, represented by X, help to facilitate some patterns that evolve from the table. Before going further with the analysis there will be a brief reminder as to what the values represent. The K, K-, and K+ values control if the ADD/SUB circuits are

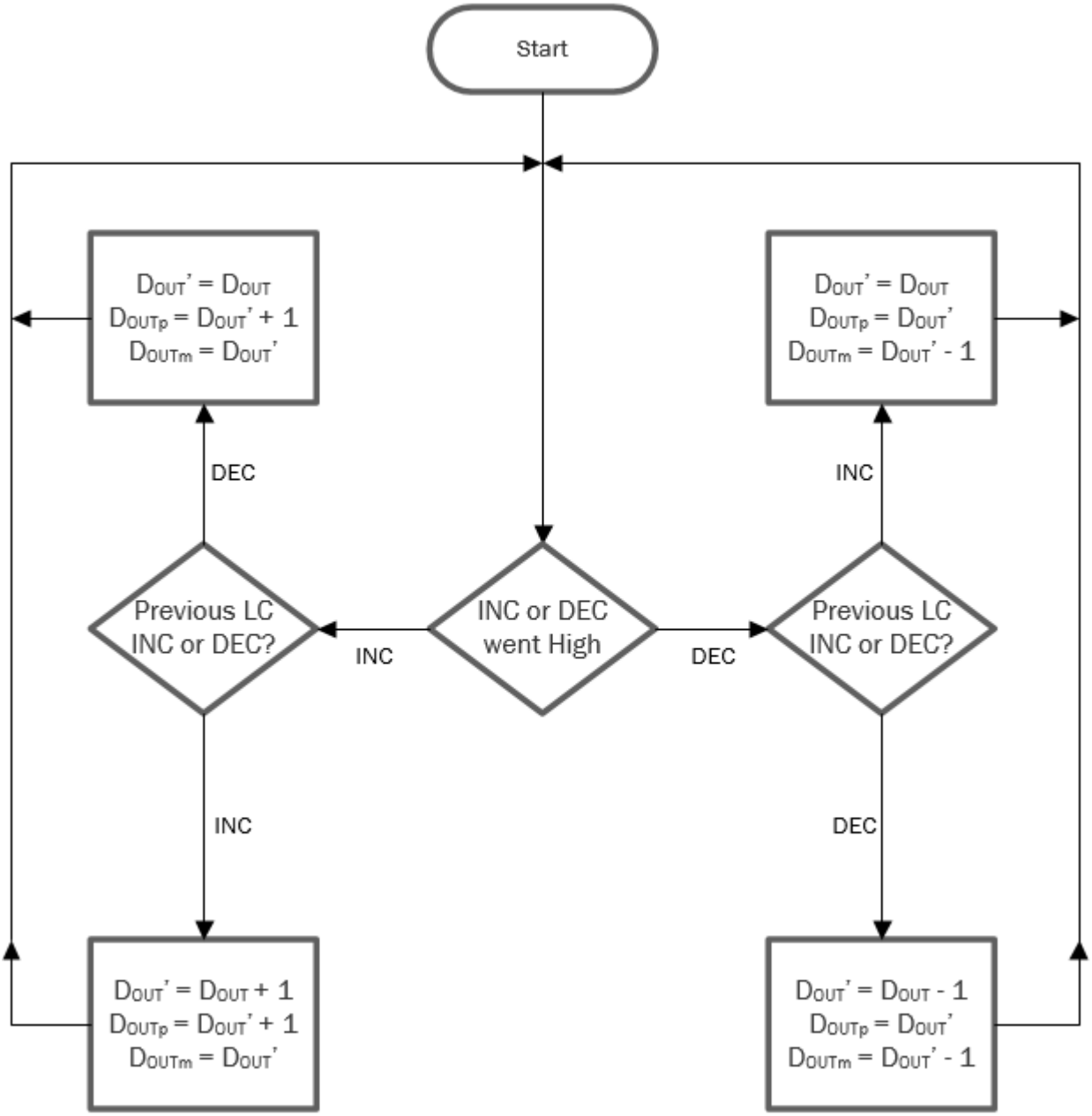

FIGURE 3.8. Keeping track of the level crossings. DOUTp = DOUT+, DOUTm = DOUT-, and DOUT' = new value of DOUT

adding or subtracting. The K values for this are 0 and 1 respectively. The B0 terms control the LSB for their respective outputs. The Track bits are there for tracking level crossings where 0 means "DEC" was triggered and an "INC" trigger is shown by a 1. Also the Track0

bit is the most recent crossing while Track1 is the crossing before that. The patterns that jump off of the page immediately are for K- and K+. K- is always high and K+ is always low no matter what the Track bits are doing. So these two values never have to change in the logic. The next bit looked at was K. This was another case where the don't care bits helped greatly in simplifying this logic. K can trivially be described as the inverse of Track0. It should also be pointed out that the other representation is the inverse of Track1, but for this design the Track0 bit was used. For B0, it can be seen that Track0 XNOR Track1 is the best way to describe it. The expression for B0- is simply realized by being the inverse of Track0. In the same vein, B0+ is just the inverse of B0-. Fig. 3.9 shows how all of these bits are laid out with actual logic gates. This figure represents the internals of the Control Bits block found in Fig. 3.4.

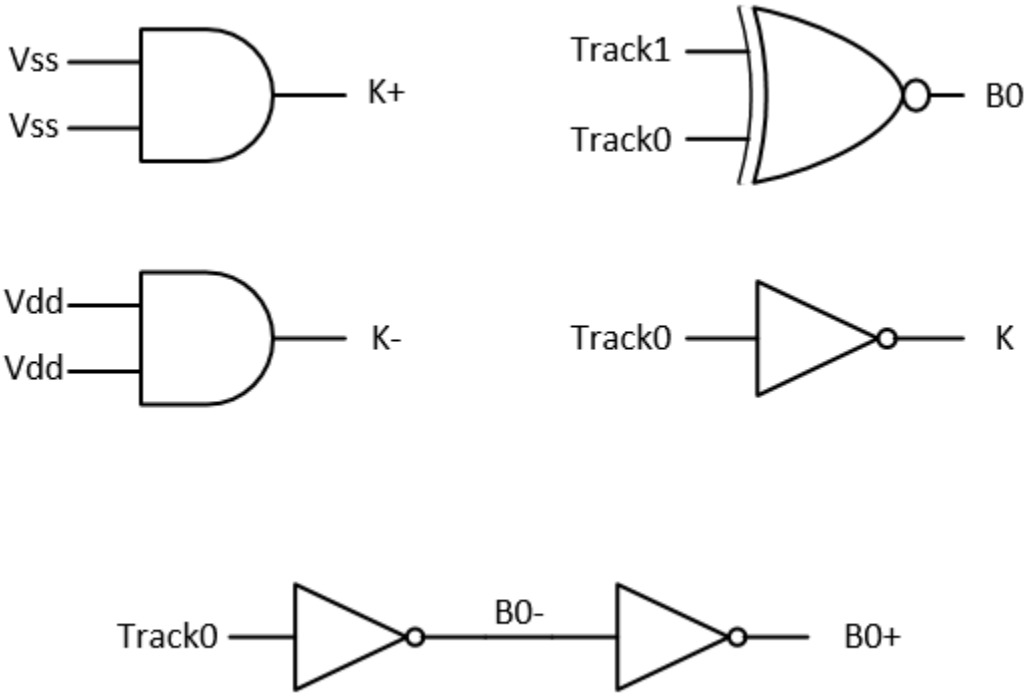

FIGURE 3.9. Control bits block

With the "DOUT", "DOUT-", and "DOUT+" logic all described that only leaves

the method for "Δt" left to be explained. This was realized through a combination of a ring oscillator and a counter circuit. For the counter circuit, a VerilogA code was written to count up to 10 bits. The code can be found in Appendix A. For the ring oscillator a current starved architecture was chosen. This allows the clock frequency to be set more accurately to the desired speed while designing it.

To explain the current starved ring oscillator more effectively a conventional ring oscillator's architecture will first be described. The traditional ring oscillator is comprised of a closed-loop of identical inverter stages that meet the Barkhausen criteria to oscillate [48]. The block diagram can be found in Fig. 3.10. In this figure, N must be odd and greater than 1 to be an effective ring oscillator.

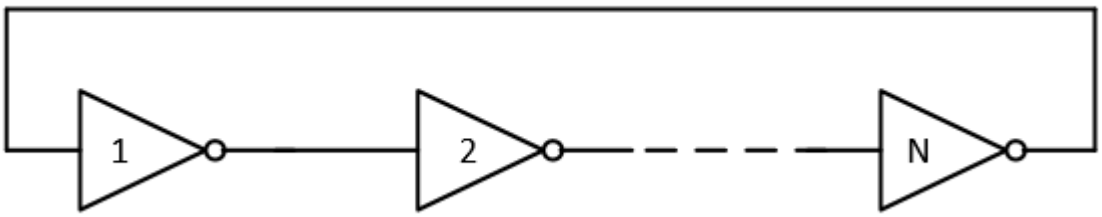

FIGURE 3.10. Traditional ring oscillator

To compute the frequency of oscillation there are only two variables required. One is the total delay time for each inverter stage, t_d. The other is the number of stages labeled as N. Eqn. 3.2 shows how to calculate oscillation frequency given the previous values [49]. The 2 found in the denominator of the equation accounts for the full low to high and high to low cycle found in the oscillator.

$$f = \frac{1}{2\,N\,t_d} \tag{3.2}$$

The current starved ring oscillator used has a similar working principle to the previously described traditional architecture with only one difference. The current starved version, as the name suggests, limits the amount of current available to the inverter stages [48]. As

the current increases in the system there is less resistance in the transistors producing a smaller delay through each stage.

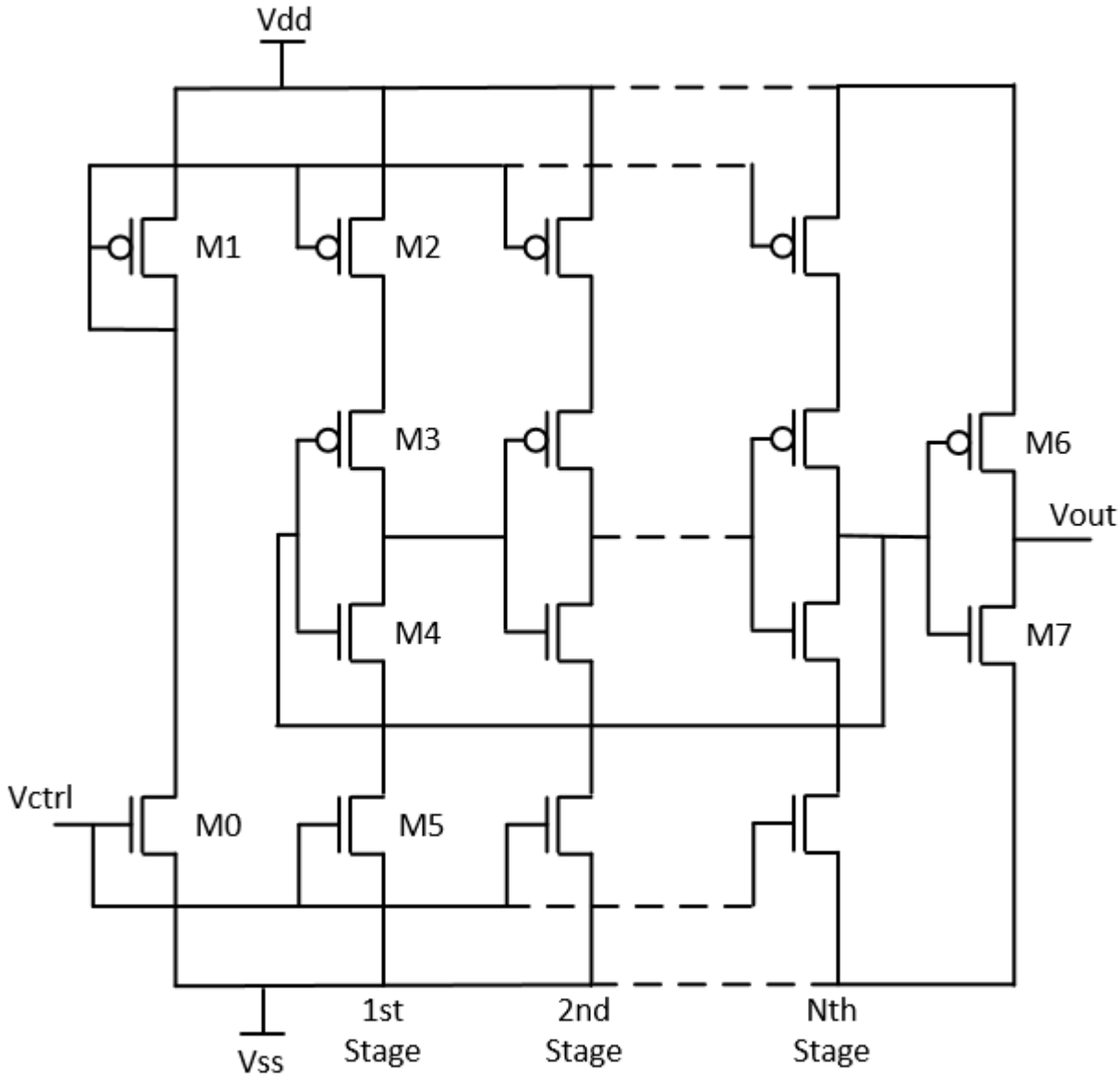

FIGURE 3.11. Current starved ring oscillator

Fig. 3.11 shows the design. Transistors M0 and M1 form the biasing stage that along with V_{ctrl} set the current available for each stage. Devices M3 and M4 form the inverter for the stage. M2 and M5 are the transistors that control the amount of current available to the inverter pair for the oscillation. Transistors M6 and M7 form a buffer for the output voltage

V_{out}. Notice that between the N^{th} stage and the out inverter that the signal gets rerouted back to the beginning of the circuit to carry on the oscillations. The value for N has the same restrictions found in Fig. 3.10. Table 3.3 shows a summary of the devices found in the ring oscillator circuit. One thing to point out that is not shown in Fig. 3.11 is that a 456 fF capacitor was introduced in between each inverter stage to help shape the delay time for each stage. The value of N was chosen to be 21 and the operating frequency of 1 MHz was achieved with a V_{ctrl} voltage of 893 mV.

TABLE 3.3. Ring Oscillator Summary

Device	**W/L**	$\mathbf{g_m/i_d}$
M_0	1 μm/500 nm	4.50
M_1	3.5 μm/500 nm	4.57
M_2	3.5 μm/500 nm	2.81
M_3	3 μm/500 nm	5.80
M_4	1.5 μm/500 nm	7.74
M_5	1 μm/500 nm	3.02
M_6	1.4 μm/300 nm	4.03
M_7	700 nm/300 nm	4.74

3.4. Simulation Results

The design for this level crossing ADC was implemented and simulated using a 180 nm CMOS process with a rail voltage of 1.8 V. This was accomplished using Cadence Virtuoso with the output data exported and processed in MATLAB to show the figures on display in this section. See Appendix B for the code used for processing.

The first simulation to be shown is for the comparator. The attributes to look for are 1) the output is high when Vin+ > Vin- and 2) the output doesn't take too long to react to a change in Vin+ and Vin-. Fig. 3.12 shows the simulation results. As can be seen the

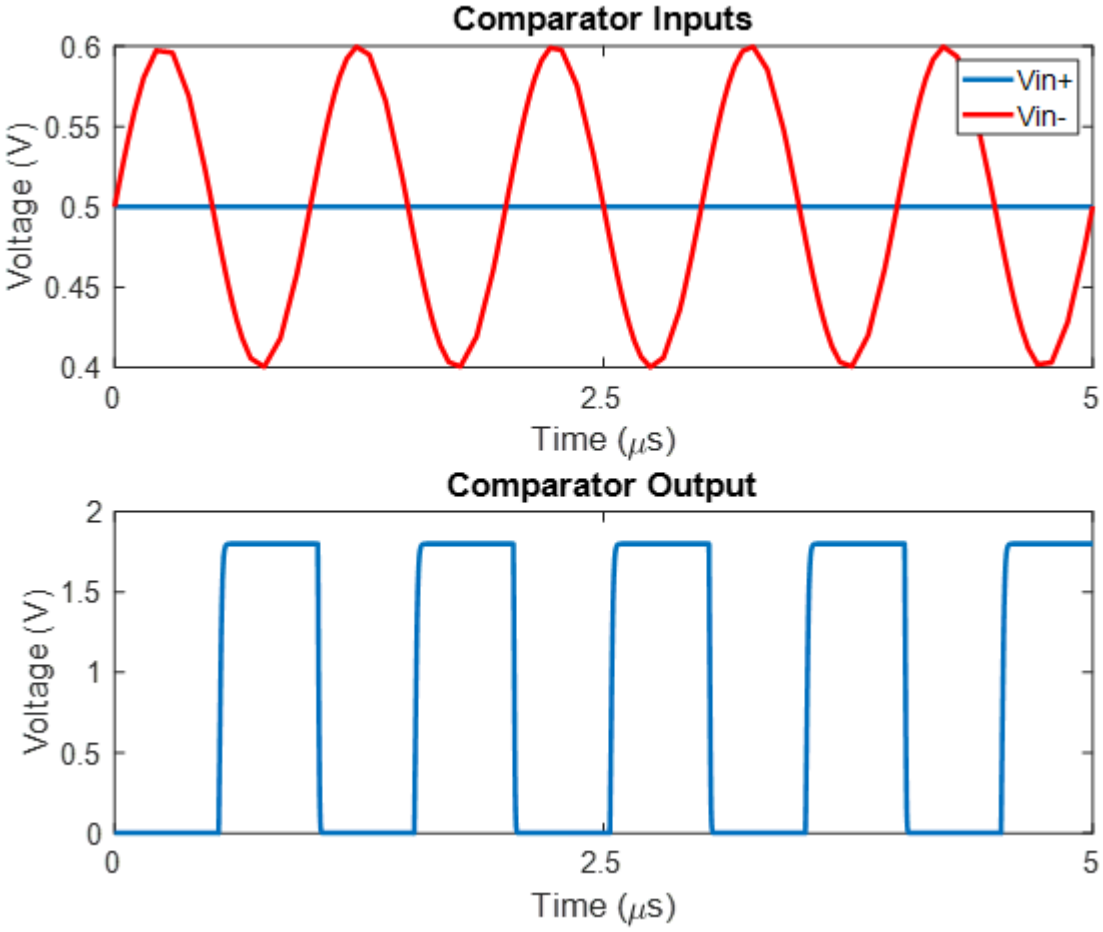

FIGURE 3.12. Comparator simulation results

comparator is only high when Vin+ is the greater signal and the circuit reacts very quickly to changes in this value.

The next result to be shown is for the DAC. The design for this paper as previously mentioned is for 10 bits. With this and the fact of the rail voltage being 1.8 V in mind, we can calculate what an expected jump in the output voltage will be for a 1 LSB increase at the input of the DAC. Plugging these values into Eqn. 3.1 we get that 1 LSB = $V_{ref}/2^{10}$ = 1.8/1024 = 1.7578 mV. Fig. 3.13 shows the simulation output.

Since displaying the full 10 bit range on one graph would be unseemly, only the first 16 digital codes for the DAC are shown. This equates to showing the 4 LSB bits for the device. The results are divided into the output voltage on top and the input code on the

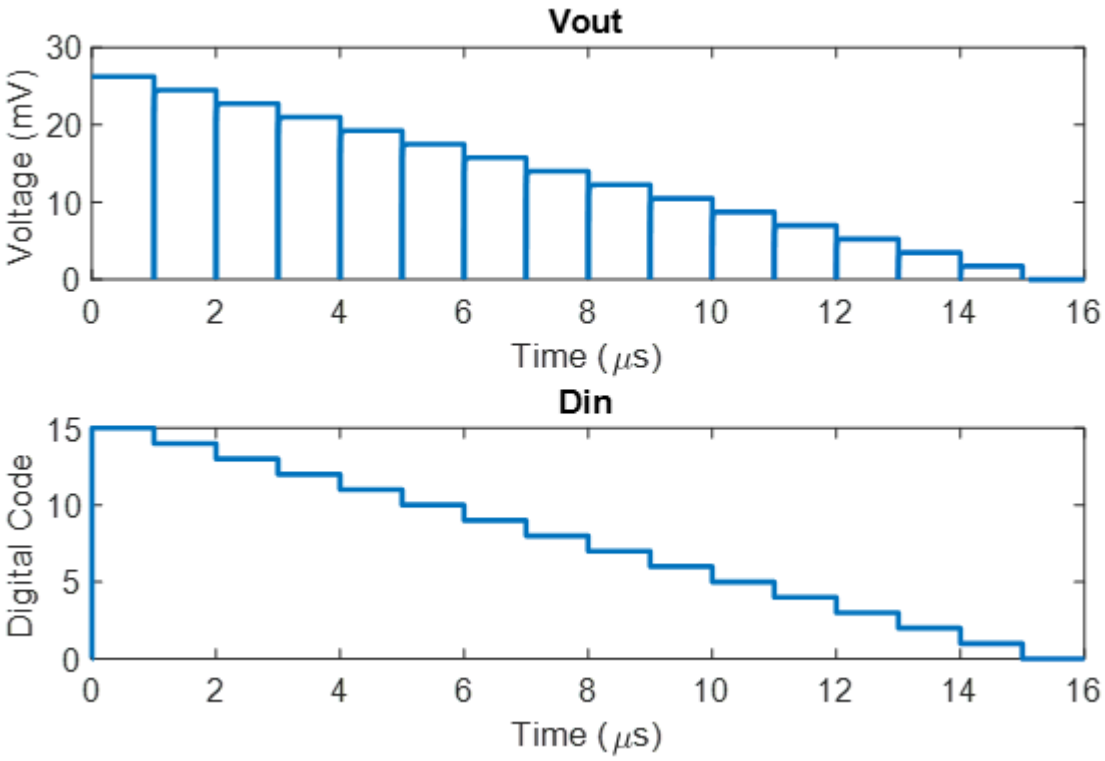

FIGURE 3.13. DAC simulation results

bottom. As can be seen we have the nice staircase effect for both graphs that we would expect. As the input code decreases by 1 LSB at each step, the output voltage decreases by approximately 1.7578 mV. To help illustrate the accuracy of the DAC a comparison of the ideal values and measured ones will be shown. See Table 3.4 for these results.

TABLE 3.4. DAC Results

Din	**Vout Ideal (mV)**	**Vout Measured (mV)**
15	26.367	26.245
14	24.609	24.495
13	22.851	22.745
12	21.094	20.994
11	19.336	19.246
10	17.578	17.496

As can be seen the ideal case and the measured values agree very well. In fact there is a pattern that emerges from this sample size. The measured values are consistently around 100 nV below what the ideal step values are. This could be used to further increase the accuracy of the ADC by noting what the actual values are in the reconstruction of the signal to be shown later. This thesis does not employ this tactic but it is something to keep in mind when accuracy at that scale is paramount.

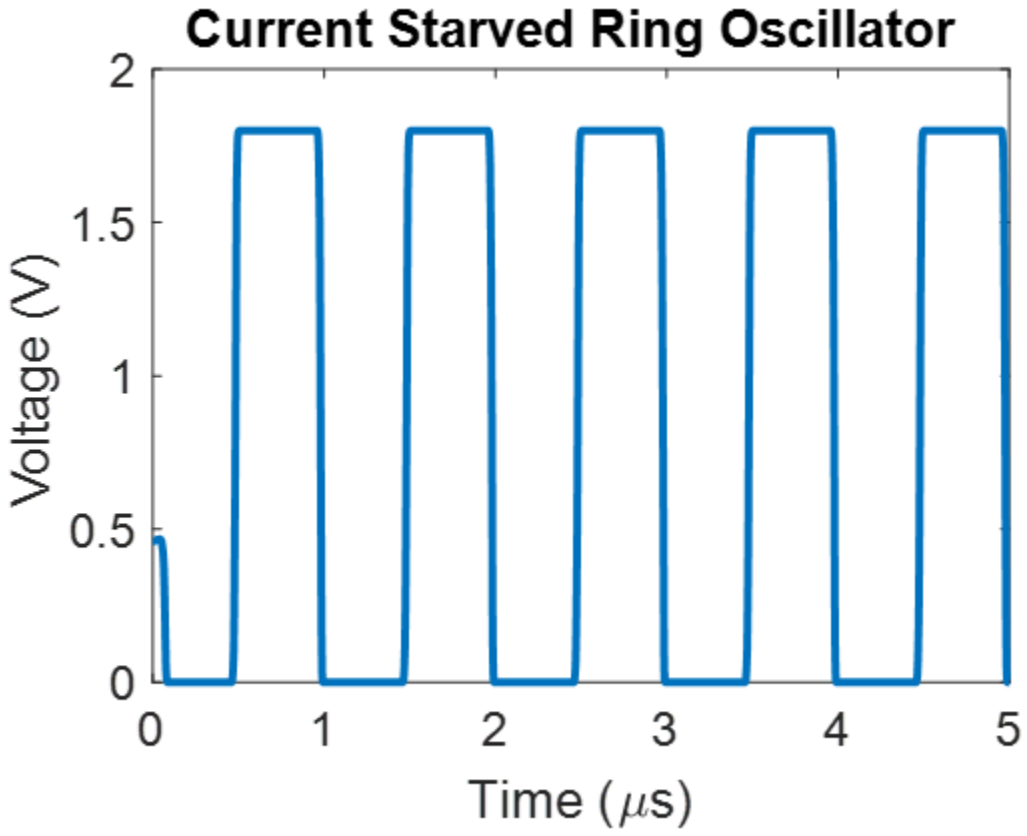

FIGURE 3.14. Ring oscillator simulation results

The next simulation to be discussed is for the ring oscillator. The timer resolution for this thesis was chosen to be 1 μs so the frequency for this circuit needed to be 1 MHz. This value was chosen so that the design can react fast enough for the high input slope parts of the input signal. The data shown in Fig. 3.14 shows that the oscillator achieves this frequency by repeating every 1 μs and thus being suitable for this design.

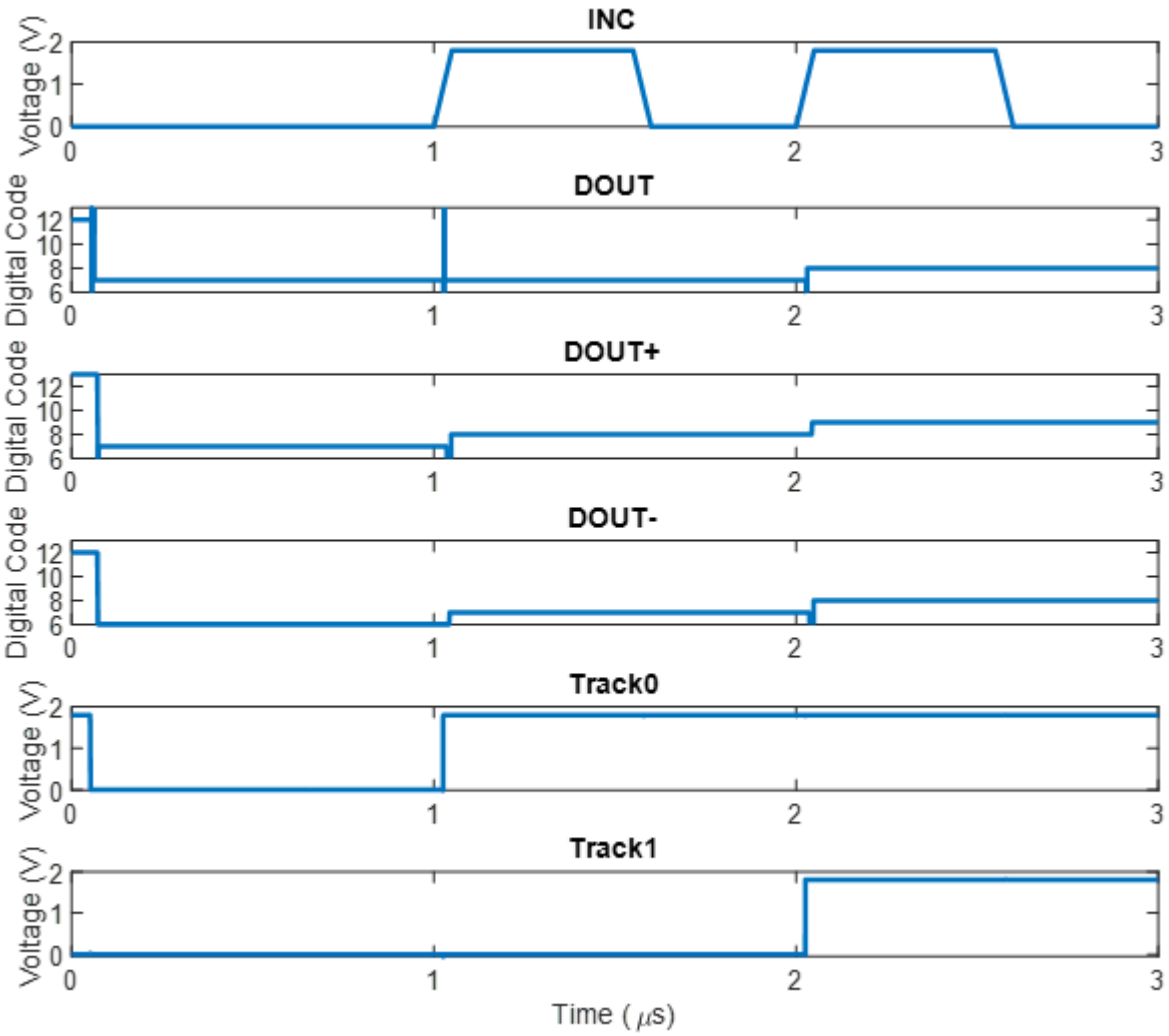

FIGURE 3.15. Digital logic simulation results 1

The last block of simulations to show are for the digital logic. To fully show that the design works as intended, the results have been split into two graphs. The first one, Fig. 3.15, shows when the Track bits both start low and then two "INC" pulses are sent into the circuit. Fig. 3.16 is similar in that it shows the Track bits starting high and then two "DEC" pulses are sent to verify the functionality of the design. One thing to note about Figures 3.15 and 3.16 is that the values at time 0 are the result of the initial states of the simulation and are quickly set to the actual starting values for the analysis.

The first case to be shown is in Fig. 3.15. The Track bits are representing that the

previous two level crossings were both "DEC". With this input to the logic, "DOUT" and "DOUT+" should be the same and "DOUT-" should be 1 below that value. From the simulation you can see that this assertion holds true. Then when the first "INC" pulse is sent at 1 μs the value for "DOUT" should stay the same while the "DOUT+" and "DOUT-" lines both move up by one to account for the changing signal. This is also confirmed in Fig. 3.15. The final "INC" pulse for this simulation is sent at 2 μs and sets both of the Track bits to high indicating the signal has increased in the past two cycles. For this case all three of the output lines "DOUT", "DOUT+", and "DOUT-" should increase by one and the simulation shows this happening.

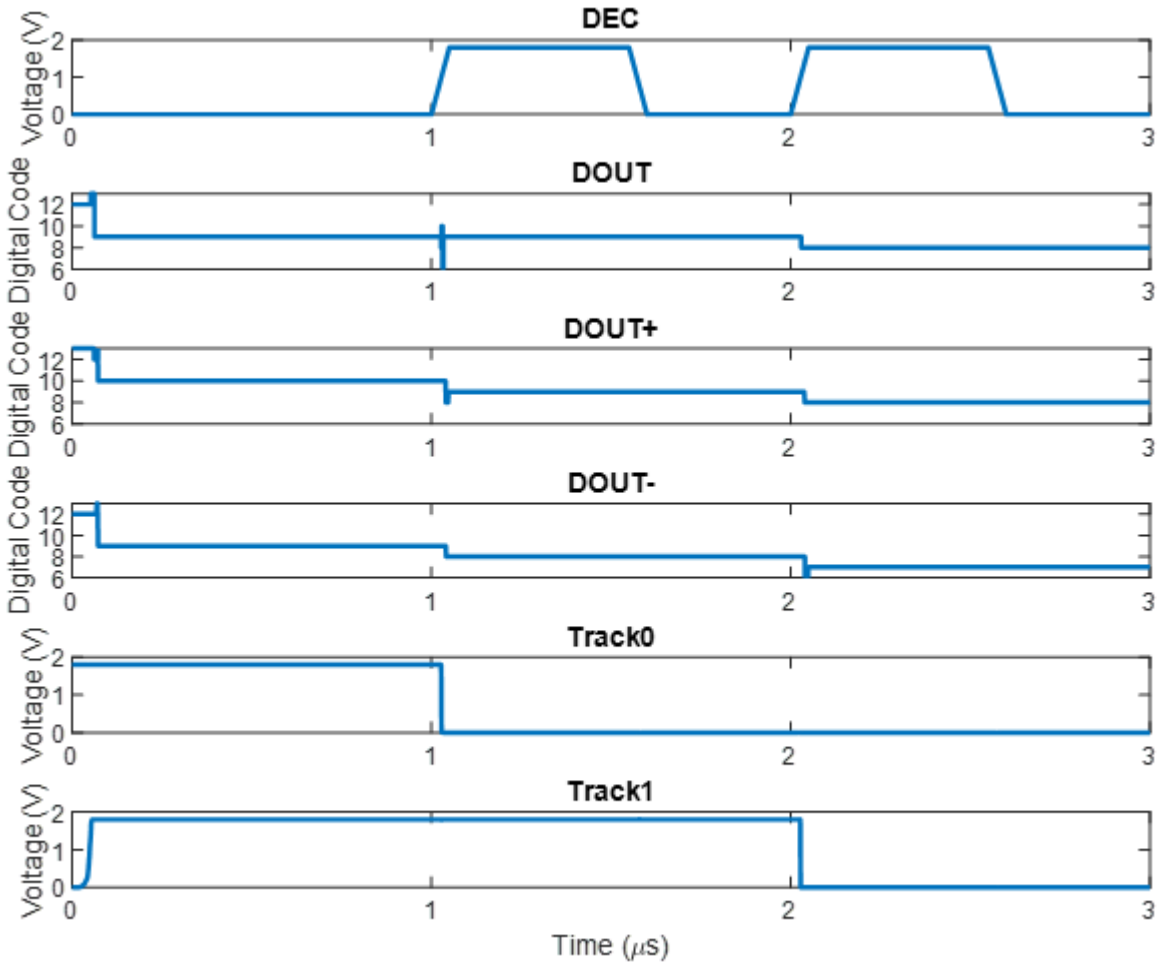

FIGURE 3.16. Digital logic simulation results 2

The last simulation to show before the overall results are presented can be found in

Fig. 3.16. In this case the Track bits are initially set to be high, showing that the signal has been increasing for the past two cycles. For this case we expect "DOUT" and "DOUT-" to be the same while "DOUT+" is one above these two. This hold up in the simulation. Then similar to the previous case when the "DEC" signal is sent at 1 μs the "DOUT" line will stay the same then "DOUT+" and "DOUT-" will both move down by one. Then on the next "DEC" pulse all three lines move down by one as expected.

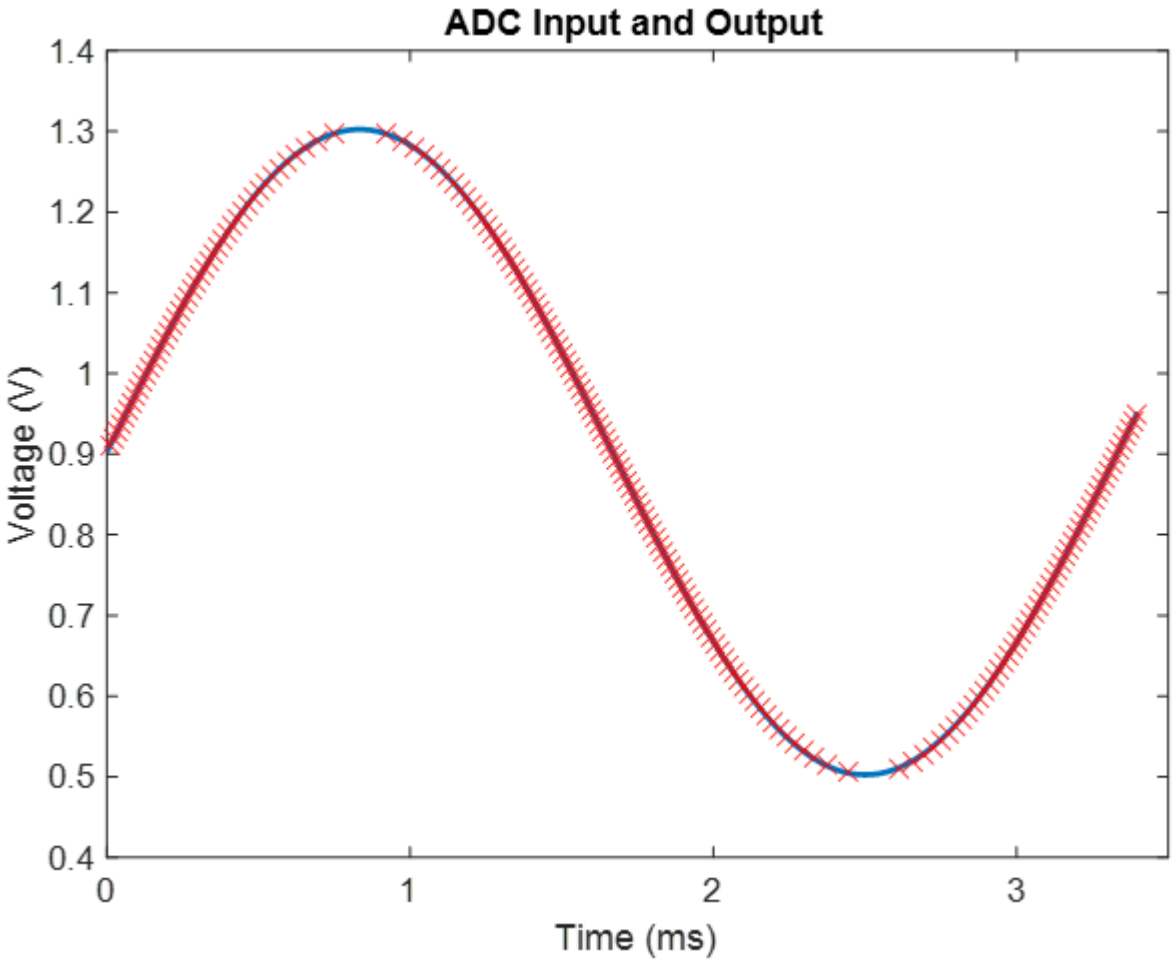

FIGURE 3.17. ADC simulation 1

Now that the individual blocks have been shown to be working as expected it is time to show how the overall ADC performs when integrated together. For this test an input sinusoid with an amplitude of 400 mV with a DC offset of 0.9 V and 300 Hz frequency was

used. This amplitude and offset mimics what would be an expected output for an analog front end (AFE) found in biomedical devices. The simulation was performed for just over a full cycle of this sin wave, so about 3.4 ms to be exact. As mentioned in the first part of this chapter the data from this simulation was exported to MATLAB and processed to gather the actual "DOUT" and "Δt" values. Appendix B goes into more detail on this process. Fig. 3.17 shows the entire cycle for this test case. The note about this figure is that only 1 in 5 samples is shown so the figure is more legible for this discussion. The full set of samples will be discussed in the next figure to be shown. The solid blue line shown is the actual input signal fed into the ADC. The red X's show a portion of the samples that the ADC took. The exact measurement of the error will be discussed later but to the naked eye there is a great deal of agreement between the ADC and the actual input signal.

Fig. 3.18 shows a more zoomed in look at output of the simulation. Unlike Fig. 3.17 this graph shows all of the samples taken in the ADC. Now the reason to show this zoomed in picture of the output is to verify that the level crossing nature of the designed ADC is functioning as intended. A reminder for the reader is that synchronous architectures do not change the amount of samples taken depending on the input signal. However, for the asynchronous design in this paper we should expect the rate of samples being taken to change as the slope of the signal increases and decreases. From Fig. 3.18 we can see this exact phenomenon happening. At the beginning of the graph the slope of the input signal is at its greatest. The amount of samples being taken in the system is also at its greatest as expected. As the signal begins to taper off at the zenith of the input, the amount of samples slows down and eventually we see a dead area at the absolute top. This is an easy sanity check to show that the proposed architecture is working as intended.

Up until this point the discussion over the results of the ADC and its individual blocks has been mostly qualitative. In the next section we will be taking a more quantitative look at this design as well as compare these performance parameters with the previous works found in the literature.

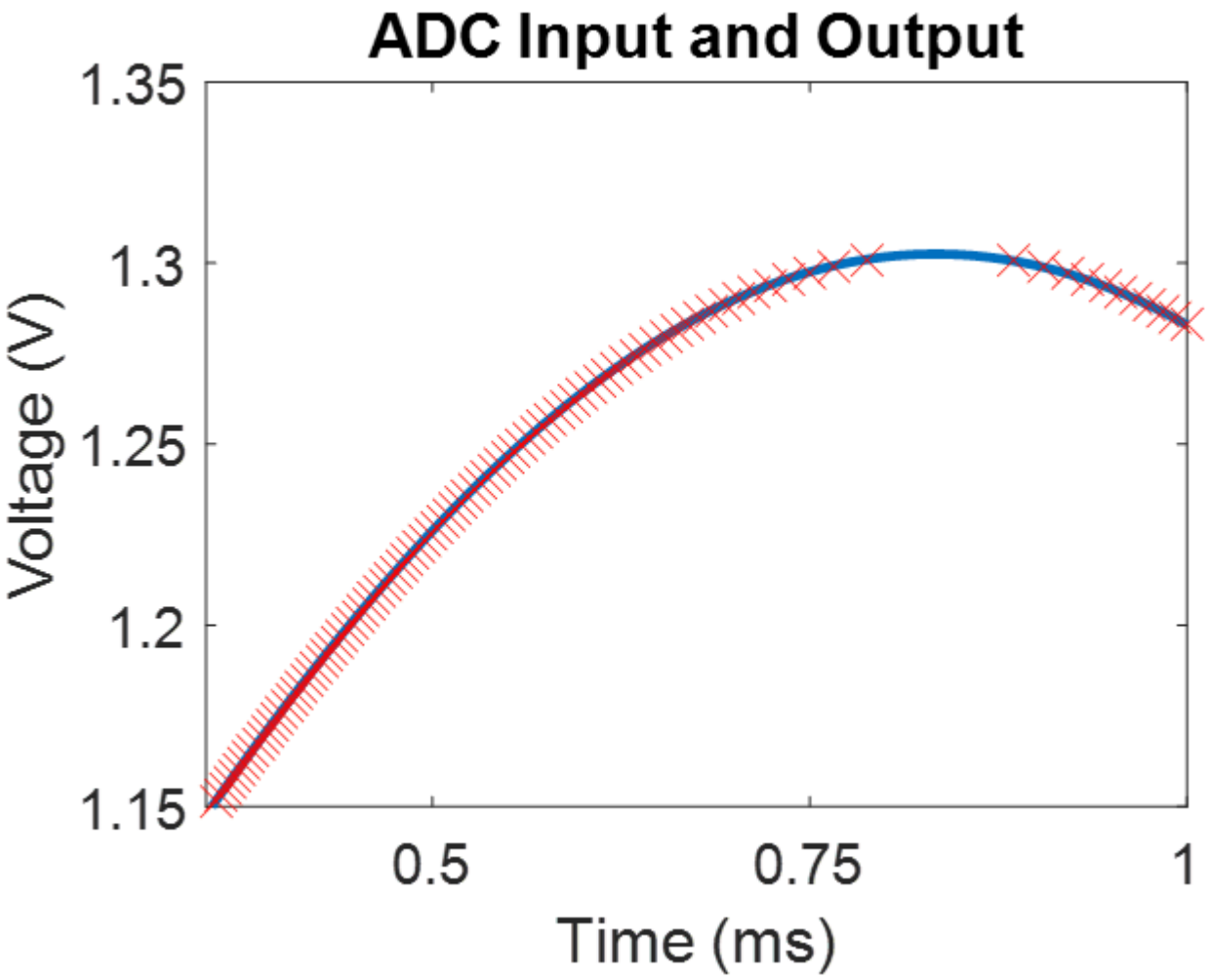

FIGURE 3.18. ADC simulation 2

3.5. Comparison With the State-of-the-Art

Before a comparison can be done we must first define how this design performs. The first metric to be looked at is the root mean square error (RMSE). While this metric isn't used extensively in the literature it used it to verify the accuracy of the ADC that Figures 3.17 and 3.18 suggested it had. To accomplish this an interpolation operation was performed on the outputs of the ADC in MATLAB. Then all of the time values from the input signal were plugged into the polynomial and compared with the actual voltage values found in the input. The code can be viewed in Appendix C. The result of this analysis found that the RMSE of the system was 0.65 mV. This confirms that the ADC has the high accuracy we thought it

had. The error found here can in part be attributed to the quantization error found in the time output.

The power calculation found in the table was calculated without the power drawn from the ring oscillator. Most of the previous works found in literature figures out the time in between samples via an outside source such as a logic analyzer or external clock signal and counter. The bandwidth was found by putting the timer resolution at 0.5 μs and estimating the fastest full scale input signal that could be put through the system without multiple crossings in the system. The ENOB and FOM presented in the table are calculated with the theoretical SNR of this system. The equations for these were found in Section 2.3.

TABLE 3.5. Performance Summary

Parameter	[30]	[50]	[37]	**This Work**
Technology (μm)	0.18	0.18	0.35	0.18
Supply Voltage (V)	0.7	0.8	1.8-2.4	1.8
Resolution (bits)	4-8	8	4-8	10
Adaptive Resolution	Yes	No	No	No
Timer Resolution (μs)	1	-	1.0-62.5	0.5
Bandwidth (kHz)	1	3.3	1	1.2
ENOB	8.4	-	6-8	9.81[b]
Power Consumption (μW)	25[a]	0.062-106	0.6-2.0	13.5[c]
FOM (pJ/conv.)	37	0.133-0.191.7	3.9	6.25[b]

[a] Without off-chip logic

[b] Calculated with derived SNR value

[c] Without ring oscillator power

Table 3.5 presents a comparison of this design with other works found in the literature. As can be seen the resolution was designed to be higher than the other mentioned works. This leads to the calculated ENOB to be higher than the other reported works. The static power consumption shown is also competitive at 13.5 μW of power draw.

3.6. Future Work

One of the aspects of a biomedical recording system that wasn't covered in this design is the amplifier at the analog front end (AFE). Designing one of these to have the appropriate amount of gain and noise suppression in the LFP and AP bandwidths for my system is high in priority. Another aspect of the design to tackle is to keep the 10-bit resolution by having an "aggressive" adaptive scheme. What is meant by this is that the resolution controller envisioned changes the resolution more often than the previously mentioned schemes as this design is for 10 bits. This scheme would be designed to try and provide a similar amount of data compression found in the papers with 4-8 bit variable resolution. The DAC structure is another design to look at changing in the future. The footprint could be reduced with the methods mentioned in Section 2.2 concerning either the fixed window comparison or the DAC alternatives. The last thing that will always be kept in mind for all designs going forward is reducing the power wherever possible. The best return on investment for this venture at first will definitely be the comparator designs as they account for over half of the simulated power. Another way to accomplish this goal is to reduce the supply voltage for the overall circuit.

CHAPTER 4

CONCLUSION

Different architectures for synchronous and asynchronous ADCs have been discussed. The application focused on in this thesis regarded neuropotential acquisition in biomedical circuits. The process of capturing both the LFP and APs found in these signals led to the design choices presented. As asynchronous level crossing ADCs have started being explored relatively recently, this thesis focused on a design for that architecture instead of the more traditional SAR ADCs used in literature. The level crossing ADC designed has 10-bit resolution that has a static power consumption of 13.5 μW.

APPENDIX A

VERILOGA CODE

```
// VerilogA for MasterThesis, VerilogA_CounterTest1, veriloga

`include "constants.vams"
`include "disciplines.vams"
`define NBITS 10
module VerilogA_CounterTest1(out, enable, clk);
inout clk;
//enable is actually used as a clear in this code
input enable;
electrical clk;
electrical enable;
output [`NBITS-1 :0] out;
electrical [`NBITS-1 :0] out;
parameter integer setval = 0 from [0:(1<<`NBITS)-1];
parameter real thresh = 0.6;
//setting up variables for cross statement
parameter real vtol = 0;
parameter real ttol = 0;
parameter real LogicHigh = 1;
parameter real enable_thresh = 0.6;
parameter real LogicLow = 0;
//setting up the voltage swings to mimic the counter circuit
parameter real tdel = 30p;
parameter real trise = 30p;
parameter real tfall = 30p;
parameter integer up = 0 from [0:1]; //0=increasing 1=decreasing
parameter integer stepsize = 1;
integer outval;

analog begin
@(initial_step) begin
    outval = setval;
    outval = (outval +(+up- !up)*stepsize)%(1<<`NBITS);
end
@(cross(V(enable)-enable_thresh,+1,ttol,vtol)) begin
    outval=0.0;
    outval = (outval +(+up- !up)*stepsize)%(1<<`NBITS);
end
@(cross(V(clk)-thresh,+1,ttol,vtol)) begin
    outval = (outval +(+up- !up)*stepsize)%(1<<`NBITS);
end
    generate j (`NBITS-1 , 0) begin
    V(out[j]) <+ transition ((!(outval &(1<<j)))*LogicHigh+!(outval&(1<<j))*LogicLow,tdel,trise,tfall);
    end
end

endmodule
```

The code shown is modified from code found in a post on the Cadence website [51]. This VerilogA code acts as an up counter to help track the time between samples. There are two inputs, labeled enable and clk, and a 10 bit output. The clk input is used to trigger the code to count up by one on a rising edge. The enable input is poorly named as in this

case it is actually used as a reset on the rising edge. The reasoning for this name choice was that this input was used for something else in previous testing of the code and simply was not changed to reflect what it's actual function in the logic is. The 10 bit output shows how many clock cycles have passed in between the reset signals. This combined with the knowledge of the clock frequency can be used to determine the time in between samples for the ADC.

APPENDIX B

MATLAB CODE FOR DATA MANIPULATION

The first code to be shown deals with how the output data of the simulations is processed to give the DOUT and Δt of the ADC. Before executing this code, the simulation data is first opened in excel and is trimmed to only show the data points shortly before and after the counter for Δt gets reset. Then this shortened simulation data file is fed into the following code to extract and plot the output of the ADC.

```
%Attempting to graph data from the output of ADC
%%
%run this setion if Table needs to transform into matrix
%NOTE: perform this section on the formatted input data
%Matrix_Name = Table_Name{:,:};
SinTest1CopyCopy = FullSinTestCopy{:,:};

%%
%Data manipulation section

%first set up parameters for the row and columns
inc = 1; dec = 2;
C9 = 3; C8 = 4; C7 = 5; C6 = 6; C5 = 7; C4 = 8; C3 = 9; C2 = 10;
    C1 = 11; C0 = 12;
D0 = 13; D1 = 14; D2 = 15; D3 = 16; D4 = 17; D5 = 18; D6 = 19; D7
    = 20; D8 = 21; D9 = 22;

%determine height of table for the while loop
RowMax = size(SinTest1CopyCopy,1);

%set up variables for the outputs
%NOTE: SampleMax is an estimate of how many samples there might be
```

```
    this is
%just to preallocate the variables to speed up the logic
SampleMax = 2000;
Dout0 = zeros(1, SampleMax);
Dout1 = zeros(1, SampleMax);
Dout2 = zeros(1, SampleMax);
Dout3 = zeros(1, SampleMax);
Dout4 = zeros(1, SampleMax);
Dout5 = zeros(1, SampleMax);
Dout6 = zeros(1, SampleMax);
Dout7 = zeros(1, SampleMax);
Dout8 = zeros(1, SampleMax);
Dout9 = zeros(1, SampleMax);

Count0 = zeros(1, SampleMax);
Count1 = zeros(1, SampleMax);
Count2 = zeros(1, SampleMax);
Count3 = zeros(1, SampleMax);
Count4 = zeros(1, SampleMax);
Count5 = zeros(1, SampleMax);
Count6 = zeros(1, SampleMax);
Count7 = zeros(1, SampleMax);
Count8 = zeros(1, SampleMax);
Count9 = zeros(1, SampleMax);


```

```
47 %logic for gathering the data points
48 %variable for while loop
49 n = 1;
50 %variable for indexing Dout and Count arrays
51 i = 1;
52 while n <= RowMax
53     if ((SinTest1CopyCopy(n, C9) == 1) || (SinTest1CopyCopy(n, C8)
           == 1)...
54             || (SinTest1CopyCopy(n, C7) == 1)...
55             || (SinTest1CopyCopy(n, C6) == 1)...
56             || (SinTest1CopyCopy(n, C5) == 1)...
57             || (SinTest1CopyCopy(n, C4) == 1)...
58             || (SinTest1CopyCopy(n, C3) == 1)...
59             || (SinTest1CopyCopy(n, C2) == 1)...
60             || (SinTest1CopyCopy(n, C1) == 1)...
61             || (SinTest1CopyCopy(n, C0) == 1))
62         %set Count variables
63         Count0(i) = SinTest1CopyCopy(n, C0);
64         Count1(i) = SinTest1CopyCopy(n, C1);
65         Count2(i) = SinTest1CopyCopy(n, C2);
66         Count3(i) = SinTest1CopyCopy(n, C3);
67         Count4(i) = SinTest1CopyCopy(n, C4);
68         Count5(i) = SinTest1CopyCopy(n, C5);
69         Count6(i) = SinTest1CopyCopy(n, C6);
70         Count7(i) = SinTest1CopyCopy(n, C7);
71         Count8(i) = SinTest1CopyCopy(n, C8);
72         Count9(i) = SinTest1CopyCopy(n, C9);
```

```
73
74          %increment n by 100 to skip count values
75          n = n + 100;
76
77          %increment i
78          i = i + 1;
79      end
80      n = n + 1;
81  end
82
83  v = 20;
84  x = 1;
85  while v <= RowMax
86      if ((SinTest1CopyCopy(v, C9) == 1) || (SinTest1CopyCopy(v, C8)
            == 1)...
87              || (SinTest1CopyCopy(v, C7) == 1)...
88              || (SinTest1CopyCopy(v, C6) == 1)...
89              || (SinTest1CopyCopy(v, C5) == 1)...
90              || (SinTest1CopyCopy(v, C4) == 1)...
91              || (SinTest1CopyCopy(v, C3) == 1)...
92              || (SinTest1CopyCopy(v, C2) == 1)...
93              || (SinTest1CopyCopy(v, C1) == 1)...
94              || (SinTest1CopyCopy(v, C0) == 1))
95          v = v - 1;
96          Dout0(x) = SinTest1CopyCopy(v, D0);
97          Dout1(x) = SinTest1CopyCopy(v, D1);
98          Dout2(x) = SinTest1CopyCopy(v, D2);
```

```
99              Dout3(x) = SinTest1CopyCopy(v, D3);
100             Dout4(x) = SinTest1CopyCopy(v, D4);
101             Dout5(x) = SinTest1CopyCopy(v, D5);
102             Dout6(x) = SinTest1CopyCopy(v, D6);
103             Dout7(x) = SinTest1CopyCopy(v, D7);
104             Dout8(x) = SinTest1CopyCopy(v, D8);
105             Dout9(x) = SinTest1CopyCopy(v, D9);
106             v = v + 100;
107             x = x + 1;
108         end
109         v = v + 1;
110 end
111
112 %now to set the final value for Dout
113 Dout0(x) = SinTest1CopyCopy(RowMax, D0);
114 Dout1(x) = SinTest1CopyCopy(RowMax, D1);
115 Dout2(x) = SinTest1CopyCopy(RowMax, D2);
116 Dout3(x) = SinTest1CopyCopy(RowMax, D3);
117 Dout4(x) = SinTest1CopyCopy(RowMax, D4);
118 Dout5(x) = SinTest1CopyCopy(RowMax, D5);
119 Dout6(x) = SinTest1CopyCopy(RowMax, D6);
120 Dout7(x) = SinTest1CopyCopy(RowMax, D7);
121 Dout8(x) = SinTest1CopyCopy(RowMax, D8);
122 Dout9(x) = SinTest1CopyCopy(RowMax, D9);
123
124 %%
125 %process Dout values to get 1's and 0's
```

```
126 %WARNING: previous section must be run to gather the data for Dout
127
128 threshold = 1;
129 j = i-1;
130 for k = 1:j
131     if Dout0(k) < threshold
132         Dout0(k) = 0;
133     else
134         Dout0(k) = 1;
135     end
136 end
137 for k = 1:j
138     if Dout1(k) < threshold
139         Dout1(k) = 0;
140     else
141         Dout1(k) = 1;
142     end
143 end
144 for k = 1:j
145     if Dout2(k) < threshold
146         Dout2(k) = 0;
147     else
148         Dout2(k) = 1;
149     end
150 end
151 for k = 1:j
152     if Dout3(k) < threshold
```

```
153             Dout3(k) = 0;
154         else
155             Dout3(k) = 1;
156         end
157     end
158     for k = 1:j
159         if Dout4(k) < threshold
160             Dout4(k) = 0;
161         else
162             Dout4(k) = 1;
163         end
164     end
165     for k = 1:j
166         if Dout5(k) < threshold
167             Dout5(k) = 0;
168         else
169             Dout5(k) = 1;
170         end
171     end
172     for k = 1:j
173         if Dout6(k) < threshold
174             Dout6(k) = 0;
175         else
176             Dout6(k) = 1;
177         end
178     end
179     for k = 1:j
```

```
180     if Dout7(k) < threshold
181         Dout7(k) = 0;
182     else
183         Dout7(k) = 1;
184     end
185 end
186 for k = 1:j
187     if Dout8(k) < threshold
188         Dout8(k) = 0;
189     else
190         Dout8(k) = 1;
191     end
192 end
193 for k = 1:j
194     if Dout9(k) < threshold
195         Dout9(k) = 0;
196     else
197         Dout9(k) = 1;
198     end
199 end
200
201
202
203
204
205 %%
206 %Setting up vin from the table import
```

```
207 %similar process for first section
208 SinTest1Copy = FullSinTestvin{:,:};
209
210 %%
211 %Pre-Graphing section
212 %WARNING: previous 2 section must be run first to gather the data
      to graph
213 %as well as import the data for vinx and viny
214
215
216 %first get rid of superfluous 0's at the ends of Count and Dout
      arrays
217
218 L = i - 1;
219 fDout0 = zeros(1, L);
220 fDout1 = zeros(1, L);
221 fDout2 = zeros(1, L);
222 fDout3 = zeros(1, L);
223 fDout4 = zeros(1, L);
224 fDout5 = zeros(1, L);
225 fDout6 = zeros(1, L);
226 fDout7 = zeros(1, L);
227 fDout8 = zeros(1, L);
228 fDout9 = zeros(1, L);
229 Dout_total = zeros(1, L);
230
231 fCount0 = zeros(1, L);
```

```
232 fCount1 = zeros(1, L);
233 fCount2 = zeros(1, L);
234 fCount3 = zeros(1, L);
235 fCount4 = zeros(1, L);
236 fCount5 = zeros(1, L);
237 fCount6 = zeros(1, L);
238 fCount7 = zeros(1, L);
239 fCount8 = zeros(1, L);
240 fCount9 = zeros(1, L);
241 Count_total = zeros(1, L);
242 Count_temp = zeros(1, L);
243
244 %setting up Vin
245 Rows = size(SinTest1Copy, 1);
246 vinx = zeros(1, Rows);
247 viny = zeros(1, Rows);
248
249 vinx_index = 1;
250 viny_index = 2;
251 for k = 1:Rows
252     vinx(k) = SinTest1Copy(k, vinx_index);
253 end
254 for k = 1:Rows
255     viny(k) = SinTest1Copy(k, viny_index);
256 end
257
258 %now to process dout and count
```

```
259 for k = 1:L
260     fDout0(k) = Dout0(k);
261 end
262 for k = 1:L
263     fDout1(k) = Dout1(k);
264 end
265 for k = 1:L
266     fDout2(k) = Dout2(k);
267 end
268 for k = 1:L
269     fDout3(k) = Dout3(k);
270 end
271 for k = 1:L
272     fDout4(k) = Dout4(k);
273 end
274 for k = 1:L
275     fDout5(k) = Dout5(k);
276 end
277 for k = 1:L
278     fDout6(k) = Dout6(k);
279 end
280 for k = 1:L
281     fDout7(k) = Dout7(k);
282 end
283 for k = 1:L
284     fDout8(k) = Dout8(k);
285 end
```

```
286 for k = 1:L
287     fDout9(k) = Dout9(k);
288 end
289
290 for k = 1:L
291     fCount0(k) = Count0(k);
292 end
293 for k = 1:L
294     fCount1(k) = Count1(k);
295 end
296 for k = 1:L
297     fCount2(k) = Count2(k);
298 end
299 for k = 1:L
300     fCount3(k) = Count3(k);
301 end
302 for k = 1:L
303     fCount4(k) = Count4(k);
304 end
305 for k = 1:L
306     fCount5(k) = Count5(k);
307 end
308 for k = 1:L
309     fCount6(k) = Count6(k);
310 end
311 for k = 1:L
312     fCount7(k) = Count7(k);
```

```
313 end
314 for k = 1:L
315     fCount8(k) = Count8(k);
316 end
317 for k = 1:L
318     fCount9(k) = Count9(k);
319 end
320
321
322 Count_LSB = 0.000001; Dout_LSB = 1.8/1024;
323 Dout_total = (fDout0 + 2*fDout1 + 4*fDout2 + 8*fDout3 + 16*fDout4
    ...
324     + 32*fDout5 + 64*fDout6 + 128*fDout7 + 256*fDout8...
325     + 512*fDout9)*Dout_LSB;
326
327 Count_temp = (fCount0 + 2*fCount1 + 4*fCount2 + 8*fCount3 + 16*
    fCount4...
328     + 32*fCount5 + 64*fCount6 + 128*fCount7 + 256*fCount8...
329     + 512*fCount9)*Count_LSB;
330 %Count_final is formatted to show the time between samples
331 %we need total time so more processing is required
332 Count_total(1) = Count_temp(1);
333 for k = 2:L
334     Count_total(k) = Count_temp(k) + Count_total(k-1);
335 end
336
337 %because the signal was delayed 1 microsecond
```

```
338 Count_total = Count_total-0.000001;
339
340 %%
341 %this section for taking the first microsecond off of vinx and
      viny
342 n = 828; %n is 1 microsecond index
343 RowMax2 = size(vinx,2);
344 temp = RowMax2-n+1;
345 vinxtest = zeros(1,temp);
346 vinytest = zeros(1,temp);
347 i = 0;
348 for k = n:RowMax2
349     i = i + 1;
350     vinxtest(i) = vinx(k);
351     vinytest(i) = viny(k);
352 end
353 vinxtest = vinxtest-0.000001;
354
355 %%
356 %section for reducing number of data points for Dout_total to
      graph
357 samp = 5; %make sure RowMax3/samp is integer
358 RowMax3 = size(Dout_total,2);
359 Dout_reduc = zeros(1,RowMax3/samp);
360 Count_reduc = zeros(1,RowMax3/samp);
361
362 for k = 1:(RowMax3/samp)
```

```
363         Dout_reduc(k) = Dout_total(k*samp);
364         Count_reduc(k) = Count_total(k*samp);
365 end
366
367
368 %%
369 %Graphing Section
370 %WARNING: make sure all data is completely ready for graphing
371
372 figure('DefaultAxesFontSize',18)
373 plot(vinxtest,vinytest,'LineWidth',3)
374 hold on
375 scatter(Count_total, Dout_total,200,'x', 'red')
376 title('ADC Input and Output')
377 xlabel('Time (ms)')
378 ylabel('Voltage (V)')
379 ylim([1.15 1.35])
380 xlim([0.00035 0.001])
381 xticks([0 0.00025 0.0005 0.00075 0.001])
382 xticklabels({'0', '0.25', '0.5','0.75', '1'})
383 hold off
384
385 %less data samples shown
386 figure('DefaultAxesFontSize',18)
387 plot(vinxtest,vinytest,'LineWidth',3)
388 hold on
389 scatter(Count_reduc, Dout_reduc,200,'x', 'red')
```

```
390 title('ADC Input and Output')
391 xlabel('Time (ms)')
392 ylabel('Voltage (V)')
393 ylim([0.4 1.4])
394 xlim([0 0.0035])
395 xticks([0 0.001 0.002 0.003])
396 xticklabels({'0', '1', '2','3'})
397 hold off
```

APPENDIX C

MATLAB CODE FOR RMSE ANALYSIS

The following code has a much simpler job than the previously laid out code. The following is for interpolating the output of the ADC and finding the RMSE that was reported in the thesis.

```
%interpolating the ADC data and finding RMSE
%first find coefficients

%taking off the first microsecond of vinx and viny
n = 828; %n is 1 microsecond index
RowMax = size(vinx,2);
temp = RowMax-n+1;
vinxtest = zeros(1,temp);
vinytest = zeros(1,temp);
i = 0;
for k = n:RowMax
    i = i + 1;
    vinxtest(i) = vinx(k);
    vinytest(i) = viny(k);
end
vinxtest = vinxtest -0.000001;

p = polyfit(Count_total,Dout_total,7);
%next evaluate the polynomial with the original vinx data
viny_interpolated = polyval(p,vinxtest);
%now find RMSE
RMSE = sqrt(mean((viny_interpolated - vinytest).^2));
%find percentage difference (Mean Absolute Percentage Error)
MAPE = mean((vinytest-viny_interpolated)./vinytest);
```

```
25
26
27 figure
28 scatter(Count_total,Dout_total)
29 hold on
30 plot(vinxtest,viny_interpolated)
31 ylim([0.4 1.4])
32 hold off
33
34 figure
35 plot(vinxtest,vinytest,'LineWidth',1.8)
36 hold on
37 plot(vinxtest,viny_interpolated, '--r','LineWidth',1.8)
38 ylim([0.4 1.4])
39 hold off
```

REFERENCES

[1] H. Tang, Z. C. Sun, K. W. R. Chew, and L. Siek, "A 5.8 nw 9.1-enob 1-ks/s local asynchronous successive approximation register adc for implantable medical device," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2221–2225, 2014.

[2] M. Pagin and M. Ortmanns, "Evaluation of logarithmic vs. linear adcs for neural signal acquisition and reconstruction," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 4387–4390, 2017.

[3] N. Tasneem and I. Mahbub, "A 2.53 nef 8-bit 10 ks/s 0.5 m cmos neural recording read-out circuit with high linearity for neuromodulation implants," *Electronics*, vol. 10, no. 5, p. 590, 2021. Copyright - © 2021. This work is licensed under http://creativecommons.org/licenses/by/3.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2021-03-09.

[4] A. T. Do, Y. Tan, C. Lam, M. Je, and K. S. Yeo, "Low power implantable neural recording front-end," in *2012 International SoC Design Conference (ISOCC)*, pp. 387–390, 2012.

[5] J. Jimenez, S. Dai, and J. K. Rosenstein, "A microwatt front end and asynchronous adc for sparse biopotential acquisition," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 503–506, 2017.

[6] C. Weltin-Wu and Y. Tsividis, "An event-driven clockless level-crossing adc with signal-dependent adaptive resolution," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 9, pp. 2180–2190, 2013.

[7] W. Kester, *The Data Conversion Handbook*. Burlington, MA: Newnes, 2005.

[8] R. J. Baker, *CMOS Circuit Design, Layout, and Simulation*. Hoboken, New Jersey: Wiley, 3rd ed., 2010.

[9] V. Zamaruiev, V. Ivakhno, and B. Styslo, "Anti-aliasing filter in digital control sys-

tem for converter with active power filter function," in *2019 IEEE 39th International Conference on Electronics and Nanotechnology (ELNANO)*, pp. 797–801, 2019.

[10] E. Crespo Marques, N. Maciel, L. Naviner, H. Cai, and J. Yang, "A review of sparse recovery algorithms," *IEEE Access*, vol. 7, pp. 1300–1322, 2019.

[11] F. Chen, A. P. Chandrakasan, and V. M. Stojanovic, "Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 744–756, 2012.

[12] Y. Mingfei and L. Shengli, "A bandwidth controllable anti-aliasing filter design method," in *2018 IEEE 18th International Conference on Communication Technology (ICCT)*, pp. 641–644, 2018.

[13] Megha R and Pradeepkumar K A, "Implementation of low power flash adc by reducing comparators," in *2014 International Conference on Communication and Signal Processing*, pp. 443–447, 2014.

[14] R. E. Rad, S. J. Kim, A. Hejazi, M. R. Ur Rehman, Z. Bai, D. Ziqi, and K. Lee, "A low power 12-bit pipeline adc with 40 ms/s using a modified op-amp," in *2020 International Conference on Electronics, Information, and Communication (ICEIC)*, pp. 1–3, 2020.

[15] V. P. Singh, G. K. Sharma, and A. Shukla, "Power efficient sar adc designed in 90 nm cmos technology," in *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pp. 1–5, 2017.

[16] V. S. Sooraj and G. M. Joseph, "Speed resolution enhancement of 12-bit sar adc," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 1655–1658, 2018.

[17] ""pipeline adcs come of age"." `https://www.maximintegrated.com/en/design/technical-documents/tutorials/6/634.html`. Accessed: 5-Mar-2021.

[18] R. Loehr, M. Kempf, F. Ohnhaeuser, J. Roeber, R. Weigel, and A. Baenisch, "Implementation of a high-speed flash adc for high-performance pipeline adcs in an 180nm cmos process," in *2015 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pp. 317–322, 2015.

[19] "Understanding pipeline analog to digital converter." https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1023.html. Accessed: 5-Mar-2021.

[20] A. Jayakumar and K. Vishnu, "A 7-bit 500-mhz flash adc," in *2014 First International Conference on Computational Systems and Communications (ICCSC)*, pp. 75–79, 2014.

[21] S. Mahdavi, F. Noruzpur, R. Ebrahimi, and Z. Alizad, "Analysis simulation and comparison different types of the sigma delta adc modulators based on ideal model system level and behavioral model using matlab," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pp. 0252–0259, 2017.

[22] D. Prasad and V. Nath, "Design of cmos difference amplifier circuit for sigma delta adc for aerospace applications," in *2017 International Conference on Information, Communication, Instrumentation and Control (ICICIC)*, pp. 1–4, 2017.

[23] "Sigma delta adcs." https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1870.html. Accessed: 5-Mar-2021.

[24] H. Inose, T. Aoki, and K. Watanabe, "Asynchronous delta-modulation system," in *Electron. Lett.*, vol. 2, pp. 95–96, 1966.

[25] P. D. Sharma, "Characteristics of asynchronous delta modulation and binary slope quantized pcm systems," in *Electron. Eng.*, vol. 40, pp. 32–37, 1968.

[26] Y. Li, D. Zhao, and W. A. Serdijn, "A sub-microwatt asynchronous level-crossing adc for biomedical applications," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 7, no. 2, pp. 149–157, 2013.

[27] M. Neugebauer and K. Kabitzsch, "A new protocol for a low power sensor network," in *IEEE International Conference on Performance, Computing, and Communications, 2004*, pp. 393–399, 2004.

[28] M. Tlili, A. Maalej, M. Ben-Romdhane, M. C. Bali, F. Rivet, D. Dallet, and C. Rebai, "Level-crossing adc modeling for wireless electrocardiogram signal acquisition system," in *2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, pp. 1–5, 2016.

[29] Y. Tsividis, "Event-driven data acquisition and digital signal processing—a tutorial," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 8, pp. 577–581, 2010.

[30] M. Trakimas and S. R. Sonkusale, "An adaptive resolution asynchronous adc architecture for data compression in energy constrained sensing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 5, pp. 921–934, 2011.

[31] R. Agarwal, M. Trakimas, and S. Sonkusale, "Adaptive asynchronous analog to digital conversion for compressed biomedical sensing," in *2009 IEEE Biomedical Circuits and Systems Conference*, pp. 69–72, 2009.

[32] C. Weltin-Wu and Y. Tsividis, "An event-driven, alias-free adc with signal-dependent resolution," in *2012 Symposium on VLSI Circuits (VLSIC)*, pp. 28–29, 2012.

[33] Y. Hou, J. Qu, Z. Tian, M. Atef, K. Yousef, Y. Lian, and G. Wang, "A 61-nw level-crossing adc with adaptive sampling for biomedical applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 1, pp. 56–60, 2019.

[34] M. Ghasemi, N. Ravanshad, and H. Rezaee-Dehsorkh, "An ultra-low power level-crossing adc for ecg monitoring application," in *2020 28th Iranian Conference on Electrical Engineering (ICEE)*, pp. 1–6, 2020.

[35] W. Tang, A. Osman, D. Kim, B. Goldstein, C. Huang, B. Martini, V. A. Pieribone, and E. Culurciello, "Continuous time level crossing sampling adc for bio-potential recording systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 6, pp. 1407–1418, 2013.

[36] Z. Tian, R. Ying, P. Liu, G. Wang, and Y. Lian, "A low power level-crossing adc for wearable wireless ecg sensors," in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 3543–3546, 2016.

[37] T. Marisa, T. Niederhauser, A. Haeberlin, R. A. Wildhaber, R. Vogel, J. Goette, and M. Jacomet, "Pseudo asynchronous level crossing adc for ecg signal acquisition," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 11, no. 2, pp. 267–278, 2017.

[38] R. R. Harrison, "The design of integrated circuits to observe brain activity," *Proceedings of the IEEE*, vol. 96, no. 7, pp. 1203–1216, 2008.

[39] "Mean squared error: Definition and example." `https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/`. Accessed 8-Jun-2021.

[40] M. Ensafdaran, *High speed successive approximation ADC and its applications*. PhD thesis, 2013. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-05-13.

[41] M. Trakimas, *Integrated Circuits and Systems for Sparse Signal Acquisition based on Asynchronous Sampling and Compressed Sensing*. PhD thesis, 2011. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2021-05-18.

[42] N. Sayiner, H. Sorensen, and T. Viswanathan, "A level-crossing sampling scheme for a/d conversion," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 43, no. 4, pp. 335–339, 1996.

[43] W. Kuang, "Correction and comment on "an adaptive resolution asynchronous adc architecture for data compression in energy constrained sensing applications"," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 4, pp. 1097–1099, 2013.

[44] "Analytical figures of merit." `https://www.sciencedirect.com/topics/chemistry/figure-of-merit`. Accessed 6-Jun-2021.

[45] R. L. Grimaldi, S. Rodriguez, and A. Rusu, "A 10-bit 5khz level-crossing adc," in *2011 20th European Conference on Circuit Theory and Design (ECCTD)*, pp. 564–567, 2011.

[46] S. Sirimasakul and A. Thanachayanont, "A logarithmic level-crossing adc," in *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 576–579, 2017.

[47] M. Trakimas and S. Sonkusale, "A 0.8 v asynchronous adc for energy constrained sensing applications," in *2008 IEEE Custom Integrated Circuits Conference*, pp. 173–176, 2008.

[48] K. Peepra and R. Gurjar, "A linear current starved voltage controlled ring oscillator with wide tuning range using 180nm cmos technology," in *2018 International Conference on Recent Innovations in Electrical, Electronics Communication Engineering (ICRIEECE)*, pp. 2925–2928, 2018.

[49] S. Suman, K. G. Sharma, and P. K. Ghosh, "Analysis and design of current starved ring vco," in *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 3222–3227, 2016.

[50] S. Polineni and A. K. Gupta, "8-bit nano watt level crossing adc for bio-medical application," in *2015 International Conference on Computer, Communication and Control (IC4)*, pp. 1–6, 2015.

[51] "Solved: binary counter in veriloga with programmable stepsize." `https://community.cadence.com/cadence_technology_forums/f/custom-ic-design/34770/solved-binary-counter-in-veriloga-with-programmable-stepsize`. Accessed 10-Mar-2021.