University of Huddersfield

Enhanced Performance of Embedded Sensor Data Acquisition using Non-linear Chaos Based Systems

A thesis submitted to the University of Huddersfield in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Philippa Kay Hazell BEng (Hons)

October 2021

Copyright Statement

Abstract

Recent research into tent map (TM) based analogue to digital converter (ADC) architectures, has demonstrated that practical implementations are able to detect small analogue signal variations over relatively large voltage ranges. However, the non-ideal nature of the fundamental TM function slope gain (μ) affects the absolute accuracy and the digital output precision. Although there has been a successful attempt at compensating for non-ideal μ, the high levels of computational resources required makes realising embedded digital system implementations, within a TM-based ADC, unfeasible. This in turn limits the prospect of real-time operation and thus viable commercial TM-based ADC devices.

This work aimed to further develop TM-based ADC performance, to enable more precise and accurate real-time operations, within a data acquisition (DAQ) system, for an ultrasonic measurement system (UMS) application. To facilitate this, an embedded digital implementation of a real-time processing μ compensation algorithm (μCA) was required to adjust the incorrect digital output signal of a TM-based ADC implementation towards the ideal digital output response for a given analogue input signal. To aid analysis of how non-ideal μ affects the TM-based ADC output accuracy, a mathematical model of a TM-based ADC, emulating an electronic implementation operational performance, was created. A novel μCA was then developed, with further compensation for non-ideal behaviours within the electronic circuit implementations of the TM function. Additionally, a VHDL implementation (for configuring a field programmable gate array (FPGA)) enabled the embedment of a digital system performing real-time μ compensation within a TM-based ADC. This digital system was tested using functional simulation and an electronic 8-bit TM-based ADC implementation.

The mathematical model of a TM-based ADC structure, comprising 7 cascaded TM and comparator stages implemented with a 12-bit commercial off the shelf (COTS) ADC digitising the final TM stage output, demonstrated that the bit accuracy improved from 5.81 bits uncompensated, to 15.68 bits after employing the μCA. This established that the proposed TM-based ADC met the UMS DAQ system specification. With the practical implementation, which was prototyped using discrete components, a bit accuracy improvement from 4.19 bits to 5 bits was observed. Both the functional simulations and practical experiments employing the VHDL/FPGA implementation of the μCA proved the concept of a standalone TM-based ADC (comprising 7 cascaded TM and comparator stages with a comparator digitising the final TM stage output) with embedded, real-time μ compensation was achievable.

Acknowledgements

I would like to thank my supervision team - Dr Peter Mather, Prof. Andrew Longstaff and Dr Simon Fletcher for the advice and support they have given me throughout my research. My thanks also extend to Dr David Upton, Dr Rajlaxmi Basu and Mr Richard Haigh for taking time to discuss their past research with me and for providing general advice on how to approach research during the early stages of this project. I am also appreciative towards the University of Huddersfield for funding this project, which was provided as part of the EPSRC Advanced Metrology Hub.

In addition, I would like to give special thanks to everyone in the SCE Research Admin and the CPT Admin teams for all their help with answering my numerous questions over the past three years; and to Dr David Halsall and Prof. Nigel Schofield for providing me with valuable advice during the research process.

I would also like to thank those in the ECMPG group for all their encouragement and support, especially Olaide Olabode, Nemwel Ariaga, Ali Iqbal, Ebube ezi, Nurudeen Alegeh, Kelechi Okegbe, Vinu Pannackal, Difference Chuku and Wencheng Pan. I would also like to thank Yiheng Hu for being a wonderful friend throughout my research.

Finally, I am very appreciative towards my mother Alison, my father Peter, my partner Toby and my brothers John, James and Matthew for all their support throughout my studies.

For John Hazell (1997 - 2015)

List of Publications

Here is a list of papers published whose contents are associated with the research discussed in this thesis:

[1] P. Hazell, P. Mather, A. Longstaff, and S. Fletcher, "A Non-linear Tent Map-Based ADC Design for Sensitive Condition Monitoring Measurement Systems," presented at the COMADEM 2019, Huddersfield, England, Sept. 3-5, 2019, 2020.

Contribution:

Developed and analysed the mathematical TM-based ADC model with different resolution and tent map gain values. Further assessed the model against non-ideal parameters such as input signal noise.

[2] P. Hazell, P. Mather, A. Longstaff, and S. Fletcher, "Digital System Performance Enhancement of a Tent Map-Based ADC for Monitoring Photovoltaic Systems," *Electronics,* vol. 9, no. 9, Sept. 2020, Art no. 1554, doi: https://doi.org/10.3390/electronics9091554.

Contribution:

Analysed the effectiveness of an algorithm to compensate for non-ideal tent map gain within a TM-based ADC. Also implemented the algorithm in VHDL and assessed, via simulation, to confirm non-ideal tent map gain compensation could be performed within one sample clock cycle of the TM-based ADC.

Table of Contents

List of Tables

List of Figures

Glossary of Terms

Acronyms and Symbols

%Δ	Percentage deviation.
μ	TM gain [3].
μ_-	Falling TM slope gain.
μ_+	Rising TM slope gain.
$\mu_\pm$	μ_+ and μ_-.
$\mu_{\pm ADC}$	μ_+ and μ_- TM slope gains within a TM-based ADC.
$\mu_{\pm algorithm}$	μ_+ and μ_- TM slope gains employed by the μ compensation algorithm.
$\mu_{\pm stage}$	μ_+ and μ_- TM slope gains for each TM stage.
μ_{ADC}	μ of a TM-based ADC.
$\mu_{algorithm}$	μ employed by the algorithm.
μ_c	Employed by the SA DM approximation method. The actual TM gain of the TM circuit.
μCA	μ compensation algorithm.
μCA-1	Fundamental μ compensation algorithm analysed in this research.
μCA-2	Enhanced version of the μ compensation algorithm (μCA) analysed in this research. Enhancements enable the μCA to accommodate non-matching TM stage and slope gain.
μCA-3	Final enhanced version of the μ compensation algorithm (μCA) analysed in this research. Enhancements enable the μCA to accommodate non-matching TM stage and slope gain, as well as the final TM stage output being digitised by a multibit sub-ranging COTS ADC.
μCS	μ compensation system

μ_o	Employed by the SA DM approximation method. The μ employed in the equation to determine the ideal DM values.
μ_{stage}	TM-stage μ.
1-D	One dimensional.
ADC	Analogue to digital converter [4].
b	Equivalent binary code representation of the TM-based ADC digital output.
CBC	Compensated binary code. Produced by the μCA-1, μCA-2 and μCA-3.
COTS	Commercial off the shelf.
D	Digital output of an ADC.
DAC	Digital to analogue converter [4].
DAQ	Data acquisition, the process which samples and converts the analogue signal (representing a physical phenomenon) into digital words [5].
DM	Difference measure. These are values employed by μCA-1, μCA-2 and μCA-3 (for each bit of the TM-based ADC digital output) to compensate for non-ideal μ.
DM_{binary}	DM values in binary code format.
DM_{mod1}	First modification in determining DM values for μCA-2 and μCA-3.
DM_{mod2}	Second modification in determining DM values for μCA-2 and μCA-3.
$DM_{theoretical}$	Theoretical DM values represented as non-integer, decimal numbers.
DNL	Differential non-linearity, which is the maximum deviation of the step width from the ideal value of 1 LSB [6].
DV	Difference value. This value (employed by μCA-1, μCA-2 and μCA-3) provides the overall magnitude and direction of the cumulative difference between the ideal TM-based ADC output and the actual TM based ADC output due to non-ideal μ output.
$DV_{polarity}$	Polarity of DV.

ECMPG	Engineering Control and Machine Performance Research Group. A research group based at the University of Huddersfield.
ENOB	Effective number of bits. This represents the number of bits that an ADC can accurately represent analogue input signals as digital words [4, 7].
FFT	Fast Fourier transform [8].
f_{in}	Input frequency.
FPGA	Field programmable gate array. This is a device consisting of a two-dimensional array of logic cells which can be configured to produce highly complex digital electronic circuits [9].
f_{sample}	Sampling frequency.
GCO	Gray code output from the TM-based ADC which is employed by the µCA-1, µCA-2 and µCA-3 to establish the DV to be applied to the digital output due to non-ideal µ.
g_n	Polarity of the TM digital output on the nth iteration (or stage if considering a series TM-based ADC configuration).
HDL	Hardware descriptive language [10].
IC	Integrated circuit [11].
INL	Integral non-linearity. This is the greatest divergence from either the line of best fit through the digital output versus analogue input plot (best straight-line INL), or the line through the two end-points of this plot (end-point INL). The latter of these two INL measurements provides the worst-case scenario, as the method always provides the greater deviation from the line [6].
LSB	Least significant bit [12]
M	A value employed to represents the number of signal cycles, in order to calculate the input frequency of a sinusoidal signal to supply an ADC when performing dynamic testing. The value should be an odd integer number in order to minimise spectral leakage [7].
MSB	Most significant bit [12].

N	A value employed, to represents the number of data points employed by the FFT, in order to calculate the input frequency of a sinusoidal signal to supply an ADC when performing dynamic testing [7].
n	Iteration (or TM stage) number of a TM-based ADC. When n = 0, this represents the initial input value [13].
PCB	Printed circuit board [11]
R	ADC resolution.
r	Resolution of the binary DM values.
SA	Scalar approximation method. Method developed to approximate the DM values employed by the µCA-1, in order to reduce resource requirements if DM values required calculating within the FPGA.
SAR ADC	Successive approximation register ADC.
SDM	Sign of difference measure. This provides the direction of the difference, for each bit of the TM-based ADC, between the ideal output and that due to the non-ideal µ for each TM stage.
SFDR	Spurious free dynamic ratio, which is the difference in magnitude between the signal (the fundamental peak) and the harmonic with the highest magnitude [14].
SINAD or SNDR	Signal to noise and distortion ratio, which is the same as SNR except the signal power is compared to the magnitudes of the noise and harmonics [4, 15].
SL&EA	Straight-line and error approximation method. Method developed to approximate the DM values employed by the µCA-1, in order to reduce resource requirements if DM values required calculating within the FPGA.
SNR	Signal to noise ratio, which is the ratio between the signal power and the average noise power (excluding the power within the signal harmonics) [4].
THD	Total harmonic distortion, which is the magnitude of the harmonics within a signal summed together [14].
TM	Tent map, which is a chaotic non-linear, folding function [16].

TM-ARCHα-n	The name given to a TM-based ADC structure. The design consists of n TM stages and employs n+1 comparators.
TM-ARCHβ-n-$R_{sub\text{-}ranging}$	The name given to a TM-based ADC structure. The design consists of n TM stages, and employs n comparators, as well as a sub-ranging ADC on the final TM stage output (the resolution of this ADC is represented by $R_{sub\text{-}ranging}$).
UBC	Uncompensated binary code. This is the binary code equivalent of the GCO employed by the µCA-1, µCA-2 and µCA-3. The determined DV is applied to this binary code to compensate for non-ideal µ within the TM-based ADC.
UMS	Ultrasonic measurement system.
VHDL	Very high-speed integrated circuits Hardware Description Language, which is a language that enables digital electronic systems to be described [17].
Vref	Partition point voltage.
x_0	Input signal to a chaotic map or system, such as a TM-based ADC [18].
x_n	Output signal from the nth iteration (or TM stage) of a TM-based ADC.
$\Delta\mu_+$	Deviation from positive TM slope gain.
λ	Lyapunov exponent, which is a measure of sensitivity dependence a chaotic map might have on the initial conditions [19].

Terminology

Accuracy	The degree of proximity to the actual value [20].
Aliasing	The effect of a higher frequency ADC input signal appearing as a lower frequency signal due to under-sampling [21].
Aperiodic	A system which does not produce a periodically repeating output [22].
ADC architecture	An overview description of how a certain class of ADCs with similar operation function [4].
Bandwidth	The range of input frequencies an ADC can accept [23].
Bernoulli map	A type of chaotic map [3].
Bifurcation diagram	Final state plot of a chaotic map over a range of control parameters (e.g., μ for a TM) [22].
Binary DM values	DM values in binary code format.
Bit accuracy	A measure of the minimum number of bits for which an ADC can accurately represent an analogue input as a digital word. Determined using the following equation. $$bit\ accuracy = R - log_2\lceil difference_{max}\rceil - 1$$ Where $difference_{max}$ is the maximum absolute quantisation error when the ADC is supplied a ramp signal whose amplitude extends across the entire valid input range. R is the ADC resolution.
Bounded	When the maximum difference between two points within the output of the system is less than infinity [22].
Chaotic	Describes a system which is deterministic and follows simple rules, but the behaviour is non-linear and complex [19].
Chaotic behaviour	A system which is deterministic and follows simple rules, but the behaviour is non-linear and complex [19].
Chaotic flow	A mathematical function representing a continuous chaotic system [19].
Chaotic map	A mathematical function representing a discrete chaotic system [19].
Chaotic system	A system which exhibits chaotic behaviour [19].

Comparator	An electronic component which compares the amplitudes of two analogue signals and outputs a digital signal whose level is dependent on which input has the lower amplitude [23].
Comparator hysteresis	A trait where two triggering levels enable the switching of the comparator output to be delayed [23].
Continuous system	Refers to a system which can be defined for all of time during a certain period [24].
Conversion speed	The time taken to convert a data sample from one format to another [25].
Data conversion	The process of converting data from one format to another [4].
Data converter	A device which converts data from one format to another [4].
Decimation	The process of deleting samples [26].
Deterministic	Produces the same output for a given input [22].
Difference equations	Mathematical method of describing the behaviour of a discrete, recursive system [19].
Discrete system	Refers to a system which can only be defined at set intervals during a certain period [24].
Dual-slope ADC	A type of integrating ADC architecture [27].
Dynamic performance	The behaviour of the ADC being supplied considerably varying input signals [4].
End-point INL	See INL.
Feedback configuration	Type of TM-based ADC configuration consisting of a single TM, whose output signal is fed back and is supplied as the next input signal.
Folding	Process of bending the signal over itself.
Full-scale error	The difference between the maximum digital output of the actual and ideal ADC [28].
Gain error	The difference between the full-scale error and the offset error [28].
Gain factor	The inverse product of the current and preceding $\mu_{\pm stage}$ values employed to establish a given digital output produced by the TM-based ADC.

Gray code	A way of representing digital data where adjacent values vary by only one bit [9].
Gray-code algorithmic ADCs	A type of ADC which employs folding and amplification circuits as part of the data conversion process, and outputs the digital data in Gray code format [29].
Integrating ADC	A type of ADC that employs integrator circuits as part of the analogue to digital conversion process [27].
Latency	Time taken to complete the data conversion process [4].
MATLAB	A software platform, with a dedicated programming language, used for numerical computing and mathematical model development [30].
Missing codes	Codes which an ADC does not produce, out of all possible digital combinations, when the input analogue signal is swept across the valid input range [28].
ModelSim	A software programme for simulating HDL designs [31].
Monotonic	An ADC is monotonic when the DNL is within the threshold of ± 1 LSB.
Multi-slope ADC	A type of integrating ADC architecture [27].
Nyquist frequency	Half of the sampling frequency of an ADC [32].
Offset error	Corresponds to the minimum input required to provide a zero output code [15], or where the transfer characteristic end-point plot of the ADC intercepts the axis representing the digital output [4].
Operational current conveyors	A device which transfers current from one impedance level to another [33].
Oversampling	When a signal is being sampled at a rate larger than double the ADC bandwidth [8].
Partition point	The minimum input signal amplitude which causes the TM to transition from one difference equation to the other [34].
Pipelined ADC	A type of ADC architecture [35].
Precision	The degree of repeatability and reproducibility of a reading [20].
Quantisation error	Also referred to as quantisation noise. The difference in LSBs between the equivalent ADC output voltage versus the input voltage [4, 36].

Resolution	The number of bits which are employed in the digital representation of the analogue sample [8].
Sampling speed	The rate at which a signal can be sampled [25].
Series configuration	Type of TM-based ADC configuration consisting of multiple TM circuits, which are connected in series.
Sigma delta ADC	A type of ADC architecture [37].
Spectral leakage	When discontinuities at the ends of the sinusoidal signal, which an FFT is being performed on, causes the peak in the resulting spectrum to spread into adjacent frequency bins and affect the spectral distribution [8].
Static performance	The behaviour of the ADC when the amplitude of the input signal is slowly changing.
Step size	The minimum change in the analogue input signal which an ADC can detect and is also the equivalent of the LSB in the digital output [38].
Sub-ranging ADC	A general purpose ADC used within a data conversion process [15, 39].
Theoretical DM values	Theoretical DM values represented as non-integer, decimal numbers.
Time-interleaved ADC	A type of ADC architecture which switches between parallel sub-ranging ADCs in order to increase the sampling speed [39].
Tolerances	The range which the value or dimension of a component must lie [11].
Transconductance amplifier	An amplifier which takes an input voltage signal and outputs a current signal [23].
Ultrasonic	Concerns frequencies which are in the acoustic bandwidth above the audible limit [40].

1 Introduction

Highly precise and high accurate signal measurement systems are essential in order to repeatably detect small signal variations across an input signal range with a high degree of confidence. Furthermore, signal measurements are generally within the analogue domain, so need converting to the digital domain to enable efficient storage, analysis or digital post processing. This requires analogue to digital converters (ADCs) with sufficient resolution to acquire and convert the minimum variation needing to be detected.

This work considers an alternative ADC architecture for employment within measurement systems that need to detect small signal variations. This architecture employs folding (bending a signal over itself) and amplification circuits, based on the tent map (TM) function, as part of the analogue to digital conversion process. Previous research has proven that TM-based ADCs are a viable option for measurement systems requiring small signal variations to be detected across the entire valid range of signal amplitude. However, the inherent, non-ideal amplification gain effects of a practical device reduce the accuracy of the digital output codes produced from such an ADC. A solution, proposed to compensate for these errors, requires unfeasibly high levels of computational resources to enable a compensation system to be embedded within the device [41, 42].

This work details an enhanced solution to this problem, through the development of a TM-based ADC, with an embedded compensation system which employs a novel compensation algorithm. This algorithm requires fewer computation resources than previous approaches and enables signal sample compensation during each TM-based ADC conversion cycle, thus allowing real-time operation. By developing the ADC for a specific type of measurement system application, this research also assesses the viability of employing this TM-based ADC,

with an embedded compensation system, within other high precision and high accuracy measurement systems. Further enhancements incorporated into the compensation algorithm will enable the development of non-ideal amplification gain compensation systems for different TM-based ADCs configurations. This will further advance the potential employment of such devices in a wide range of applications, where consistently detecting small signal variations, across the whole valid signal range, is required.

1.1 Background

Many signal measurement systems needing to produce repeatable measurements with a low degree of error require high precision (high degree of repeatability and reproducibility) and high accuracy (high degree of proximity to the actual value) [20] signal measurement. This requires the measurement systems to be capable of reliably detecting small variations within the signal being measured.

If these signal measurements of an analogue signal also require storing, analysis or digital post-processing, then the signal measurements need converting to the digital domain. When the measured physical phenomenon needs displaying or transmitting as digital values, measurement systems require data acquisition (DAQ). DAQ is the process which samples and converts the analogue signal (representing a physical phenomenon) into digital words [5]. The measured value tends to be represented by an analogue output signal from a sensor, therefore the DAQ system employed by the measurement set-up requires an ADC [4] to perform the conversion to the digital domain [5].

1.2 Analogue to Digital Converters

When assessing the performance of an ADC, there are four categories which are considered:

- resolution, which refers to the number of bits which are employed in the digital representation of the analogue sample [8];
- speed, which concerns both the sampling speed (the rate at which a signal can be sampled) [25] and the conversion speed (the time taken to convert a sample to the digital domain) [25];
- power consumption; and
- the silicon die area required to fabricate the ADC if an integrated circuit (IC) [11] is being produced [43].

Ideally the performance of an ADC should simultaneously achieve:

- high resolution to enable small signal variations to be represented in digital format;
- high sampling speed to allow high frequency input signals to be acquired and converted;
- high conversion speeds, so digital data can be obtained and employed faster; low power consumption in order to reduce energy requirements, as well as methods of heat dissipation, which minimises costs; and
- low fabrication area, which will keep fabrication costs down [43].

Simultaneously meeting all these requirements is challenging, especially when approaching the current extremes of any one requirement; being at such a point often has a negative effect on other characteristics [43] and can increase cost.

For measurement systems to be able to detect smaller signal variations over a larger signal range, the ADC employed needs to have a sufficient resolution to distinguish those variations. The ADC should also have a linear response when detecting those variations across the whole signal range. The minimum change in the analogue input signal which an ADC can detect, and is also the equivalent of the least significant bit (LSB) in the digital output, is known as a step size [38]. (1-1) details how the step size can be determined from the ADC resolution (R) and the full-scale input signal range which an ADC can accept [15]. An increase in the valid input signal range requires a higher resolution ADC in order to maintain the same step size.

$$LSB \ = step\ size = \frac{full\ scale\ input\ range}{2^R} \tag{1-1}$$

Simultaneously achieving high resolutions and one of high speeds, low power, low cost or low area of fabrication is challenging and results in trade-offs in the other categories. For example, ADCs which simultaneously achieve high resolutions and high sampling speeds are more expensive than those with lower sampling speeds, as illustrated in Figure 1-1 which compares the normalised cost of ADCs of different resolution with the fastest and slowest sampling speeds [44-55]. Also the majority of higher resolution ADCs currently on the market employ architectures (a description of the circuit operation) [17] which hinder the conversion speed, as the data conversion process establishes each bit (or a small group of bits) in order from the most significant bit (MSB) to the LSB, rather than concurrently. This in turn impacts the ADC latency (time taken to complete the data conversion process) [4], and often the sampling speed if the architecture requires one sample to be fully converted to the digital domain prior to the next sample being acquired.

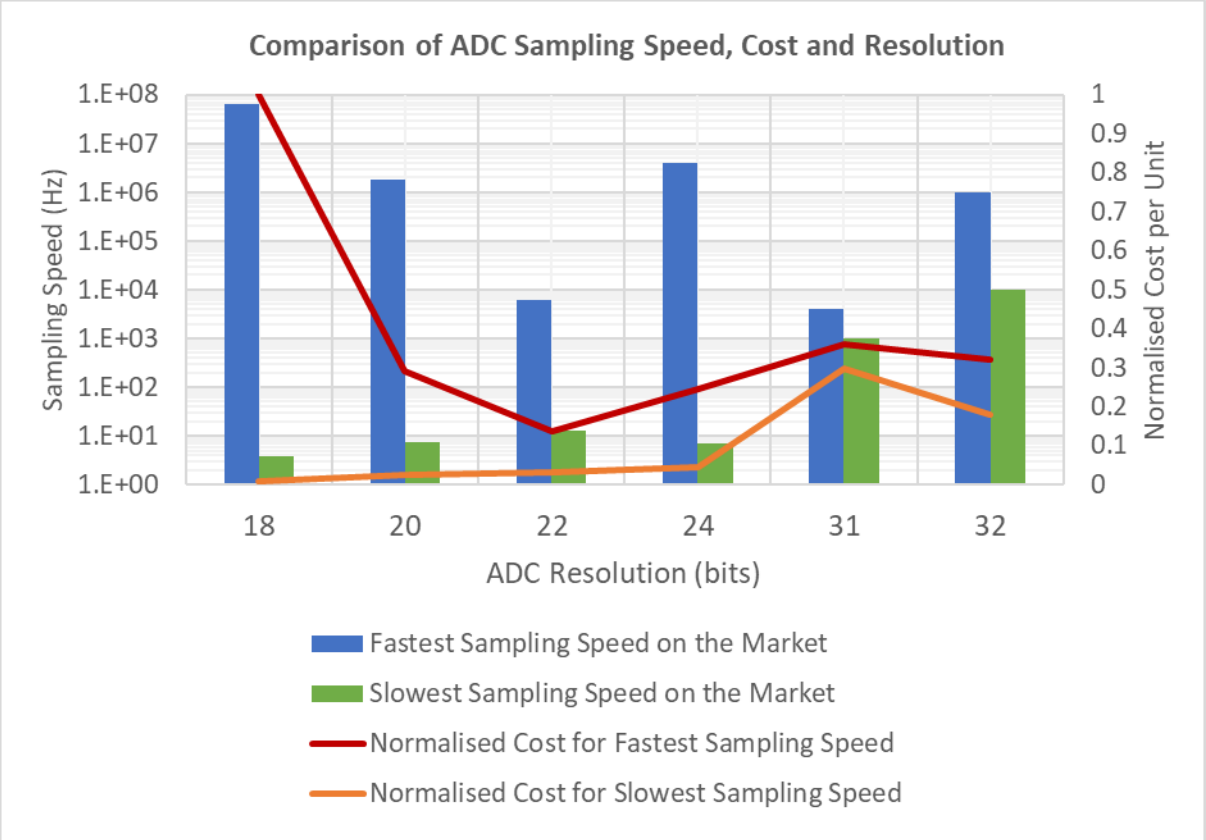


Figure 1-1: Comparing sampling speed, cost and resolution of ADCs available on the market [44-55].

1.3 Overview of Tent Map Based ADCs

Alternative ADC architectures, other than those found in commercial off the shelf (COTS) ADCs, which employ folding functions called tent maps (TM) within the data conversion (in this scenario, the analogue signal samples are converted to the digital domain) [4] process have been explored. The TMs within these type of ADC architectures fold (in effect bend the signal over itself) and amplify the sampled analogue signal. The digital representation is then determined by feeding the folded signals through comparators[1] or sub-ranging ADCs[2] [13, 16, 56, 57].

[1] Comparators are electronic components which compares the amplitudes of two signals and outputs a digital signal whose level is dependent on which input has a lower amplitude [23].
[2] Sub-ranging ADCs are general purpose ADCs used within the analogue to digital data conversion process [39].

A TM is a discrete, one-dimensional (1-D) mathematical model representing one type of chaotic system [58]. A chaotic system is deterministic and follows simple rules, but the behaviour is non-linear and complex [19]. A TM is summarised by the difference equations (a mathematical method of describing the behaviour of a discrete, recursive system) shown in (1-2) [19].

$$x_{n+1} = \begin{cases} \mu x_n \\ \mu(1 - x_n) \end{cases} \quad \text{when} \quad \begin{matrix} x_n \leq 0.5 \\ x_n > 0.5 \end{matrix} \tag{1-2}$$

x_n and x_{n+1} represent the input and output of the TM respectively (the original input, x_0, is referred to as an initial condition), whilst n signifies the number of iterations. μ is the TM gain and the TM will not exhibit chaotic behaviour if $\mu > 2$ or ≤ 1 (as will be explained in more detail in Section 2.3 [3]). The value of x_n where the TM transitions between the two difference equations is known as the partition point [34]. When the TM exhibits chaotic ($1 < \mu \leq 2$) behaviour both x_n and x_{n+1} are bounded between 0 and 1 [41, 42]. If $\mu < 1$, the TM output goes towards zero with each iteration, whilst when $\mu > 2$ the TM output becomes unbounded and goes towards -∞.

Analysis of research, conducted over the past 8 years [13, 16, 56, 57], suggests that TM-based ADCs are well-suited for detecting small variations within signals with a wide range of input signal amplitudes [16]. This trait, which is desired in signal measurement systems, is due to the TMs effectively zooming into the analogue sample during the data conversion process [16]. Research also suggests that different configurations of the TM-based ADC architecture can be employed reduce different combinations of trade-off costs [13, 16, 56, 57], which increases the potential of the architecture being employed to a wider range of applications.

Berberkic developed two types of TM-based ADCs which measured and converted the difference in magnitude between two successive samples to the digital domain (rather than representing each analogue sample as an absolute digital value). One of these ADCs had the TMs arranged in a series configuration and employed an inexpensive COTS 10-bit sub-ranging ADC to acquire the TM output signals and determine the digital output. The prototype of this configuration was capable of detecting 5 μV changes in successive samples (although the error was > 10% with variations < 50 μ V), over a relatively large voltage range of 0 – 10 V (the equivalent of 20 bits resolution) [16], which highlighted the ability to achieve sufficient resolutions to detect small signal variations. Configuring the TMs in series also enabled higher sampling speeds, which were less restricted by the conversion speed, as the next analogue sample was acquired after the preceding sample had been processed by the first TM stage, and the TM output signal acquired by the corresponding sub-ranging ADC [16]. This TM-based ADC was also inexpensive to prototype, when compared to ADCs fabricated onto silicon, as the integral TMs blocks were constructed from discreet, COTS components [16].

The other TM-based ADC configuration proposed by Berberkic also achieved sufficient resolution to detect small signal variations, although not to the same level as the series configuration [16]. This configuration employed a single TM circuit that fed the output signal back to the input and used a sub-ranging ADC to convert the folded signal to the digital domain in order to determine the difference between successive samples. The prototype of this configuration detected 50 μV changes in successive samples (although the error was > 10% with variations < 200 μ V), over a range of 0 – 10 V, thus achieving a resolution of 17 bits [16]. The small reduction in resolution is reimbursed in terms of smaller circuit area (if this TM-based ADC were to be fabricated onto silicon) and hence cost, because the feedback configuration required less circuitry than the series configuration.

Upton developed a TM-based ADC, which employed comparators in place of sub-ranging ADCs, producing a simpler, but lower resolution, design. Unlike Berberkic [16], the TM-based ADC determined the absolute digital representation of each sample through the addition of a single comparator at the input of the first TM stage. This design reduced power consumption by implementing a trigger circuit which only enabled the clocking signals to the ADC when a signal was present. The design also enabled the next sample to be acquired after the MSB of the digital representation had been determined, enabling higher sampling speeds which were less restricted by the conversion speed. This meant that several additional samples were acquired (and conversion started) during the time taken to convert the first sample to the digital domain. The ADC devised by Upton was also inexpensive to prototype, being constructed out of discreet components [56, 57].

Finally, Liu et al. developed a TM-based ADC which employed a single TM circuit in a feedback configuration [13]. Liu et al. employed a comparator to determine each bit of the digital representation, before the signal went through the TM, in order to produce the digital representation of the individual samples. The sampling rate was restricted by the conversion speed, as the analogue sample had to be completely digitised before the next sample was acquired. However, this variation of the TM-based ADC required less circuitry as only one TM stage and comparator were required, reducing the silicon area required for fabrication. This enabled multiple versions of this ADC variation to be employed on an IC to allow parallel sampling of a tactile sensor outputs without becoming too costly in terms of the circuitry and the fabrication area required [13].

In order to establish the benefits and trade-offs of the different TM-based ADC structures observed in literature, a comparison through ranking was undertaken as part this research. Table 1-1 compares the resolution, sampling speed, conversion speed and potential silicon area the four TM-based ADCs discussed above would require, if fabricated in silicon. How each ADC was ranked was based on the information provided in the literature, however some sources did not declare the sampling speed, potential fabrication area, conversion speed and power consumption information. With the former three points, the relative performance in these two categories was estimated from the TM-based ADC circuitry. None of the literature provided enough information to determine the power consumption of most of the TM-based ADCs to provide a meaningful comparison, so this category was omitted in the comparison.

	Ranking TM-based ADCs based on Relative Performance			
	Resolution (1 = lowest 5 = highest)	***Sampling Speed (1 = slowest 5 = fastest)***	***Conversion Speed*[3] *(1 = slowest 5 = fastest)***	***Fabrication Area*[3] *(1 = largest 5 = smallest)***
Berberkic's Series Configuration	5	2	2	1
Berberkic's Feedback Configuration	4	1	1	4
Upton's Series Configuration	1	5	5	3
Liu et al's feedback configuration	1	3[4]	3	5
TM-ARCHβ-7-12 ADC presented in this work	3	5	5	2

Table 1-1: Comparison of the TM-based ADCs observed in literature. Aided by [13, 16, 56, 57].

[3] This information was not provided in the literature [13, 16, 56, 57] and was estimated from the TM-based ADC structure.

[4] This information was not provided in the literature [13] and was estimated from the TM-based ADC structure.

1.4 Tent Map Based ADC Output Accuracy and Tent Map Gain

All the TM-based ADCs found in literature, except for the examples developed by Berberkic [16], convert an analogue sample to the digital domain using the same method [13, 56, 57]. The input signal (x_0) is compared with the partition point voltage using a comparator to determine the MSB. Then the amplitude of the output signals after each TM stage are also compared to the partition point voltage to determine the remaining bits. (1-3) demonstrates this process [13, 42, 56, 57].

$$D(n) = \begin{cases} 0, & x_n \leq 0.5 \\ 1, & x_n > 0.5 \end{cases} \tag{1-3}$$

Where n represents the TM iteration for a feedback configuration (or TM stage output for a series configuration) and D(n) represents the equivalent bit produced for x_n (the TM iteration output voltage, or the original input signal if n equals 0). The format of the digital data produced by the TM-based ADC is in Gray code representation [9].

TM-based ADCs found in literature rely on amplifying the folded signals back to full-scale, thus the μ must be exactly two. A μ less than 2 affects the output accuracy of the TM-based ADC, and this can be demonstrated using a theoretical MATLAB model of a series configuration of a TM-based ADC developed for this research (developed script given in B.1.1). Figure 1-2 and Figure 1-3 presents the input signal (x_0) and the output of three TM stages, when μ = 2 and 1.9 respectively, when x_0 is set at different amplitudes.

These plots highlight how a difference in the value of μ results in the TM producing a different output for a given x_0 with each TM stage. This includes the amplitude of the maximum and minimum points produced by the TM outputs, when supplied a ramp input signal. Where the

TM outputs cross the partition point voltage differs depending on the μ employed, as highlighted in Figure 1-4 which is a plot of the third TM stage outputs when μ = 2 and μ = 1.9, along with the partition point voltage.

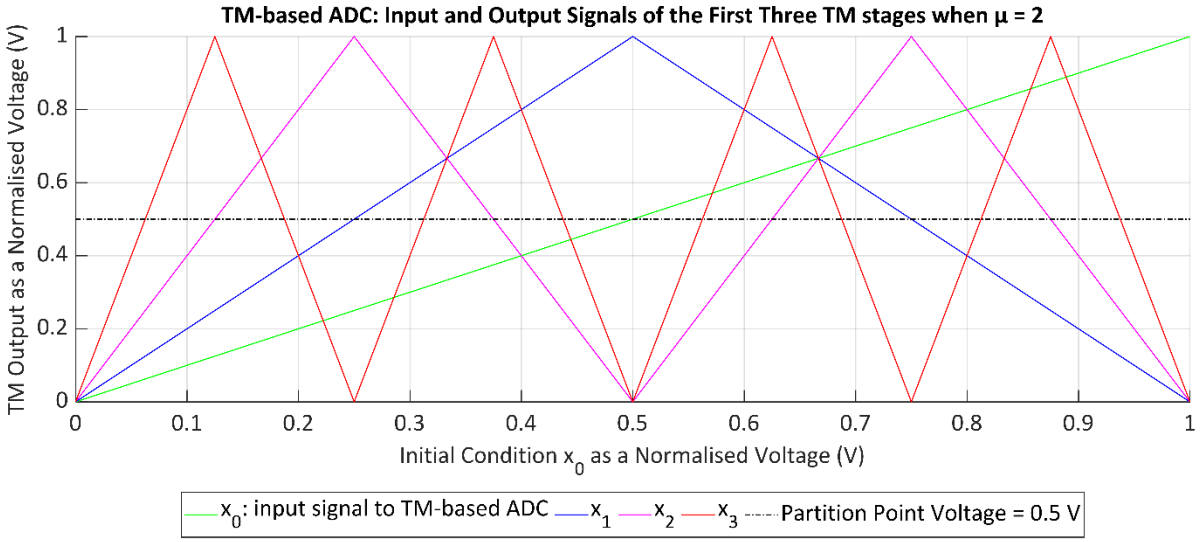


Figure 1-2: Plot of the input and output signals of 3 TM stages within a theoretical 4-bit TM-based ADC, when μ = 2.

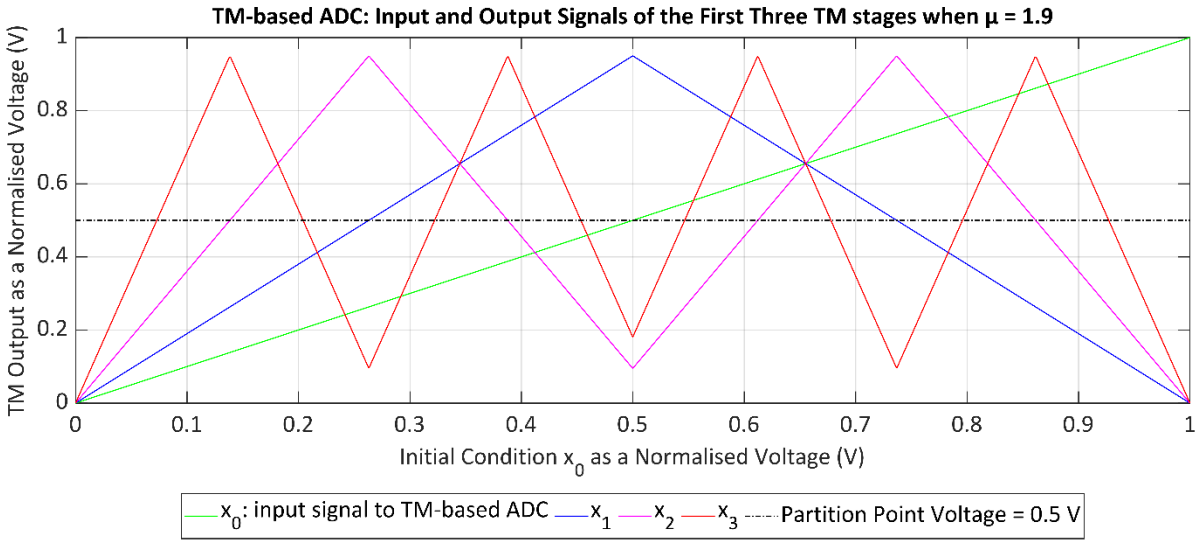


Figure 1-3: Plot of the input and output signals of 3 TM stages within a theoretical 4-bit TM-based ADC, when μ = 1.9.

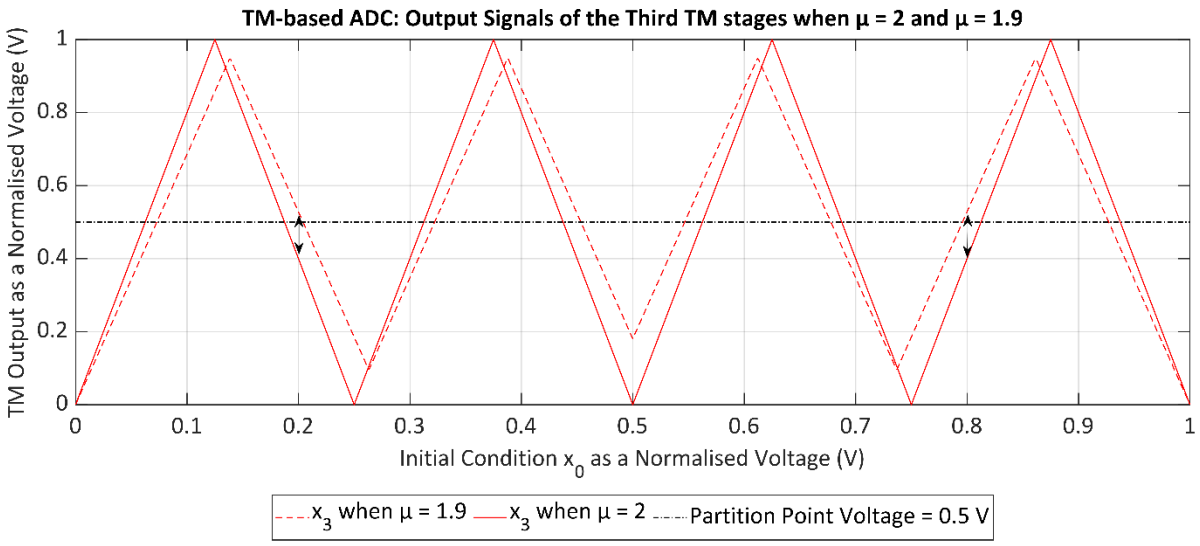


Figure 1-4: Plot comparing output signals of the third TM stage when μ = 2 and μ = 1.9.

Table 1-2 summarises the equivalent Gray code output produced by the theoretical TM based ADC (which has a resolution of 4 bits) for μ = 1.9 and 2, when $0 \leq x_0 \leq 1$ V (in increments of 0.1 V). The rows highlighted in red show when the digital representation differs for the two μ values used, caused by the value of μ changing where the TM outputs cross the partition point voltage. These points are also highlighted using double ended arrows on the plot presented in Figure 1-4.

	Two values of μ demonstrating the impact on the output accuracy of a 4-bit TM-based ADC							
	μ = 2				μ = 1.9			
x_0 (V)	D(0)	D(1)	D(2)	D(3)	D(0)	D(1)	D(2)	D(3)
0	0	0	0	0	0	0	0	0
0.1	0	0	0	1	0	0	0	1
0.2	0	0	1	1	0	0	1	0
0.3	0	1	1	0	0	1	1	0
0.4	0	1	0	1	0	1	0	1
0.5	0	1	0	0	0	1	0	0
0.6	1	1	0	1	1	1	0	1
0.7	1	1	1	0	1	1	1	0
0.8	1	0	1	0	1	0	1	1
0.9	1	0	0	1	1	0	0	1
1	1	0	0	0	1	0	0	0

Table 1-2: Gray code output of a 4-bit TM-based ADC when μ = 2 and 1.9.

Certain output codes given for set values of x_0 are different when $\mu \neq 2$, thus a deviation from the ideal μ significantly affects the accuracy of the TM-based ADCs [1]. Figure 1-5 presents a plot highlighting how reducing the μ employed in a theoretical 16-bit TM-based ADC results in the equivalent binary output of the ADC producing a less accurate representation of the original input signal. This is due to the generation of incorrect digital output codes and missing codes (an ADC with no missing codes presents all possible digital combinations when the input analogue signal is swept across the valid input range [28]).

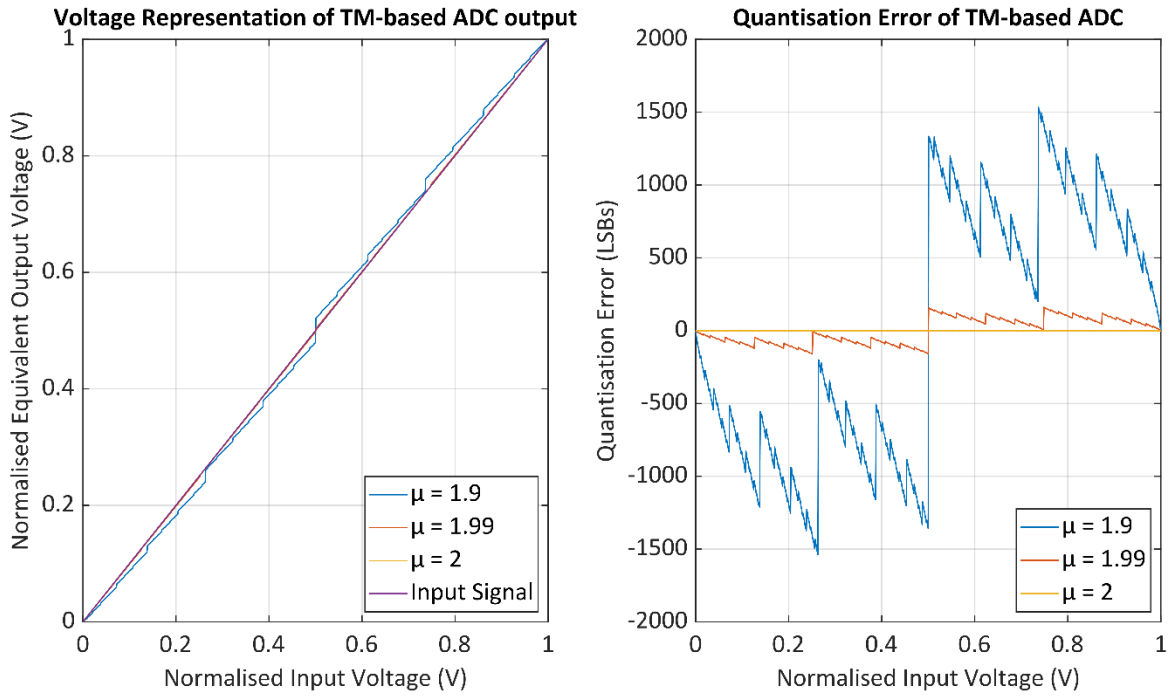


Figure 1-5: Graphs showing the effects non-ideal μ has on the accuracy of a 16-bit TM-based ADC.

The left-hand plot in Figure 1-5 presents the equivalent normalised voltage of a 16-bit TM-based ADC output versus the input voltage with different μ values. The voltage equivalent of the ADC output was determined from the sum of weightings of the binary code equivalent to the original Gray code. (1-4) summarises the conversion process from Gray to binary code, whilst (1-5) details how the sum of weightings is determined from the binary code [59, 60].

$$b(n) = \begin{cases} D(0), & n = 0 \\ D(n-1) \oplus D(n), & n \geq 1 \end{cases} \tag{1-4}$$

$$Equivalent\ normalised\ voltage = \sum_{n=0}^{N} \frac{b(n)}{2^n} \tag{1-5}$$

Where:

- b(n) refers to the bit of the binary code representing x_n,

- D(n) is the respective bit of the Gray code representation,
- n refers to the TM stage (or initial condition if n = 0) and
- N is the total number of TM stages in the TM-based ADC.

The μ values examined in Figure 1-5 were 2 (the ideal value), 1.99 and 1.9 (0.5 % and 5 % less than the ideal value respectively). The TM-based ADC output when μ = 2 produced an accurate reproduction of the original input signal, whilst the reconstructed output signal when μ = 1.9 is a noticeably less accurate representation of the ramp input signal. The right-hand plot in Figure 1-5 presents the quantisation error (also referred to as quantisation noise), which is the difference, in LSBs, between the equivalent ADC output voltage and the input voltage and is calculated using (1-6) [4, 36].

$$Quantisation\ error = \frac{equivalent\ output\ voltage - input\ voltage}{step\ size} \qquad (1\text{-}6)$$

The maximum positive and negative quantisation error values for each μ are summarised in Table 1-3, along with the bit accuracy of the TM-based ADC output. The ADC bit accuracy is a measure of the minimum number of bits for which an ADC can accurately represent an analogue input as a digital word and defined in this research using (1-7), from the absolute quantisation error.

TM Gain	Quantisation error (LSBs)	Bit accuracy (bits)
2	±1	15.00
1.99	±162	7.66
1.9	±1543	4.41

Table 1-3 Reduction in the TM-based ADC accuracy due to μ.

$$bit\ accuracy = R - log_2\lceil difference_{max}\rceil - 1 \quad (1\text{-}7)$$

In (1-7), $difference_{max}$ is the maximum absolute difference (in LSBs) between the input signal and the voltage representation of the digital output (i.e., the maximum absolute quantisation error). An additional bit is subtracted from the ADC resolution (R) as $\log_2|difference_{max}|$ equates to zero if $difference_{max}$ = 1 bit, but the bit accuracy will be one bit less than the stated resolution of the ADC, due to quantisation error [4].

The results highlight how a small reduction of 0.5 % in μ (1.99) results in a deviation from the ideal, equating to the loss of 8 bits in accuracy (although this loss in accuracy was less visibly noticeable in the reconstructed output signal of the TM-based ADC). This loss of accuracy increases the further μ deviates from the ideal value.

The μ of the electronic TM implementations employed within the TM-based ADCs seen in [13, 16, 56, 57] were set by resistors. Resistors have tolerances (the range which the value or dimension of a component must lie [11]) which makes practically achieving precise and accurate μ values more challenging than when silicon matched parts are used. Resistors with tight tolerance bands can be employed to achieve good μ precision, but this generally leads to an increase in cost. Also the stability of the μ value will remain affected by uncontrollable factors such as resistance varying over time, due to the resistors aging, as well as fluctuations in operating temperature [61]. An alternative solution to the use of precision components is to evaluate the effects a non-ideal μ has on the performance of a TM-based ADC output. The deviation from the ideal digital codes can then be compensated for, by processing the non-ideal ADC output, to estimate the initial conditions (the input signal(s) to the system) [22], thus enabling the correction of incorrect codes and improving the ADC accuracy.

TMs are non-linear and non-invertible (each output value could have been formed from two input values rather than one) as illustrated in Figure 1-2 and Figure 1-3, which makes estimating the initial conditions of these chaotic maps challenging. Basu successfully estimated the initial conditions of TM-based ADCs [41, 42], and can be considered to be compensating for non-ideal μ in the process. However the algorithm required significant computational resources, as division was employed to estimate the initial condition (division being computationally resource intensive) [62]. Moreover the algorithm developed was not implemented as an electronic system, but was proven using off-line batch processing of data acquired from a TM-based ADC [41, 42].

This thesis details a new algorithm requiring less computation resources, to estimate the initial conditions (and thus compensate for non-ideal μ within a TM-based ADC), compared to the one presented by Basu [41, 42]. The viability of implementing the algorithm, as an embedded electronic system, that compensates the output of a TM-based ADC in real-time (the ADC output for a given sample is compensated for, whilst the digital output for the subsequent sample is being converted), was also investigated. The feasibility of applying this real-time TM-based ADC, with embedded μ compensation system, to a specific real-world application (an ultrasonic measurement system), was also considered.

Figure 1-6 presents a block diagram of the TM-based ADC and μ compensation algorithm (μCA) implemented as an electronic solution. An adaption of the TM-based ADC design by Upton [56, 57], (consisting of a printed circuit board (PCB) [11]) was connected to a field programmable gate array (FPGA, which is a device comprising a two-dimensional array of logic cells which can be configured to produce highly complex digital electronic circuits) [9]. The PCB comprised the analogue circuitry for performing the data conversion, while the FPGA

both coordinated this process via clock signals, as well as aligning and converting the Gray code data from the PCB to binary code [56, 57]. In this research the µCA was embedded in the FPGA in order to produce a standalone TM-based ADC and µ compensation system (µCS).

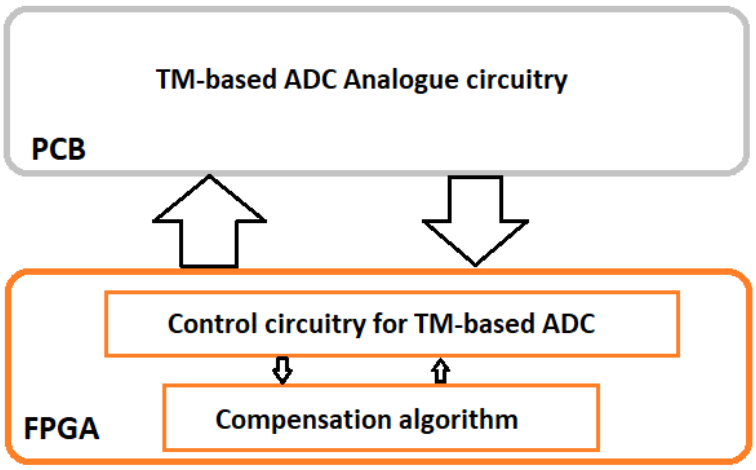


Figure 1-6: Block diagram of how the TM-based ADC and µCA were integrated.

A later adaption of the TM-based ADC developed was also based on the design proposed by Upton [56, 57] and, employed techniques used by Berberkic [16]. This ADC was analysed by simulation. Table 1-1 compares the design to other TM-based ADCs found in literature over the past 8 years. This TM-based ADC achieves higher resolution than the designs proposed by Upton [56, 57] and Liu et al. [13], as well as matching the high sampling and conversion speeds (when compared to the other TM-based ADCs found in literature) achieved by Upton [56, 57]. An enhanced version of the µCA was also developed for the adapted TM-based ADC design.

1.5 Ultrasonic Measurement System

Researchers, within the Engineering Control and Machine Performance Research Group (ECMPG), at the University of Huddersfield are developing a highly sensitive Ultrasonic Measurement System (to be referred to as UMS) for detecting temperature variations in metal undergoing precision manufacturing cutting processes. This typically needs a dimensional error less than 5 µm [63]. The temperature of metal undergoing cutting varies and causes expansion, which in turn introduces errors. Such temperature variations can increase by 10 °C, which for a 200 mm part of tungsten (a high density metal renown for a high melting point [64]) can result in a 9.2 µm expansion [65].

The UMS will enable precise, non-invasive, in-process monitoring of the metal temperature and allow the process to compensate for errors introduced from metal expansion caused by temperature variations. This will produce higher quality work pieces, as the compensation for temperature variation will improve the accuracy of the cutting process. In addition, more work pieces will meet the required specifications, resulting in a higher yield, less material wastage and a fall in rework time [63].

The proposed UMS employs an ultrasonic, piezoelectric transceiver to transmit and receive sinusoidal waves from the piece of metal under observation (Figure 1-7 outlines the set up) [63]. The transmitted and received sinusoidal waves then go through a phase detection board [66] which outputs a voltage signal (Φ1) representing the phase difference. A DAQ board acquires this signal and transmits a digital representation to a computer, where the temperature variations in the metal are established [40, 63].

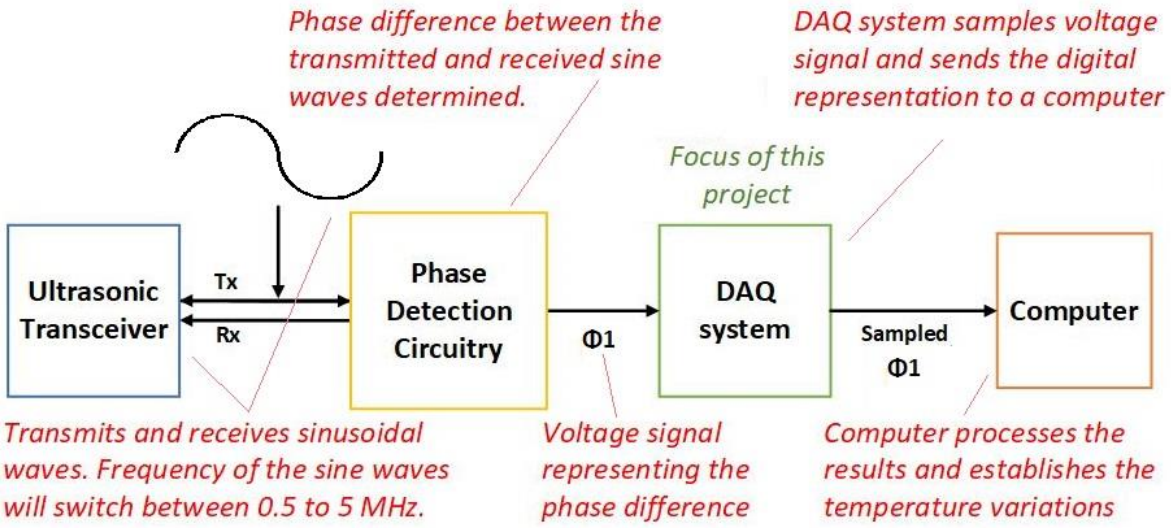


Figure 1-7: Block diagram of the UMS proposed by the ECMPG. Aided by [63].

Ideally, the UMS should establish temperature variations to an accuracy of at least 0.1 °C to enable the manufacturing process to compensate for metal expansion and achieve precision machining [63]. Early research suggested a temperature variation of 0.1 °C equated to a 1 mV change in the output voltage signal from the phase detection circuitry (over a 0 - 1.8 V range) [66], but the preference is for the system to detect 100 µV variations. To detect these variations in the output signal, of the phase detection circuitry, a DAQ board containing an ADC with a minimum resolution of 15 bits will be required. (1-8), which was derived from (1-1) [15], highlights how the minimum resolution was determined.

$$minimum\ resolution = \left\lceil log_2\left(\frac{voltage\ range}{voltage\ variation}\right)\right\rceil = \left\lceil log_2\left(\frac{1.8-0}{100\times10^{-6}}\right)\right\rceil = 15\ bits \quad (1\text{-}8)$$

Where voltage range refers to the input voltage signal range which the ADC can accept, whilst voltage variation indicates the minimum change within the input signal that must be detected

and digitised. For this application, the voltage variation needed to be 100 µV and the voltage range must equal that of the phase detection circuitry output.

The maximum frequency of the transmitted signals will be 5 MHz, thus the sampling frequency of the ADC needed to exceed 10 MHz in order to meet the Nyquist criterion. The Nyquist criterion states the maximum input frequency should be less than half the sampling frequency in order to avoid aliasing, which is the effect of a higher frequency signal appearing as a lower frequency signal due to under-sampling [21, 25].

The research project described in this thesis explores the viability of a stand-alone TM-based ADC with embedded µ compensation (performed in real-time), within a DAQ system, which meets the requirements for the above application. There was an additional requirement that the ADC needed to be constructed from discrete components, in order to be low-cost to develop when compared to an ADC fabricated in silicon.

1.6 Aim and Objectives

This project assesses the viability of a standalone TM-based ADC, with an embedded digital implementation of a compensation algorithm for non-ideal μ, to be employed within a DAQ system for an UMS application. The objectives are to:

- **Develop a mathematical model of a TM-based ADC to emulate the operational performance of an electronic implementation.**

  An adaption of the TM-based ADC design developed by Upton [56, 57] was chosen and a mathematical model developed. This model was developed to determine the effects non-ideal μ had on the ADC output accuracy as well as to aid the assessment of the μ compensation algorithm (μCA).

- **Develop a compensation algorithm for a non-ideal μ, to increase the accuracy of a TM-based ADC, without the requirement of off-line computational processing.**

  The μCA was initially developed in MATLAB (a software platform, with a dedicated programming language, used for numerical computing and mathematical model development) [30] and applied to the data produced by the mathematical TM-based ADC model to analyse the effectiveness in terms of improving accuracy. In order to embed a μ compensation system (μCS), comprising the μCA, within the FPGA used in the electronic implementation of the TM-based ADC, the algorithm was implemented in VHDL (Very high-speed integrated circuits Hardware Description Language, which is a language that enables digital electronic systems to be described [17]). The operation of the embedded μCS was then verified via a functional simulation using ModelSim (a software programme for simulating Hardware Descriptive Language (HDL) [10] designs) [31].

- **Assess the viability of implementing a physical DAQ system which employs a stand-alone TM-based ADC with embedded µ compensation.**

  The mathematical model of the TM-based ADC with µCA was analysed to determine if a standalone TM-based ADC with embedded, real-time µ compensation was viable. A physical standalone TM-based ADC with embedded, real-time µ compensation was then produced and analysed.

1.7 Originality of Research

The research project discussed in this thesis has led to the following original contributions:

- The development of a µ compensation algorithm (µCA) which can be embedded within a standalone TM-based ADC and perform real-time compensation. This embedded µCA improved the output accuracy of a TM-based ADC to such an extent that this data converter could be employed within a measurement system required to consistently detect small signal variations across a relatively large dynamic range.
- Further enhancements to this development produced three techniques to compensate for additional, non-ideal behaviour in the electronic implementation of the TM circuits within the TM-based ADC and enabled the future production of µ compensation systems (µCS) which can be adapted to suit different configurations of TM-based ADCs. These three techniques:
  - enabled compensation of non-ideal µ within a TM-based ADC, when the µ of the TM circuits were not identical.
  - enhanced the µCA to account for non-matching slope µs within each TM stage.

- enabled compensation for μ within a TM-based ADC when a sub-ranging ADC was employed to acquire the output of a TM.

1.8 Document Structure

This document is structured as follows:

- Chapter 2 inspects methods of evaluating ADC performance and the common architectures observed with higher resolution ADCs currently available on the market. Basic chaos theory relating to TMs is also discussed, and two literature reviews (on TM-based ADCs and initial conditions estimation of TMs respectively) are presented.
- Chapter 3 provides an overview of the proposed TM-based ADC structures and μCS.
- Chapter 4 discusses in more detail the key components, operation and implementation of the proposed TM-based ADC structures and μCS.
- Chapter 5 presents the analysis of a TM-based ADC structure with the fundamental μCA.
- Chapter 6 presents the analysis and results of the TM-based ADC structures with the enhanced μCAs.
- Chapter 7 discusses the results from chapter 6.
- Finally, chapter 8 concludes the work presented and proposes suggestions for further work.

2 Theory and Literature Review

Some of the material in this chapter was previously published in the journal paper [2].

This chapter provides the key underpinning theory and presents literature reviews undertaken as part of this work. The purpose of this research was to assess the improvement in output accuracy resulting from a µ compensation algorithm (µCA) being embedded within the TM-based ADC. Therefore Section 2.1 covers the parameters which can be employed to establish the output accuracy of an ADC when slow moving and fast changing input signals are supplied respectively.

Signal measurement systems which need the measurements to be digitised require sufficiently high-resolution ADCs in order to detect small signal variations within the analogue signal being measured. Section 2.2 covers the current architectures employed in high resolution COTS ADCs and then compares them to TM-based ADCs found in the literature in Section 2.4.3.

Section 2.3 discusses the theory of chaos and defines the conditions under which a TM, and hence a TM-based ADC, can be classed as chaotic. This definition follows through to Section 2.4 which explores the literature on TM-based ADCs and highlights other ADCs which fall under the category of TM-based ADCs.

A review of methods estimating the initial input signals of TM-based ADCs with non-ideal µ is provided in Section 2.5. Finally, Section 2.6 summarises key findings and how the information presented in preceding sections determined the course of this research.

2.1 Assessing Performance of ADCs

There are a wide variety of parameters which can be employed to assess and evaluate the performance of an ADC. These parameters fall into one of two categories: dynamic performance and static performance. Static performance concerns the behaviour of the ADC when the amplitude of the input signal is slowly changing, while dynamic performance is related to input signals with a higher rate of change [4].

The following sub-sections provides an overview of the parameters relevant to assessing the static and dynamic performance of ADCs. The third sub-section details additional parameters (observed in ADC reviews and textbooks concerning data conversion [43, 67-72]) which are relevant in assessing the performance and capabilities of ADCs for this work.

2.1.1 Static Performance

Gain error, offset error, integral non-linearity (INL) and differential non-linearity (DNL) are four parameters used to assess the static performance of an ADC [4]. Figure 2-1 present the transfer characteristic of a non-ideal 4-bit ADC and illustrates how some of these parameters are determined.

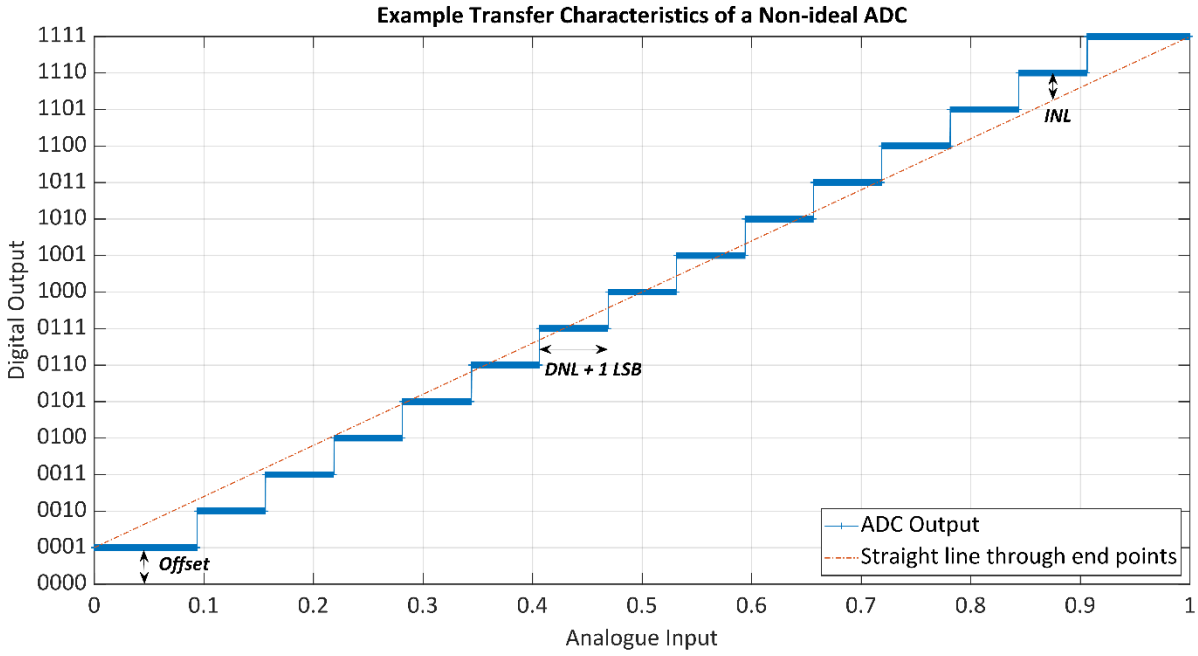


Figure 2-1: An illustration of a transfer characteristic plot of a non-ideal 4-bit ADC. Redrawn based on [4].

DNL is the maximum deviation of the step width from the ideal value of 1 LSB (which is also the ideal step size), whilst the INL is the greatest divergence from either the line of best fit through the digital output versus the analogue input plot (best straight-line INL), or the line through the two end-points of this plot (end-point INL) [6]. The end-point INL gives the worst-case scenario, as this method always provides the greater deviation from the line (Figure 2-1 also illustrates how the line is produced for the end-point INL method) [6]. Equations (2-1) and (2-2) detail how the DNL and end-point INL are calculated respectively [6, 15, 73].

$$DNL(k) = (V_{D+1} - V_D)/V_{LSB_{ideal}} - 1 \tag{2-1}$$

$$INL(i) = (V_{actual}(D) - V_{ideal}(D))/V_{LSB_{ideal}} \tag{2-2}$$

Where:

- D is the digital output code;
- V_D is the analogue value which corresponds to D;
- $V_{LSBideal}$ is the ideal step size;
- k represents each potential digital output ($2^{resolution}$ in total);
- i represents each change in the analogue input; and
- V_{actual} and V_{ideal} (ideal refers to the end-point plot) represent the minimum actual and ideal voltage values which produce D [6, 15, 73].

The offset error of an ADC corresponds to, the minimum input required to generate a zero output code [15] or, where the transfer characteristic end-point plot of the ADC intercepts the axis representing the digital output [4]. The latter of these definitions is represented in Figure 2-1 and (2-3) and will be employed in this work [4].

$$offset = DigitalOutput(Analogue\ Input = 0) \quad (2\text{-}3)$$

With (2-3), DigitalOutput(Analogue Input) refers to the analogue input to digital output transfer function, which details the digital code produced by the ADC for a given analogue input.

The gain error of an ADC is the difference between the full-scale error and the offset as represented in (2-4). The full-scale error is the difference between the maximum digital output of the actual and ideal ADC [28].

$$GainError = FullScale_{error} - offset$$

$$= \left(max(DigitalOutput_{actual}) - max(DigitalOutput_{ideal})\right) \quad (2\text{-}4)$$

$$- offset$$

$DigitalOutput_{actual}$ represents the digital codes produced by the ADC under assessment, whilst $DigitalOutput_{ideal}$ refers to the digital codes which would be generated by a theoretically ideal ADC.

- Table 2-1 below summarises all the static parameters discussed in this sub-section.

Static Parameter	**Definition**
Differential non-linearity (DNL)	The maximum deviation of the step width from the ideal value of 1 LSB [6].
Full-scale error	The difference between the maximum digital output of the actual and ideal ADC [28].
Gain error	The difference between the full-scale error and the offset error [28].
Integral non-linearity (INL)	The greatest divergence from either the line of best fit through the digital output versus analogue input plot (best straight-line INL), or the line through the two end-points of this plot (end-point INL). The latter of these two INL measurements provides the worst-case scenario, as the method always provides the greater deviation from the line [6].
Offset error	Corresponds to the minimum input required to provide a zero-output code [15], or where the transfer characteristic end-point plot of the ADC intercepts the axis representing the digital output [4].

Table 2-1: Summary of static parameters.

2.1.2 Dynamic Performance

The dynamic performance of the ADC can be assessed using the following parameters:

- signal to noise ratio (SNR) [4];
- spurious free dynamic range (SFDR) [14];
- total harmonic distortion (THD) [14] and
- signal to noise and distortion ratio (SINAD or SNDR) [4, 15].

All of these parameters can be established by supplying the ADC with a clean, sinusoidal input signal and taking a Fast Fourier Transform (FFT) of the digital output, then analysing the resulting spectrum [4, 8, 14].

The SNR is the ratio of the input signal power and the average noise power (excluding the power within the signal harmonics) [4]. SINAD is the same as SNR except the power of input signal is compared to the magnitudes of the noise and harmonics (the latter is represented by the parameter THD) [4, 14, 15]. The SFDR, meanwhile, is the difference in magnitudes between the input signal (the fundamental peak) and the harmonic with the highest magnitude [14].

When performing a FFT to determine the parameters stated above, the ADC needs to be supplied a sinusoidal input signal, which meets the criteria given in (2-5). The amplitude range of the sinusoidal input must match the valid input voltage range of the ADC [7].

$$\frac{f_{in}}{f_{sample}} = \frac{M}{N} \tag{2-5}$$

With (2-5), f_{in} is the input frequency, f_{sample} is the sampling frequency, M represents the number of signal cycles and N the number of data points. To minimise spectral leakage (when discontinuities at the ends of the sinusoidal signal, which an FFT is being performed on, causes the peak in the resulting spectrum to spread into adjacent frequency bins and affect the spectral distribution [8]), M should be an odd integer number, whilst N must be a power of 2 (the higher the value, the more accurate the FFT) [7].

From the SINAD measurement, the effective number of bits (ENOB) can also be calculated using (2-6) [4]. ENOB represents the number of bits to which an ADC can accurately convert an analogue input into a digital word [7].

$$ENOB = \frac{SINAD - 1.76}{6.02} \tag{2-6}$$

- Table 2-2 below summarises all the dynamic parameters discussed in this sub-section.

Dynamic parameter	**Definition**
Effective number of bits (ENOB)	This represents the number of bits that an ADC can accurately represent analogue input signals as digital words [4, 7].
Spurious free dynamic ratio (SFDR)	The difference in magnitude between the signal (the fundamental peak) and the harmonic with the highest magnitude [14].
Signal to noise and distortion ratio (SINAD or SNDR)	The same as SNR except the signal power is compared to the magnitudes of the noise and harmonics [4, 15].
Signal to noise ratio (SNR)	The ratio between the signal power and the average noise power (excluding the power within the signal harmonics) [4].
Total harmonic distortion (THD)	The magnitude of the harmonics within a signal summed together [14].

Table 2-2: Summary of dynamic parameters.

2.1.3 Other Performance Parameters

This research focuses on the ADC output accuracy. However, there are some additional parameters, of relevance to this work, which are also useful when selecting an ADC for a data acquisition application. Table 2-3 summaries these additional parameters, which are discussed in other publications concerning ADCs [43, 67-72].

Parameter Term	**Definition**
Bandwidth	The range of input frequencies the ADC can accept [23].
Conversion rate/frequency	Rate ADC can convert an analogue sample to the digital domain [25]
Latency	The time delay in the conversion and transmission of a signal sample to the digital domain [74].
Quantisation error	Error between input and equivalent output signals in terms of LSBs [4].
Sampling frequency	Number of samples per second [15].
Stated resolution or resolution	The total number of quantisation bits of the ADC. Can also be thought as the total number of bits in the digital words an ADC produces [15].

Table 2-3: Summary of additional parameters which can be employed to assess ADC performance.

2.2 Overview of Main High Resolution ADC Architectures

2.2.1 Research Procedure

The purpose of the research discussed in this thesis was to assess whether the improvement in output accuracy of TM-based ADCs, after employing a µ compensation system (µCS), would make them a good candidate for high precision and high accuracy measurement systems. Such measurement systems need to detect small signal variations across the whole valid input signal range, which requires ADCs with sufficient resolution to acquire and convert the minimum variation needing to be detected.

For this reason, searches were conducted in early July 2021 to establish the architectures being employed in higher resolution COTS ADCs and to enable comparisons with the TM-based ADC architecture. Five distributors of electronic components were used in this investigation[5].

The highest and lowest ADC resolutions available were 32 and 1 bits respectively. With this investigation the upper half of the resolution range were considered (ADC resolution was 16.5 bits < R ≤ 32 bits). Five architectures were observed across this range of resolutions, these being: sigma delta, dual-slope, multi-slope, pipelined and successive approximation register (SAR). The following four sub-sections explain how these architectures operate.

[5] The retailers were RS Components Ltd., Arrow Electronics Inc., Premier Farnell Ltd., Digi-key Electronics and Mouser Electronics Inc. [75 - 79].

2.2.2 Sigma Delta ADCs

The sigma delta (also known as oversampling) ADC uses the techniques oversampling (when a signal is being sampled at a rate larger than double the ADC bandwidth [8]), digital low pass filtering and decimation (the process of deleting samples [26]) to improve the resolution of its conversions [37]. The circuitry is shown in Figure 2-2. The operation of this ADC architecture involves feeding the sampled input signal into a difference amplifier (U1) and integrating the output signal using an integrator circuit (U2). The slope of the integrator output signal then determines whether the 1-bit ADC outputs a 1 or a 0. This result is then sent both to the digital filter and 1-bit digital to analogue converter (DAC), the latter determines whether the negative input of the difference amplifier (U1) is connected to a positive or negative reference voltage [80]. The digital filter removes most of the quantisation noise (this being shifted to the higher frequency spectrum by the integrator [80]), before transmitting the processed signal to the decimator which removes specific samples in order to reduce the output data rate [80]. The sigma delta ADC architecture can have more than one integrator in its circuitry (the number of integrators determines the order of the ADC) [43].

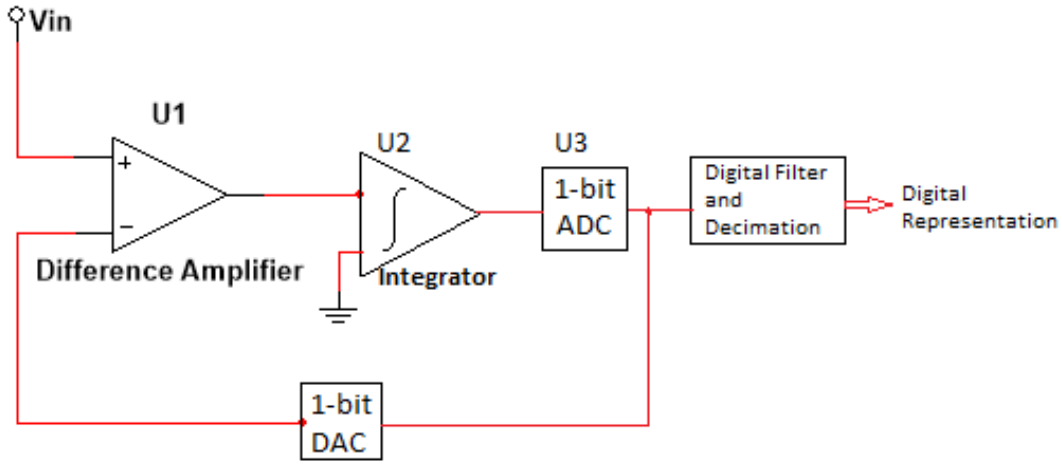


Figure 2-2: Sigma delta architecture. Redrawn from [43].

2.2.3 Dual-slope and Multi-slope ADCs

The dual-slope and multi-slope ADC architectures are subsets of the integrating ADC architecture class [27]. The simplest integrating ADC architecture is the single-slope ADC, whose architecture forms the basic operation of all integrating ADCs. Figure 2-3 presents the circuitry of a single slope ADC, which has an integrator, comparator and counter. The sampled input signal feeds into integrator circuit (U1), causing the capacitor within the circuit to charge. The comparator (U2) is employed to compare the integrator output with a known reference voltage and to change output state when the integrator output exceeds this reference. The counter tracks the time taken for the comparator output to change state: this time duration is proportional to the input signal and is used to determine digital representation of the sampled input signal [25, 43].

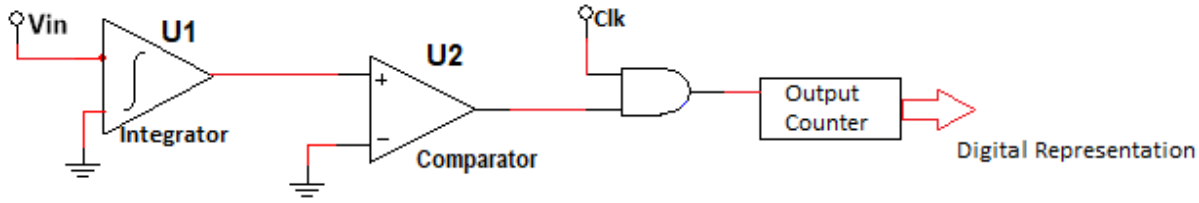


Figure 2-3: Single-slope integrating type ADC. Redrawn from [43].

With the dual slope ADC (Figure 2-4) the sampled input signal is feed into the integrator for a set period. The capacitor within the integrator then discharges (by switching the input voltage to U1 from Vin to Vref) causing the integrating circuit to de-integrate. The counter increments whilst the integrator de-integrates, and the final value produces the digital word output of the sampled analogue input signal [25, 43].

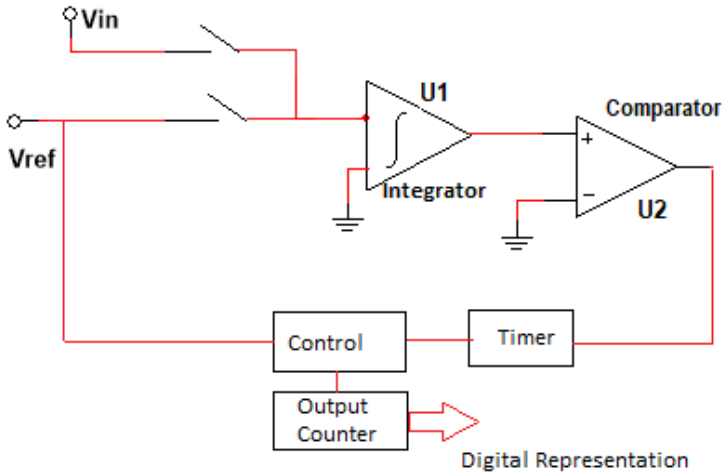


Figure 2-4: Dual-slope integrating type ADC. Redrawn from [43].

Multi-slope ADCs add a further level of complexity to the integrating architecture. Such architectures use a single integrating and de-integrating cycle to determine a set number of MSBs for the digital output. The final voltage level given by the integrator circuit is then

amplified by a set amount and de-integrated again to produce the final bits of the digital output. This latter part can be repeated to further increase the resolution [27].

2.2.4 Pipelined ADCs

The pipeline ADC architecture (see Figure 2-5) consists of several stages arranged in series (each stage processes a sample of the input signal for one clock cycle before sending the signal onto the next stage for further processing). Although this method does introduce significant latency, the throughput (and hence potential sampling frequency) is high [35].

The processing provided by each stage involves finding the sum of a set of reference voltages that equal the amplitude of the sampled signal. By sequentially subtracting each of the reference voltages from the sample until the remainder voltage (the residue) is close to zero, the correct combination of reference voltages required to represent the sample can be found. Each reference voltage represents a bit in the final digital word. To further improve the accuracy of the conversion the residue is amplified between stages by a factor of 2^k (where k is the number of bits determined by the pipelined stage) [8, 35].

Every time the sample passes through a processing stage the computed value for that bit goes to a bit aligning stage. When the sample has passed through all the pipelining stages, the bit aligning stage groups the relevant bits for the sample and outputs them as a single digital word [35].

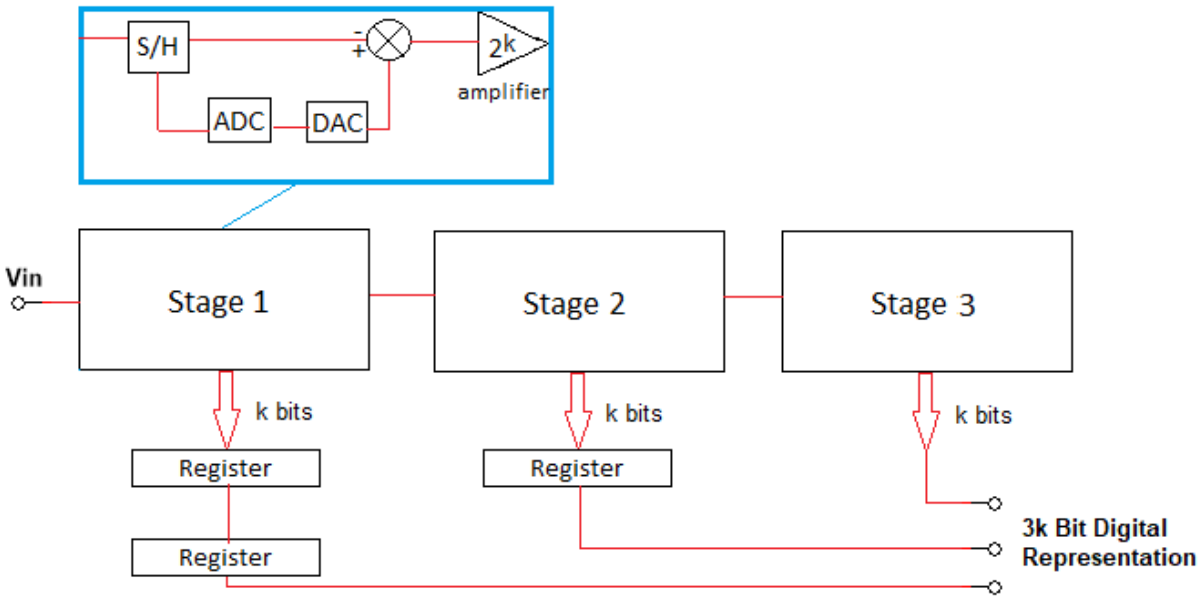


Figure 2-5: Pipeline architecture. Redrawn from [8].

2.2.5 SAR ADCs

The SAR ADC uses a type of binary search algorithm known as the successive approximation algorithm to perform the conversion [81]. Figure 2-6 shows the circuitry for this architecture. Each time a sample of the analogue input signal is taken, the ADC determines whether the output digital word should have a high or low MSB. This is achieved by comparing the sampled input signal with the DAC output signal (the amplitude of this latter signal is determined by the digital code supplied by the decision register) [43, 81].

The comparator output representing the MSB then determines the next digital code supplied to the DAC in order to produce a different output signal. This signal is again compared to the sampled input signal to determine the second MSB. This process is then repeated until the LSB has been established [43, 81].

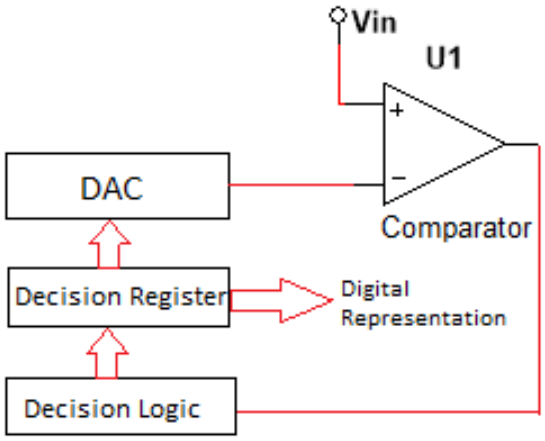


Figure 2-6: SAR architecture. Redrawn from [4, 43].

2.2.6 Higher Resolution ADC Architectures Analysis

Sigma delta ADCs offer high resolution and can be low cost due to the data conversion process being more reliant on digital, rather than analogue circuitry, thus precision components are not required. This has the added benefit of requiring no calibration or trimming [15, 27]. The architecture is also very stable, even at higher orders, due to stable lower order loops in multi-stage noise shaping modulators (MASH) [82]. Oversampling however limits the sampling speed of this architecture, and a large circuit die area is required to fabricate the ADC if a high-order or multibit system is employed [43, 71]. Despite this, sampling frequencies of sigma delta ADCs have risen from hundreds of MHz to the GHz region due to technology scaling, supporting bandwidths greater than 100MHz [70]. At very high resolutions though, sampling speeds are still restricted: the fastest on the market in Summer 2021 being at 1 MHz [49].

Dual-slope and multi-slope integrating ADCs offer low power consumption [8] but are unable to reach very high resolutions, unlike sigma delta ADCs [43]. Both these architectures

however have long conversion times (for dual slope it is $2^{(R+1)}$ cycles) [43]. In common with sigma delta ADCs, dual-slope ADCs have good rejection of 50/60 Hz signals, as the conversion process rejects frequencies which are multiples of the integration rate. This type of ADC architecture is good for low level signal conversion but, unlike sigma delta ADCs, requires external components (resistors and capacitors) to convert voltage signal to current signal and set reference voltages [27]. Trimming and calibration is also required for integrating ADCs to compensate for errors resulting from the analogue circuitry [27]. Integrating ADCs also have very low bandwidths and are restricted to acquiring 100 samples per second [27].

Pipeline architectures are the only ADC architecture capable of meeting the most demanding specifications for data conversion, and can achieve good resolution, power consumption and sampling speed simultaneously [43, 70, 71, 83]. When being compared to parallel ADC architectures which determine all bits of the digital output within one clock cycle, the pipelined architecture is noticeably slower, but still faster than other serial ADC architectures [43]. The main limitation with the pipelined architecture is the high power consumption associated with the op-amps required for the residue amplification with high resolution and high speed designs [83].

SAR architectures are renowned for power efficiency which is continuing to improve [43, 70]. However, there is a trade-off between speed, power efficiency and resolution [43, 70]. This architecture is suitable for low speed operations, but can also be employed as a sub-ranging ADC in other ADC architectures (for example, pipelined) to achieve higher sampling rate [15]. SAR ADCs have a higher bandwidth than integrating ADCs, but are not suited for low level signals and have poorer line rejection [27]. Like integrating ADCs, SAR ADCs also need

trimming and calibration ADCs to compensate for errors resulting from the analogue circuitry [81].

All the ADC architectures discussed in this section are summarised in Table 2-4 below and ranked in terms of resolution, sampling speed, conversion speed and fabrication area. The resolution and sampling speed were ranked based on the maximum performance achieved by higher resolution ADCs currently on the market [75-79]. Many commercial ADCs did not advertise information on conversion rate and fabrication area, thus for these two categories, the four ADC architectures were ranked based on published studies on different ADC architecture types [43, 71]

ADC architecture (alphabetical order)	Process of Determining Digital Codes Summarised	Resolution (4 = highest 1 = lowest)	Sampling Speed (4 = fastest 1 = slowest)	Conversion Speed (4 = fastest 1 = slowest) [43, 71]	Fabrication Area (4 = largest 1 = smallest) [43, 71]
Integrating (Dual-slope and Multi-slope)	Digital codes determined from the time taken for the integrated signal amplitude to cause the comparator output to toggle.	2	1	1	3[6]
Sigma Delta	Digital codes determined from the duration of pulses, produced by the analogue circuitry.	4	3	3	2
Pipelined	The cascaded stages each determine a set number of bits for the digital codes generated.	1	2	4	1
SAR	Digital codes determined by repeatedly comparing a DAC output signal (amplitude is set by a digital code) with the sampled input signal. Digital code is determined from the value which results in the DAC output having the closest amplitude to the sampled analogue input..	3	4	2	3[6]

Table 2-4: Summary of main high resolution ADC architectures [43, 71, 75-79].

[6] Studies declared SAR and integrating ADCs as having low fabrication area but did not indicate which of the two architectures could achieve the smallest circuit area [43, 71].

2.3 Chaos and the Discrete One-Dimensional Chaotic Tent Map

A chaotic system is a system which shows chaotic behaviour. A system exhibiting chaotic behaviour is deterministic (produces the same output for a given input [22]) and follow simple rules, but the behaviour is non-linear and complex. The complexity of the system is due to internal dynamics rather than random external influences [19].

Chaotic systems can be represented using mathematical models, which are known as flows when a continuous (can be defined for all of time during a certain period) dynamical system is being represented, or maps for discrete (can only be defined at set intervals during a certain period) dynamical systems [19, 24]. Maps require fewer computing resources than flows, as difference mathematical equations, which employ iteration, are utilised rather than differential equations [19].

Maps can be multi-dimensional; however one-dimensional (1-D) maps are the simplest to work with and analyse as only a single parameter needs to be considered [16, 19]. The TM function is one of numerous discrete, 1-D mathematical models of a chaotic system [58] and can be summarised using the difference equations given in (1-2), which has been reproduced below [19].

$$x_{n+1} = \begin{cases} \mu x_n & \text{when } x_n \leq 0.5 \\ \mu(1 - x_n) & \text{when } x_n > 0.5 \end{cases} \quad (1\text{-}2)$$

x_n and x_{n+1} represent the input and output of the TM respectively (the original input, x_0, is referred to as an initial condition), whilst n signifies the number of iterations. μ is the TM gain

and the value of x_n where the TM transitions between the two difference equations is known as the partition point [34].

A system experiencing chaotic behaviour needs to be:

- deterministic;
- bounded (the maximum difference between two points within the output of the system is less than infinity);
- aperiodic (the system does not produce a periodically repeating output); and
- sensitive to initial conditions (the input signal(s) to the system) [22].

The rest of this section sets out to prove a TM can be a chaotic system under certain conditions.

The TM can be explained using the difference equations in (1-2), which proves the map is deterministic as the same output will be produced for a given input. Confirming whether a chaotic map is bounded or not can be achieved by plotting a bifurcation diagram (which plots the final states of a chaotic map over a range of control parameters (e.g., μ for a TM)). A bifurcation diagram confirms that a chaotic map stays between two limits or goes towards $\pm\infty$ as well as providing an insight on the state of the system with different control parameter values (e.g., stationary, periodic, or chaotic). Figure 2-7 shows the bifurcation diagram of a TM which confirms the TM remains bounded between 0 and 1 over a range of $1 < \mu \leq 2$. The diagram also highlights the TM final state is at zero when $\mu < 1$, while when $\mu \geq 1$ the plot creates a stroboscopic effect which highlights chaotic behaviour [22].

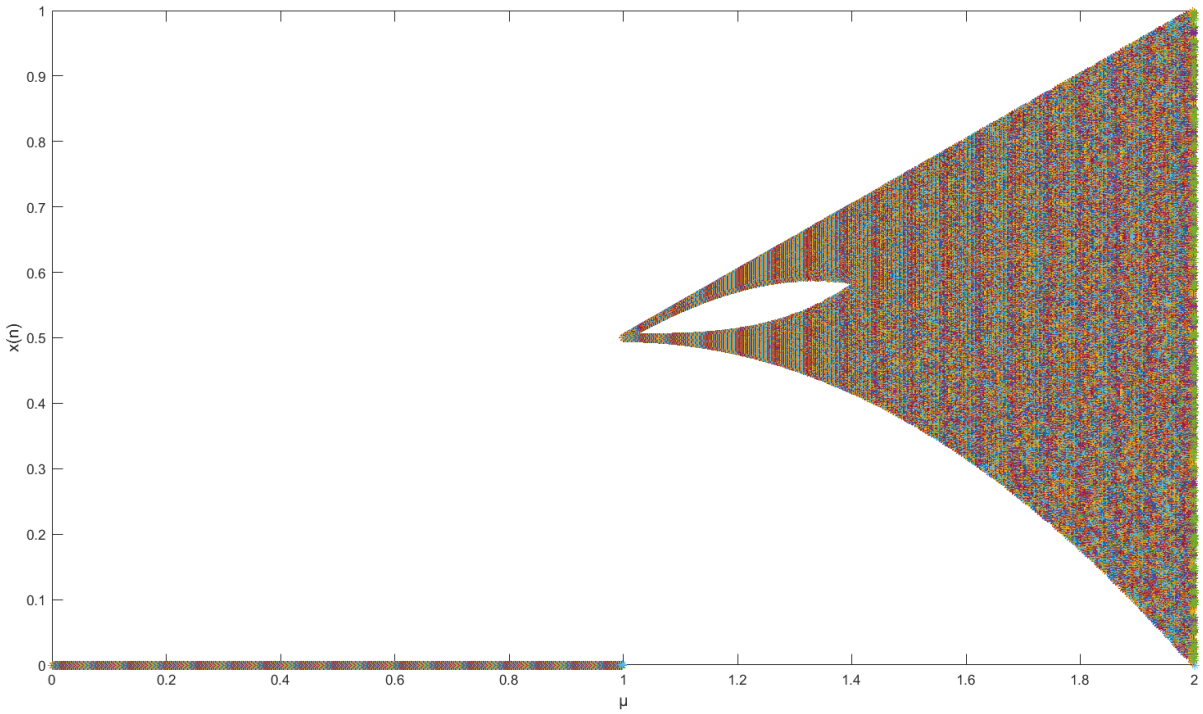


Figure 2-7: Bifurcation diagram of a TM. Redrawn based on [16].

Observing evidence of aperiodic behaviour over a range of μ can be obtained by plotting the output of a TM during 100's of iterations [22]. Figure 2-8 presents a range of plots of iterated TM output signals for $0 \leq \mu \leq 2$, when the initial condition was 0.5 (this value was chosen due to being a non-zero number within the range 0 to 1). The plots highlight the TM output signal is aperiodic when $0 \leq \mu \leq 2$, as the values plotted for each gain do not repeat on a periodic basis. The exception is when $\mu = 1$, where a constant signal (which is classed as a periodic signal with an undefinable period [84]) is produced.

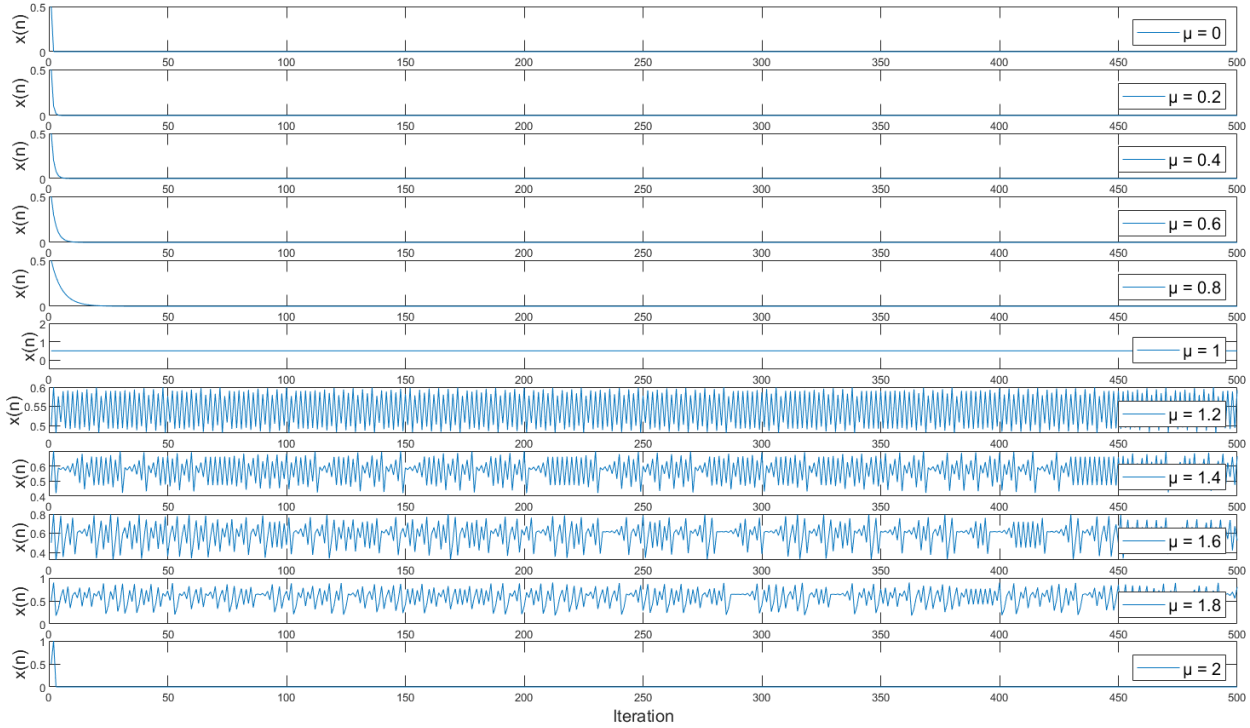


Figure 2-8: Plots of TM output, with different values of μ, after 100s of iterations, which are employed to find evidence of aperiodic behaviour. Based on method presented in [22].

Finally determining whether a chaotic map is sensitive to initial conditions can be achieved by calculating the Lyapunov exponent (λ). The Lyapunov exponent is a measure of sensitivity dependence a chaotic map might have on the initial conditions [19]. (2-7) presents the calculation for the Lyapunov exponent, which when the result is positive shows the map is sensitive to initial conditions [19].

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} ln|f'(x_i)| \tag{2-7}$$

Where n refers to the iteration number of the chaotic map being considered, f'(x_i) is the derivative of the chaotic map for a given input (x_i) and λ represents the Lyapunov exponent.

Figure 2-9 plots the Lyapunov exponent of a TM over a range of gains showing that the TM is sensitive to initial conditions when $\mu > 1$ and supports the observations made in Figure 2-7 [19].

Figures 2-7 to 2-9 and (1-2) highlight that the TM only exhibits chaotic behaviour over the TM gain range $1 < \mu \leq 2$ as this is the range where the TM is deterministic, bounded, aperiodic and sensitive to initial conditions [22].

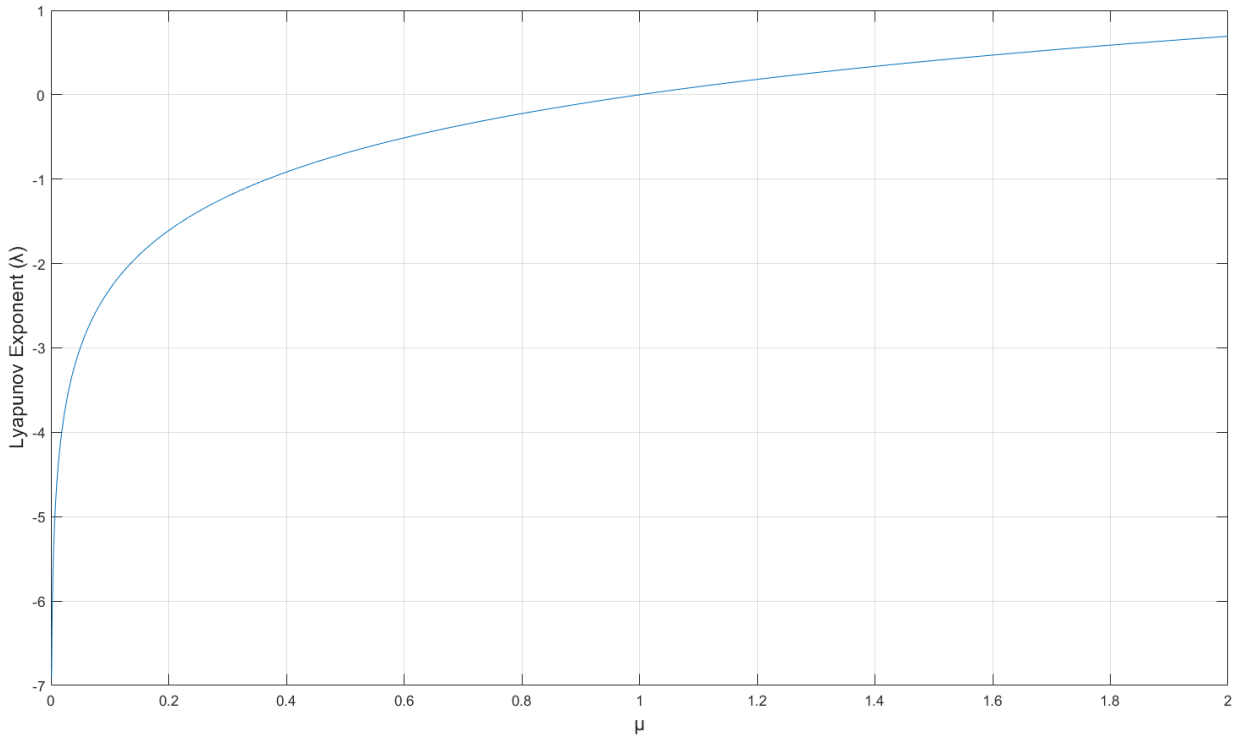


Figure 2-9: Plot of Lyapunov exponent of a TM. Redrawn based on [85].

2.4 Tent Map Based ADCs

2.4.1 Classification of TM-based ADCs

The literature published on ADCs employing TMs is limited. Only TM-based ADCs were considered for this research because although ADCs which employ different types of 1-D chaotic maps (e.g., Bernoulli maps and Logistic maps) have been previously analysed, past research has determined that TM-based ADCs achieve better performance [16, 34]. A TM-based ADC with a $\mu = 2$ has been proven to achieve a more linear output response across the whole valid input signal range than a logistic map based ADC with the same amplification factor [16]. Chaotic ADCs which employ Bernoulli maps in place of TMs have worse output accuracy when the amplification gain and partition point voltages are not the ideal values of 2 and the mid-point of the valid input voltage range respectively [34].

Certain types of Gray-code algorithmic ADCs and folding ADCs employ folding and amplification circuits that can be classified as TMs. Therefore, these types of ADC architectures can also be classed as TM-based ADCs. Kennedy argued any algorithmic converter is a discrete-time dynamical system and those employing TMs will output Gray code [29]. However, not all Gray-code algorithmic ADCs employed the TM function to perform the conversion (some, for example, inverted the TM function and produced reverse Gray code [86, 87], as illustrated in Figure 2-10). The µCA developed for this work required the Gray code output from a non-inverting TM-based ADC. Only non-inverting TM-based ADCs were considered in this research and literature reviews, because the µCA developed operated by processing non-reverse Gray code generated by such ADCs, although it should be noted that the µCA could be adapted to cope with reverse Gray code.

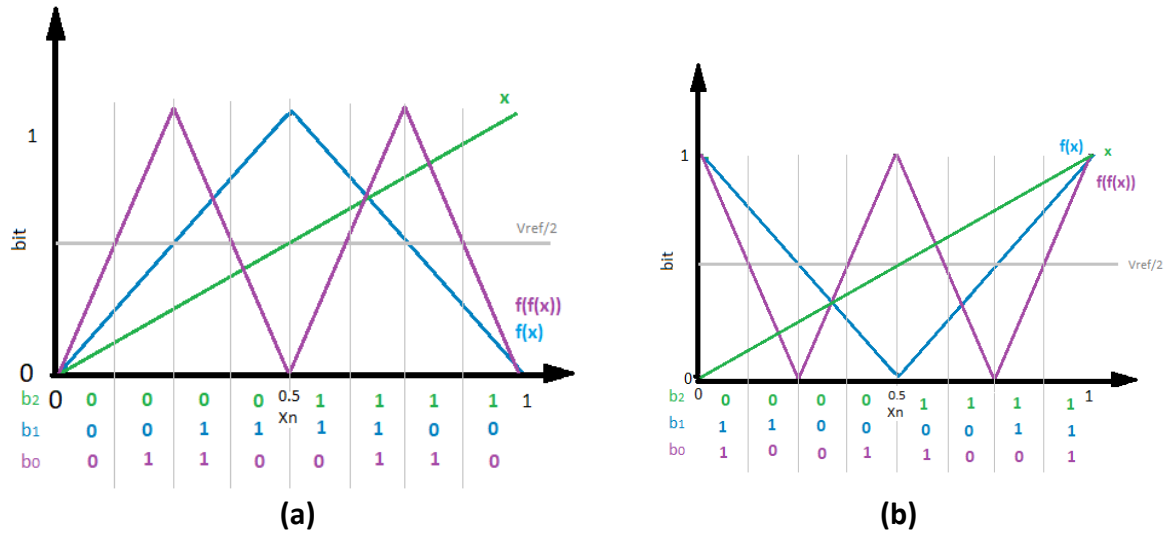


Figure 2-10: (a) Analogue-to-digital conversion using the Gray-code algorithm. Based on [86]. (b) Analogue-to-digital conversion using the reverse Gray-code algorithm. Based on [86].

Most texts define folding ADCs as an architecture which employ non-amplifying folding circuits [4, 8, 15]. The exception to this definition is the one given by Kester, whose definition is identical to the Gray-code algorithmic ADC [88].

Some literature on TMs state the μ should be exactly 2 [22, 89, 90], while others believe it should be within the range of $1 < \mu \leq 2$ [3, 19]. For the literature review performed, the latter, more inclusive, of these two definitions was considered [3, 19], in the hope of widening the scope of acceptable literature.

Most of the TM-based ADCs studied followed the same operation, which involved taking an input signal and employing comparators to establish the MSB, by comparing the amplitude with a fixed reference voltage representing the partition point. The input signal was also symmetrically folded, amplified and forwarded to the next TM stage so the subsequent MSB could be established. The ADCs required the μ to be exactly two and the partition points to be precisely halfway between the full-range of the input signals to prevent encoding errors [34].

The ADCs employed the TMs in either a series or feedback configuration (illustrated in Figure 2-11 and Figure 2-12). Some architectures used a series of TMs and comparators [56, 57], while another architecture involved a single comparator and TM function in a feedback circuit [13].

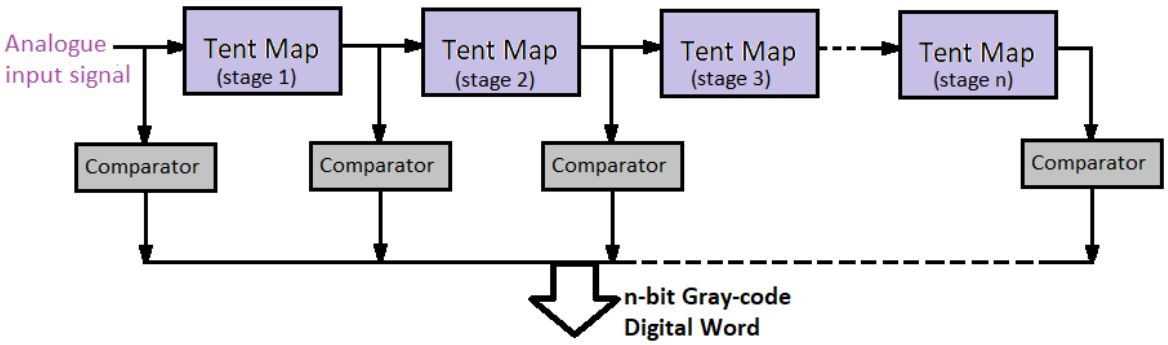


Figure 2-11: Series TM configuration. Based on [56, 57].

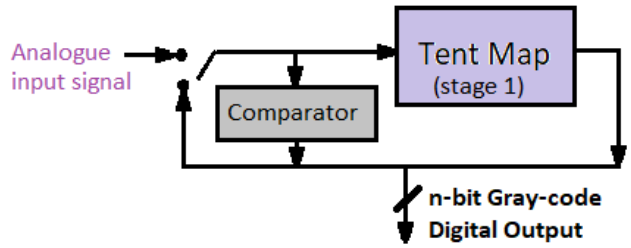


Figure 2-12: Feedback TM configuration. Based on [13].

2.4.2 TM-based ADCs

The earliest example of a TM-based ADC found was proposed by Smith in 1956 who suggested both a series and feedback configuration. One TM circuit design given comprised op-amps, resistors, and diodes (one op-amp had no feedback gain and acted as a comparator). The feedback configuration employed two additional capacitors with the TM circuit to store the input and output voltages (then switched the latter to the TM input for the next iteration and released the other capacitor to store the new output). The TM circuits cascaded together to create the series configuration but required the input to remain constant as the TM stages had no respective sample and hold circuit. This lack of individual sample and hold circuits combined with the series configuration, restricted the sampling rate as the signal could not be pipelined, but was still faster in terms of sampling rate and conversion speed when compared to the feedback configuration [91].

A range of TM-based ADCs were developed by a group of researchers in Thailand which involved TM folding of current signals to perform analogue to digital conversion [92-95]. The first design was implemented in CMOS technology and involved the analogue to digital conversion of a current input signal [92]. This design was analysed via simulation [92]. The next version of the ADC employed transconductance amplifiers to change voltage to current signals and used a voltage comparator to produce the digital output. This design was constructed as a 4-bit practical circuit and despite having a relatively high voltage supply of ±10 V for some of the amplifiers, the valid input voltage range was limited to 0 - 1 V [93]. This voltage range was increased to 0 - 5 V in a later redesign (which also reduced components for each TM stage), although only a 4-bit practical implementation was demonstrated [94].

The circuits employing operational transconductance amplifiers needed careful design to minimise transfer errors in converting the input voltage to the output current signal. The next version of the TM-based ADC used a voltage to current converter, which employed an op-amp and operational current conveyor (a device which transfers current from one impedance level to another [33]), rather than operational transconductance amplifiers, to perform the folding operation, as this converter was proven to have a theoretically lower transfer error. An 8-bit prototype was constructed using discreet components, which showed the error between the output and input signals was less than 5%. However interpreting the effective resolution of the ADC, and whether the device was capable of detecting a step size change in the input signal, from the results is impossible [95]. All of these variations were series configurations which required the input signal to remain constant during each conversion cycle [92-95].

Both Litovski et al. and Liu et al. employed the same feedback configuration design, which was constructed from switch-capacitor circuitry [13, 96]. Litovski et al. verified the TM-based ADC operation via simulation and analysed the effect non-ideal behaviour from imprecise gain, partition points and op-amp impedance had on accuracy [96]. Liu et al meanwhile employed this TM-based ADC design to produce multiple sub-ranging ADCs, which simultaneously acquired multiple voltage output signals from a tactile sensor, and assessed the performance by testing a fabricated design [13]. The simulated ADC of Litovski et al. performed 20 iterations (20-bits resolution) while Liu et al. only used 8 iterations (8-bits resolution) for every sub-ranging ADC. Both articles failed to confirm whether the TM-based ADC detected the minimum step changes (954 nV for the 20-bit ADC [96] and 3.91 mV for an individual 8-bit sub-ranging ADC [13]). This design also employed area saving switch-capacitor circuitry in order to be fabricated as an IC [13, 96, 97].

Upton and Berberkic both developed TM-based ADCs constructed using discrete components [16, 56, 57], making them cheaper to prototype and develop. Upton developed a series TM-based ADC configuration designed to acquire the amplitude of sporadic pulses [56, 57], which required the ADC to have a sufficiently high conversion rate and fast operating internal comparators. For this reason the comparators were configured without hysteresis (a circuit trait where two triggering levels enable delayed switching [23]), making them susceptible to noise (the outputs switched between states even when the TM-based ADC was supplied a constant input) [56, 98].

Berberkic explored two TM-based ADCs designs. One used a feedback configuration (Figure 2-13), while the other employed a series configuration (Figure 2-14), but unlike the designs discussed above, the comparators were replaced with 10-bit ADCs. The series approach allowed smaller changes in the input signal to be detected (noise limits the sensitivity of the feedback system) but required more hardware to detect smaller changes. Meanwhile the feedback system offered more flexibility regarding the number of iterations that could be performed and was easier to control. However, the sampling speed was slower and errors were introduced by the sample and hold, and switching circuitry [16].

Both designs by Berberkic offered higher sensitivity (each detected 50 µV changes over a 0 - 10 V range [16]) than the other ADCs reviewed in this section and had potential of offering higher bandwidth. Folding increases signal frequency by a factor of 3.14 (known as frequency multiplication) [99]. Employing mid-resolution ADCs, instead of comparators, to acquire the output signals from each TM, meant fewer folding stages (iterations) were required, resulting in a lower frequency multiplication rate. As the op-amps employed by the TMs have a

bandwidth limit, a lower frequency multiplication rate allows the TM-based ADCs to handle higher frequency input signals [99] and achieve high-resolutions.

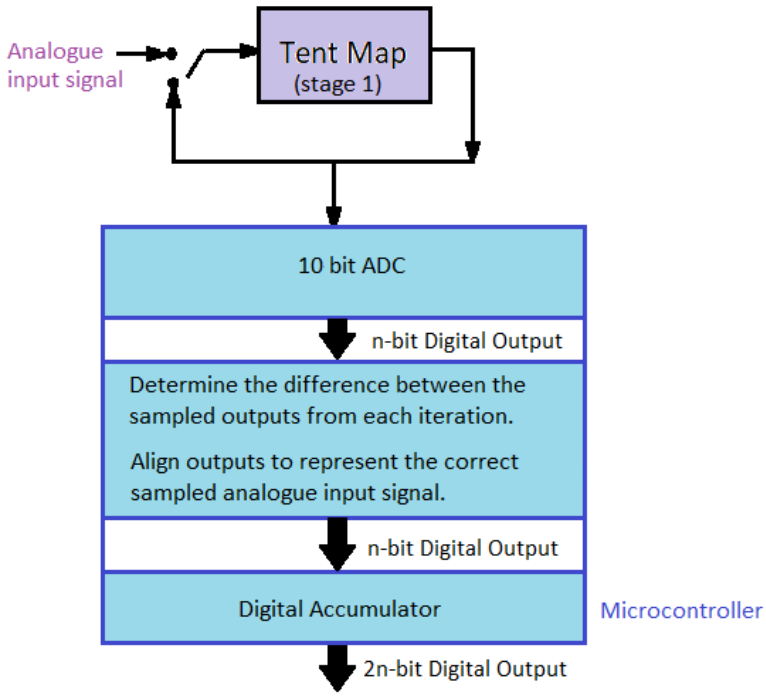


Figure 2-13: Feedback configuration of the TM-based ADC by Berberkic. Reproduced from [16].

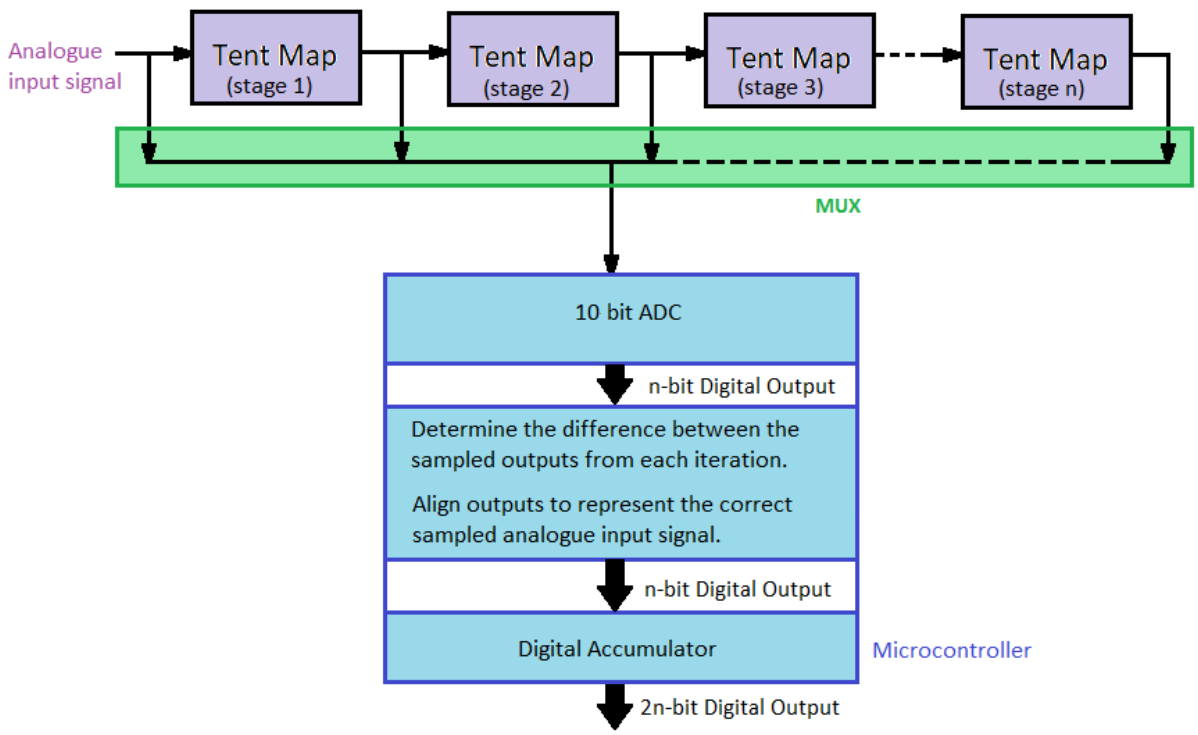


Figure 2-14: Series configuration of the TM-based ADC by Berberkic. Reproduced from [16].

- Table 2-5 below summarises all the TM-based ADCs discussed in this section.

Literature (chronological order)	**Maximum resolution (bits)**	**Implementation of proposed design**	**Analysis of proposed design**
Smith (1956) [91]	Not Applicable.	Theoretical	Not Applicable.
Arayawat et al. (2004) [92]	7	Theoretical	Simulation
Litovski et al. (2006) [96]	20	Theoretical	Simulation
Chaikla, Arayawat and Riewruja (2006) [93]	4	Discrete components	Testing of Practical circuit
Arayawat et al. (2008) [94]	4	Discrete components	Testing of Practical circuit
Petchmaneelumka and Julsereewong (2010) [95]	8	Theoretical and Discrete components	Simulation and Testing of Practical circuit
Liu et al. (2013) [13]	8	Fabricated CMOS Switch-capacitor.	Testing of Practical circuit
Berberkic (2014) [16]	19	Discrete components	Testing of Practical circuit
Upton et al. (2020) [56]; Upton (2018) [57]	8	Discrete components	Testing of Practical circuit

Table 2-5: Summary of the TM-based ADCs found in literature.

2.4.3 Comparison of TM-based ADCs and Other Higher Resolution ADC Architectures

When compared to the integrating, dual-slope and multi-slope ADC architectures the TM-based ADC is able to achieve higher conversion and sampling rates [16, 56, 57, 75-79]. Also, certain variations of the TM-based ADC architecture can achieve higher resolutions than current COTS integrating dual-slope and multi-slope ADCs [16, 75-79].

Most pipelined ADCs establish multiple bits per stage, but the conversion rate of the individual stages restricts the sample rate of the ADC. The TM-based ADC design proposed by Upton [56, 57], which comprises a series configuration, and employs comparators to digitise the TM input and output signals, could be classed as a one-bit per stage, pipelined ADC, and may be capable of higher sampling speeds than a traditional pipelined ADC, as resolving one bit tends to be faster than resolving multiple bits. However, more one-bit stages (opposed to a few multi-bit stages) may increase the latency, and thus the overall conversion rate of the ADC.

Both the TM-based ADC and SAR architecture require a fixed number of conversion cycles to convert an analogue sample to a R-bit digital word. However, the feedback TM-based ADC configuration is not limited by the DAC settling time or digital logic latency, unlike the SAR ADC [81]. Therefore, a TM-based ADC may potentially achieve fast conversion rates. Also, despite the increase in circuitry, the series configuration of the TM-based ADC could enable a faster sampling rate if each TM stage had a respective sample and hold circuit.

Sigma delta ADCs are available on the market which have higher resolutions than the TM-based ADCs observed in literature [46-49], but have lower sampling rates than that achieved by the lower resolution TM-based ADC designed by Upton [57]. By combining work by Berberkic and Upton, a TM-based ADC could be designed to employ a sigma delta ADC and

further enhance the performance of both architectures [16, 56, 57]. Higher resolution sigma delta ADCs tend to have lower sampling speeds than the lower resolution counterparts. By employing a TM-based ADC to determine a set number of the MSBs, a high-speed, mid-resolution, sigma delta ADC could acquire the final TM output thus establishing the remaining bits. This could enable a faster, higher resolution ADC using a hybrid of the two ADC architectures.

- Table 2-6 provides a summary of how the current performance of TM-based ADCs compare to, and could be advantageous over, the mainstream ADC architectures discussed in Section 2.2.

Mainstream Architecture (alphabetical order)	**Potential Benefits of TM-based ADCs, based on Current Performance Achievements**
Integrating	TM-based ADCs, compared to integrating ADCs, have achieved higher conversion rates, sampling rates and resolution.
Pipelined	Most series TM-based ADCs which employ comparators for the digitization process, may be capable of higher sampling speeds at higher resolutions than pipelined ADCs (resolving one bit tends to faster be than multiple bits).
SAR	Both TM-based and SAR architectures require a fixed number of conversion cycles. However, TM-based ADC are not limited by the DAC settling time or digital logic latency, so could achieve faster conversion rates. Series TM-based ADCS configurations could also achieve faster sampling rates than SAR architectures as the conversion of the preceding sample does not need to be complete before the next sample is acquired.
Sigma Delta	Sigma delta ADCs have higher resolutions than current TM-based ADCs, but lower sampling rates than the TM-based ADC by Upton [57]. Combining work by Berberkic [16] and Upton [57] could produce a hybrid TM-based and sigma delta ADC and further enhance the performance of both architectures.

Table 2-6: Potential advantages of TM-based ADCs over mainstream ADC architectures.

2.5 Estimating Initial Conditions of Tent Maps with Non-ideal Gain

A significant portion of the literature that discusses initial condition estimation of TMs concerns determining the chaotic signals produced by TMs when superimposed with noise (a problem for chaotic communication systems) [100-102]. Although noise is an issue for TM-based ADCs [103] these proposed techniques were aimed at different variations of the TM function than the one presented in (1-2)) [100-102].

Some literature presented methods estimating the initial conditions of TM-based ADCs when the partition points of the TM circuits employed were displaced [34, 96, 104]. However, the focus of this research was for non-ideal μ compensation only.

Past research has been undertaken to estimate the initial conditions of the TM function given in (1-2), but methods considering non-ideal μ are sparse and limited in practise. For example, one method proposed by Xi et al. [105], estimated the initial conditions using the sawtooth and Bernoulli map functions on the Gray code output. This approach is simple to implement as a digital system, but unsuitable for compensating non-ideal μ, as the method assumed an ideal TM function with a $\mu = 2$ was being employed [105].

An accurate estimation method, which estimated the initial conditions of a symmetrical TM-based ADC, with non-ideal μ, was proposed by Basu [41, 42]. This method calculated the effective difference between the voltage levels for ideal and non-ideal μ that each ADC output bit represented. However, this approach was complex and challenging to implement as a digital system, as the algorithm involves calculating compensation values by using multiplication and division for n iterations (n is R - 1) and required more computational overhead than the previous example, which only required Boolean logic [105]. Also, this method was only implemented in the mathematical software, MATLAB, therefore a batch of

digital data had to be acquired from the TM-based ADC, then run through the algorithm separately in order to estimate the initial input signal [41, 42].

2.6 Summary

This chapter discussed key parameters available when assessing the ADC output accuracy and detailed current COTS ADC architectures employed in higher resolution data conversion. The underpinning theory on chaos and TMs was also presented, followed by literature reviews into TM-based ADCs, and estimating the initial conditions of TM-based systems, respectively.

The purpose of this research was to assess how a μ compensation algorithm (μCA) embedded within the TM-based ADC improved the output accuracy. For this reason, mainly static and dynamic parameters concerning the assessment of ADC output accuracy were determined when analysing the employed TM-based ADC design (see Chapters 5 and 6).

Comparisons of research into COTS ADC architectures with resolutions greater than 16.5 bits and TM-based ADCs highlighted that the latter could achieve better performance in certain categories or enhance the performance of mainstream architectures. For example, an ideal TM-based ADC (one with a $\mu = 2$) can achieve a more linear output response across the whole valid input signal range [16] when compared to a SAR ADC, which is unsuited for low level signal amplitudes [27]. TM-based ADCs have also achieved faster sampling and conversion speeds than integrating ADCs [27, 56, 57]. Merging the architectures of a TM-based ADC and a sigma delta ADC could also produce a faster, high resolution data converter.

This work focussed on developing a TM-based ADC with an embedded μ compensation system (μCS), comprising a novel μCA, for the UMS application detailed in Section 1.5. This

application required the ADC to: detect 100 µV signal variations over a voltage range of 0 - 1.8 V; sample at a minimum rate of 20 MHz; and be relatively cheap to prototype.

The TM-based ADC designs proposed by Berberkic and Litovski et al. [16, 96], in theory, were capable of detecting a change of 100 µV, but only the designs by Berberkic were proven, through practical implementation, to have achieved this criterion [16]. However, the TM-based ADC designs by Berberkic digitised the difference between consecutive samples, rather than provide a digital representation of the absolute value of the analogue sample [16]. Both of the TM-based ADC designs by Berberkic could be rectified by employing a front-end comparator to establish the MSB of the digital output [16].

Only Upton [56, 57] was found to have achieved a sampling rate greater than 20 MHz, but the proposed design only detected 11 mV signal variations over a 0 - 3 V range. By increasing the resolution of this design to at least 15 bits (this resolution was established using (1-8)), the signal variation size which could be detected will reduce to a theoretical value of 100 µV.

Smith, Berberkic, Upton and a group of cross-institute researchers in Thailand, proposed TM-based ADC designs constructed from discrete components, making them cheap to prototype and develop [16, 56, 57, 91, 93-95]. The remaining TM-based ADCs discussed in Section 2.4.2 were designed to be constructed from area saving, switch-capacitor circuitry, and thus needed to be produced using the expensive process of fabrication [11].

The underlying TM-based ADC design for this work is an adaptation of the design by Upton [56, 57]. This design was chosen due to: meeting the sampling rate requirement; being low cost to prototype; and having the potential to increase the resolution to meet the requirement of detecting 100 µV variations.

A later TM-based ADC design employed techniques used by Berberkic [16] to achieve sufficiently high resolution and maintain the sampling rate achieved by Upton [56, 57], whilst reducing the trade-off in conversion speed. The comparator on the final TM stage output was replaced by a 10-12 bit COTS ADC with a sampling rate employed by the TM-based ADC. This reduced the number of TM stages needed to achieve the required resolution, which in turn increased the conversion rate. This TM-based ADC design achieves higher resolution than the designs proposed by Upton [56, 57] and Liu et al. [13], while matching the highest sampling and conversion speeds (when compared to the other TM-based ADCs found in literature). The design is also less likely to be affected by frequency multiplication (caused by the TM circuits folding the signals) which improves the acceptable bandwidth range [99].

The operation of the fundamental μCA assessed during this work requires sufficiently lower computational resources, than the one proposed by Basu [41, 42], to enable embedment within the FPGA, coordinating the operation of the TM-based ADC. This enabled the compensation to be performed on the ADC digital output data prior transmission.

Enhancements were made to the fundamental μCA in this work to enable compensation for non-matching TM stage μ within a series configuration, as well as when the TM stage was digitised using a multi-bit sub-ranging ADC. This makes the final μCA suited to a wider range of TM-based ADC configurations, including series configurations. This was because another limitation of the μCA developed by Basu (as well as the fundamental μCA in this work) was being only suited for a feedback TM-based ADC configuration that employed a comparator to perform the data conversion. This was due to the μCA designed by Basu, requiring a constant TM stage μ and for the output of each iteration to be digitised to one bit only.

The next chapter presents, and describes the operation of, the underlying TM-based ADC structure and the fundamental µCA (which forms the basis of the embedded µCS) for the research project. An overview of the adaptations made to both the TM-based ADC structure and µCA are also given.

3 Proposed Tent Map Based ADC Structures and Gain Compensation Algorithms

Some of the material in this chapter was previously published in the journal paper [2].

This chapter presents the proposed TM-based ADC structures and the μ compensation algorithm (μCA) developed to form the basis of a μ compensation system (μCS) embedded within the ADC. The fundamental concept of the μCA was provided by Dr Peter Mather (research supervisor). Section 3.1 first presents and details the operation of the TM-based ADC designs employed in this work and Section 3.2 details the operation and the development of the μCA.

The underlying, fundamental TM-based ADC circuitry was based on the design proposed by Upton [56, 57]. An adapted version of this TM-based ADC design employed techniques used by Berberkic [16] to achieve sufficiently high resolution and to maintain the sampling rate achieved by Upton [56, 57], whilst reducing the trade-off in conversion speed. For ease of reading, these two TM-based ADC structures will be distinguished as TM-ARCHα and TM-ARCHβ respectively.

The fundamental (core) μCA was developed to analysis the TM-based ADC Gray-code output and establish the value which needed to be added or subtracted from the equivalent binary code, in order to compensate for non-ideal μ within the ADC. Enhancements to the fundamental μCA were then produced to make the μCA suitable for different TM-based ADC configurations and to be able to compensate for additional, non-ideal behaviour in the electronic implementation of the TM circuits. The different variations of the μCAs will be numbered to aid readability.

Figure 3-1 comprises a block diagram, showing how the µCA is embedded within the TM-based ADC, forming a µCS. This diagram also highlights how the two systems work together with the ADC Gray code (and binary equivalent) output being passed to the µCS. The µCS analyses the Gray code digital data and applies the relevant compensation to the equivalent binary code. The compensated binary code data is then transmitted back to the ADC control logic, which then outputs the compensated data.

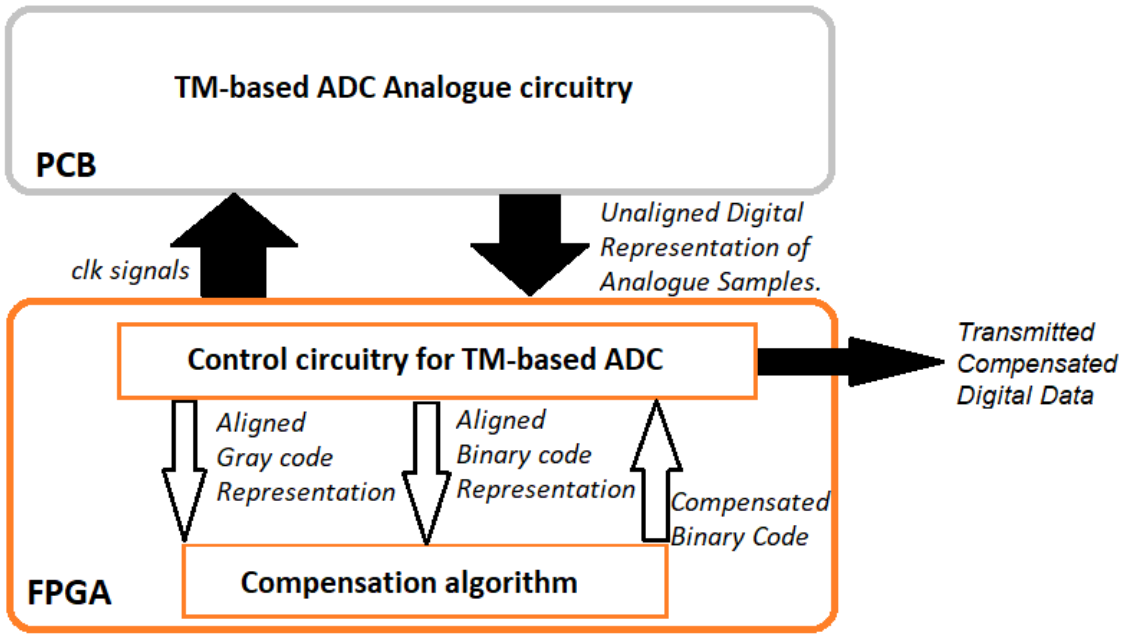


Figure 3-1: A more detailed block diagram of how the TM-based ADC and µ compensation algorithm were integrated.

3.1 Tent Map Based ADC Structures

3.1.1 Underlying TM-based ADC Structure

The fundamental µCA was developed to determine the amount of compensation required from the Gray code output of a TM-based ADC. This required the TM-based ADC to produce a Gray code output. Figure 3-2 presents the underlying TM-based ADC structure, employed to assess the fundamental µCA, which was based on the design proposed by Upton [56, 57].

The ADC consists of an initial sample and hold circuit, which samples the input analogue voltage signal. This signal is then fed to a comparator to determine the MSB, and to a TM circuit whilst the next sample is acquired. The output from the first (and subsequent) TM circuits are fed to the next comparator and TM stage. This is repeated until the output signal of the nth TM stage is terminated by a final standalone comparator.

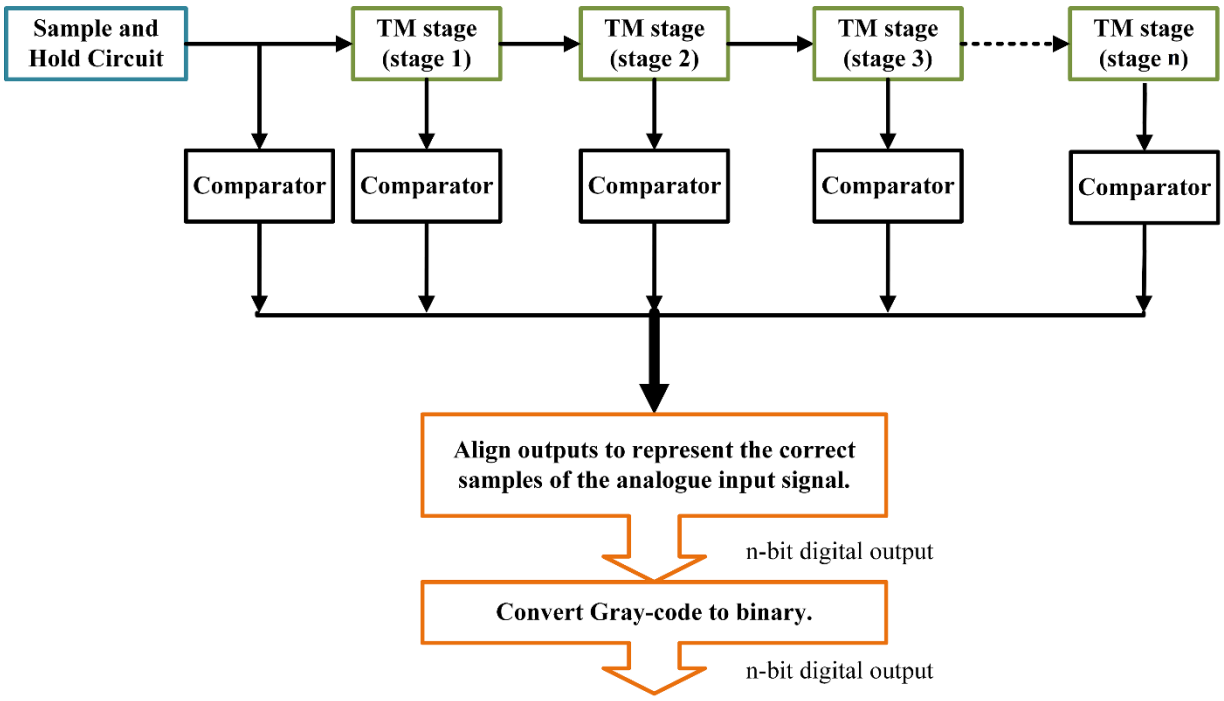


Figure 3-2: Proposed underlying TM-based ADC structure (to be referred to as TM-ARCHα-n). Based on [56, 57].

The comparator outputs are acquired by an FPGA for alignment, to represent the correct analogue sample. The symmetrical folding in the TM-based ADC results in the comparator output states producing Gray code, so the FPGA also converts the aligned digital data to binary code.

To avoid confusion with the adapted TM-based ADC structure discussed in the following sub-section, this TM-based ADC from this point on will be referred to as TM-ARCHα-n, where n refers to the number of TM stages. Section 4.1.1 in Chapter 4 further details the operation and circuit implementation of the key TM-ARCHα-n ADC stages.

3.1.2 Adapted TM-based ADC Structure

An adapted TM-based ADC structure was proposed to assess the viability of achieving sufficiently high resolution and of maintaining the sampling rate achieved by Upton [56, 57], whilst reducing the trade-off in conversion speed. This involved replacing the comparator on the seventh (final) TM stage output with a 10–12-bit COTS ADC with the same sampling rate as the TM-based ADC. This removed the need of an additional 10-12 TM stages and 11-13 comparators to achieve the same resolution, which in turn reduced the conversion rate. This TM-based ADC design (to be referred to as TM-ARCHβ-n-$R_{sub\text{-}ranging}$, where n refers to the number of TM stages, whilst $R_{sub\text{-}ranging}$ notates the resolution of the COTS ADC) was an adaptation of the design proposed by Upton [56, 57], and employed techniques used by Berberkic [16]. The structure is presented in Figure 3-3.

The TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC was designed to employ seven comparators and TM stages, while a 10-12-bit COTS ADC was used to digitise the final TM output. The COTS ADC produced

a binary code representation of the final TM output, whilst the 7 MSBs of the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC were represented in Gray code.

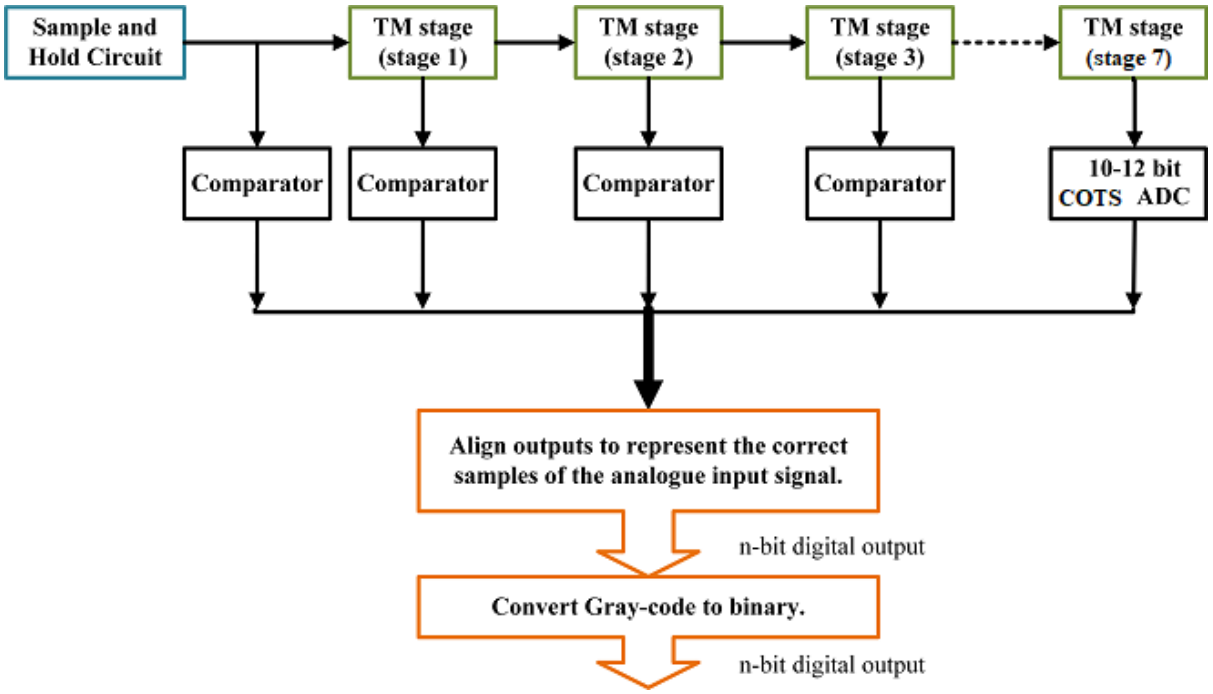


Figure 3-3: Proposed adapted TM-based ADC structure (to be referred to as TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC). Based on [56, 57] and [16].

3.2 The Tent Map Gain Compensation Algorithms (μCAs)

3.2.1 Fundamental μCA

Figure 3-4 presents a flow-diagram describing the operation of the fundamental μCA (to be referred to as μCA-1) being assessed and refined. The fundamental concept of μCA-1 was provided by Dr Peter Mather (research supervisor). This algorithm was developed to take the Gray code output of a TM-ARCHα-n ADC to establish the compensation value to be added/subtracted from the binary code equivalent of the ADC output. This μCA formed the basis of the μCS embedded within the same FPGA that was used to control the operation of the TM-ARCHα-n ADC.

The μCA was designed to compensate for non-ideal μ within a chaotic TM-based ADC, which requires the μ of the TM functions to be > 1 and ≤ 2, as proven in Section 2.3. A system must simultaneously be deterministic, bounded, aperiodic and sensitive to initial conditions to be classed as chaotic. While the sensitivity to initial conditions characteristic of a chaotic TM-based ADC is responsible for the noticeable loss in output accuracy with higher resolution designs when the μ is a non-ideal value; having a TM-based ADC that produces chaotic behaviour also enables the loss in output accuracy to be compensated for. This is because the chaotic TM-based ADC will have a bounded and deterministic output, and these two characteristics enable the initial conditions to be estimated and the digital output to be compensated for.

The μCA-1 has 4 key stages (as summarised in Figure 3-4). The first stage employs the Gray code output to determine whether the compensation values for each bit (the difference measure (referred to as DM)) should be added or subtracted. This sign for difference measure

(SDM) establishes the direction of the difference between the ideal output and that produced due to the non-ideal gain for each TM stages.

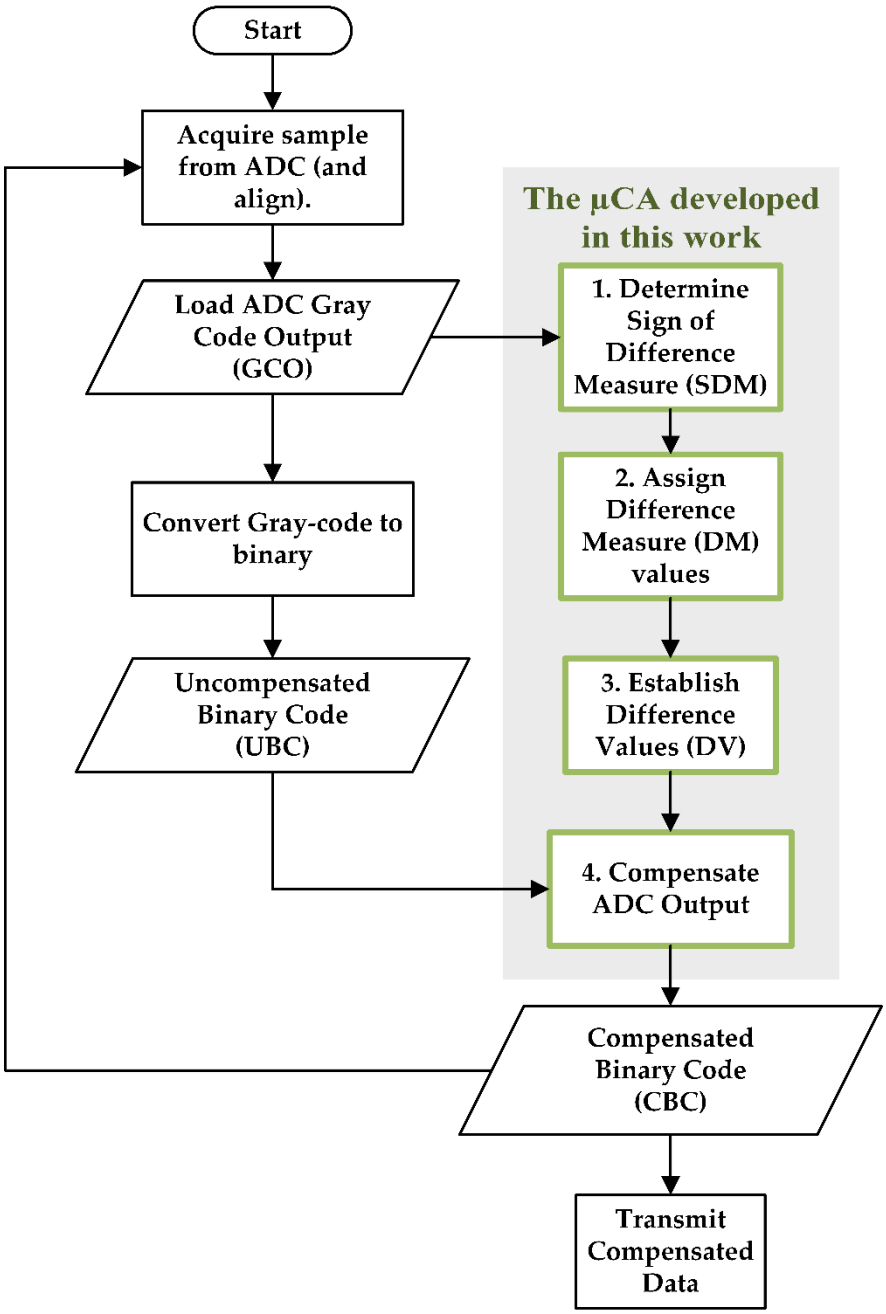


Figure 3-4: A flowchart giving an overview of µCA-1.

The MSB of the Gray code output is produced from the signal that has not been processed by the TM function circuit, and thus requires no compensation. The SDM for the second MSB will always be configured so the DM value will be added to the equivalent bit weighting, because if $\mu < 2$, the output signal of the first TM stage will be lower than the ideal (as illustrated by x_1 in Figure 3-5). With subsequent bits, the DM value will only be added if there are an odd number of 1's from the second MSB to the bit that is being analysed. If there are an even number of 1's, the SDM is configured such that the DM value is subtracted from the equivalent bit weighting.

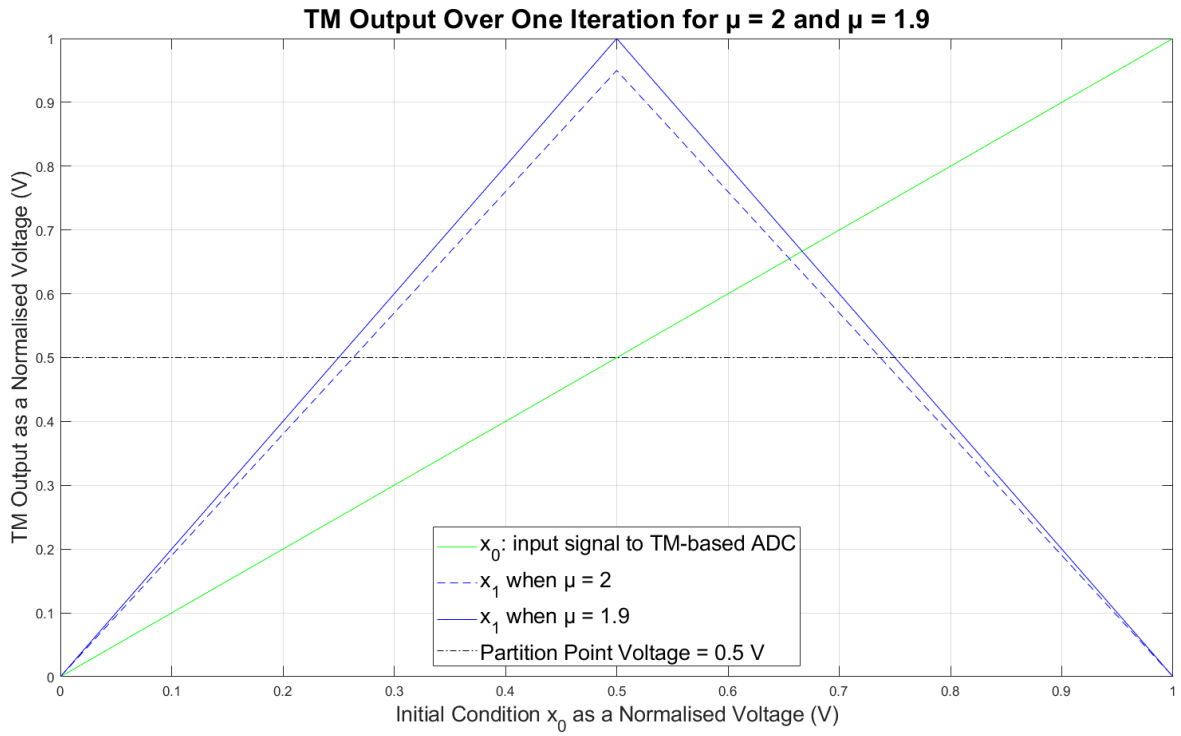


Figure 3-5: Input and output signals of the first TM stage of TM-ARCHα-n ADC when μ = 2 and μ = 1.9.

The second stage calculates the magnitude of the DM per bit (the difference between the ideal output and the output due to the non-ideal μ for each TM stage output). The ideal bit weighting of the comparator output is $\frac{g_n}{2^n}$, where n is the iteration and g is the polarity of the

TM output. The actual polarity of the last n bits of the digital word will be $\frac{g_n}{\mu^n}$ (the MSB is determined before the analogue signal goes through the first TM stage and thus is unaffected by non-ideal μ). Therefore, the DM can be determined as shown in (3-1). Figure 3-6 presents the DM versus μ plots for the second to fifth MSBs of a TM-based ADC output, of a μ range 1.9 to 2.

$$DM = \left(\frac{g_n}{\mu^n} - \frac{g_n}{2^n}\right) \tag{3-1}$$

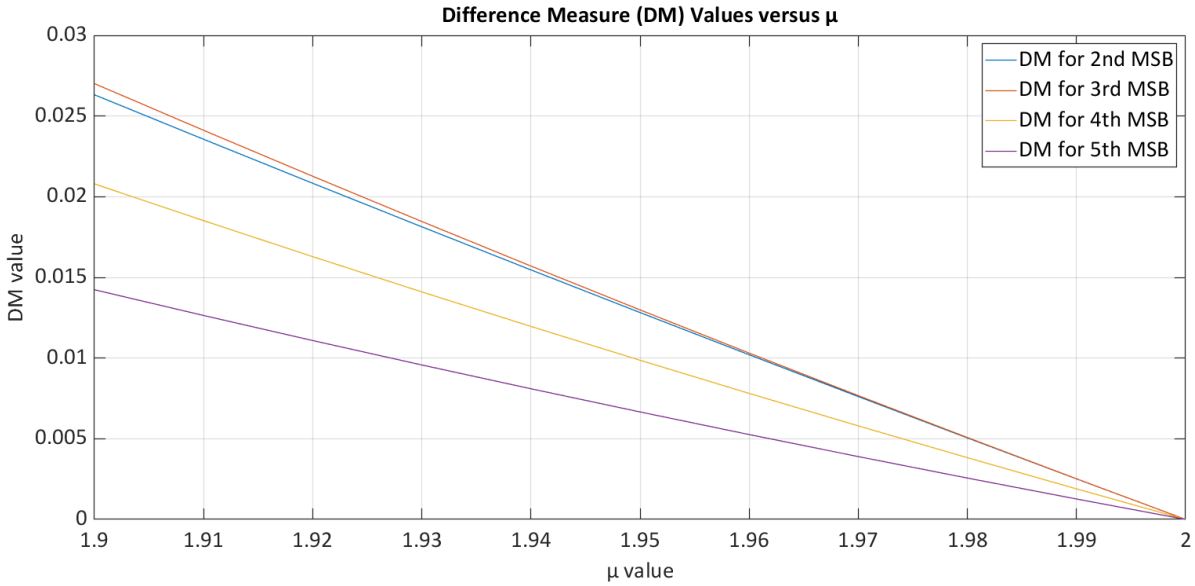


Figure 3-6: DM versus μ plots for the second to fifth MSBs of a TM-based ADC output.

Meanwhile the third stage computes the difference value (DV), which provides the overall magnitude and direction of the cumulative difference between the non-ideal μ of the TM based ADC output and the ideal. The magnitude of the DV (referred to as |DV|) is established as shown in (3-2), by multiplying the SDM with the DM for each bit (except the MSB) and summing the products together. The polarity of the DV value (represented as $DV_{polarity}$ in (3-2)) is then determined by the MSB of the TM-based ADC Gray code output. If the MSB = 0, the

DV value needs adding to the binary TM-based ADC output code, else subtracting if the MSB = 1.

$$DV = DV_{polarity} \times |DV| = DV_{polarity} \times (\sum_{i=1}^{n} SDM_i \times DM_i) \quad (3\text{-}2)$$

Finally, the DV value is applied to the binary code equivalent of the TM-based ADC output to compensate for non-ideal μ. Figure 3-5 highlights that x_1 crosses the partition point (Vref) with a higher x_0 value when μ = 1.9, than when μ = 2, whilst x_0 < Vref. This will result in most of the x_0 < Vref values being represented digitally by lower value digital codes when μ = 1.9, than if μ = 2. Meanwhile, when x_0 > Vref x_1 crosses the Vref partition point with a lower x_0 value when μ = 1.9, than when μ = 2. This means x_1, when μ = 1.9 and x_0 > Vref, will be higher than when μ=2, thus the majority of the x_0 > Vref values will be represented digitally by higher value digital codes.

On completion of a compensation cycle the μCA-1 passes the compensated binary code back to the control logic within the FPGA coordinating the operation of the TM-ARCHα-n ADC. The compensation process is then repeated for the next digital value. Figure 3-7 provides a more detailed overview of the operation of the μCA-1, and a MATLAB code listing of the μCA is provided in Appendix B.2.1. Section 4.2.1 of Chapter 4 discusses the operation of the key stages of the μCA-1, as well as the implementation.

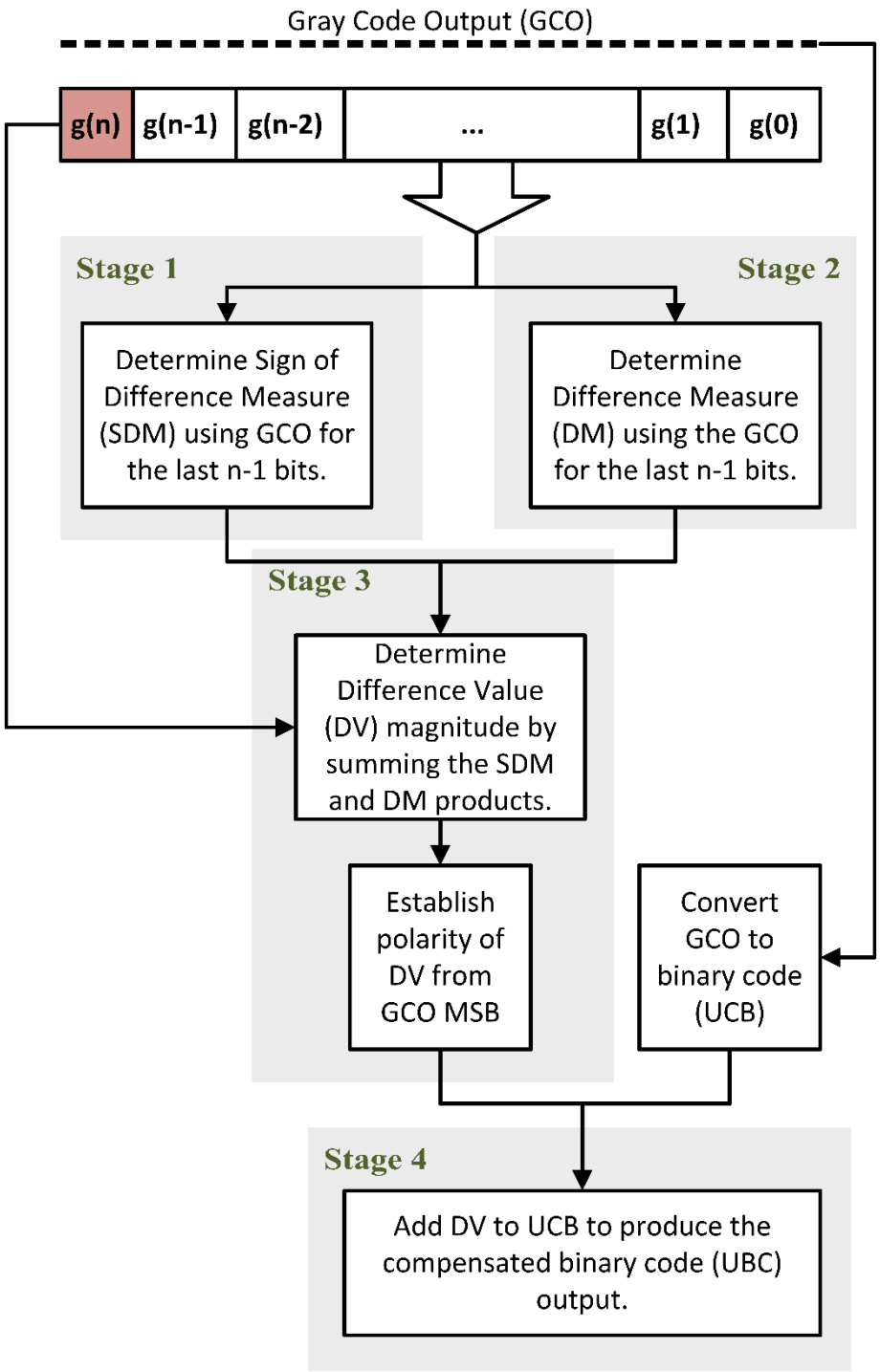


Figure 3-7: Diagram providing overview of µCA-1 operation.

3.2.2 Enhancements to the Fundamental µCA

An assumption was made for µCA-1 that the µ of each TM stage were identical and the µ of the two difference equations forming the TM were matching (see (1-2)). Although simple to implement through simulation software, in a practical implementation of a series TM-based ADC configuration (as presented in Section 3.1.1), achieving identical µ for each TM stage was impossible due to component tolerances (this is discussed in Section 4.1). This meant µCA-1 was only suitable for a feedback TM-based ADC configuration, as presented in Figure 2-12 in Section 2.4, because the µ for the single TM stage remains constant for each iteration.

Further enhancements were made to the µCA-1 adaptions to realise an embedded compensation system, for a practical TM-based ADC, for non-matching µ, within the TM-based ADC, and within each TM stage. This enhanced version of the µCA-1 will be referred to as µCA-2.

Additional adaptions were also required in order to enable the µCA to cope with the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC. The µCA-1 and µCA-2 were only suitable for TM-based ADC structures, like the TM-ARCHα-n ADC, which employ TM circuits and single bit producing comparators that generate the digital output data in Gray code format. Therefore, the algorithm required further adaptation to accommodate the binary code digital output produced by the multibit, sub-ranging COTS ADC in the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC described in Section 3.1.2. The version of the µCA will be referred to as µCA-3.

Section 4.2.2 in Chapter 4 discusses the enhancements made to the µCA-1 to accommodate these limitations to produce µCA-2 and µCA-3. How these enhanced µCAs were implemented is also discussed.

3.3 Summary

This chapter presented the TM-based ADC structures and the μ compensation algorithm (μCA) developed to form the basis of a μ compensation system (μCS) embedded within the ADC. The underlying TM-based ADC structure (referred to as TM-ARCHα-n ADC, where n notates the number of TM stages) was based on that proposed by Upton [56, 57]. A further adaptation of the TM-based ADC design (referred to as the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC, where $R_{sub\text{-}ranging}$ was the resolution of the sub-ranging COTS ADC employed) employed techniques used by Berberkic [16]. The fundamental μCA was developed to employ the Gray code output from the TM-ARCHα-n ADC to establish the compensation to be applied to the equivalent binary output, and then was enhanced twice to cope with non-matching TM stage and slope μ, as well as the employment of a sub-ranging COTS ADC to convert a TM stage output.

The TM-ARCHα-n ADC structure consisted of a sample and hold circuit for acquiring regular samples of the input analogue signal, which was proceeded by a chain of cascaded comparators and TM circuits, to generate each bit of the digital Gray code output. The TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC was developed to achieve sufficiently high resolution, while maintaining the sampling rate of the TM-ARCHα-n ADC but reducing the conversion rate for the given resolution. The adaption involved converting the final TM stage output signal to the digital domain, using a multibit sub-ranging COTS ADC (with a resolution of $R_{sub\text{-}ranging}$), which allows the TM-based ADC to achieve higher resolutions using less TM and comparator stages, thus increasing the conversion speed. The digital outputs of both TM-based ADCs structures were acquired and aligned by an FPGA to generate the Gray code representations of the analogue samples. The FPGA then converted the Gray code digital data to binary code.

The fundamental μCA, μCA-1, consisted of four key stages and employed the Gray code output from the TM-ARCHα-n ADC to determine whether fixed values should be added or subtracted from each bit to compensate for non-ideal μ. μCA-2 was an enhanced version of μCA-1, which accommodated for non-identical TM stage μ and for μ of the two difference equations forming the TM not matching. Another enhanced version of the μCA, μCA-3, was developed to cope with the TM output being acquired by a sub-ranging COTS ADC, which digitised the signal in binary code.

The next chapter will discuss the implementation of the TM-ARCHα-n and TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADCs and provide additional details on how these two TM-based ADC structures operate. The implementation and more detailed discussions on the operation of the μCA-1, μCA-2 and μCA-3 will also be given

4 Tent Map Based ADC Structures and Gain Compensation Algorithms Implementation

Some of the material in this chapter was previously published in the journal paper [2].

This chapter discusses the implementation of the TM-ARCHα-n and TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADCs (terms α and β employed to distinguish between the two TM-based ADC structures) and provides additional details on how these two TM-based ADC structures operate. The implementation and more detailed discussion on the operation of the µCA-1, µCA-2 and µCA-3 (numbered to distinguish the different µCA variations and aid readability) are also given. The fundamental concept of µCA-1 was provided by Dr Peter Mather (research supervisor).

Section 4.1 details how the TM-ARCHα-n ADC (introduced in Chapter 3) was implemented in order to perform analysis on the ADC output, with, and without, the µCA developed within this work. Descriptions of the different TM-ARCHα-n ADC versions produced for this work are also given. Then, the implementation of the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC structure, proposed to assess the viability of achieving sufficiently high resolution, while maintaining the sampling rate of the TM-ARCHα-n ADC, and reducing the conversion rate for the given resolution, by employing sub-ranging COTS ADCs to convert the TM stage outputs, are discussed.

Section 4.2 covers the implementation of the fundamental µCA, µCA-1, along with the two enhanced variations µCA-2 and µCA-3. The µCA-2 compensated for non-identical TM stage µ, as well as for the µ of the two difference equations forming the TM not matching each other. The µCA-3 also accommodates for the final TM stage being digitised using a multibit sub-ranging ADC rather than a single digital bit producing comparator.

4.1 Tent Map Based ADC Structures

4.1.1 TM-ARCHα-n ADC

Figure 4-1 presents a more detailed block diagram of the TM-ARCHα-n which comprises a sample and hold circuit, n TM circuit stages, n+1 comparators and an FPGA. The sample and hold circuit is capable of sampling at a rate of 25 MHz. The acquired samples pass through a comparator and TM circuit which determines the first bit of the digital output, as well as applying the TM function to the sampled signal. The output of the first TM stage then passes through further comparators and TM stages n-1 times.

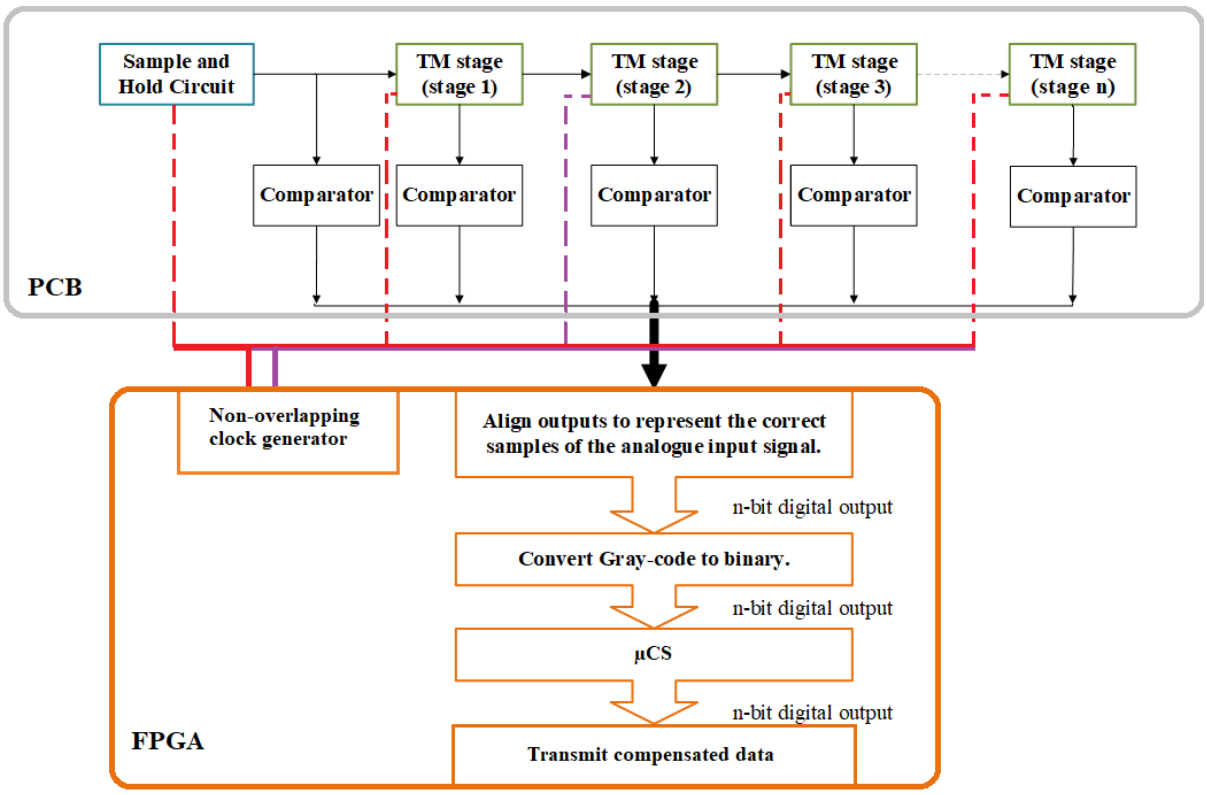


Figure 4-1: More detailed block diagram of the TM-ARCHα-n ADC. Based on [57].

Initially a 16-bit mathematical model of the TM-ARCHα-n (TM-ARCHα-15) ADC was developed in MATLAB (developed script given in B.1.1). The purpose of the model was to assess whether

the μCA-1 could compensate for non-ideal μ and that the μCA-2 could compensate for non-identical TM stage μ, and for the μ of the two difference equations forming the TM not matching. If electronic circuit simulation software had been employed to develop the TM-ARCHα-15 model, the outcome would have taken into consideration non-ideal characteristics of the individual components (e.g., resistors, op-amps etc.). This would have produced a complex model that considered non-ideal behaviour not relating to the μ of the TM stages and would have made assessing the effectiveness of the μCAs challenging. Employing MATLAB, a theoretical model was developed, which could later be edited to reflect additional characteristics associated with the practical implementation, once the effectiveness of the μCA at compensating for non-ideal μ had been proven.

An 8-bit electronic implementation of the TM-ARCHα-n ADC (TM-ARCHα-7) was also developed where a PCB was designed that comprised the analogue to digital conversion circuitry, which was constructed from discrete, COTS components, and connected to an FPGA development board. The FPGA development board was employed to supply the clock signals needed to drive the circuit, as well as to acquire, align and convert the digital outputs from the analogue to digital circuitry (the VHDL code to achieve this was developed by Richard Haigh [106]). Figure 4-2 and Figure 4-3 presents the schematic for the sample and hold circuit and the TM (folding) circuits. The full schematic and list of components for the TM-ARCHα-7 ADC are in Appendix A.1, whilst and VHDL code is presented in Appendix B.5.1.

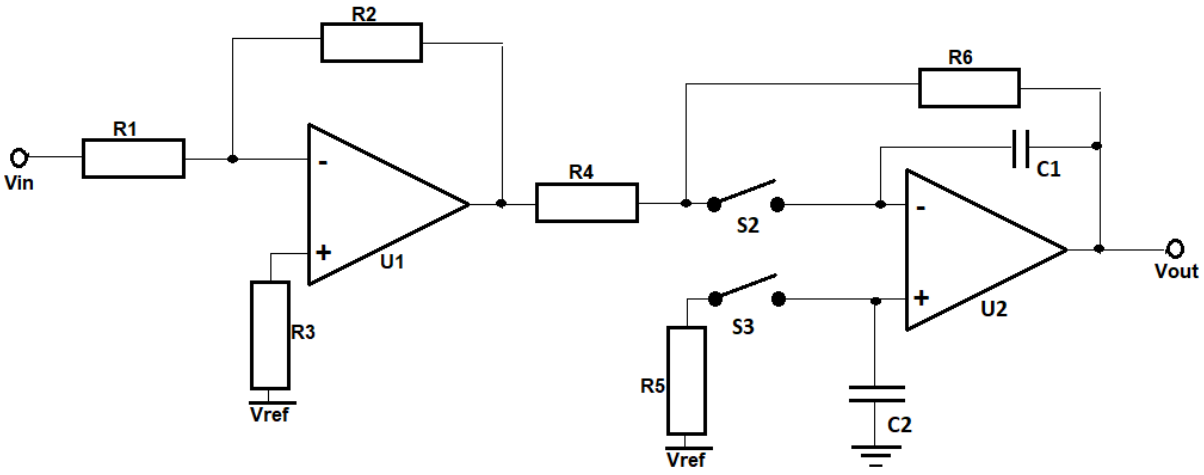


Figure 4-2: Schematic for the sample and hold circuit. Reproduced from [57].

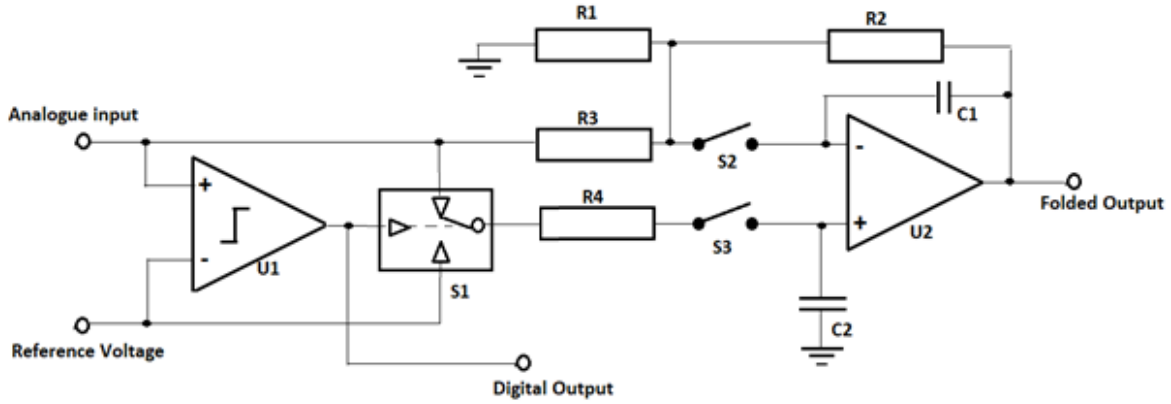


Figure 4-3: TM circuit employed. Reproduced from [57].

The sample and hold circuit (Figure 4-2) consists of two cascaded inverting amplifiers which both offset the output signal by a set voltage (this being the partition point voltage, Vref, which is employed by the TM circuits). The second of these amplifiers also samples the two input signals to the op-amp.

The TM circuit (Figure 4-3) functions by identifying whether the input is greater than the partition point voltage. When Vin < Vref, the sample signal is amplified using a non-inverting

op-amp. If Vin > Vref, Vref is included in the inverting amplification process. (4-1) summarises how the input and reference signals are amplified [57].

$$Vout = \begin{cases} \left(1 + \frac{R2}{R1}\right) Vin, & Vin \leq Vref \\ \left(1 + \frac{R2}{R1}\right) Vref - \frac{R2}{R3}(Vin - Vref), & Vin > Vref \end{cases} \quad (4\text{-}1)$$

Where Vin equates to the "Analogue Input" in Figure 4-3, Vref is the "Reference Voltage" and Vout is the "Folded Output". In (4-1), $\left(1 + \frac{R2}{R1}\right) = \frac{R2}{R3}$, which makes the format of this equation identical to that of the TM function shown in (1-2) [57].

Two key modifications were made to the TM circuit design by Upton for this research project [56, 57]. The first involved placing external hysteresis around the comparators (the upper and lower limits of the hysteresis voltages chosen were Vref ± half the step size voltage). This modification was needed due to the switching noise induced into the analogue TM circuit signals, to and from the TM circuits, causing the comparator outputs to oscillate.

The second modification added resistors in series with R1, R2 and R3 so the gain of each stage was changeable. R1, R2, R3 and R4 were produced using resistor arrays, yielding gains close to, and possibly over, the ideal of 2. The µCA was developed for non-ideal µ values less than 2 (but greater than 1) only, as this resulted in a chaotic TM-based ADC. Therefore, resistors needed to be added to decrease the gain to prove that the algorithm could compensate for the inevitable non-ideal gain, associated with a practical implementation of the TM-based ADC. (4-2) presents the updated equation (adapted from (4-1) [57]).

$$Vout = \begin{cases} \left(1 + \frac{R2 + \Delta R2}{R1 + \Delta R1}\right) Vin, & Vin \leq Vref \\ \left(1 + \frac{R2 + \Delta R2}{R1 + \Delta R1}\right) Vref - \frac{R2 + \Delta R2}{R3 + \Delta R3}(Vin - Vref), & Vin > Vref \end{cases} \quad (4\text{-}2)$$

Where ΔR1, ΔR2 and ΔR3 represent the resistors that were added in series with R1, R2 and R3 respectively. Figure 4-4 presents the updated schematic of the TM circuit.

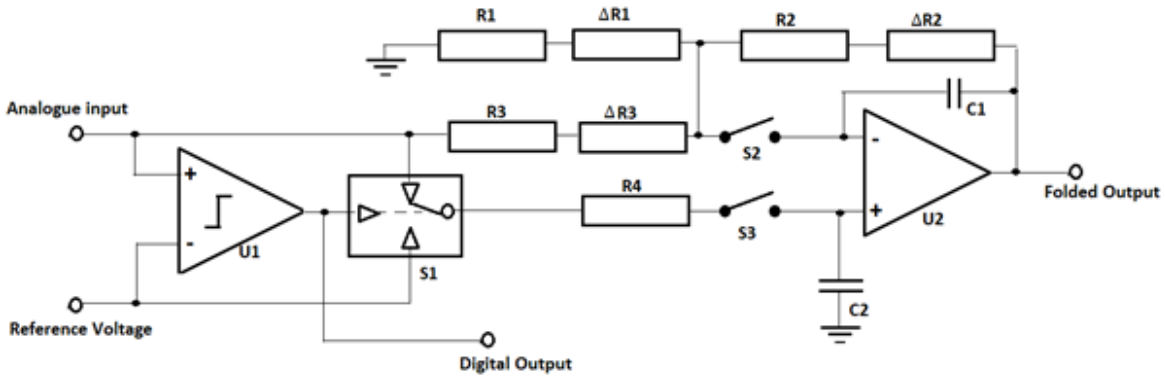


Figure 4-4: TM circuit employed, which has additional resistors to alter μ+ and μ-. Adapted from [57].

There was an assumption with the circuit implemented by Upton that the gains created by R1, R2 and R3 were identical (i.e., $\left(1 + \frac{R2}{R1}\right) = \frac{R2}{R3}$) [57]. Although simple to implement through simulation software, achieving identical gains in practical circuits is impossible due to component tolerances. Therefore, not only will the μ of each TM stage not be identical, but the μ values employed in the two difference equations of the TM function will not match either. These additional non-ideal characteristics will further affect the output accuracy of the TM-ARCHα-7 ADC.

To notate the different gains (to be referred to as slope gains), in the TM equation, (1-2) was modified to reflect the electronic circuit TM implementation more closely. (4-3) presents the modified TM function (modified from (1-2) [19]).

$$x_{n+1} = \begin{cases} \mu_{n+} x_n \\ \mu_{n+} V_{ref} - \mu_{n-}(x_n - V_{ref}) \end{cases} \text{ when } \begin{matrix} x_n \leq 0.5 \\ x_n > 0.5 \end{matrix} \quad (4\text{-}3)$$

Where μ_+ and μ_- produces the positive and negative slopes respectively, and n represents the TM stage.

4.1.2 TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC

A more detailed block diagram of the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC is presented in Figure 4-5. This structure is identical to the one presented in Figure 4-1, except the final TM stage output passes through the COTS ADC to determine the last 10-12 bits of the digital representation of the sampled analogue input signal. The comparator outputs represent the first 7 bits of the digital word in Gray code, while the last 10-12 bits are represented in binary code.

The TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC structure was developed as a 7 TM stage with a 12-bit sub-ranging COTS ADC (a TM-ARCHβ-7-12 ADC) and analysed as a mathematical model in MATLAB (developed script shown in C.3). The practical PCB implementation of the TM-ARCHα-7 ADC structure discussed in Section 4.1.1 was also designed, (although not implemented) so the final TM stage could be connected to a breakout board containing a THS1030 10-bit ADC [107]. Appendix A.2 presents the schematic and list of components of the THS1030 10-bit ADC breakout board, although due to challenges faced when testing the practical implementation of the TM-ARCHα-7 ADC (discussed in Section 6.6), the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ was never tested as a practical implementation. As an electronic implementation of this TM-based ADC structure was not produced, the VHDL code was not amended to handle the modification.

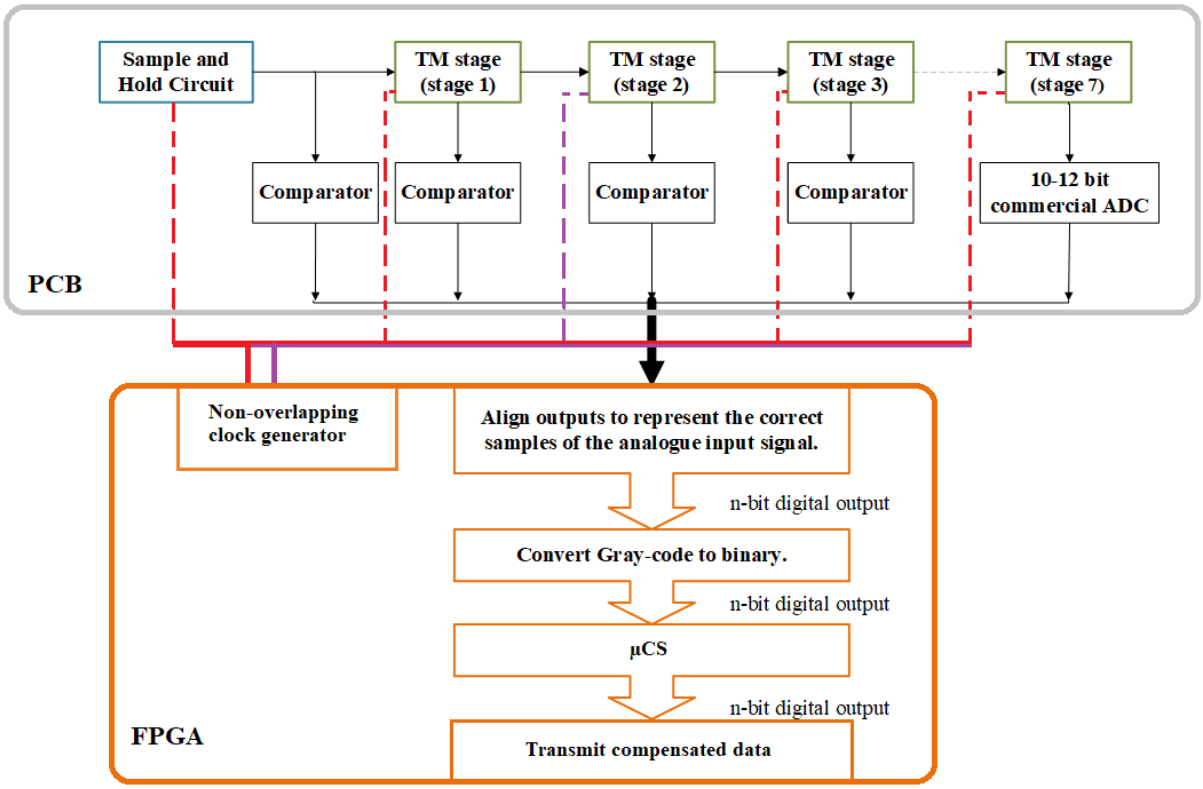


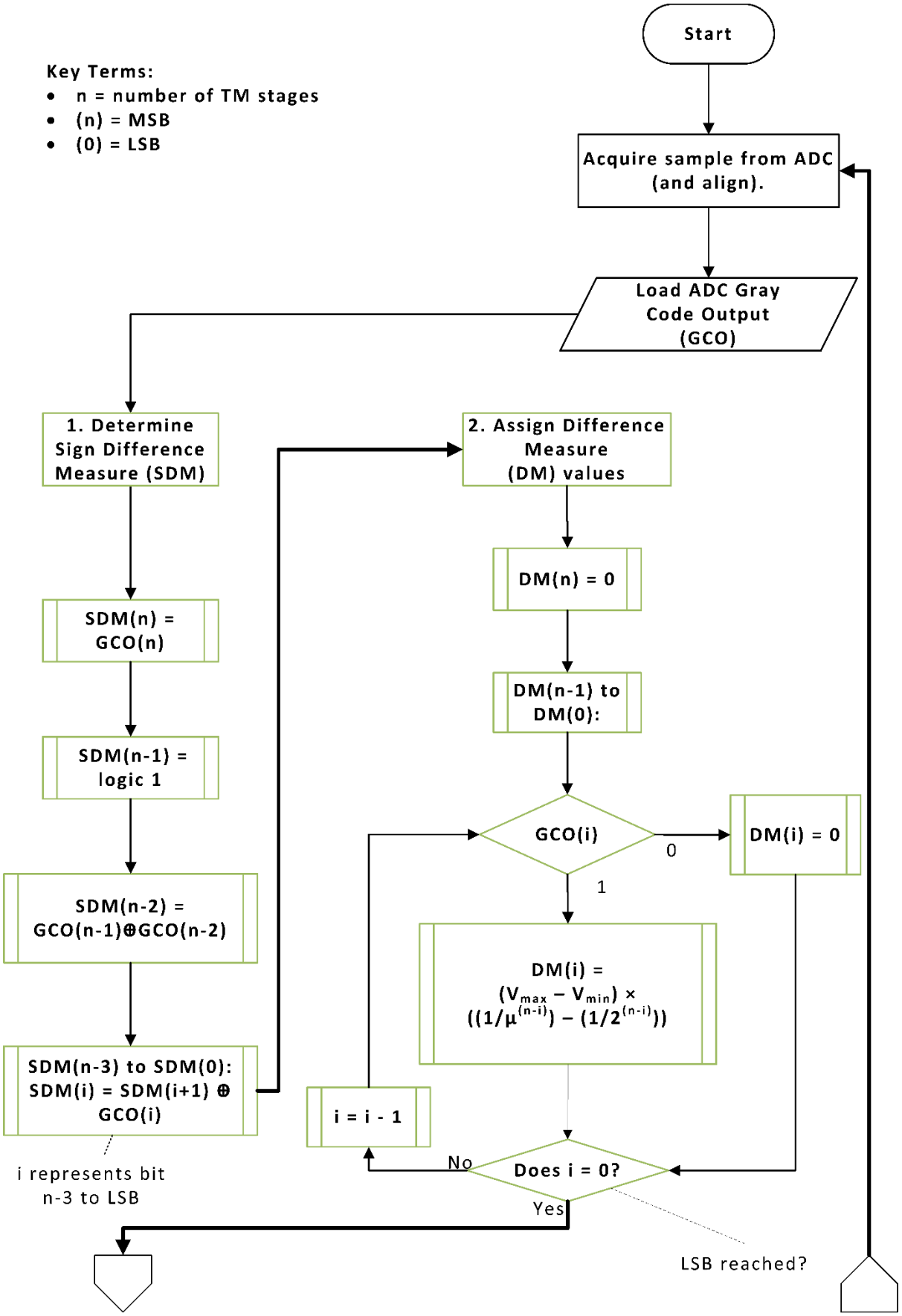
Figure 4-5: TM-ARCHϐ-n-$R_{sub\text{-}ranging}$ ADC structure. Based on [57] and [16].

4.2 The Tent Map Gain Compensation Algorithms

4.2.1 The µCA-1

Figure 4-6 presents a more detailed overview of the µCA-1 presented in Section 3.2, the fundamental concept of which was provided by Dr Peter Mather (research supervisor). This algorithm required the partition point voltage to be halfway between the valid ADC input range, and for the µ to be determined, and be less than 2. There was also an assumption that the TM circuits had matching stage and slope gains.

The first part of the algorithm determines the sign of difference measure (SDM), which in turn determines whether the precalculated compensation values (DM values) are added or subtracted to each bit of the TM-based ADC output, using the Gray code ADC output. The difference measure (DM) values are the equivalent deviation from the correct weighting for each bit. The SDM values are represented as digital logic, where a '1' represents addition and a '0' subtraction, whilst the DM values are fractional values (represented fixed point binary values when the µCA-1 was implemented as a digital µCS system).


Key Terms:
• n = number of TM stages
• (n) = MSB
• (0) = LSB
Start
Acquire sample from ADC (and align).
Load ADC Gray Code Output (GCO)
1. Determine Sign Difference Measure (SDM)
SDM(n) = GCO(n)
SDM(n-1) = logic 1
SDM(n-2) = GCO(n-1)⊕GCO(n-2)
SDM(n-3) to SDM(0): SDM(i) = SDM(i+1) ⊕ GCO(i)
i represents bit n-3 to LSB
2. Assign Difference Measure (DM) values
DM(n) = 0
DM(n-1) to DM(0):
GCO(i)
0
DM(i) = 0
1
DM(i) = (V_max − V_min) × ((1/μ^(n-i)) − (1/2^(n-i)))
i = i - 1
No
Does i = 0?
Yes
LSB reached?

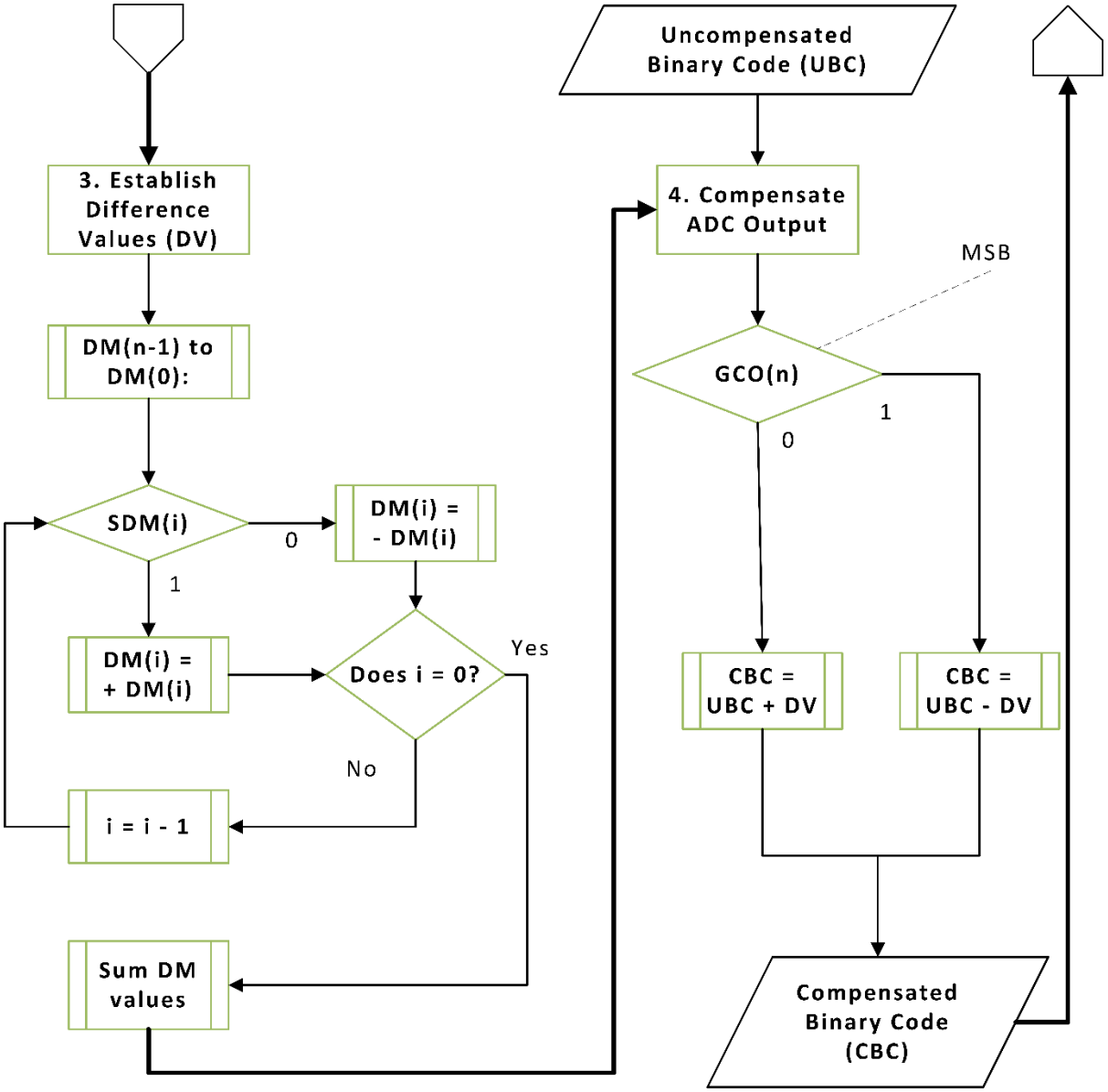


Figure 4-6: A more detailed flowchart of the µCA-1.

The MSB is established before the input signal goes through the first TM stage and so requires no compensation, but the polarity of this bit establishes whether the final difference value (DV) should be added or subtracted from the uncompensated binary code. The SDM representing the second MSB will always be positive (logic high) as µ will be less than 2, resulting in a DM value always being added to compensate for the equivalent weighting of this bit. The SDM for bit 3 is dependent on whether the current TM output was produced

using the same TM equation as the previous bit and is derived from the XOR operation on the Gray code output for bits 2 and 3. For the remaining bits the SDM is determined from XORing the previous SDM bit with the current Gray code bit. If the current Gray code bit and previous Gray code (or SDM) bit match, then the TM difference equation employed for the next TM stage will be the same as the current TM stage, and the DM value for that bit is subtracted. If the opposite is true, the following TM stage will use a different difference equation to the current TM stage, and the DM value will be added. The code listing in Figure 4-7 (which relates to stage 1 of the µCA presented in Figure 4-6) demonstrates how the SDM calculations were implemented in MATLAB.

```
%% Sign for Difference Measure (SDM)
for i = 1: 1: length(y) %Samples of input signal
    SDM(1, i) = Dout(g, 1,i); %MSB of Gray code output
    SDM(2, i) = 1;          %1 shows adding function
    if xor(Dout(g,2,i), Dout(g,3, i)) % find 3rd bit of SDM
        SDM(3,i) = 1;
    else
        SDM(3,i) = 0;
    end
    for res = 4: 1: resolution   % gives remaining bits of SDM
        if xor(SDM(res-1,i), Dout(g, res, i))
            SDM(res,i) = 1;
        else
            SDM(res,i) = 0;
        end
    end
end
```

Figure 4-7: Code extract of stage 1 of µCA-1, which determines the SDM values.

The magnitude of each DM value is calculated in the second stage of the algorithm. Only bits set to 1 (excluding the MSB which does not require compensation) in the TM-based ADC Gray code output have DM values assigned to them. These DM values, which are the difference between the ideal and actual weighting of each bit, are calculated using (4-4).

$$DM = (V_{max} - V_{min}) \times (\frac{1}{\mu^{i-1}} - \frac{1}{2^{i-1}}) \quad (4\text{-}4)$$

Where the two extremes of the valid input voltage range are denoted by V_{max} and V_{min}, the TM gain by μ, and the TM-stage output being considered by i (where 1 is the MSB). The code listing in Figure 4-8 (which relates to stage 2 of the μCA presented in Figure 4-6) demonstrates how the SDM calculations were implemented in MATLAB.

```
for g = 1: 1: gain_size
    %% Ideal DM values - look up table
    VHDL_bits = resolution + 8;
    for i = 1:1:(resolution - 1)
        LUT_theory(i) = (1/mpower(gain(g), i))-(1/pow2(i)); %Calculate
difference value
    end

                                [...]

    %% Difference Measure: selected for each respective gray code bit

    for i = 1: 1: length(y) %Samples of input signal
        DV_theory(1,i) = 0; %Ideal as it hasn't passed through a TM
        for res = 2: 1: resolution   % gives remaining bits of DM
            if (Dout(g,res, i) > 0)
                DV_theory(res, i) = LUT_theory(res - 1);
            else
                DV_theory (res, i) = 0;
            end
        end
    end
```

Figure 4-8: Code extract of stage 2 of μCA-1, which determines the DM values.

The third stage of the μCA-1 sums the DM values (taking into consideration the respective polarity represented by the SDM) to calculate the DV, which needs to be added (or subtracted) from the binary code equivalent of the TM output. Stage four takes the ADC output (converted to binary code) and adds/subtracts the DV to compensate for the non-ideal μ during the data conversion process, depending on the polarity of the MSB of the ADC Gray code output. Figure 4-9 and Figure 4-10 are code extracts relating to stages 3 and 4 the μCA

presented in Figure 4-6 respectively, illustrating how both theses stages were implemented in MATLAB.

```
    %% Signed Difference Value

    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV_theory(res, i) = DV_theory(res, i);
            else
                SDV_theory(res, i) = -DV_theory(res, i);
            end
        end
    end

    % Determine DV
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum_theory(i) = sum(SDV_theory(:,i));
    end
```

Figure 4-9: Code extract of stage 3 of µCA-1, which determines the DV values.

```
    %% Implement correction
    %uncompensated output
    for i = 1: 1: length(y) % converting Gray-code representation of
samples, to binary
        gray_code_vector = Dout(g,:,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution                 %convert binary values
to the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep;
        end
        output_representation(g, i) = decimal_rep  ; %modify decimal
value so it lies within the input voltage range
    end
    voltage_representation(g, :) = output_representation(g, :)*(Vmax -
Vmin);
    %compensated ADC output
    for i = 1: 1: length(y) % compensate output
        if (SDM(1,i) == 1)
            corrected_output_theory(g,i) = output_representation(g,i) -
SDV_sum_theory(i);
        else
            corrected_output_theory(g,i) = output_representation(g,i) +
SDV_sum_theory(i);
        end
    end
```

Figure 4-10: Code extract of stage 4 of µCA-1, which shows how uncompensated ADC output is modified using the DV values.

To analyse the effectiveness of this method of compensating for non-ideal μ, this μCA was modelled in MATLAB (code listing is presented in B.2.1), and mostly tested using the data produced from the TM-ARCHα-15 ADC MATLAB model (one test with this μCA was repeated with a TM-ARCHα-7 ADC MATLAB model). Once the effectiveness of the algorithm had been established the algorithm was implemented in VHDL code (implementation is given in B.5.2). The VHDL code was used to configure an FPGA and produce a digital μCS to compensate each data word acquired from the TM-based ADC within one clock cycle. The DM values were precalculated, converted to binary code and stored as an array within the VHDL code, requiring less FPGA resources than repeatedly calculating the DM values. This HDL design was then validated using simulation, by employing digital data produced by an HDL module emulating a TM-ARCHα-7 ADC (code listing given in B.5.4).

4.2.2 The μCA-2 and μCA-3

Enhancements were made to the μCA-1, to cope with non-matching stage μ and non-matching μ_+ and μ_- ($\mu_\pm$) in each TM stage. This enhanced version of the μCA-1 will be referred to as μCA-2. A further enhancement was made to μCA-2, to be referred to as μCA-3, to accommodate the employment of a multibit, sub-ranging COTS ADC to acquire and digitise (in binary code format) a TM output.

To compensate for varying TM stage gain, the way which the DM values were calculated was altered. (4-5) presents the adapted function developed during this work.

$$DM_{mod1} = (V_{max} - V_{min}) \times ((\prod_{1}^{i} \frac{1}{\mu^{i-1}}) - \frac{1}{2^{i-1}}) \\ = (V_{max} - V_{min}) \times (GF - \frac{1}{2^{i-1}}) \quad (4\text{-}5)$$

Where *i* is the current "bit" of the data word and takes into consideration the previous μ_{stage} (represented by μ^{i-1}). The two extremes of the valid input voltage range for each TM stage are denoted by V_{max} and V_{min}. The new method employs the same equation to calculate the DM values as shown in (4-4), however $\frac{1}{\mu^{i-1}}$ is now replaced by a gain factor (GF). This gain factor is shown in (4-6).

$$GF = \prod_{1}^{i} \frac{1}{\mu^{i-1}} \quad (4\text{-}6)$$

A further adaption was made to the gain factor (GF_{mod}) to compensate for non-matching $\mu_{\pm}$. (4-7) presents this modification.

$$GF_{mod}(i) = \begin{cases} \frac{1}{\mu_{+}^{i-1}}, & g(0) = 0 \; AND \; i = 1 \\ \frac{1}{\mu_{-}^{i-1}}, & g(0) = 1 \; AND \; i = 1 \\ GF_{mod}(i-1) \times \frac{1}{\mu_{+}^{i-1}}, & g(i-1) = 0 \; AND \; i > 1 \\ GF_{mod}(i-1) \times \frac{1}{\mu_{-}^{i-1}}, & g(i-1) = 1 \; AND \; i > 1 \end{cases} \quad (4\text{-}7)$$

Where g(i-1) is the previous bit of the ADC Gray code output, μ_{+} is the rising slope gain and μ_{-} the negative slope gain. This results in the gain factor being produced from the relevant slope gains employed when determining the preceding bits of the Gray code output.

The final adapted function to calculate the DM values is presented in (4-8). This modification assumes the negative slope of the TM is produced by a single gain factor, as shown in (4-9) (modified from (1-2) [19]). However the actual TM circuit implementation employs both slope gains (see (4-3)).

$$DM_{mod2} = (V_{max} - V_{min})*((GF_{mod}(i)) - \frac{1}{2^{i-1}}) \quad (4\text{-}8)$$

$$x_{n+1} = \begin{cases} \mu_{n+} x_n & when \quad x_n \leq 0.5 \\ \mu_{n-} V_{ref} - \mu_{n-}(x_n - V_{ref}) & \quad x_n > 0.5 \end{cases} \quad (4\text{-}9)$$

To accommodate the multibit sub-ranging COTS ADC at the output of the final TM stage of the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC, the output of the COTS ADC needs converting to Gray code to match the 7 MSBs produced by the preceding comparator outputs. The compensation algorithm then treats the digital output produced by the COTS ADC as if the bits had continued to be produced by a series of comparators and TMs with a μ_{stage} of exactly 2.

The µCA-2 and µCA-3 were implemented in MATLAB (respective code listings presented in C.1.1 and C.1.2) and analysed with the mathematical models of the TM-ARCHα-n and TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADCs respectively. A VHDL implementation of µCA-2 was also created and combined with the VHDL code developed by Richard Haigh [106] to drive the physical TM-ARCHα-7 ADC (implementation given in C.5.2). An FPGA was then configured with this adapted design and tested with the PCB implementation of the TM-ARCHα-7 ADC.

4.3 Summary

This chapter discussed the implementation of the TM-ARCHα-n and TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADCs and provided additional details on how these two TM-based ADC structures operate. The implementation and more detailed discussions on the operation of the µCA-1, µCA-2 and µCA-3 were also given.

The TM-ARCHα-n ADC structure, where n refers to the number of TM stages within this data converter, was implemented both as a MATLAB model and as a physical electronic device. Two versions of this TM-based ADC structure were investigated during this work: a 16-bit TM-ARCHα-15 ADC was modelled within MATLAB only; and 8-bit TM-ARCHα-7 ADC consisting of 7 TM stages was modelled within MATLAB and as an HDL module, as well as being constructed as a practical circuit. The data conversion circuitry of the TM-ARCHα-7 ADC was constructed out of discrete components, and was connected to an FPGA development board, which generated clock signals to control the operation of the conversion circuitry and well as acquire and align the digital output codes from the ADC.

To achieve a higher resolution with fewer TM and comparator stages in order to improve the conversion speed, a second TM-based ADC structure was explored. The TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC replaced the comparator on the final TM output with a sub-ranging, COTS ADC with a resolution of $R_{sub\text{-}ranging}$. A TM-ARCHβ-7-12 variation of this TM-based ADC structure was only developed as a MATLAB mathematical model during this work.

The fundamental µ, µCA-1, was primarily implemented, and mainly assessed in MATLAB with a TM-ARCHα-15 ADC model (one test was repeated with a TM-ARCHα-7 ADC model). The

μCA-1 was then implemented in VHDL code and validated via simulation with an emulated TM-ARCHα-7 ADC.

The μCA-2, an enhanced version of μCA-1 which compensates for non-matching stage μ and for non-matching $\mu_{\pm}$ in each TM stage, was also assessed in MATLAB with a TM-ARCHα-15 ADC model and occasionally a TM-ARCHα-7 ADC model. The μCA-2 was then implemented in VHDL code and validated via simulation with an emulated TM-ARCHα-7 ADC. This VHDL implementation of the μCA-2 was then used to configure an FPGA that was part of a physical TM-ARCHα-7 ADC, producing an embedded μCS which was then tested.

With the μCA-3, an enhanced version of the μCA-2 that accommodates for the final TM stage being acquired, using a multibit sub-ranging COTS ADC, was implemented and analysed in MATLAB only using digital data produced by a TM-ARCHβ-7-12 ADC model.

The next chapter analyses through MATLAB modelling, as well as through simulating HDL implementations, how the fundamental μCA, μCA-1, affects the performance of the TM-ARCHα-15 ADCs and TM-ARCHα-7 ADCs.

5 Simulated Performance Analysis of a Tent Map Based ADC with the Fundamental Compensation Algorithm

Some of the material in this chapter was previously published in the journal paper [2].

This chapter analyses how the fundamental µCA, µCA-1, affects the performance of the TM-ARCHα-n ADC structure, through MATLAB modelling, as well as through simulating HDL implementations. Tests performed in MATLAB employed the 16-bit TM-ARCHα-15 ADC (which required 15 TM stages), as the resolution of this ADC, assuming a µ = 2, meet the specification for the UMS (discussed in Section 1.5), after taking into consideration the potential loss of up to one bit due to quantisation error [3]. Also, the higher TM-based ADC resolution better illustrated the effectiveness of the µCA. With the VHDL implementation of the µCA-1 (and some tests in MATLAB), the TM-ARCHα-7 ADC model was employed, as this TM-based ADC structure matched the one employed in the physical electronic implementation used when later testing and assessing the µCA-2 (discussed in Chapter 6).

Sections 5.1 and 5.2 analyse the output accuracy of the MATLAB TM-ARCHα-15 ADC model, with different µ, without and with the µCA-1 being applied to the digital output data respectively. The output accuracy analysis consisted of three sets of tests, which were:

1. Bit accuracy predictions: established the quantisation error and the bit accuracy of the TM-based ADC.
2. Static performance predictions: determined the DNL, end-point INL, offset error and gain error of the TM-based ADC.
3. Dynamic performance predictions: established the SNR, SINAD, SFDR and THD parameters of the TM-based ADC.

A sensitivity analysis, presented in Section 5.3 investigated the effects variations in the μ employed by the μCA-1, compared to the μ employed by the TM-based ADC, had on the compensated output. Section 5.4 compares the bit accuracy predictions when the μCA-1 and the μCA developed by Basu [41, 42] were applied to a TM-ARCHα-15 ADC model.

The simulation results from the VHDL implementation of the μCA-1, developed for a TM-ARCHα-7 ADC, are shown in Section 5.5. Section 5.6 gives the final analysis with the μCA-1, which compares methods of approximating the difference measures (DMs) for different μ values.

The TM-based ADC MATLAB models were configured with a 25 MHz sampling frequency and a 0 - 3 V valid input voltage range to match the design this ADC was based on [56, 57]. All but the dynamic performance tests (discussed later in the chapter) supplied the ADCs with a 0 - 3 V ramp input signal with a relatively low input frequency (when compared to the sampling rate). The ramp input signal frequency was set so $2^{(R+2)}$ samples (where R is the ADC resolution) were acquired during one ramp cycle, enabling an ideal, TM-based ADC (μ = 2) to sample every step change within the signal. In the dynamic performance tests, faster, sinusoidal input signals were provided.

The MATLAB scripts and VHDL developed for the tests described in this chapter are presented in Appendix B (where the code presented in each sub-section of Appendix B corresponds to the respective sub-section within in this chapter). Appendix D.1 also presents additional results from the tests.

5.1 Uncompensated Tent Map Based ADC Output Accuracy Analysis

This section presents the output accuracy analysis undertaken to assess the effects non-ideal μ has on the TM-ARCHα-15 ADC model, prior to being compensated by the μCA-1. With the bit accuracy predictions test, the number of missing codes were also established, in addition to calculating the quantisation error and the bit accuracy.

5.1.1 Bit Accuracy Predictions

Figure 5-1 presents the TM-ARCHα-15 model bit accuracy when the μ value for every TM stage was incremented in 0.005 steps from 1.7 to 2 inclusive. A lower limit of 1.7 was chosen for the range of μ values tested, as this was considered a large enough deviation from the ideal value of 2 to illustrate the impact non-ideal μ had on the bit accuracy of the TM-ARCHα-15 ADC. Only μ values which produced a chaotic ADC were considered, hence why the upper μ limit was set to 2, as when $\mu > 2$ the TM stage outputs become unbounded which makes recovering the initial conditions impossible. The results demonstrate that a small reduction in μ results in a significant, exponential loss in bit accuracy, where a μ of 1.995 (point A) results in a bit accuracy of 8.64 bits.

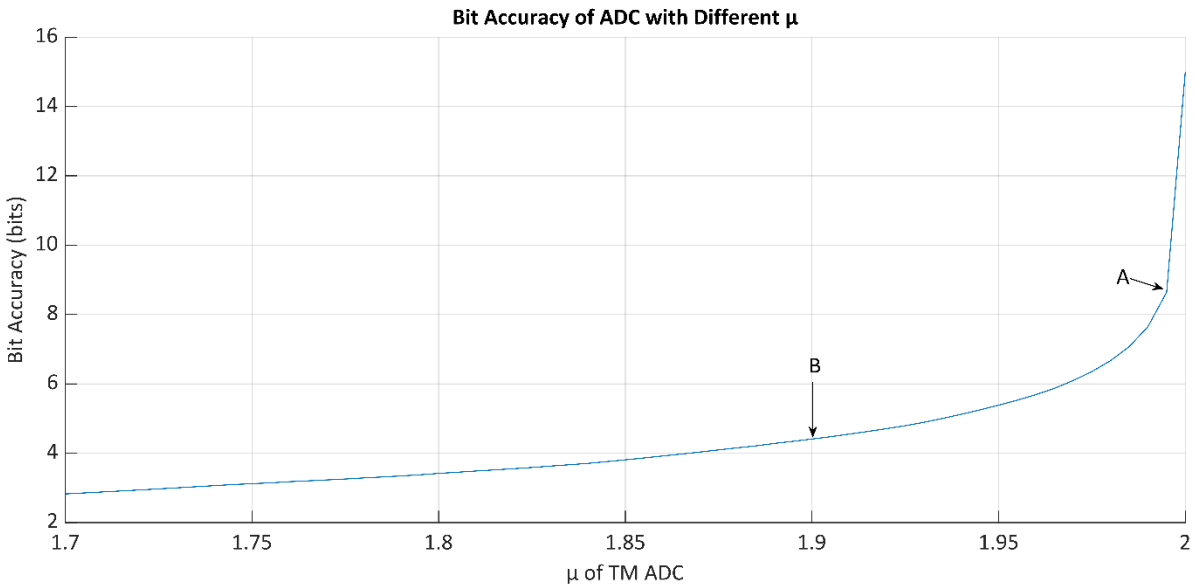


Figure 5-1: Graph illustrating the effects non-ideal μ have on bit accuracy.

Figure 5-2 illustrates the effects non-ideal μ has on the TM-ARCHα-15 ADC output and the deviation from the ideal representation of the input signal. The set-up was the same as the first test, however, a smaller μ range of $1.9 \leq \mu \leq 2$, and larger increments of 0.02 steps, were employed, to better differentiate the effects of non-ideal μ. Table 5-1 summarises the maximum absolute quantisation error, along with the number of missing codes.

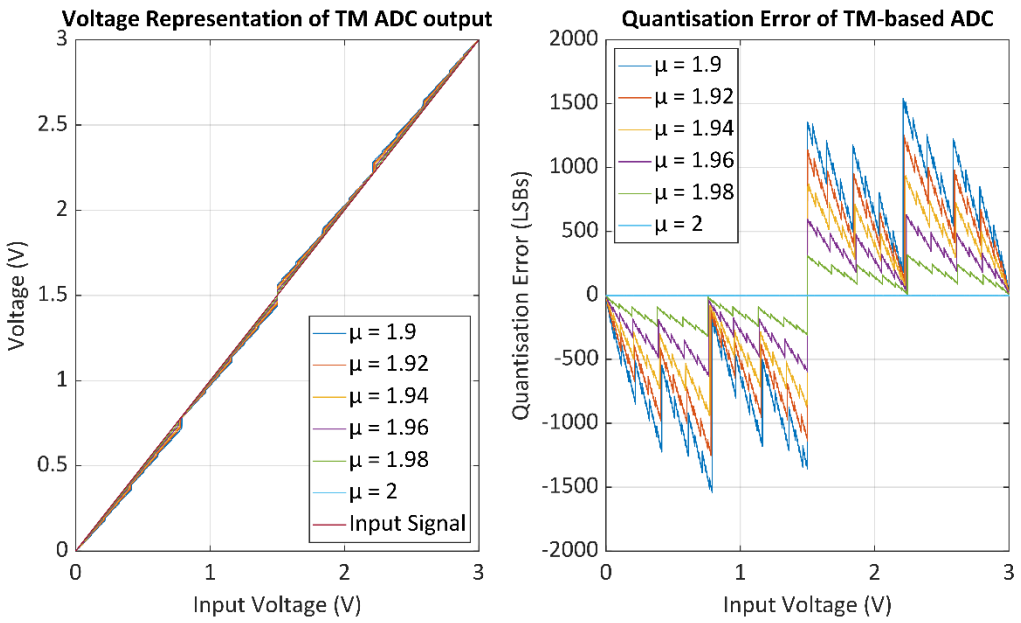


Figure 5-2: Output response and quantisation error of the TM-ARCHα-15 ADC due to different μ.

μ	1.9	1.92	1.94	1.96	1.98	2
Maximum Absolute Quantisation Error (LSBs)	1542.75	1253.50	940.25	634.25	321.25	1.00
Missing Codes (% of codes available)[7]	39.6	34.0	23.8	16.9	7.9	0.0

Table 5-1: Maximum quantisation error between the TM-ARCHα-15 ADC output and input signal.

Figure 5-2 and Table 5-1 demonstrate a 0.02 reduction in μ resulted in an average increase of approximately 308 LSBs in the maximum absolute quantisation error. The number of missing codes also highlight a reasonably linear increase in missing codes as the μ value deviates from 2. The results highlight an approximate increase of 5188 missing codes per 0.02 reduction in μ.

[7] There were 65536 (2^{16}) possible digital output codes.

The purpose of this research was to improve the output accuracy of the TM-ARCHα-15 ADC by compensating for non-ideal μ. Large deviations of μ from 2 (the ideal value) are not desirable, as this results in a high volume of missing codes which cannot be compensated for and equate to data loss. As such, subsequent analysis and test results focussed on μ values which were kept within the range of $1.9 \leq \mu \leq 2$ (point B highlights the lower μ value being considered in Figure 5-1), limiting the number of missing codes. Figure 5-3 presents a histogram of the binary digital output codes produced by the TM-ARCHα-15 ADC when the μ was 1.9 and 2. The plot highlights the gaps (which represent the missing codes) across the range of digital codes which were produced when the $\mu = 1.9$. The codes which were produced by the TM-ARCHα-15 ADC, when the μ was 1.9, were observed more frequently when compared to the codes produced by the TM-ARCHα-15 ADC with a μ of 2. The missing codes cluster, and these clusters increase the further μ deviates from the ideal value of 2. These clusters of missing codes are also partly responsible for the distorted digital representation, of the ramp input signal supplied to the TM-ARCHα-15 ADC observed in Figure 5-2.

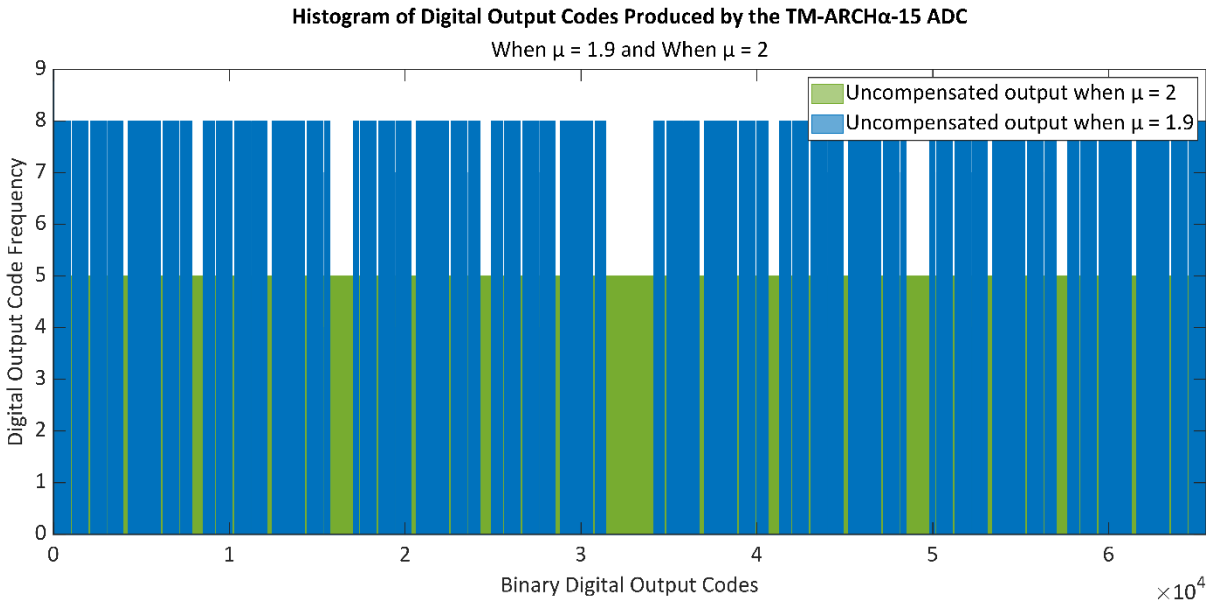


Figure 5-3: Histogram of digital codes produced by the TM-ARCHα-15 ADC with a μ = 1.9 and μ = 2.

5.1.2 Static Performance Predictions

To analyse the static performance of the TM-ARCHα-15 ADC with different μ, the DNL, INL, offset error and gain error were calculated using equations (2-1) to (2-4) respectively over a $1.9 \leq \mu \leq 2$ range (increased in 0.005 increments). The INL was measured using the end-point method (see Section 2.1.1), to provide the worse-case figure, highlighting the maximum potential deviation the ADC output had from the ideal digital output versus analogue input transfer function.

Figure 5-4 presents the maximum and minimum DNL and INL for each μ, along with the offset and ADC gain error. The plots highlight how a decrease in μ affects the INL and DNL of the TM-ARCHα-15 ADC. When $\mu > 1.925$, the ADC is monotonic as the DNL is within the threshold of ± 1 LSB. A fall in μ results in a linear increase and decrease of the maximum and minimum INL respectively, at a rate of 440.5 LSBs per 0.01 drop in μ. No offset or gain error is produced even with different μ values, because the minimum and maximum digital outputs were produced when the respective minimum and maximum voltage inputs were supplied to the TM-ARCHα-15 ADC.

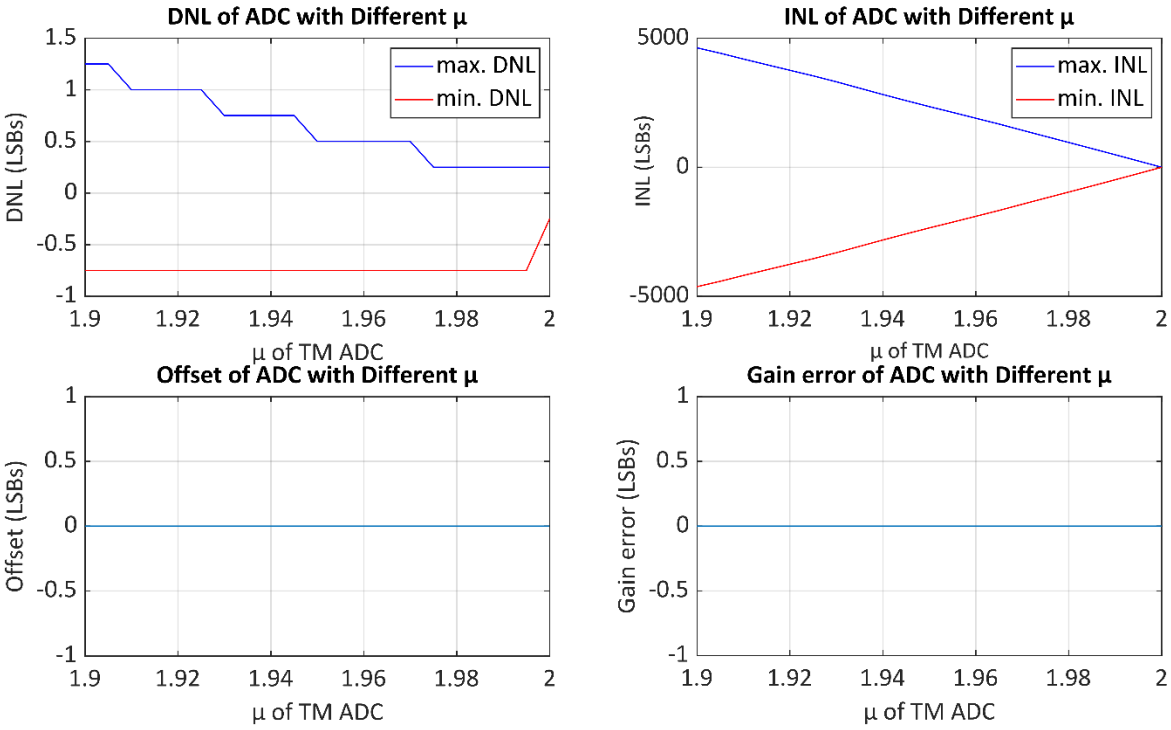


Figure 5-4: Static performance test results of the *TM-ARCHα-15 ADC.*

5.1.3 Dynamic Performance Predictions

To analyse the dynamic performance of the TM-ARCHα-15 ADC with different µ, the SNR, SINAD, SFDR and THD MATLAB functions were employed, which all performed an FFT to determine the dynamic performance parameter value [108]. This test was performed to determine how great the deviation in the SNR, SINAD, SFDR and THD values were over a range of input frequencies, and near to the extremes of the non-ideal µ range of interest.

Initially, µ was set to 1.9 and the dynamic performance parameters were found when the sinusoidal input signal to the ADC was set to have a frequency close to the Nyquist frequency (12.5 MHz, this being half of the sampling frequency) as possible, whilst still meeting the criteria given in (2-5). Therefore, for this test N was set to 262144 (2^{18}), a power of 2^2 higher than the power of 2^R, whilst odd number 131071 was used for the value of M.

The output signal from the TM-based model was then processed by the THD, SNR, SFDR and SINAD MATLAB functions. With the THD function, only the first five harmonics were considered, as this is standard practise when evaluating ADC performance [7]. To assess how the dynamic performance varies over a range of input frequencies, the test was repeated using input frequencies which were approximately a factor of 10 lower than the previous. The frequencies and M values employed were: 1.25 MHz (M = 13109); 125 kHz (M = 1307); 12.5 kHz (M = 131) and 1.24 kHz (M = 13). This test was then repeated with $\mu = 1.995$.

Figure 5-5 presents the results and shows that the parameter values were approximately the same over the range of input frequencies tested (for both μ values under test), except for the one that was effectively the Nyquist frequency**.** This was due to the aliasing of the harmonics, which had frequencies above the Nyquist frequency.

The results also supported the findings shown in the previous two sub-sections, that reducing μ from the ideal value significantly impacts the TM-based ADC performance. The 25 dB reduction in SNR and SINAD, over the 1.24 kHz to 1.25 MHz input frequency range, showed the digital representation of the input sinusoidal wave produced by the TM-ARCHα-15 ADC becomes more distorted the further the μ value deviates from 2. This increase in distortion is highlighted by the THD plot. The SFDR plot also supports these results by highlighting the reduced difference in magnitude between the sinusoidal input signal, reproduced by the TM-ARCHα-15 ADC, and the first harmonic, the further the μ value deviates from 2.

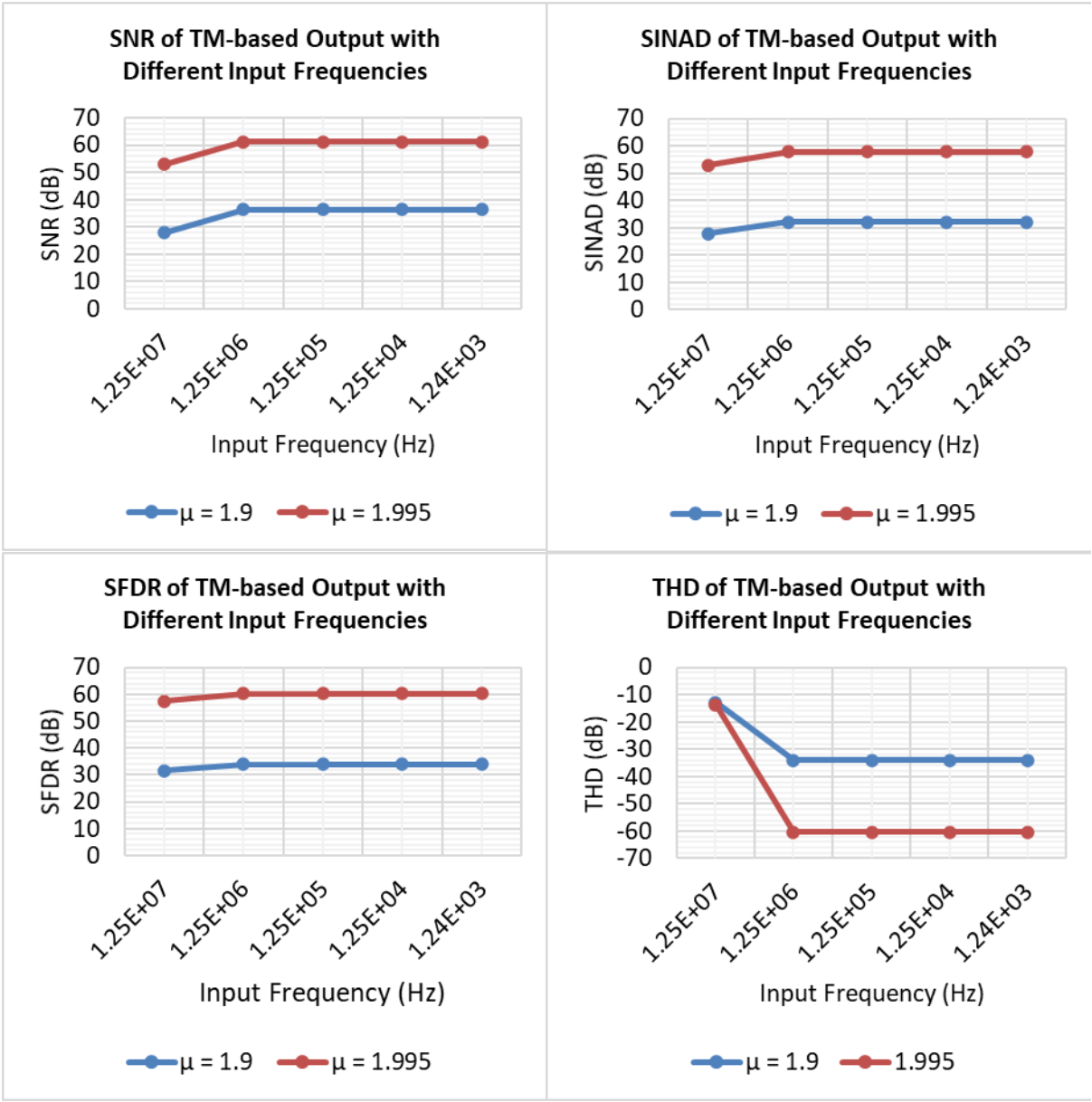


Figure 5-5: SNR, SINAD, SFDR and THD for μ = 1.9 and μ = 1.995 over a range of input frequencies.

5.2 Tent Map Based ADC with the Fundamental Tent Map Gain Compensation Algorithm Output Accuracy Analysis

This section presents the analysis undertaken to assess how the µCA-1 improves the output accuracy of a TM-ARCHα-15 ADC model, over a range of µ values. With the dynamic performance predictions test, the ENOB was also calculated from the established SINAD parameter values, to establish how the input frequency also affected the number of bits for which the ADC could accurately represent a sampled signal as a digital word.

5.2.1 Bit Accuracy Predictions

The µCA-1 was initially tested with the data produced from the TM-ARCHα-15 ADC. The test set-up was similar to the second one described in Section 5.1.1, apart from the µCA-1 was applied to the TM-ARCHα-15 ADC digital output data, and the compensated bit accuracy calculated and noted.

Figure 5-6 presents the bit accuracy versus µ plot, which highlights an improvement in the bit accuracy after compensation over the $1.9 \leq \mu \leq 1.99$ range. The average increase in bit accuracy was 7.15 bits, with a maximum improvement of 9.45 bits. The compensated bit accuracy also falls the further µ deviates from the ideal, due to the increase in missing codes restricting the ability of the µCA-1 to improve the ADC output accuracy.

The top plot in Figure 5-7 compares the ADC analogue input to the equivalent output voltage, before and after compensation, when $\mu = 1.9$, while the bottom plot compares the uncompensated and compensated quantisation error. Figure 5-8 presents the quantisation error of the compensated TM-ARCHα-15 ADC output only. These plots highlight the non-

linear nature of TMs, and the effectiveness of the µCA-1, as both plots show how the maximum absolute quantisation error has reduced from 1542.8 LSBs to 2.2 LSBs due to compensation.

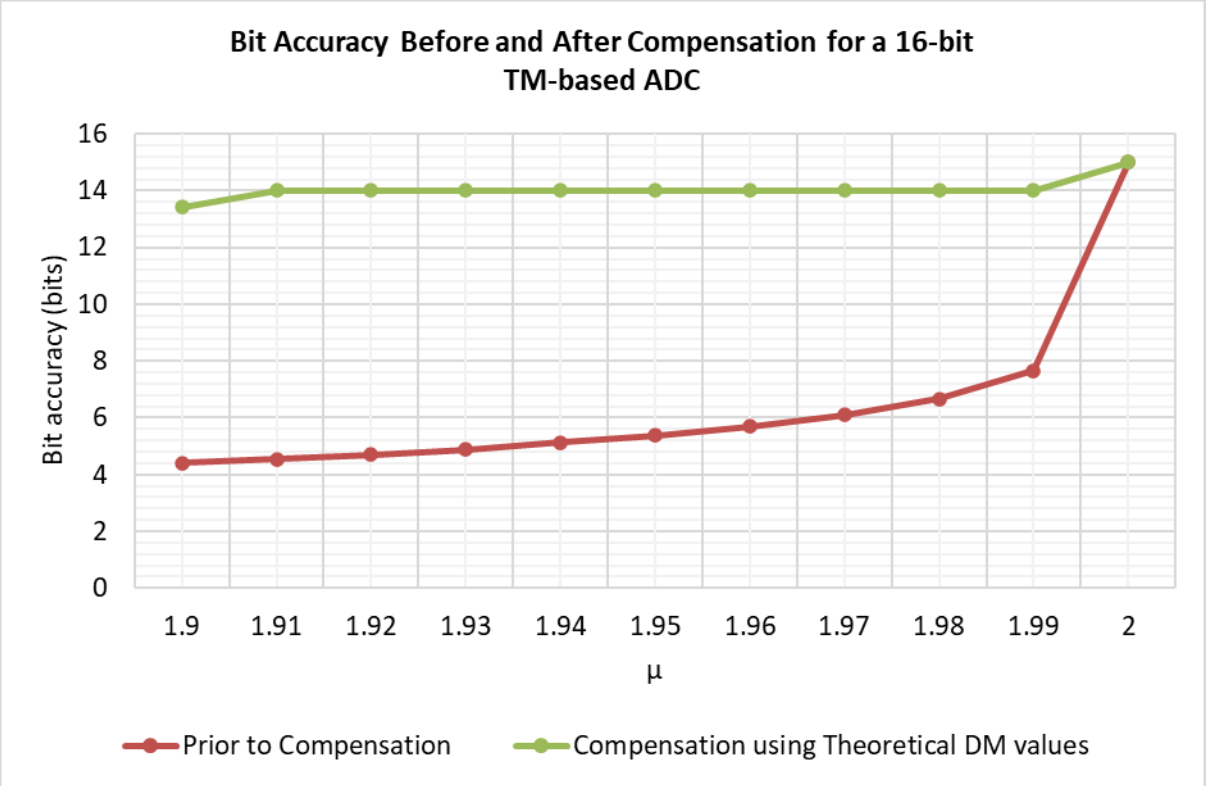


Figure 5-6: Bit accuracy of a TM-ARCHα-15 ADC before and after compensation using theoretical DM values.

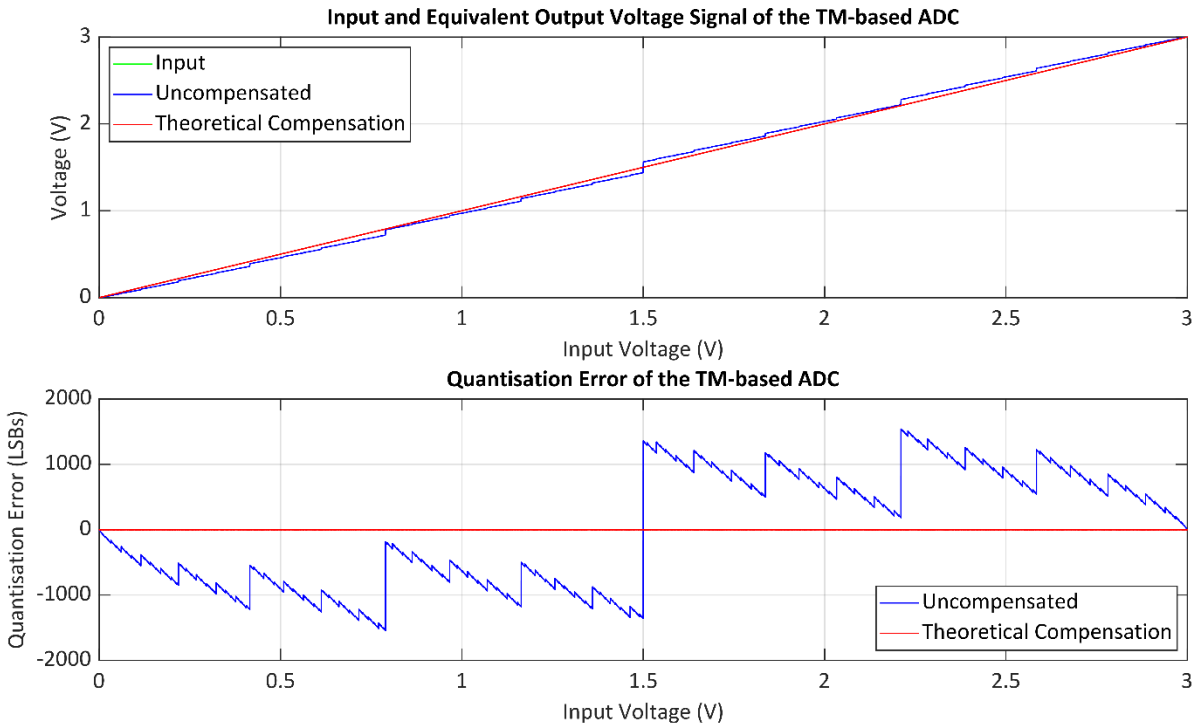


Figure 5-7: Quantisation error of a TM-ARCHα-15 ADC, before and after compensation, when μ = 1.9.

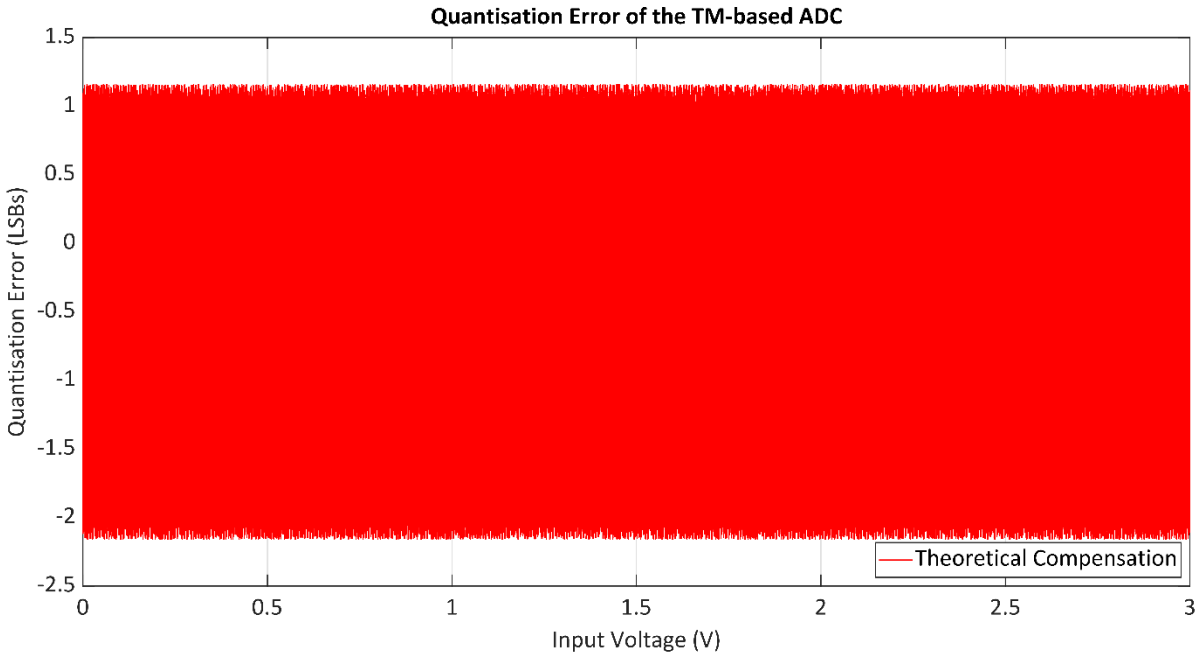


Figure 5-8: Quantisation error of a TM-ARCHα-15 ADC, after compensation, when μ = 1.9.

Figure 5-9 is a histogram detailing the frequency a selection of 150 of the 665536 possible digital outputs, produced by the TM-ARCHα-15 ADC, occurred when the μ was 1.9, before and after the μCA-1 was applied to the digital output data. The plot highlights the observed digital codes are more evenly distributed across the range of possible output codes after compensation, which supports the earlier findings presented in Figures 5-6 to 5-8 that the μCA-1 helps reduce distortion caused by the clusters of missing codes in the digital signal representation of the ramp input signal. The frequency of the observed codes after compensation has increased, due to the μ compensation process altering certain digital output codes and causing them to match other pre-existing digital combinations, which in turn increases the number of missing codes. Yet the maximum absolute quantisation errors of the respective digital codes produced by the compensated TM-ARCHα-15 ADC across the range of μ investigated are lower prior to compensation, confirming the μCA-1 does improve the ADC output accuracy.

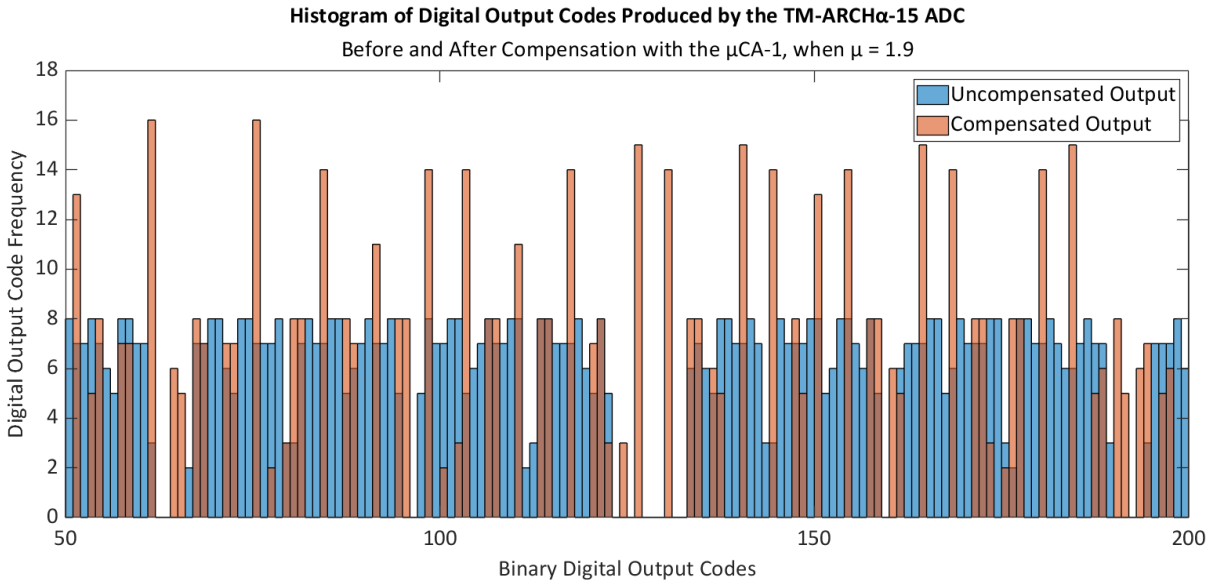


Figure 5-9: A histogram of a selection of 150 digital codes which could be produced by the TM-ARCHα-15 ADC with a μ = 1.9, before and after the μCA-1 was applied to the output data.

To reduce the FPGA resource requirements when creating a VHDL implementation of the μCA-1 (see Section 5.5), the DM values were converted to binary code formant. (5-1) presents the method employed to achieve this [109].

$$DM_{binary} = \lfloor DM_{theoretical} \times 2^{r} \rfloor \quad (5\text{-}1)$$

Where:

- r is the chosen resolution for the binary DM values;
- $DM_{theoretical}$ represents the theoretical, decimal calculation (which is always less than 1) of a DM value; and
- DM_{binary} gives the integer value produced which can then be presented as binary code.

To find the optimal value of r that achieved the same improvement to the TM-ARCHα-15 ADC accuracy as the theoretical DM values, the first test was repeated with binary DM values. The r value was initially set to 17 bits and incremented until the binary DM values reached the same levels of bit accuracy (across the observed range of $1.9 \leq \mu \leq 2$) as the theoretical DM values.

Table D-1 in Appendix D presents the results from this test, showing that to achieve the same improvement in the TM-ARCHα-15 ADC bit accuracy across the range $1.9 \leq \mu \leq 2$, r must be 24 bits. However, the results also highlight that TM-based ADCs with higher μ values (e.g., $\mu \geq 1.96$) can employ a lower r value (such as 18 bits), reducing the FPGA resource requirements further due to the lower resolution DM values being stored. For the remaining tests with the μCA-1 presented in Section 5.2, binary DM values with r values of 24 bits were employed.

5.2.2 Static Performance Predictions

The static performance tests described in Section 5.1.2 were repeated on the compensated output of the TM-ARCHα-15 ADC model. Figure 5-10 presents the maximum and minimum DNL and INL for each μ, before and after compensation. The plots highlight how the μCA-1 reduced the magnitude of the maximum and minimum INL of the TM-ARCHα-15 ADC across the μ range being considered (including a fall in INL of approximately 4620 LSBs when the μ = 1.9). This is because the reconstructed ADC output signal is compensated to represent the original input signal more closely.

The offset and gain error remained at zero after compensation (thus not plotted), as the μCA-1 does not alter the minimum and maximum digital outputs which can be produced by the TM-ARCHα-15 ADC. The DNL was negatively impacted by the μCA-1, as the magnitude of both the minimum and maximum DNL increased across the considered μ range for the TM-ARCHα-15 ADC. The TM-based ADC structure was only monotonic when $\mu > 1.99$ (compared to the uncompensated ADC which was monotonic when $\mu > 1.925$). This is a result of the μCA-1 altering certain digital output codes, causing them to match other pre-existing digital combinations.

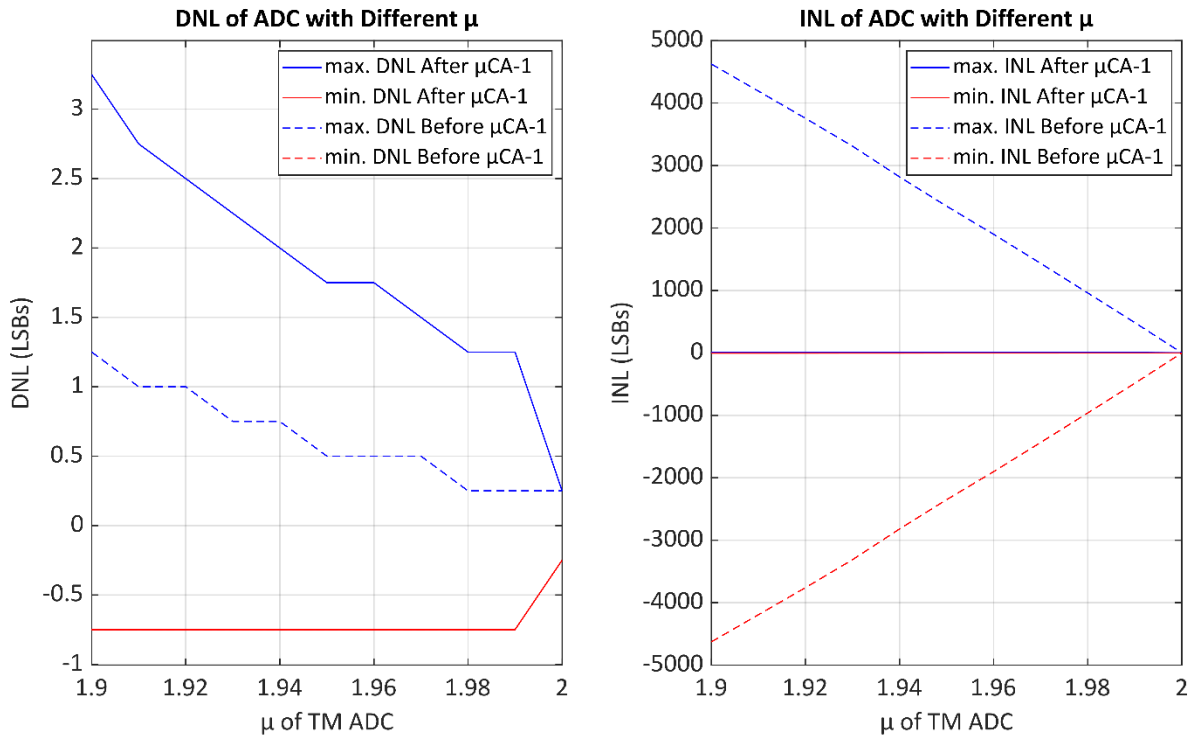


Figure 5-10: Static performance of the uncompensated and compensated TM-ARCHα-15 ADC with binary DM values (r = 24 bits).

5.2.3 Dynamic Performance Predictions

A variation in the dynamic performance tests described in Section 5.1.3 were performed on the compensated output of the TM-ARCHα-15 ADC model. Section 5.1.3 highlighted the dynamic performance of the TM-ARCHα-15 ADC model was similar when the input frequency was equal to, or lower than, 1.25 MHz. Therefore, only the two highest frequencies (12.5 MHz and 1.25 MHz) were employed in these tests. The tests were performed across the whole $1.9 \leq \mu \leq 2$ range (increased in 0.005 increments) and the ENOB was also calculated, using (2-6), after obtaining the SINAD measurement.

Figures 5-11 to 5-15 present the results for the SNR, SINAD, SFDR, THD and ENOB respectively. The plots highlight the improvement with these parameters after the TM-ARCHα-15 output had been compensated using the μCA-1 over a $1.9 \leq \mu \leq 1.995$ range. The SNR, SINAD, SFDR

and ENOB (Figures 5-11, 5-12, 5-13 and 5-15 respectively) did worsen the more μ deviated from the ideal value of 2, but the deterioration was less than that before the μCA-1 was applied. There was also an improvement in the THD and SFDR across the range $1.9 \leq \mu \leq 1.995$, although the improvement in THD was restricted when the Nyquist frequency was supplied due to the aliasing of the harmonics further distorting the ADC output signal.

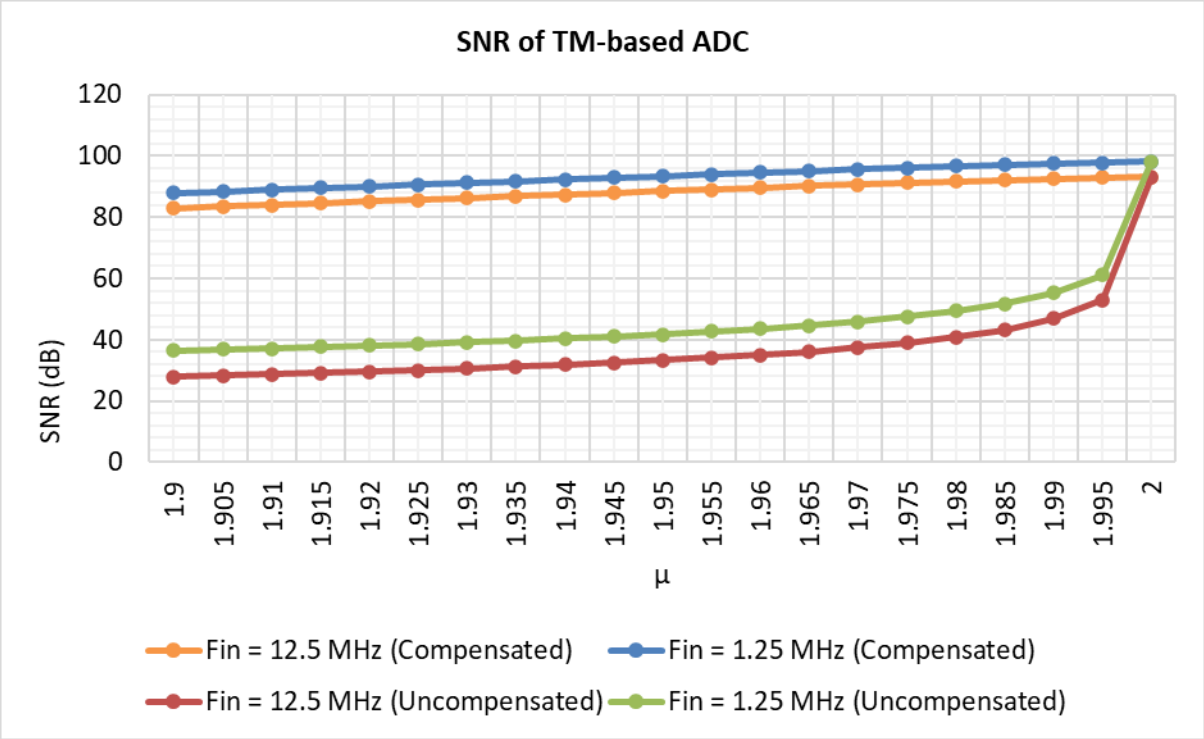


Figure 5-11: SNR plot of a TM-ARCHα-15 ADC before and after compensation.

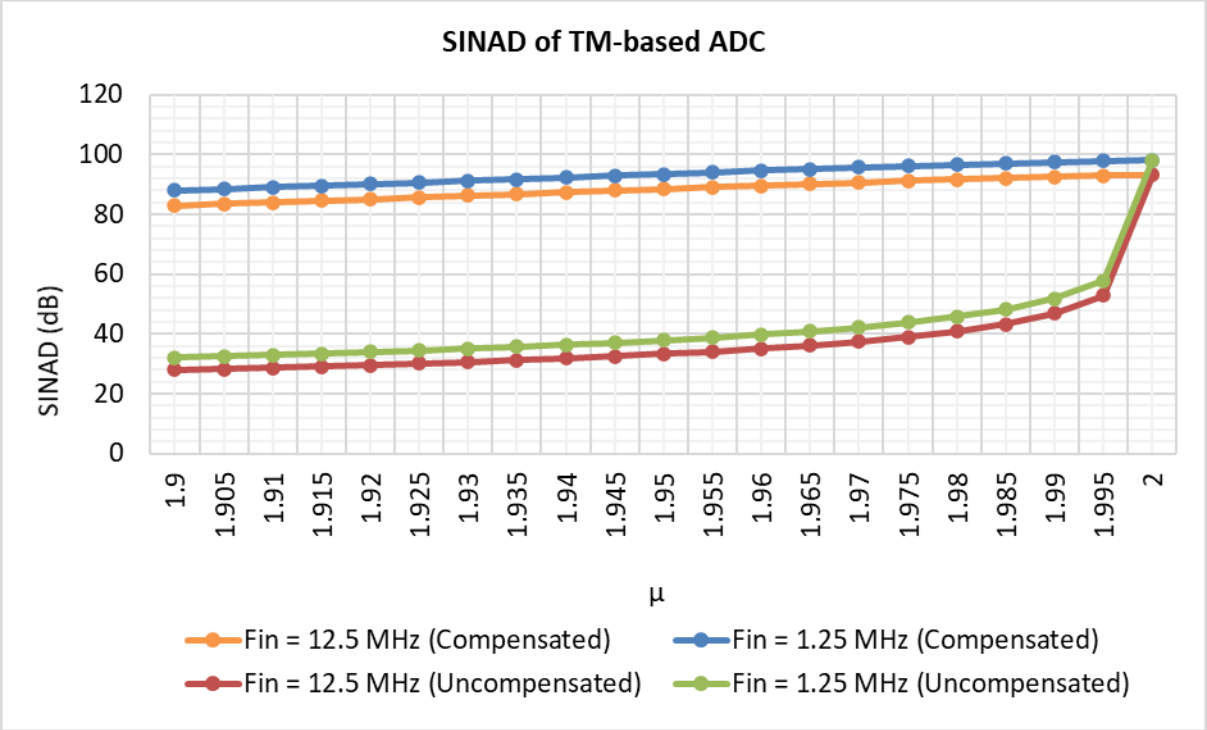


Figure 5-12: SINAD plot of a TM-ARCHα-15 ADC before and after compensation.

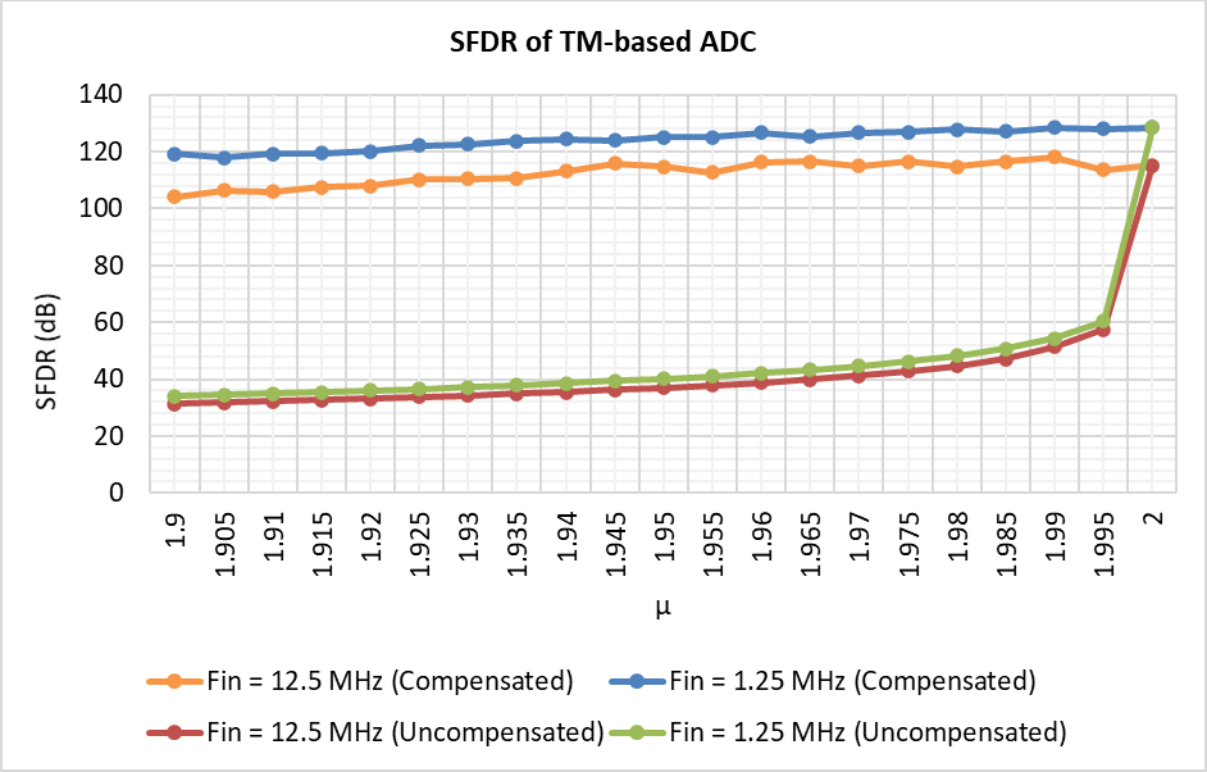


Figure 5-13: SFDR plot of a TM-ARCHα-15 ADC before and after compensation.

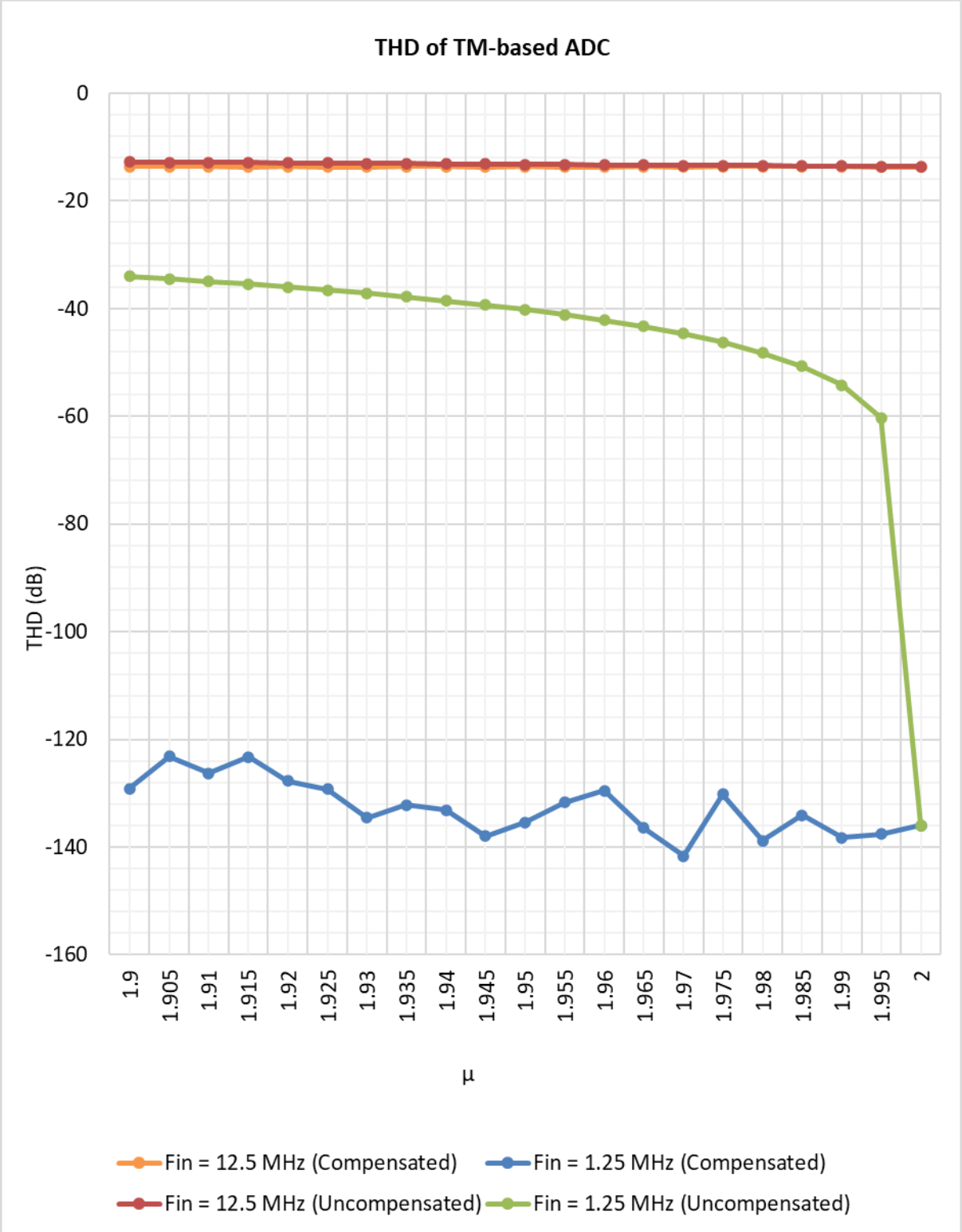


Figure 5-14: THD plot of a TM-ARCHα-15 ADC before and after compensation.

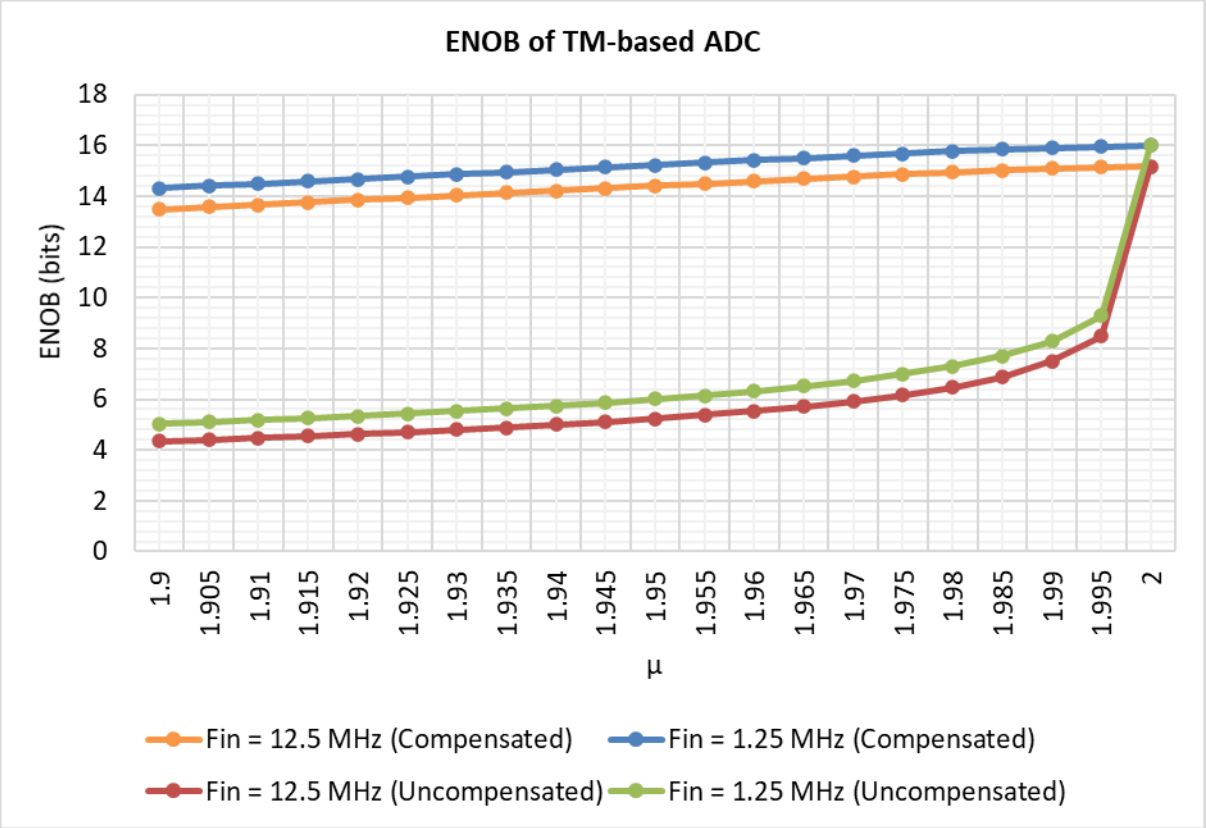


Figure 5-15: ENOB plot of a TM-ARCHα-15 ADC before and after compensation.

5.3 Sensitivity Analysis of the Fundamental Tent Map Gain Compensation Algorithm

There will always be uncertainty when measuring the μ of a TM-based ADC. This will restrict the ability of the μCA-1 to compensate for non-ideal μ because, the DM values employed will be calculated from the measured μ value. To quantify the effects variations between the μ values employed by the μCA-1, and the actual μ of the TM-based ADC, had on the ability of the μCA-1 to compensate for non-ideal μ within the ADC, a sensitivity analysis was performed.

The μ of the TM-ARCHα-15 ADC (μ_{ADC}) was initially set at the lowest value of non-ideal μ of 1.9 being investigated by this analysis, and the μCA-1 applied to the ADC output. The range of μ employed by the μCA-1 ($\mu_{algorithm}$) had a percentage deviation (%Δ) from μ_{ADC} over a range of -2.5% ≤ $\mu_{algorithm}$ %Δ from μ_{ADC} ≤ 2.5%, in increments of 0.1 %Δ. This test was repeated with

a higher non-ideal μ_{ADC} value of 1.99 to determine how TM-based ADCs, with higher μ_{ADC} values, were affected when the $\mu_{algorithm}$ was not identical.

Figure 5-16 presents the results of the sensitivity analysis for μ_{ADC} = 1.9 and 1.99 respectively. Both curves highlight a sharp roll-off in compensated bit accuracy when $\mu_{algorithm}$ deviated from μ_{ADC} by ± 0.1 % (there was a reduction from 13.42 to 9.91 bits when μ_{ADC} = 1.9 and from 14 to 9.91 bits when μ_{ADC} = 1.99). The rate of reduced improvement to bit accuracy was reduced for the lower $\mu_{ADC,}$ but the difference in bit accuracy between the two μ_{ADC} values for the same $\mu_{algorithm}$ deviation remained less than 0.06 bits.

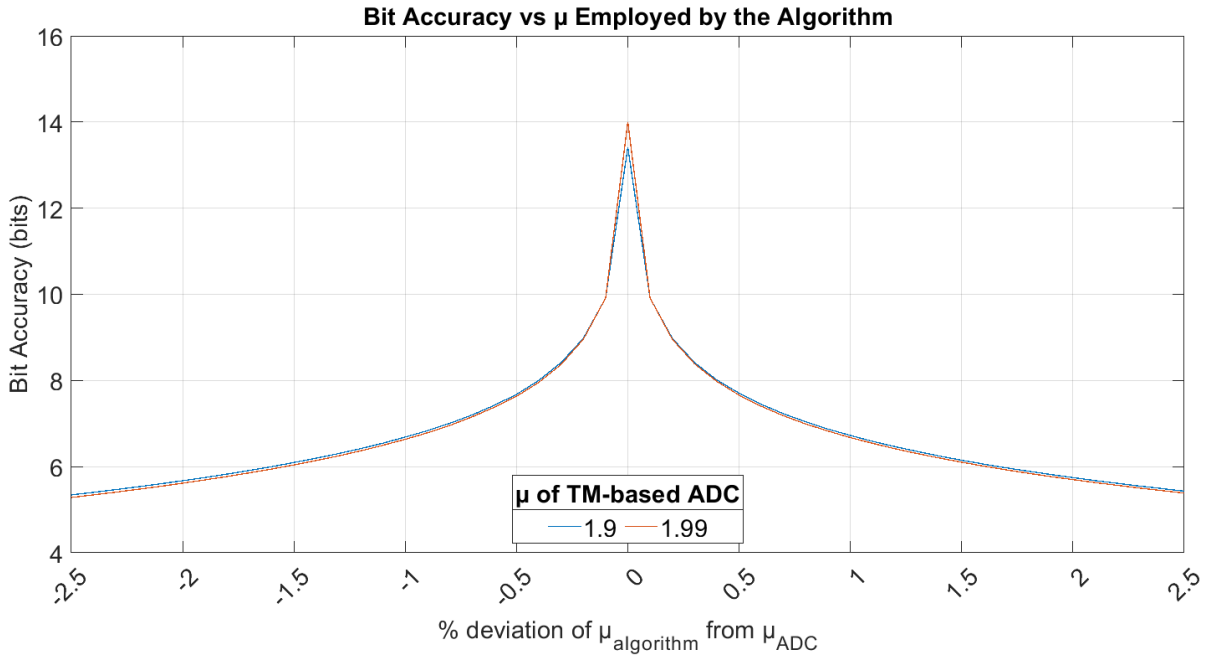


Figure 5-16: Sensitivity analysis results from a TM-ARCHα-15 ADC when μ_{ADC} = 1.9 and μ = 1.99.

A small deviation in $\mu_{algorithm}$ from μ_{ADC} results in a significant reduction in the improvement to the TM-ARCHα-15 ADC output accuracy. This rapid reduction is undesirable, as there is uncertainty when measuring the μ_{ADC} of an electronic implementation of a TM-based ADC.

The electronic implementation of the TM-ARCHα-7 ADC, which the enhanced version of μCA-1 was being tested with (see Section 6.6 of the next chapter), had an 8-bit resolution. The sensitivity analysis was repeated for an TM-ARCHα-7 ADC, so the effects uncertainty in measuring the μ_{ADC} have on the ability of the μCA-1, to compensate for non-ideal μ within a lower resolution TM-based ADC, could be determined. The test procedure was identical to the one performed on the TM-ARCHα-15 ADC.

Figure 5-17 presents the results. The reduction in improved bit accuracy is more gradual for a lower resolution TM-based ADC. The maximum $\mu_{algorithm}$ can deviate from μ_{ADC} to maintain the highest bit accuracy is -0.5 % to +0.6 %, when μ_{ADC} = 1.9, and -0.8 % to +0.9 %, when μ_{ADC} = 1.99. Unlike the TM-ARCHα-15 ADC, a higher μ_{ADC} resulted in a less acute reduction in bit accuracy for the TM-ARCHα-7 ADC.

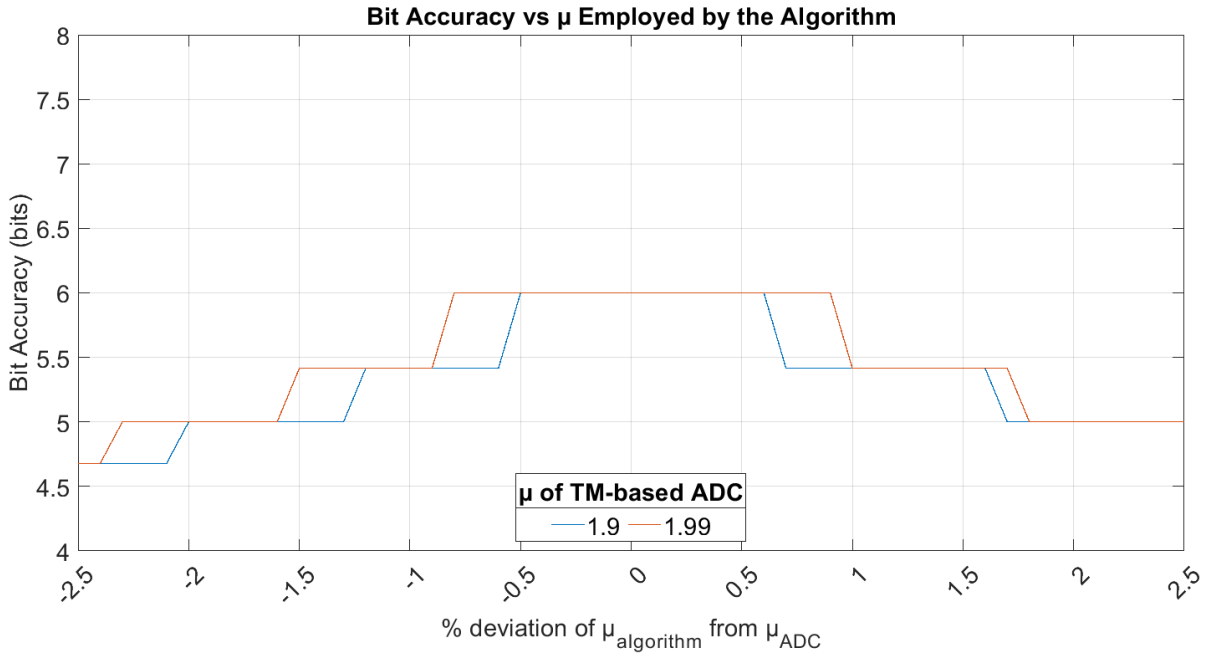


Figure 5-17: Sensitivity analysis results from a TM-ARCHα-7 ADC.

5.4 Comparison with the Tent Map Gain Compensation Algorithm by Basu

The effectiveness of the µCA-1 from this research project was compared with that developed by Basu [41, 42] using the MATLAB code provided in [41]. This was to confirm that the lower number of computational resources, employed by the µCA-1, did not diminish the ability to improve the TM-based ADC output accuracy, when compared to the µCA by Basu [41, 42].

Figure 5-18 is a graph comparing the bit accuracy of the TM-ARCHα-15 ADC model over a $1.9 \leq \mu \leq 2$ range compensation, with compensation using the µCA-1, and with compensation using the µCA developed by Basu [41]. The results show that the µCA-1 achieves identical performance, in regard to improving the bit accuracy of the ADC, when compared to the µCA developed by Basu [41, 42]. The µCA by Basu determined the compensation values using iterative calculations involving division, which is computationally resource intensive [62]. Meanwhile, the µCA-1 determines the compensation values by adding pre-calculated DM values, which requires less computational resources, and can be implemented as a single digital system (as proven in the following section) to compensate the output data of a TM-based ADC prior to being transmitted.

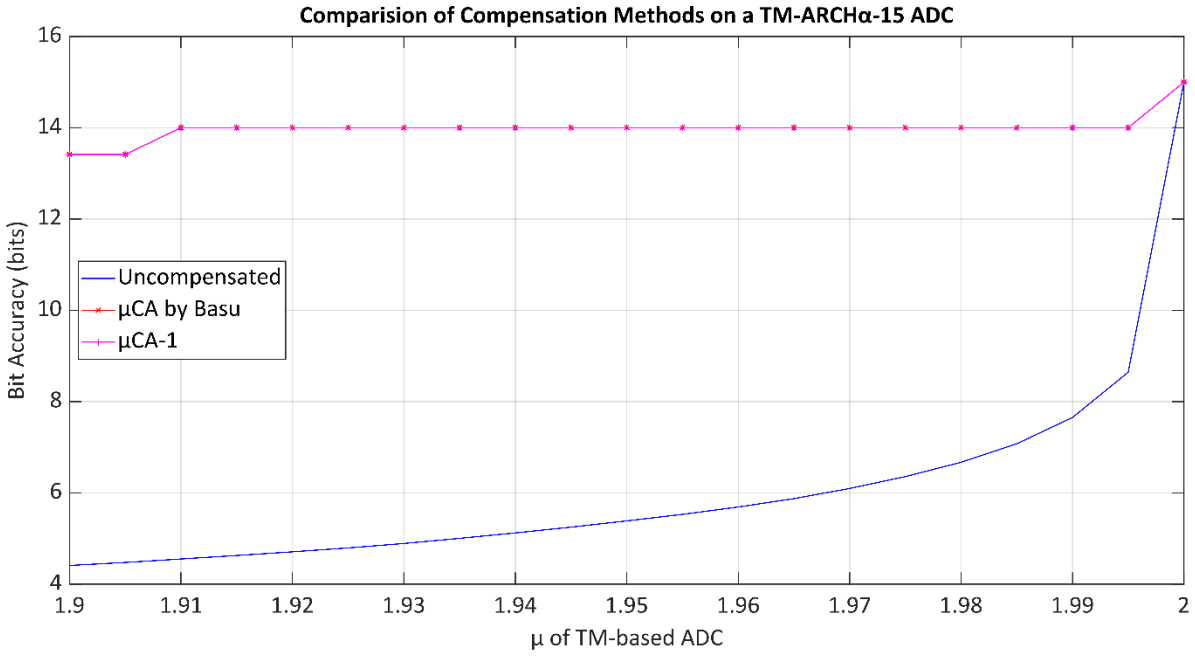


Figure 5-18: Comparing μCA performance on the TM-ARCHα-15 ADC model.

5.5 VHDL Implementation of the Fundamental Tent Map Gain Compensation Algorithm

A VHDL implementation of a μCS, comprising of the μCA-1, was developed for a TM-ARCHα-7 ADC with a μ_{ADC} = 1.9. This VHDL implementation, which could configure the same FPGA used to coordinate the operation of a TM-ARCHα-7 ADC, was assessed via simulation to prove a μCS comprising the μCA-1 could be embedded within a TM-based ADC and perform real-time compensation.

The μCA-1 was implemented such that the compensation was applied to the converted TM-ARCHα-7 ADC sample before the next input sample was converted to binary, delivering real-time compensation for non-ideal μ. The DM values, used by the μCA-1 for each bit, were precalculated using (4-4), converted to binary using (5-1), and hard coded as an array within the VHDL code. This was to reduce FPGA resource requirements, so the DM values were not

repeatedly calculated within the µCS. The r value was set to 10 bits, as tests show that this value achieves the same improvement in bit accuracy, for an TM-ARCHα-7 ADC, as using theoretical DM values (see Table D-2 in Appendix D).

A practical TM-ARCHα-7 ADC signal emulator, developed in VHDL to test the µCA-1, mimicked the ADC sampling and outputting of the acquired data. The signal emulator generated an equivalent output to a practical TM-ARCHα-7 ADC, with $\mu_{ADC} = 1.9$, when supplied with a ramp input signal.

A test bench VHDL program was also developed to write and save the uncompensated and compensated values from the implemented µCA-1 into two separate texts files during simulation (using the ModelSim FPGA software). Figure 5-19 and Figure 5-20 show how these VHDL components were connected for simulation.

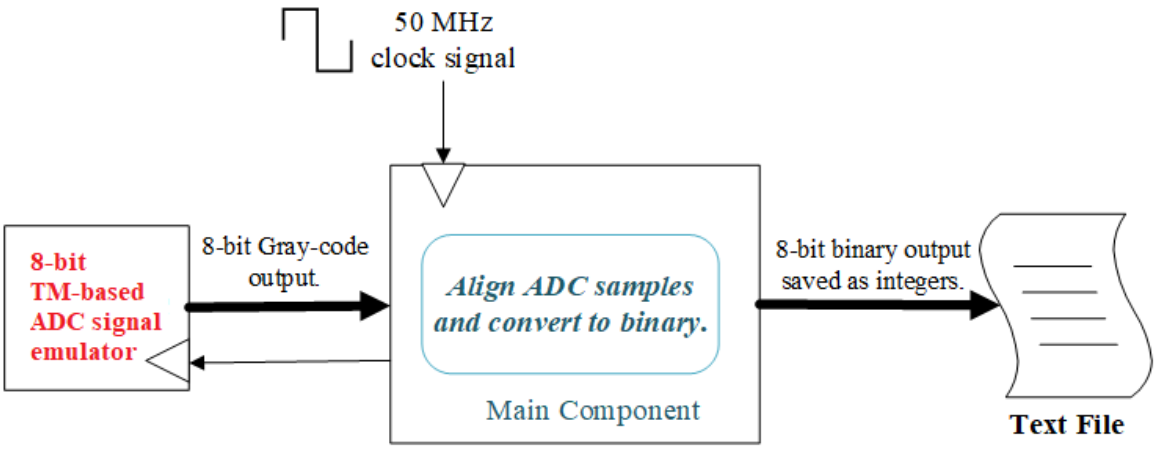


Figure 5-19: Testing the ADC output without applying the µCA-1.

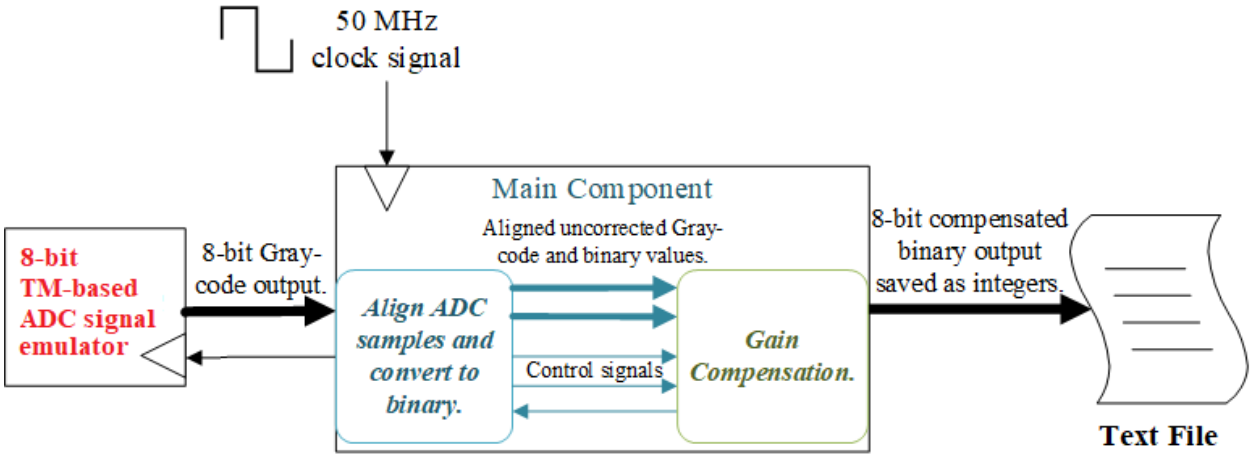


Figure 5-20: Testing the ADC output with the µCA-1.

Figure 5-21 presents a graph of the quantisation error of the simulated TM-ARCHα-7 ADC VHDL module, with, and without, the applied µCA-1. Figure 5-21 highlights the implemented µCA-1 reduced the maximum quantisation error from 6 bits to 1 bit, thus enhancing the bit accuracy of the model TM-based ADC from 4.19 bits to 6 bits. In addition, the µCA-1 was performed on each newly acquired sample of the input signal while the next sample was obtained, confirming that real-time compensation of non-ideal µ is possible.

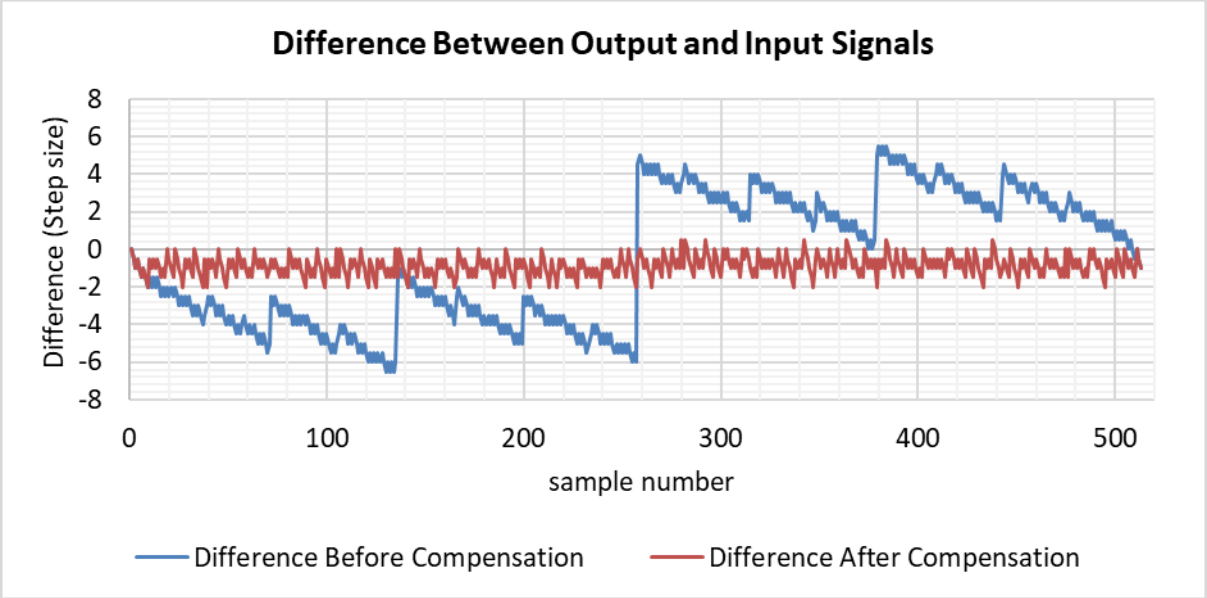


Figure 5-21: Quantisation Error of the TM-ARCHα-7 ADC VHDL model before and after compensation.

5.6 Approximating Difference Measure Values for the Fundamental Tent Map Gain Compensation Algorithm

The focus of this research was to compensate for non-ideal μ, within a practical TM-based ADC, using an embedded compensation system. This work could be taken further by producing a self-calibrating TM-based ADC, which monitors the inevitable drifts in μ_{ADC} overtime (drifting is due to the resistors which alter the μ values, over time, and with temperature [61]) by estimating the change from the ADC output and then applying compensation to the digital output.

The μCA for a self-calibrating TM-based ADC needs the DM values to be calculated within the FPGA requiring complex division performing circuitry [110]. This is especially the case for higher resolution TM-based ADCs as the quantity (and resolution) of DM values to be calculated increases. Therefore, a comparison of two methods to approximate the DM values, which were more resource efficient, was performed.

The first method, referred to as the Straight-line and Error Approximation (and abbreviated to SL&EA), involved taking a straight-line approximation of the ideal DM values, calculated using (4-4), versus μ curves (over a $1.9 \leq \mu \leq 2.0$ range) for each bit. The gradients of the straight-line approximations were calculated using the end points of the DM versus μ plots (as shown in Figure 5-22 for the LSB) as this approach was found to achieve closer approximations of the DM values than estimating the gradient using a line of best fit. The constant for the main straight-line approximation was determined using the DM values when $\mu = 2$.

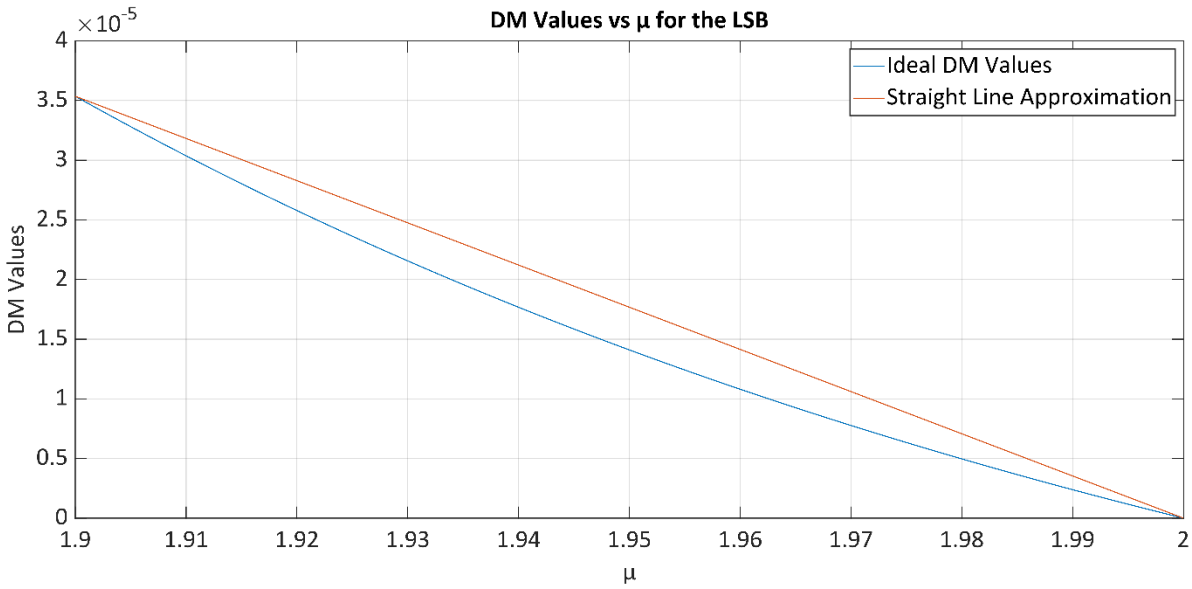


Figure 5-22: Establishing straight-line approximation of the LSB DM values.

The difference between the approximated and ideal DM values were then determined to calculate the errors. Two straight-line approximations of the error versus μ curves were also taken and added to the first straight-line approximation to improve accuracy. The constant value chosen for both straight-line approximations of the error was the error value at the mid-range μ (μ = 1.95).

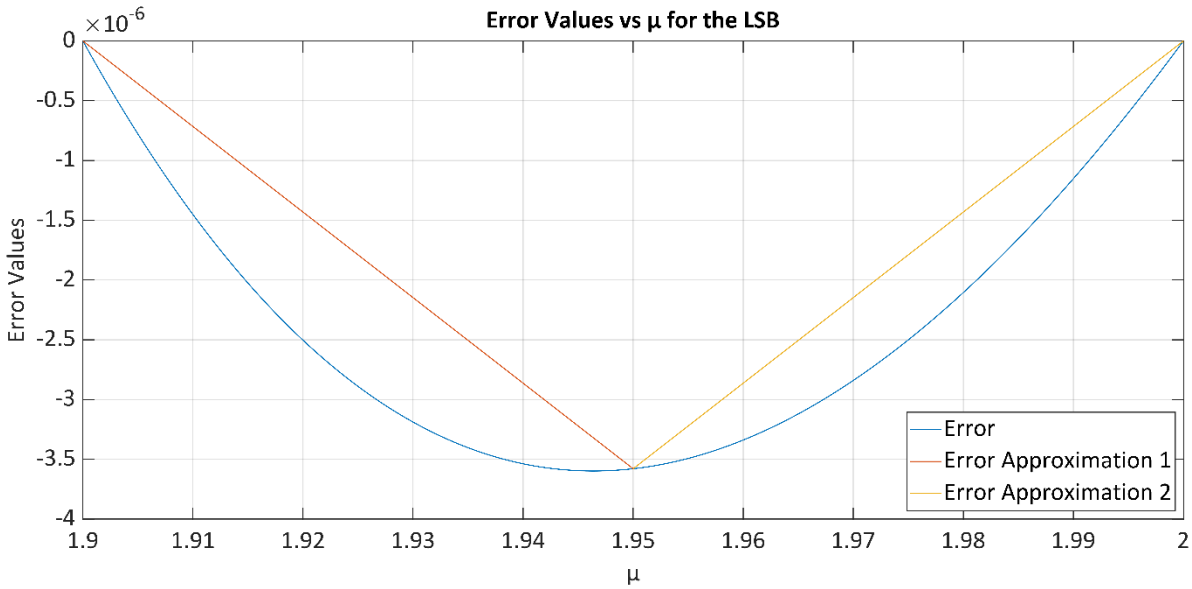


Figure 5-23: Establishing straight-line approximation of the LSB error values.

The second method, referred to as the scalar approximation method (and abbreviated to SA), involved taking the ideal DM values for a chosen μ (μ_o) and multiplying by a scalar value determined using (5-2).

$$\text{Scalar} = 1 - \frac{\mu_C - \mu_o}{2 - \mu_o} \tag{5-2}$$

Where μ_c is the actual gain of the TM circuit and μ_o is the gain employed in the ideal DM determined from (4-4). The fundamental concept of (5-2) was provided by Dr Peter Mather (research supervisor).

Both the SL&EA and SA methods were tested over a $1.9 \leq \mu \leq 1.99$ range in 0.01 increments. The SA method was tested using the μ_o value of 1.95, as this value was the mid-point of the μ range employed in the test. The test outlined in Section 5.2.1 was repeated using these two methods to approximate the DM values and Figure 5-24 presents the results. In this test, the DM values were not converted to binary as the focus was to analyse the effectiveness of the approximation methods. Overall, the SL&EA method attained the most accurate results, although the SA method (which requires fewer FPGA resources) achieved identical performance when $\mu \geq 1.94$.

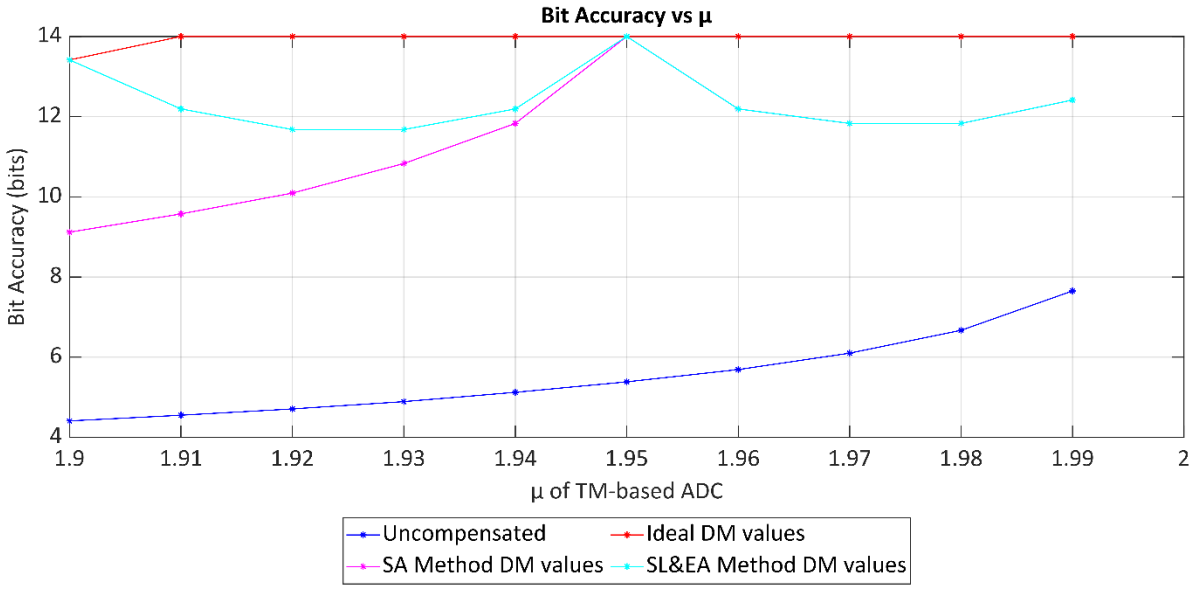


Figure 5-24: Results from DM approximation tests.

5.7 Summary

This chapter analysed through MATLAB modelling, as well as through simulating HDL implementations, how the fundamental μCA, μCA-1, affected the performance of the TM-ARCHα-15 ADCs and TM-ARCHα-7 ADCs.

The output accuracy analysis of the uncompensated TM-ARCHα-15 ADC highlighted an increase in the maximum absolute quantisation error of approximately 308 LSBs per 0.02 deviation in μ from the ideal value of 2. There was also a linear increase of around 5188 missing codes per 0.02 decrease in μ. A small deviation of -0.25 % from the ideal μ also resulted in a drop in bit accuracy (the minimum number of bits for which an ADC can accurately represent an analogue input as a digital word) from 15 bits to 8.64 bits. This consistently significant reduction in the TM-ARCHα-15 ADC output accuracy, as the μ deviates further from the ideal value of 2, was also observed with the static performance and dynamic performance tests.

A noticeable improvement in the output accuracy of the TM-ARCHα-15 ADC model was observed after the μCA-1 processed the digital output data. The average bit accuracy increased from 5.52 to 13.94 bits over a $1.9 \leq \mu \leq 1.99$ range, and the maximum INL error was reduced by approximately 4618 LSBs when the $\mu = 1.9$. The dynamic performance parameters also highlighted an improvement in the ADC output accuracy. Only the static performance parameter DNL was found to have been negatively affected by the μCA-1. This was because the μCA-1 modified some digital output codes to other binary values, which increased the range of analogue voltages associated with certain digital codes. Even after compensation for non-ideal μ, the TM-ARCHα-15 ADC output accuracy deteriorated the further μ deviated from the ideal, due to the increased distortion of the equivalent uncompensated output signal and volume of missing codes. Therefore, with practical implementations of TM-based ADCs, the closer μ is to 2, the better output accuracy which can be achieved after compensation.

The μCA-1 was also proven to be as effective as the μCA developed by Basu [41, 42] with no difference in the bit accuracy of the compensated TM-ARCHα-15 ADC output. The advantages of the μCA-1 were the reduction in the required computation resources and the ability of being implemented as a single digital system to compensate the output data of a TM-based ADC in real-time prior to being transmitted. This latter point was confirmed by simulation of a VHDL implementation of the μCA-1.

This work could be taken further by producing a self-calibrating TM-based ADC, which monitors the inevitable drifts in μ_{ADC} overtime (the drift being due to the resistors which altered the μ values, over time, and with temperature [61]) by estimating the change in μ from the ADC output and then applying compensation to the digital output. The μCA for a self-calibrating TM-based ADC requires the DM values to be calculated within the FPGA,

requiring complex division performing circuitry [110] which will increase resource requirements, especially with higher resolution designs. Therefore, a comparison of two methods to approximate the DM values for μCA-1, which were more resource efficient, was performed. The results showed the method providing the maximum improvement in the TM-ARCHα-15 ADC bit accuracy across the $1.9 \leq \mu \leq 2$ range, was the SL&EA method. However, the SA method, which requires fewer FPGA resources, achieved identical performance when the $\mu \geq 1.94$. Therefore, the SA method is better to implement if the non-ideal μ of the TM-ARCHα-15 ADC is greater than 1.94.

The next chapter analyses the enhanced μCAs, μCA-2 and μCA-3, through MATLAB modelling. The TM-ARCHα-15 ADC and TM-ARCHα-7 ADC models are employed with the μCA-2 analysis, while a TM-ARCHβ-7-12 ADC model is used to assess the μCA-3. A VHDL implementation of the μCA-2 is also assessed via simulation using a TM-ARCHα-7 ADC structure, then tested through practical experiments.

6 Performance Analysis of Tent Map Based ADCs with the Enhanced Compensation Algorithms

This chapter analyses the enhanced μCAs (μCA-2 and μCA-3) through MATLAB modelling. A VHDL implementation of the μCA-2 is also assessed, firstly via simulation, and then tested through practical experiment.

Section 6.1 details the bit accuracy analysis of the μCA-2 and μCA-3, developed to take into account non-matching slope gains (μ_+ and μ_-) for each TM stage ($\mu_{\pm stage}$), and the incorporation of a multibit, sub-ranging COTS ADC at the final TM stage output, respectively. Sensitivity analysis of the μCA-2 and μCA-3 compensated, TM-based ADC outputs are then presented in Section 6.2.

Section 6.3 details the output accuracy analysis of the TM-ARCHβ-7-12 ADC MATLAB model, utilising the enhanced compensation algorithm μCA-3. Section 6.4 then details the noise analysis performed on the same MATLAB model.

Finally, the simulation results from the μCA-2 VHDL implementation, developed for a TM-ARCHα-7 ADC, are presented in Section 6.5. Section 6.6 then provides the test results from the electronic implementation of TM-ARCHα-7 ADC with an embedded μ compensation system (μCS) that comprises the μCA-2.

Three TM-based ADC structures were employed in analysing the enhanced μCAs. Initially, the 16-bit TM-ARCHα-15 ADC (which required 15 TM stages) was used to assess the effectiveness μCA-2 (Section 6.1.1) in compensating for non-matching $\mu_{\pm stage}$, as the higher TM-based ADC resolution better highlighted the improvement in output accuracy. For the remaining simulation tests with μCA-2, the TM-ARCHα-7 ADC model was employed, as this structure

matched the physical electronic implementation. With assessing the µCA-3, which was developed to accommodate a multibit sub-ranging ADC digitising the final TM stage output, a model of the TM-ARCHβ-7-12 ADC was employed.

Table 6-1 presents the test conditions of the TM-based ADC models employed in the MATLAB analysis. All the tests performed in MATLAB, except for a dynamic performance prediction test (Section 6.3.3), supplied the TM-based ADC model with a 0 - 3 V ramp input signal with the frequency set so $2^{(R+2)}$ samples were acquired for one ramp cycle. This relatively low ramp rate (when compared to the sampling frequency) would enable an ideal, TM-based ADC ($\mu_{\pm stage} = 2$) to sample every step change within the signal.

Test Conditions of MATLAB Models	Reason for Test Condition
Sampling frequency = 25 MHz	Inherited from the underlying TM-based ADC structure [56, 57] and matched those of the electronic TM-ARCHα-7 ADC implementation.
Valid input voltage range = 0 - 3 V	
$\mu_{\pm stage}$ values = random values over a $1.9 \leq \mu \leq 1.99$ range chosen by the rand() function in MATLAB	Range of µ chosen to be consistent with tests presented in Chapter 5. Random values chosen to prove the enhanced µCAs could compensate for non-matching $\mu_{\pm stage}$ values.
Resolution of Difference Measure (DM) values = non-integer decimal representation of the DM values was employed.	The DM were not converted to binary code format as the focus was to determine how the µCA-2 and µCA-3 improved the digital output bit accuracy of the TM-based ADCs.

Table 6-1: Summary of test conditions used in the MATLAB analyses.

The simulated and physical TM-ARCHα-7 ADC, for testing the VHDL implementation of the µCA-2, had mostly identical test conditions. The key difference was the DM values being converted to a binary format. With the VHDL simulation and practical testing, binary DM values with an r value of 10 bits were chosen as previous tests showed this r value achieved

the same improvement in bit accuracy for an TM-ARCHα-7 ADC as using theoretical DM values (see Table D-2 in Appendix D).

The MATLAB and VHDL code developed for the performance analysis is provided in Appendix C (where the code presented in each sub-section of Appendix C corresponds to the respective sub-section within in this chapter). Meanwhile, Appendix D.2 presents key data obtained during the practical implementation analysis presented in Section 6.6.

6.1 Initial Bit Accuracy Predictions of the Enhanced Tent Map Gain Compensation Algorithms

This section presents the bit accuracy predictions determined when testing the μCA-2 and μCA-3. The tests assessed whether the enhanced μCAs compensated for non-matching stage μ (μ_{stage}) and non-matching $\mu_{\pm}$, as well as the employment of a multibit, sub-ranging COTS ADC at the TM-based ADC output.

A more complex model of the TM-based ADCs, than was employed in Chapter 5, was implemented in MATLAB. This model took into consideration non-matching $\mu_{\pm stage}$, as well as the effects the power supply voltage may have on the TM circuits. The TM stages were implemented as shown in (4-9), to assess the enhanced μCA's ability to compensate for non-matching $\mu_{\pm stage}$. Section 6.2.1 presents the analysis on the μCA-2 using the TM-ARCHα-15 ADC model, while Section 6.2.2 gives the results for the μCA-3 with the TM-ARCHβ-7-12 ADC model.

6.1.1 Analysis of the µCA-2: Varying TM-stage Gain and Varying TM-slope Gain

For this test, the µCA-2 was applied to the TM-ARCHα-15 ADC output data, to compensate for non-ideal $\mu_{\pm stage}$, and the uncompensated and compensated bit accuracies were noted. The test was performed 100 times with different random $\mu_{\pm}$ combinations for each TM stage.

The µCA-2 was found to be effective at compensating for $\mu_{\pm stage}$ within the TM-ARCHα-15 ADC model. With this test, the bit accuracy was consistently increased to 14 bits (which equated to an improvement from the uncompensated bit accuracy of 9 to 10 bits). Figure 6-1 presents the quantisation error versus input voltage plots for the final test, which illustrates the dramatic reduction in absolute quantisation error, which went from 1184.5 LSBs uncompensated to 1.7 LSBs after compensation.

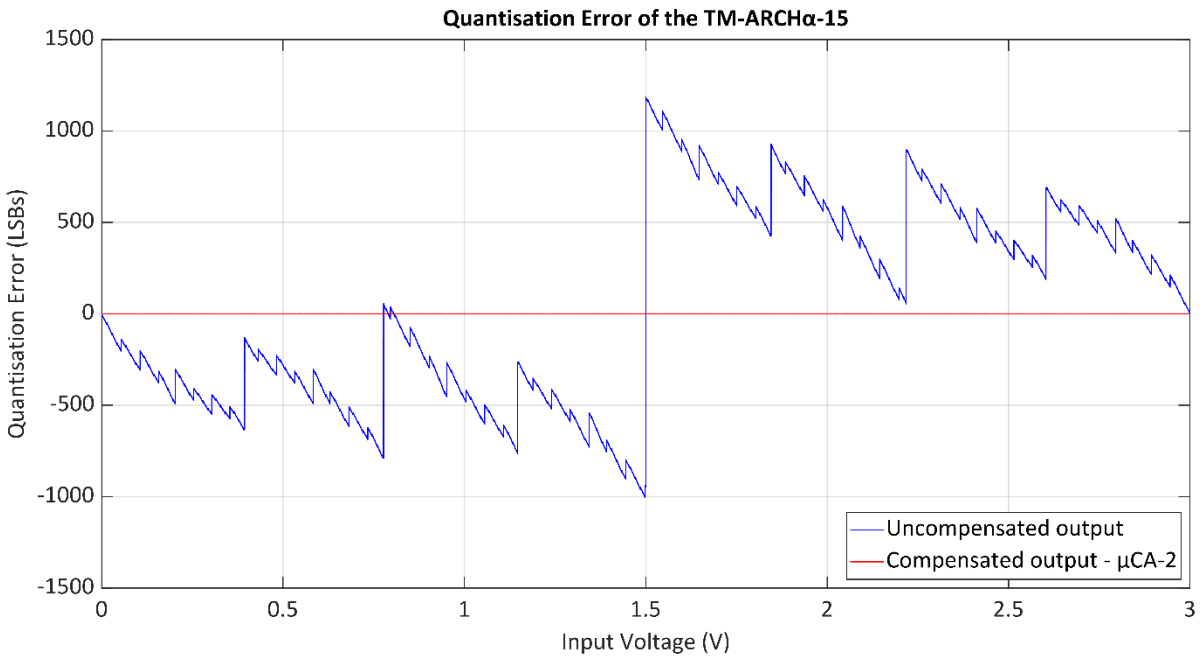


Figure 6-1: Comparison of quantisation error for TM-ARCHα-15 ADC model (with different slope gains) before and after compensation with the µCA-2.

6.1.2 Analysis of the µCA-3: Sub-ranging ADC Acquiring TM Stage Output

The TM-ARCHβ-7-12 ADC was then modelled and tested with the µCA-3. The same methods discussed in Sections 6.1.1 were employed to configure the non-matching $\mu_{\pm stage}$. Also, each of these tests were repeated 10 times with different random combinations of $\mu_{\pm stage}$, as a higher number of combinations exceeded the resource capacity of MATLAB.

Figure 6-2 presents the quantisation error versus input voltage plot of the TM-ARCHβ-7-12 ADC, for the final set of tests performed, before and after the µCA-3 was applied to the output digital data. The maximum absolute quantisation error was again found to have significantly reduced from 1184.5 LSBs, uncompensated to 1.6 LSBs after compensation. The results demonstrated that the µCA-3 was capable of consistently increasing the TM-ARCHβ-7-12 ADC bit accuracy to 17 bits for all 10 tests, from an uncompensated bit accuracy range of 4 to 6 bits.

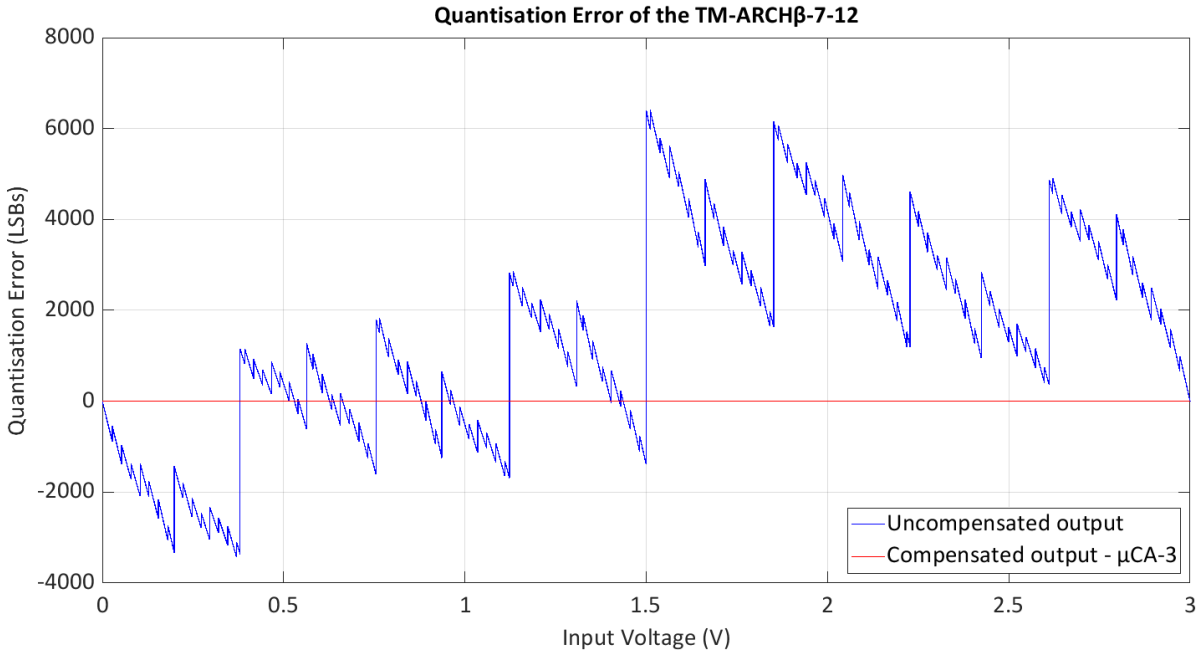


Figure 6-2: Quantisation error of the TM-ARCHβ-7-12 ADC model ((4-9) TM implementation with different $\mu_{\pm stage}$) before and after compensation.

6.2 Sensitivity Analysis of the Enhanced Tent Map Gain Compensation Algorithms

Section 6.1 highlights the effectiveness of μCA-2 and μCA-3 at compensating for $\mu_{\pm stage}$. However, the negative slope circuitry of the electronic TM function implementation (as shown by (4-3)) uses both μ_{+} and μ_{-}. This results in a Vref(μ+ - μ-) offset (where Vref is the partition point voltage) being introduced by the TM stage, then often amplified and reintroduced by successive TM stages. For the μCA-2 and μCA-3 to be effective with this TM implementation, there had to be a limit on the variation between the $\mu_{\pm stage}$.

The following two sub-sections present the sensitivity analysis tests performed to determine the maximum difference that could occur between $\mu_{\pm stage}$, as well as how much the $\mu_{\pm stage}$ employed by the μCA-2 and μCA-3 could deviate from μ_{ADC}, for both the TM-ARCHα-7 and TM-ARCHβ-7-12 ADCs. For the TM-ARCHβ-7-12 ADC, the bit accuracy of the compensated output needed to be at least 15 bits, to meet the UMS application specification (see Section 1.5). For the lower resolution TM-ARCHα-7 ADC, the purpose of the analysis was to establish the maximum difference between $\mu_{\pm stage,}$ as well as how much the $\mu_{\pm stage}$ employed by the μCA-2 could deviate from μ_{ADC} before the maximum possible compensated bit accuracy started diminishing.

6.2.1 Deviation Between $\mu_{\pm}$ Within the ADC

This sensitivity analysis investigated what the maximum difference between $\mu_{\pm stage}$ could be for the TM-ARCHβ-7-12 ADC to achieve a bit accuracy of 15 bits, after the μCA-3 had been applied to the digital output data. The maximum difference between $\mu_{\pm stage}$ for the lower resolution TM-ARCHα-7 ADC, used for testing the μCA-2, before the maximum possible compensated bit accuracy started diminishing was also established.

With testing the TM-ARCHβ-7-12 ADC, the μ_- values deviated from the μ_+ for each TM stage over a range of $-0.2 \times 10^{-3} \leq \Delta\mu_+ \leq +0.2 \times 10^{-3}$ (where $\Delta\mu_+$ defines the deviation from μ_+) in increments of 10×10^{-6}. For the TM-ARCHα-7 ADC, the set-up was identical, except the μ_- values deviated from the μ_+ for each stage over a range of $-0.1 \leq \Delta\mu_+ \leq +0.1$, in increments of 0.1×10^{-3}, as lower resolution TM-based ADCs are less sensitive to μ deviations.

Figure 6-3 and Figure 6-4 present the sensitivity analysis results for the TM-ARCHβ-7-12 ADC and TM-ARCHα-7 ADC respectively. The results highlight that the higher resolution TM-ARCHβ-7-12 ADC is more sensitive to deviation between $\mu_{\pm}$, despite an ideal ADC (rather than a series of additional TM stages) being employed to determine the last 12 bits of the digital output. This is because the COTS ADC is considered an ideal TM-based ADC by the μCA-3, thus the output of the 7th TM stage of the TM-based ADC is still effectively being amplified by multiples of 2. This results in the deviation from the ideal TM output becoming more pronounced with higher resolution TM-based ADCs.

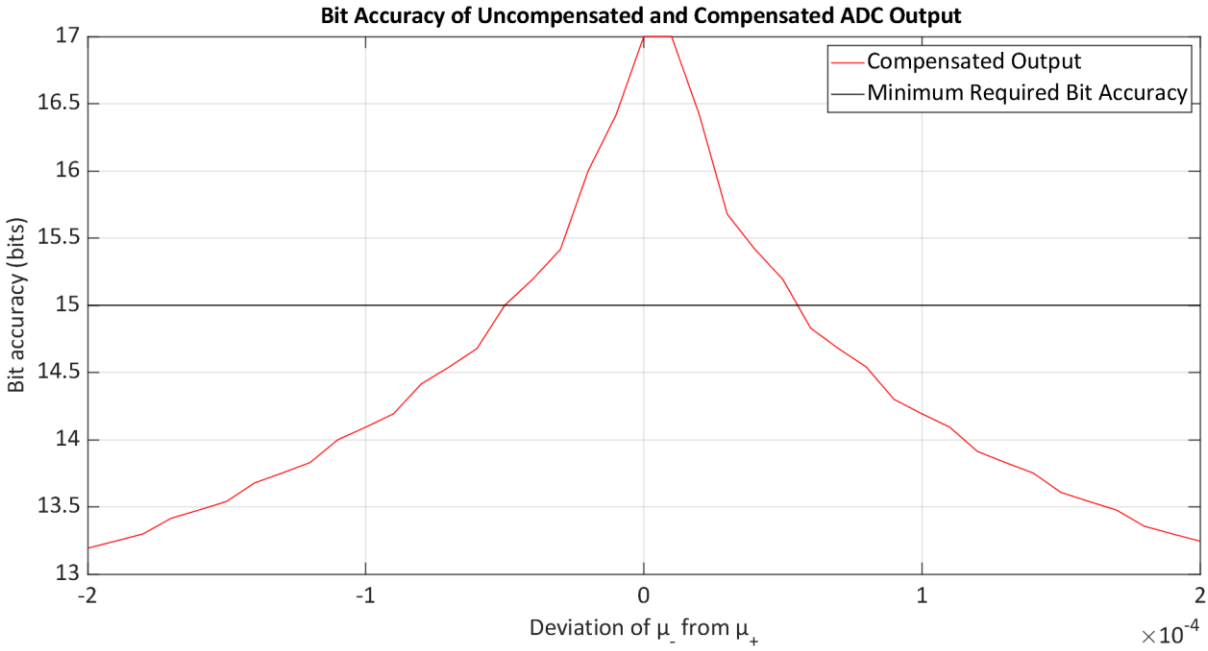


Figure 6-3: Sensitivity analysis of TM stage slope gain deviation within the TM-ARCHβ-7-12 ADC.

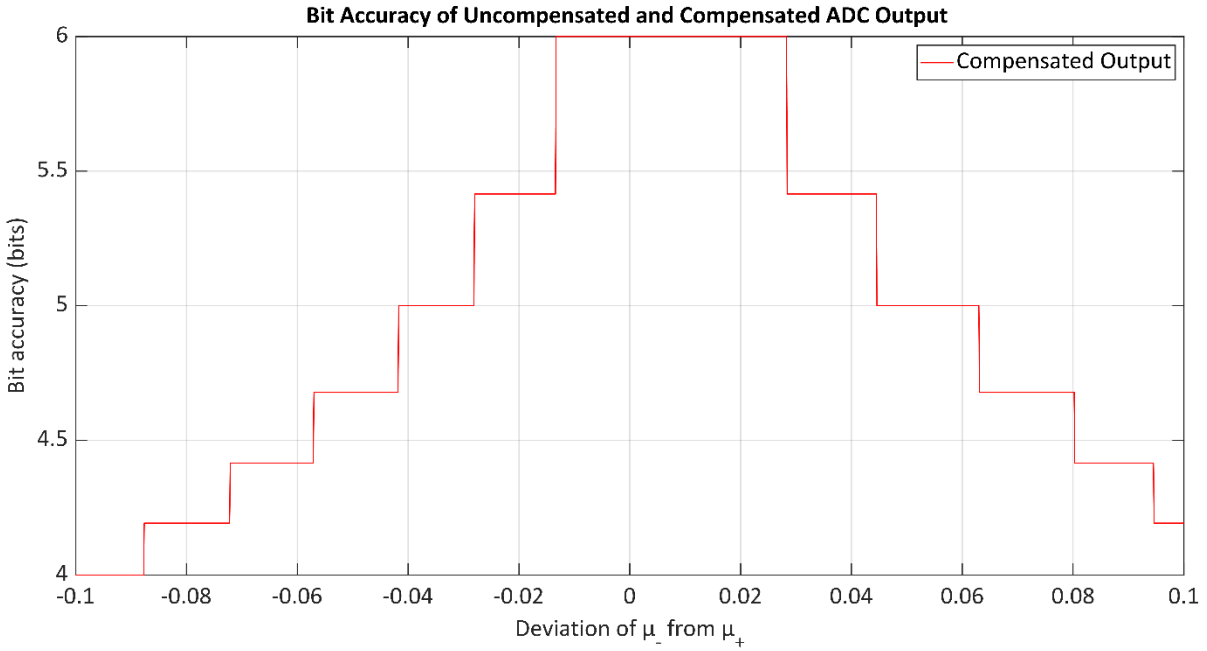


Figure 6-4: Sensitivity analysis of TM stage slope gain deviation within the TM-ARCHα-7 ADC.

For the UMS application the TM-ARCHβ-7-12 ADC needs to have a minimum bit accuracy of 15 bits. From the results presented in Figure 6-3, the maximum amount μ_- can deviate from

μ_+ is ± 50 x 10^{-6}. For the TM-ARCHα-7 ADC to maintain the maximum bit accuracy of 6 bits after compensation, the maximum μ_- can deviate from μ_+ is $-0.0133 \leq \Delta\mu_+ \leq +0.0283$ (see Figure 6-4).

6.2.2 Deviation in $\mu_\pm$ Employed by µCA-2 and µCA-3

Two sensitivity analysis tests to assess how accurately the $\mu_{\pm stage}$ needed to be determined for the µCA-2 and µCA-3, in order for the µCS to still be effective, were also performed (similar to the sensitivity analysis discussed in Section 5.3). The first test was for the TM-ARCHβ-7-12 ADC with the µCA-3, whilst the second test was for the TM-ARCHα-7 ADC with the µCA-2.

For the TM-ARCHβ-7-12 ADC, the μ_+ for the first 7 TM stages were again generated using the rand() function within the range $1.9 < \mu_+ < 1.99$. The extent the μ_- values could deviate from the µ+ values were also generated by the rand() function and the range of deviation was set to ± 50 x 10^{-6}, based on the results presented from the previous sub-section. With the TM-ARCHα-7 ADC the set-up was identical except the range µ- could deviate from μ_+ from was $-0.0133 \leq \Delta\mu_+ \leq 0.0283$.

The $\mu_\pm$ values employed by the µCA-3 ($\mu_{\pm algorithm}$) originally deviated from the actual $\mu_\pm$ values of the TM-ARCHβ-7-12 ADC ($\mu_{\pm ADC}$) by -0.1 %. The bit accuracy of the compensated TM-ARCHβ-7-12 ADC output was then recorded. The deviation of $\mu_{\pm algorithm}$ from $\mu_{\pm ADC}$ was then incremented by 0.001 % until the final deviation was +0.1 %. With the TM-ARCHα-7 ADC, the test procedure was identical except the range of deviations was increased to ±2 % for the µCA-2.

Figure 6-5 and Figure 6-6 present the bit accuracy versus deviations from $\mu_{\pm ADC}$ plots for the TM-ARCHβ-7-12 ADC and TM-ARCHα-7 ADC respectively. With the TM-ARCHβ-7-12 ADC (Figure 6-5), the maximum amount the $\mu_{\pm algorithm}$ values could deviate from the $\mu_{\pm ADC}$ values is -0.001 % to +0.003 %, in order to improve the uncompensated ADC output bit accuracy to 15 bits and meet the UMS specification. For the TM-ARCHα-7 ADC to achieve a bit accuracy of 6 bits after compensation, the maximum deviation is -0.491 % to +1.167 %. The results also show that the higher resolution TM-ARCHβ-7-12 ADC is still more sensitive to deviations between $\mu_{\pm algorithm}$ and $\mu_{\pm ADC}$, despite an ideal ADC determining the last 12 bits, compared to a series of cascaded comparators and TM stages.

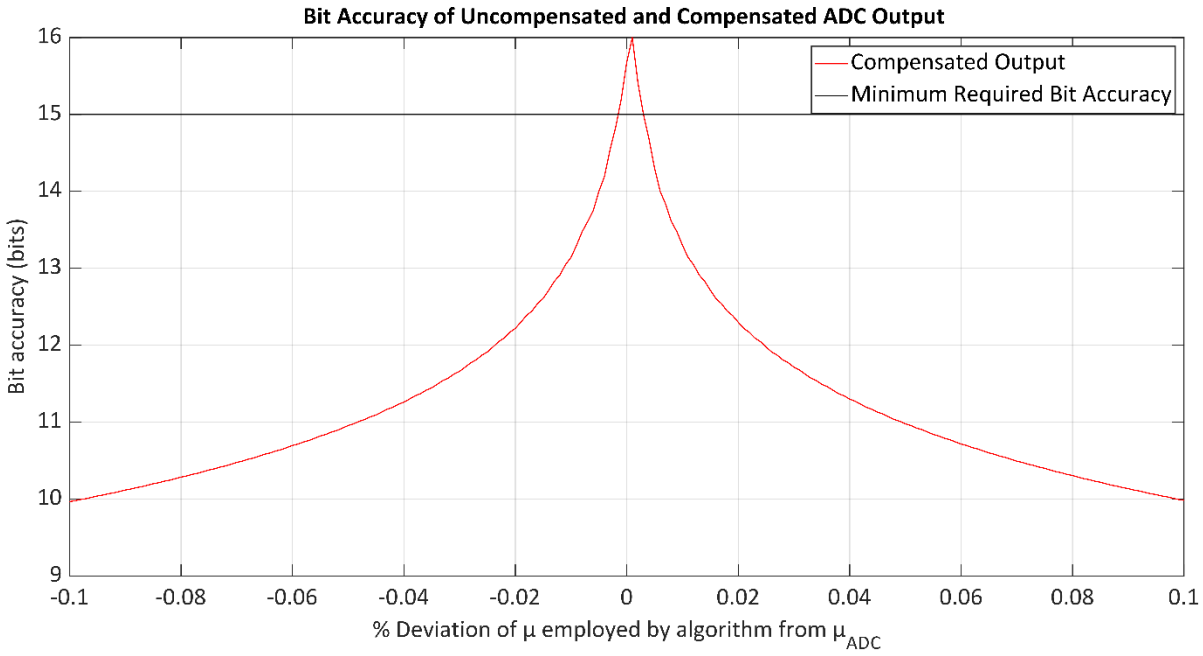


Figure 6-5: Sensitivity analysis of $\mu_{\pm algorithm}$ deviating from $\mu_{\pm ADC}$ for the TM-ARCHβ-7-12 ADC.

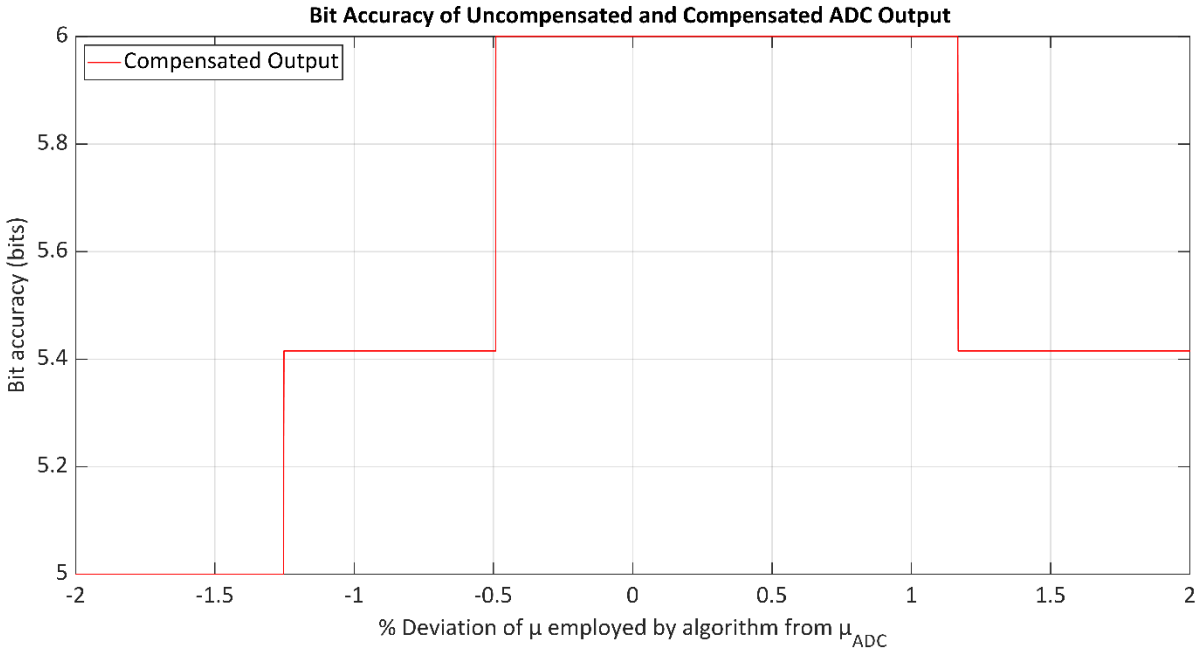


Figure 6-6: Sensitivity analysis of $\mu_{\pm algorithm}$ deviating from $\mu_{\pm ADC}$ for the TM-ARCHα-7 ADC.

6.3 Simulated Output Accuracy Analysis of the Adapted Tent Map Based ADC with the Enhanced Tent Map Gain Compensation Algorithm

The initial model of the TM-ARCHα-n ADC presented in Chapter 5 was refined to take into consideration the following non-ideal characteristics of the physical implementation of both the TM-ARCHα-n ADC and TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC structures:

- Non-matching μ_{stage};
- Non-matching $\mu_{\pm}$;
- The negative slope of each TM stage being created by both slope gains μ_{+} and μ_{-} (as highlighted in (4-3));
- A multibit sub-ranging COTS ADC being added to the final TM stage (TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC only); and

- The external hysteresis added to the comparators, which was set to be ± half a step size.

With the exception of the external hysteresis and the negative slope of each TM stage being created by both $\mu_{\pm}$, all the above characteristics were considered in the TM-based ADC models employed in Sections 6.1.2 to 6.2.2. This section presents the output accuracy analysis of the TM-ARCHβ-7-12 ADC model, which takes into consideration all the above non-ideal characteristics, and the μCA-3.

The analysis investigates the improvement in bit accuracy, as well as the changes in static and dynamic performance, after applying the μCA-3 to the digital output data from the TM-ARCHβ-7-12 ADC model and assesses whether this TM-based ADC structure met the UMS specification. Three sets of analysis tests were performed on the TM-ARCHβ-7-12 ADC, which were:

1. Bit accuracy predictions: confirmed bit accuracy is greater than, or equal to, 15 bits, after the μCA-3 is applied to the digital output data of the TM-ARCHβ-7-12 ADC.
2. Static performance predictions: analysed the static performance of the TM-ARCHβ-7-12 ADC before and after compensation, by determining the DNL, INL, offset error and gain error.
3. Dynamic performance predictions: examined the dynamic performance of the TM-ARCHβ-7-12 ADC, before and after compensation, by determining the SNR, SINAD, SFDR, THD and ENOB.

Based on the results obtain in the sensitivity analysis in Section 6.2.1, the maximum deviation between μ_- from μ_+ for each TM stage was set to $\pm$ 50 x 10^{-6}. With the dynamic performance

tests, faster sinusoidal input signals of 12.5 MHz and 1.25 MHz were provided (this selection being consistent with the tests presented in Section 5.2.3).

6.3.1 Bit Accuracy Predictions

The analysis performed on the most refined TM-ARCHβ-7-12 ADC model with the µCA-3, was similar to that presented in Section 6.1.2. With this analysis, only one test was performed, as previous tests suggested that the final bit accuracy after compensation was consistent.

Figure 6-7 presents the quantisation error in terms of LSBs before and after compensation, and Figure 6-8 presents the quantisation error after compensation only. The plot highlights the significant improvement in the TM-ARCHβ-7-12 ADC output accuracy in terms of representing the original input signal. The bit accuracy of the TM-ARCHβ-7-12 ADC increases from 5.81 bits to 15.68 bits (due to a reduction in the maximum absolute quantisation error from 4657 LSBs to 4.7 LSBs), showing the µCA-3 enabled this TM-based ADC design to meet the specification for the UMS application.

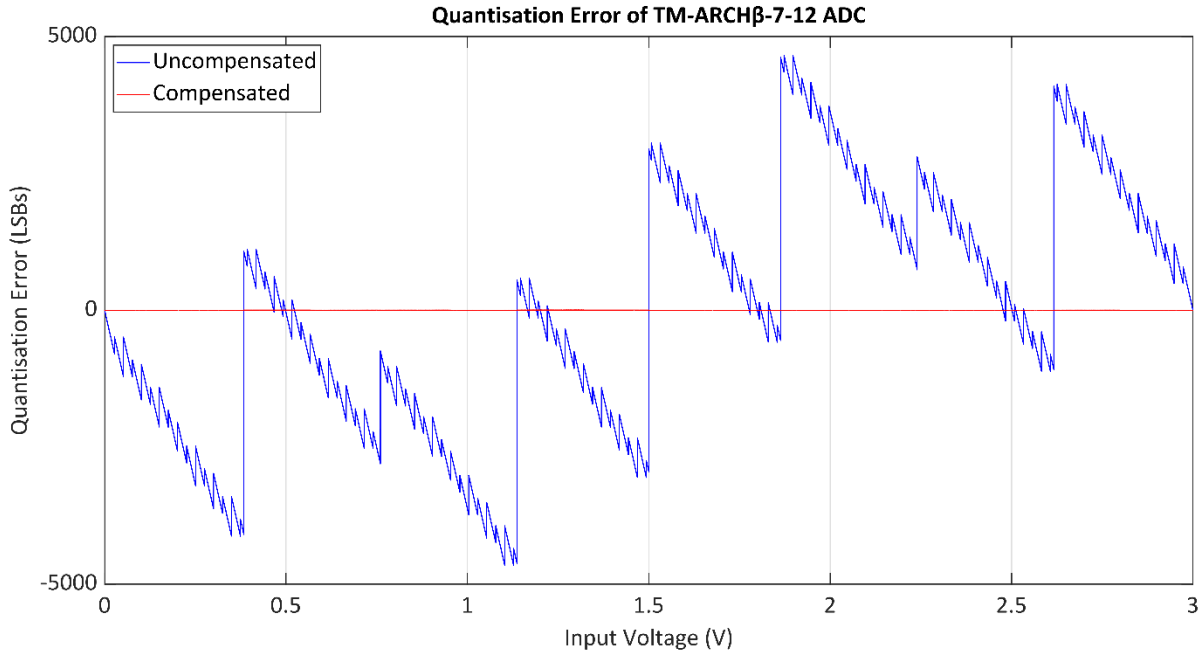


Figure 6-7: Quantisation error of the refined TM-ARCHβ-7-12 ADC model before and after compensation.

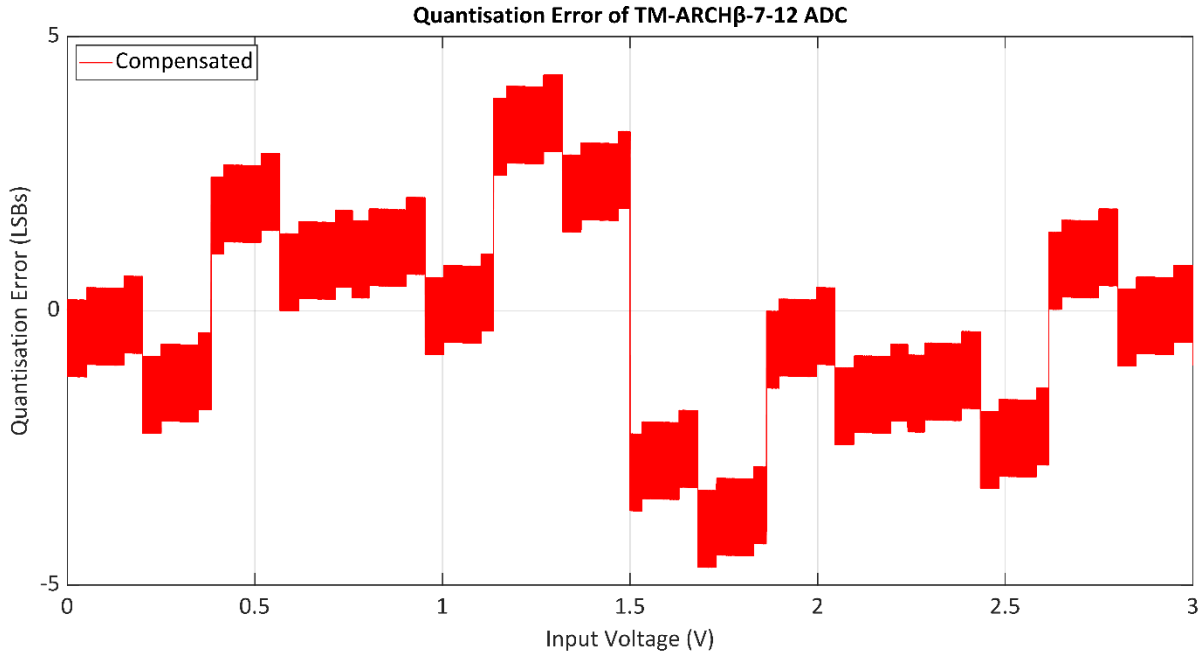


Figure 6-8: Quantisation error of the refined TM-ARCHβ-7-12 ADC model after compensation.

6.3.2 Static Performance Prediction

The same procedure detailed in Sections 5.2.2 was employed when establishing the DNL, INL, offset error and gain error of the TM-ARCHβ-7-12 ADC output digital data before and after compensation. The µCA-3 produced a significant reduction in magnitude with the maximum and minimum end-point INL error of approximately 13956 LSBs (the minimum INL went from -13971 LSBs to -14.25 LSBs, and the maximum INL fell from 13967.25 LSBs to 12 LSBs). This agrees with the results observed in the previous sub-section, as both bit accuracy and INL are a measure of how well the realised ADC output matches the ideal output. The offset error and gain error were both 0 LSBs before and after compensation, thus being unaffected by the non-ideal characteristics explored in this test.

With the maximum and minimum DNL error, a slight increase was observed for the maximum DNL error after compensation from 0.25 LSBs to 1.25 LSBs (the minimum DNL was maintained at -0.75 LSBs). As with the results observed in Section 5.2.2, this was due to the µCA-3 altering certain digital output codes to other binary values. However, the rate which the DNL increased was less than that observed in Section 5.2.2, despite the TM-ARCHβ-7-12 ADC having the higher resolution of 19 bits compared to the 16-bit TM-ARCHα-15 ADC model employed in the earlier test. This is because the COTS ADC acts as an ideal, 11 stage TM-based ADC, so the µ of these equivalent TM stages is 2 (whilst the µ of the first 7 stages was < 2). The high number of TM stages with a µ = 2 resulted in the DM values calculated for this TM-ARCHβ-7-12 ADC being smaller in value than those calculated for the TM-ARCHα-15 ADC in Section 6.1.1. Smaller DM values resulted in lower DV values, which in turn reduced the probability of the compensation values modifying those digital output codes to match other pre-existing digital combinations and resulted in a smaller increase in DNL.

6.3.3 Dynamic Performance Predictions

For this test the same procedure as described in Sections 5.2.3 was employed to find the dynamic parameters of the uncompensated and compensated ADC output. Different values for M and N were used to create the input frequency (see (2-5) in Section 2.1.2) due to higher resolution of the TM-ARCHβ-7-12 ADC. N was set to 209712 ($N=2^{(R+2)}$, where R equals 19 bits), while M was configured to 104851 and 1048573 which produced input frequencies of 1.25 MHz and 12.5 MHz respectively.

Figure 6-9 presents the results, before and after compensation, for the SNR, SINAD, SFDR and THD. The results highlight an increase in magnitude for all four of these parameters after the µCA-3 had been applied. This shows that the µCA-3 improved the performance of the TM-ARCHβ-7-12 ADC, as the ratio between the main output signal and noise distortion had increased, meaning the fundamental signal was more prominent than the noise and distortion, as the compensated signal more closely represents a sinusoidal signal. The improvement in THD was restricted when the Nyquist frequency was supplied due to the aliasing of the harmonics further distorting the ADC output signal.

The improvement in ENOB was found to have increased from approximately 6.5 bits to 16 bits for both input frequencies. These results confirm that the µCA-3 enables the TM-ARCHβ-7-12 ADC to meet the specification for the UMS over the required signal bandwidth, which requires the ADC to have a minimum resolution of 15 bits with input frequencies of up to 5 MHz.

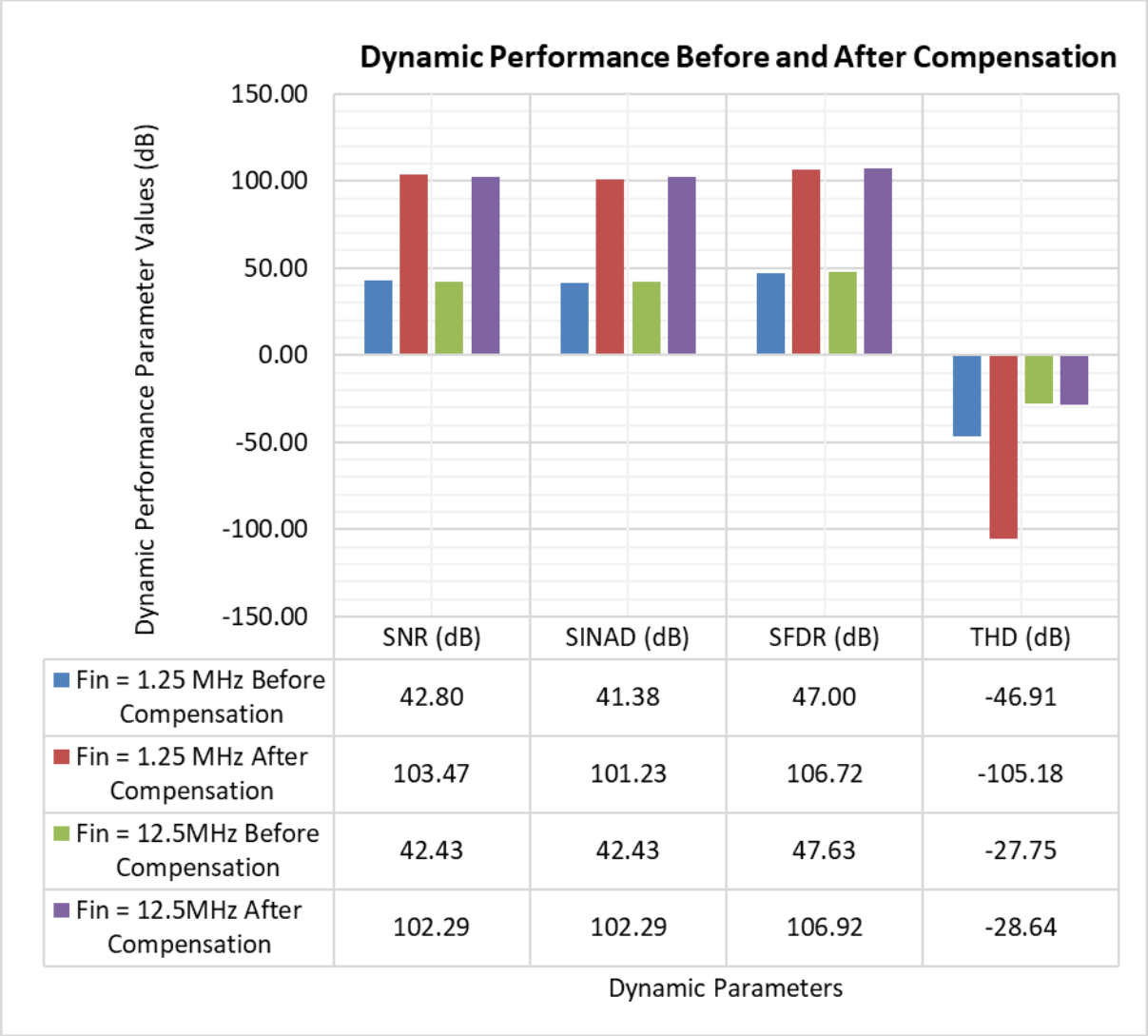


Figure 6-9: SNR, SINAD, SFDR and THD before and after compensation.

6.4 Noise Analysis Simulation

Internal noise within chaotic ADCs effects the performance more noticeably at higher resolutions [103]. Noise can be caused by: distortion in sampling circuits; output referred noise of each TM caused by internal noise of the TM circuit (e.g. thermal noise); long term drifts of circuit component parameters; and input noise [103]. Dominant errors are input noise and output referred noise, as techniques have been developed that reduce distortion within sampling circuits [111] as well as for estimating variation within the gains due to component long term drift, [112] which in turn enable μ compensation [2, 41, 42, 113] making these other two noise sources negligible [103].

For this reason, a simple analysis on the additional effects noise might have on the TM-ARCHβ-7-12 ADC output accuracy was performed. Figure 6-10 presents a block diagram summarising the set-up employed for the noise analysis. Noise was injected onto the input signal to the TM-ARCHβ-7-12 ADC (by superimposing white gaussian noise on the signal of interest), as well as onto the output signals of each TM stage. No noise was injected within the COTS ADC model as this was being simulated as an ideal ADC. The same magnitude of noise, which ranged from 0 to 4 step sizes (a step size is the equivalent of 1 LSB [38]) and was incremented in 0.5 step sizes, was superimposed on each signal of interest within the TM-ARCHβ-7-12 ADC model. To simplify modelling, output referred noise of each TM stage was modelled to have the same magnitude as the input noise.

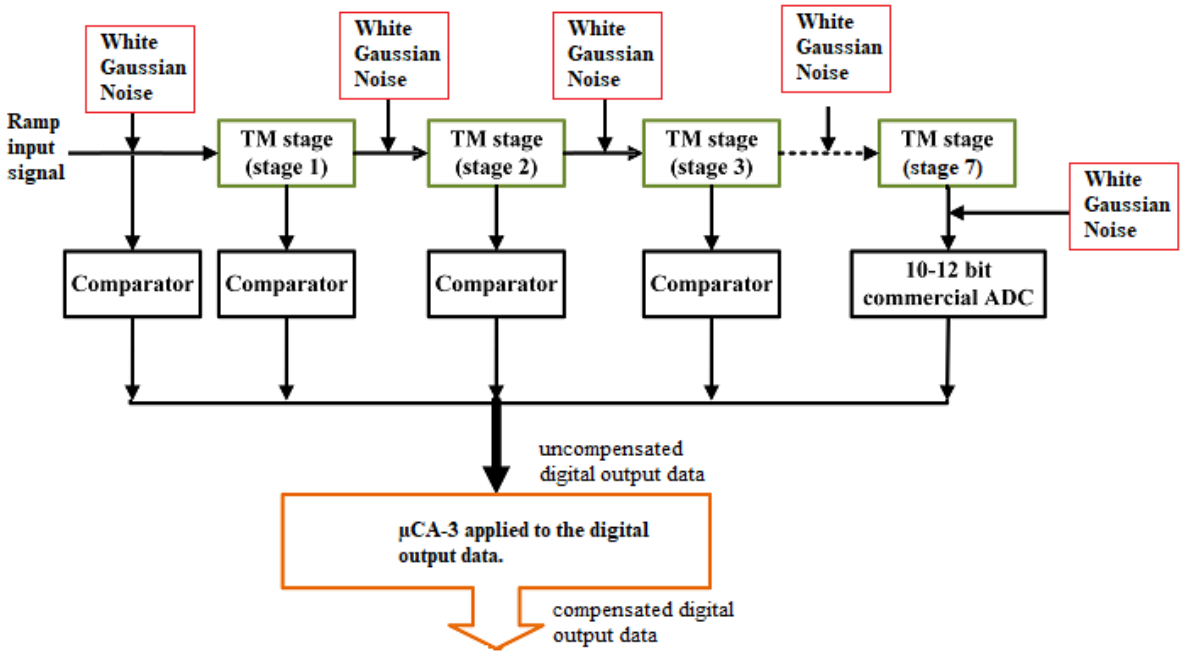


Figure 6-10: Block diagram of noise analysis test set-up.

The results demonstrated that even the addition of half a step size of noise reduces the improvement in bit accuracy after compensation from 15.7 to 15.4 bits. However, the compensated bit accuracy still met the UMS specification (≥ 15 bits), until the noise level exceeded 2 step sizes. Also, the uncompensated bit accuracy was unaffected by the noise range being investigated (consistently being 5.81 bits to 2 d.p.), showing that non-ideal µ has the more pronounced effect on the ADC output accuracy than noise across this noise range.

6.5 VHDL Implementation of an Enhanced Tent Map Gain Compensation Algorithm

A VHDL implementation of a µCS, comprising of the µCA-2, was implemented in VHDL code, as the plan was to initially test a TM-ARCHα-7 ADC, which has a comparator, instead of a sub-ranging COTS ADC, on the final TM stage. This µCS, developed to configure the same FPGA being employed to coordinate the operation of a TM-ARCHα-7 ADC, was tested using the same method to that detailed in Section 5.5.

A MATLAB script was used to select seven random µ+ values between 1.9 and 1.99 (one for each TM stage). The corresponding μ_{-} values for each TM stage were also generated at random and were configured to fall within the range of $-0.0133 \leq \Delta\mu_{+} \leq +0.0283$ (based on the maximum limits found from tests detailed in Section 6.2.1). The TM-ARCHα-7 ADC VHDL model (described in Section 5.5) was modified to emulate the predicted output, when comprising the generated $\mu_{\pm stage}$ values and supplied a ramp input signal. The DM values, used by the µCA-2 for each bit were also calculated from the generated $\mu_{\pm stage}$ values using (4-8), converted to binary, and hard coded as an array within the VHDL code.

Figure 6-11 presents a graph of the simulated VHDL module quantisation error, with and without compensation. The results show the implemented µCA-2 enhanced bit accuracy of the VHDL TM-ARCHα-7 ADC model, which increased from 5.42 bits to 6 bits. In addition, the µCA-2 processed each newly acquired sample of the input signal, while the next sample was obtained, confirming that real-time compensation of non-ideal and non-matching $\mu_{\pm stage}$ was still achievable with the µCA-2.

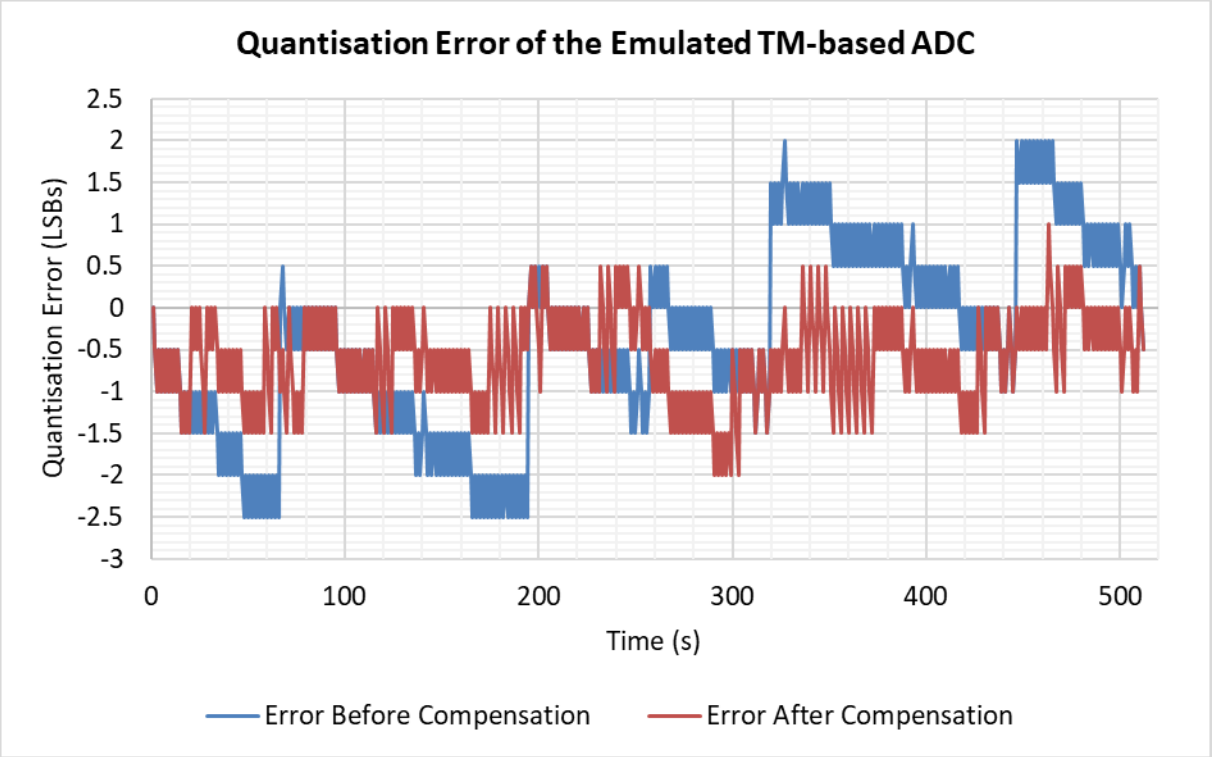


Figure 6-11: Quantisation error of the TM-ARCHα-7 ADC VHDL model before and after compensation.

6.6 Practical Implementation of a Tent Map Based ADC with an Embedded Tent Map Gain Compensation System

The µCS comprising the µCA-2 was tested with a physical, electronic implementation of a TM-ARCHα-7 ADC to confirm that real-time µ compensation was achievable. The VHDL implementation of the µCA-2 configured the same FPGA which was being used as part of an electronic implementation of the TM-ARCHα-7 ADC.

Appendix A.1 presents the schematic and list of components for the physical implementation of the TM-ARCHα-7 ADC. Initially the resistors which produced ΔR1, ΔR2 and ΔR3 (see (4-2)) were set to 0 Ω, before being changed to bring the $\mu_{\pm stage}$ below 2. The sensitivity analysis in Section 6.2.1 determined the $\mu_{\pm stage}$ should fall within the ideal $-0.0133 \leq \Delta\mu_{+} \leq +0.0283$ range,

however precisely determining these values was not possible, but all the $\mu_{\pm stage}$ values were less than 2.

The $\mu_{\pm stage}$ were determined by supplying the TM-ARCHα-7 ADC with DC input voltages and measuring the input and output voltages from each TM, along with the Vref voltage (the partition point voltage). The input voltage to the ADC was set so the input voltage, to the TM stage under observation, started at 0.7 V, then increased by 0.025 V, until 0.8 V was reached. Using (6-1) (which was derived from (1-2) [19]) and the TM input and output voltages measured (x_n and x_{n+1} respectively), μ+ (the positive slope gain) was calculated for each 0.025 V increment.

$$\mu_+ = \frac{x_{n+1}}{x_n}\,, \qquad x_n < Vref \tag{6-1}$$

The midrange of the five calculated μ_+ values was determined for each TM stage. This measure of centre was found to achieve the greatest improvement in the TM-ARCHα-7 ADC output accuracy (when compared to the average and median μ_+ values).

The above test was repeated to determine the μ- (the negative slope gain) value for the TM stage under consideration. The DC signal to the TM stage was set to 2.2 V and incremented to 2.3 V, again by 0.025V steps. (6-2) (also derived from (1-2) [19]) shows how μ- was calculated using the measured Vref voltage, as well as the calculated midrange μ_+ value for the same TM stage. The midrange value of the five μ_- values calculated for each TM stage was then determined.

$$\mu_- = \frac{(Vref \times \mu_+) - x_{n+1}}{x_n - Vref}\,, \qquad x_n > Vref \tag{6-2}$$

Table D-3 in Appendix D presents the final $\mu_{\pm stage}$ values determined; these were then used to calculate the DM values for the µCA-2 VHDL implementation. These $\mu_{\pm stage}$ values were also placed within a MATLAB model of the TM-ARCHα-7 ADC, which was used to make predictions on the digital output the ADC produces. The comparator hysteresis threshold voltages were also modified to reflect how the hysteresis had been implemented within the practical TM-ARCHα-7 ADC.

The TM-ARCHα-7 ADC was tested with the µCS. The FPGA controlling the analogue circuitry of the TM-ARCHα-7 ADC was originally programmed to generate the uncompensated output. A 0 V DC voltage signal was supplied to the ADC and the binary output captured before the FPGA was reprogrammed to generate the compensated output and the new binary output recorded. The DC input signal was then incremented by 0.1 V and the process described above repeated until the input signal reached 3 V.

Figure 6-12 presents the difference between the uncompensated output, ideal ADC output and the compensated output using the midrange $\mu_{\pm stage}$ values. The results show that the µCS reduces the difference between ADC and ideal output (the maximum deviation was reduced from ±7 LSBs to ±4 LSBs) and improves the bit accuracy of the electronic implementation of the TM-ARCHα-7 ADC from 4.19 bits to 5 bits.

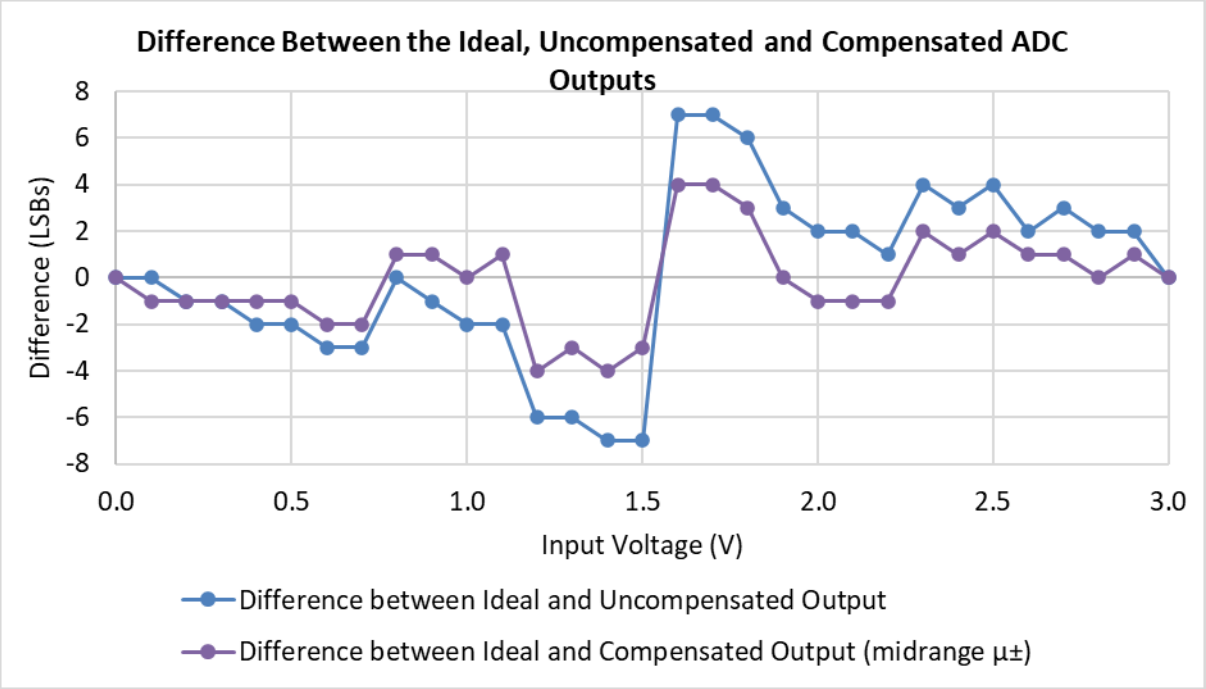


Figure 6-12: Plot between the physical TM-ARCHα-7 ADC output and ideal output, before and after compensation.

The µCA-2 improved the bit accuracy of the TM-ARCHα-7 ADC, although not to 6 bits, as predicted by MATLAB model. The reduced improvement in the ADC output accuracy is linked to the limited accuracy and precision when measuring the $\mu_{\pm stage}$ values. The difference between the uncompensated electronic TM-ARCHα-7 ADC output and the predicted uncompensated TM-ARCHα-7 ADC output (produced from the MATLAB model) was -2 to +3 bits, while for the compensated data the maximum difference was - 3 bits to + 4 bits. These results highlight that the $\mu_{\pm stage}$ values calculated from the measured TM voltages were not sufficiently accurate for the MATLAB model to predict the uncompensated digital output codes, nor for calculating the DM values required for the µCA-2. The $\mu_{\pm stage}$ values also seemed to vary depending on the amplitude of the TM input voltage, which may also have limited the ability of the embedded µCS to improve the TM-ARCHα-7 ADC output accuracy.

Limitations in getting the practical TM-ARCHα-7 ADC with μCA-2 to reach sufficiently high-quality performance restricted further work regarding the development of an electronic TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC implementation and a VHDL implementation of the μCA-3. However, the results have demonstrated that real-time embedded μ compensation within a TM-based ADC is feasible.

6.7 Summary

This chapter analysed the enhanced μCAs, μCA-2 and μCA-3, through MATLAB modelling. The TM-ARCHα-15 ADC and TM-ARCHα-7 ADC models were employed with the μCA-2 analysis, while a TM-ARCHβ-7-12 ADC model was used to assess the μCA-3. A VHDL implementation of the μCA-2 was also assessed via simulation using a TM-ARCHα-7 ADC structure, then tested through practical experiments.

Analysis of the μCA-2 and μCA-3 showed promising results in compensating for non-matching $\mu_{\pm}$ for each TM stage ($\mu_{\pm stage}$), over a range of $1.9 \leq \mu_{\pm stage} \leq 1.99$, within a TM-ARCHα-15 ADC. When μ_{+} produced the positive slope of the TMs and only μ_{-} was employed to create the negative slope within the TM-ARCHα-15 ADC, the compensated bit accuracy rose to 14 bits from an uncompensated bit accuracy of 9 to 10 bits. Meanwhile, the μCA-3, which can accommodate a multibit sub-ranging ADC acquiring the final TM stage output signal, was found to improve the compensated bit accuracy of the TM-ARCHβ-7-12 ADC structure to 17 bits, from an uncompensated bit accuracy of 4 to 6 bits.

The negative slope circuitry of the TM function employed in the physical TM-ARCHα-7 ADC (as shown by (4-3)) employs both μ_{+} and μ_{-}. For the μCA-2 and μCA-3 to be effective, a limit on the variation between the $\mu_{\pm}$ for each TM stage ($\mu_{\pm stage}$) was required to minimise the

resulting offsets. The sensitivity analysis in Section 6.2.1 suggested the maximum μ_- can deviate from μ_+ per TM stage, for the TM-ARCHβ-7-12 ADC, should be ± 50 x 10^{-6}, but lower resolution TM-based ADC designs can cope with a larger difference. The second sensitivity analysis presented in Section 6.2.2 highlighted that uncertainty in measuring the μ_{ADC} values needed to be minimised for μCA-2 and μCA-3 to be effective.

Simulation results show that the TM-ARCHβ-7-12 ADC can achieve the specification for the UMS through employing the μCA-3. The results highlighted an increase in bit accuracy from 5.81 bits to 15.68 bit, whilst the dynamic performance tests proved the ENOB, when the input frequency was greater than 5 MHz (the maximum frequency of the signals being employed within the UMS application), was also greater than the minimum requirement of 15 bits. The static performance tests also demonstrated a dramatic reduction in the INL error of 13956 LSBs. The increase in DNL error after compensation was lower than that observed for the compensated TM-ARCHα-15 ADC output in Chapter 5, due to the sub-ranging COTS ADC in the TM-ARCHβ-7-12 ADC acting as the equivalent of an ideal 12-bit TM-based ADC. This sub-ranging ADC reduced the level of compensation required as the equivalent TM stages were modelled with an ideal μ of 2, which in turn reduced the magnitude of the calculated DMs and the probability of digital levels being increased or decreased due to the compensation.

The noise analysis into the theoretical TM-ARCHβ-7-12 ADC model with the μCA-3 highlights that input signal noise and output referred noise to the TM-based ADCs reduces the improvement in bit accuracy. However, the simple noise analysis suggests noise less than two step sizes in amplitude enables the TM-ARCHβ-7-12 ADC to achieve a bit accuracy of 15 bits after compensation, as required by the UMS specification.

Finally, the simulated VHDL implementation of a TM-ARCHα-7 ADC with the μCA-2 improved the bit accuracy from 5.42 bits to 6 bits, whilst with the electronic implementation the bit accuracy was 4.19 bits uncompensated, and 5 bits compensated. With the electronic implementation, limitations in obtaining precise and accurate TM input and output voltage measurements restricted the ability in determining the $\mu_{\pm stage}$ values and thus achieving better compensation. Nevertheless, the results proved that embedded, real-time μ compensation of a physical TM-based ADC digital output data was possible.

The next chapter discusses the results from the simulated and practical TM-based ADC designs with the enhanced μCAs.

7 Discussion

Theoretical analysis and practical experimentation have demonstrated the feasibility of a real-time μCS for TM-based ADCs. A theoretical, mathematical model of a TM-ARCHβ-7-12 ADC with a μCS (comprising the μCA-3) has shown the bit accuracy can be increased to a minimum of 15 bits accuracy (achieving the requirement of detecting 100 μV signal variations), making this design viable to be employed within the DAQ system for the UMS application. Practical experimentation with a TM-ARCHα-7 ADC has confirmed that a μCS (comprising the μCA-2) can be embedded within the TM-based ADC and perform real-time compensation for non-ideal μ on the digital output data. Three techniques were also created, whilst developing the compensation algorithm, for coping with: non-matching TM stage μ; non-matching $\mu_{\pm stage}$; and a multibit sub-ranging ADC converting a TM output signal to the digital domain rather than a single bit producing comparator.

Simultaneously meeting high resolution, high sampling speed, high conversion speed, low power and low fabrication area is challenging, especially at the extremes of a particular category. Different TM-based ADC structures have been shown to have potential in reducing combinations of these trade-offs [13, 16, 56, 57], but a key limitation of such ADCs is the impact non-ideal μ within the TM circuitry has on the output accuracy. This research has developed a novel algorithm, which can be embedded within TM-based ADCs, to perform real-time μ compensation on the digital output data, to counteract this problem.

As part of this work, the viability of a standalone TM-based ADC, with an embedded digital implementation of a μCA, for the employment within a DAQ system for a UMS application was assessed. This work also demonstrates the feasibility of employing TM-based ADCs within other measurement systems requiring small signal variations to be consistently detected

across a relatively large signal range. The UMS application required the TM-based ADC to have a bit accuracy of 15 bits after compensation. The mathematical model development of a TM-ARCHβ-7-12 ADC structure (consisting of a 7 TM stage ADC with a 12-bit COTS ADC to acquire the final TM output) emulated the operational performance of an electronic implementation and demonstrated that the improvement in bit accuracy, after the applied μCA-3, was sufficient to meet the UMS application specification. The VHDL implementation of the μCA-1 and μCA-2 also proved compensating for non-ideal μ, without the requirement of offline computational post-processing, was viable. Instead, simulation and electronic experimentation proved a μCS can be embedded within a TM-based ADC and perform real-time compensation on the output digital data prior to transmission. These tests, along with a mathematical model, demonstrated the viability of a physical DAQ system which employs a stand-alone TM-based ADC with embedded μ compensation.

Techniques were also developed and proven, through a combination of theoretical simulation and electronic implementation, to compensate for non-matching TM stage μ, as well as non-matching $\mu_{\pm stage}$ values. A μ compensation technique for when the TM output was digitised using a multibit ADC, rather than a single bit producing comparator, was also developed. These three techniques enable a variety of TM-based ADC structures to be compensated for non-ideal μ. The μCA by Basu, in addition to not being performed prior to the ADC data being transmitted, was only suitable for a feedback TM-based ADC configuration with a comparator acquiring the previous output (as shown in Figure 2-12), as this enabled the μ for each iteration (equivalent to each TM stage in a series configuration) to be constant [41, 42]. The μCA-2 and μCA-3, however, can be employed to compensate for non-ideal μ within series and feedback TM-based ADC configurations that employ both comparators and sub-ranging ADCs to digitise TM input and output signals.

The μCA and techniques developed during this research have demonstrated the viability of employing TMs for analogue to digital data conversion. Due to how the TM circuits were electronically implemented in the TM-based ADC design developed by Upton [56, 57] (shown in (4-3)) , the TM negative slope circuitry introduced an offset of Vref(μ+ - μ-), which was often amplified by successive TM stages. For the enhanced μCAs to be effective at compensating for non-matching $\mu_{\pm stage}$ values, the difference between the $\mu_{\pm stage}$ values had to fall within an acceptable tolerance (established via the sensitivity analysis in Section 6.2.1), which becomes more critical at higher resolutions. With higher resolution TM-based ADCs, there may also be a case for fabricating the design onto silicon, enabling the $\mu_{\pm stage}$ to be better matched (when compared to a discrete component implementation), alleviating the limitation posed by the offset introduced. Another solution is investigating different circuit configurations of TM implementations whose operation match that shown in (4-9), as better compensation for non-matching $\mu_{\pm stage}$ values with this circuit configuration can be achieved. Alternatively, a technique to compensate for this offset could be investigated.

The sensitivity analyses in Section 6.2.2 highlighted the need to significantly reduce the difference between $\mu_{\pm ADC}$ and $\mu_{\pm algorithm}$ for higher resolution TM-based ADCs, for the enhanced μCAs to remain effective. This requires measuring the $\mu_{\pm stage}$ values of the TM-based ADC accurately and precisely, which needs to be achieved with the electronic TM-ARCHα-7 ADC, before the μCA-3 can be implemented in VHDL and trialled with the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC, tested in theoretical simulations. For a discrete component implementation of the TM-based ADC, using higher precision measurement equipment and the method to determine $\mu_{\pm stage}$ values described in Section 6.6 might be acceptable. Methods of further reducing noise in the TM-based ADC would also improve measurement precision and thus the accuracy of the $\mu_{\pm stage}$ measurements. Noise reducing methods could include reducing

switching noise by lowering the sampling speed (involves adjusting the sample and hold capacitors and resistors) or redesigning the PCB to enable further noise reduction. If the TM-based ADC was fabricated as an IC, this method of establishing the $\mu_{\pm stage}$ values maybe infeasible as the analogue signals to and from the TM circuits might not be accessible for measurement. Therefore, research into employing the output digital data, such as exploring the adaptation of the method suggested by Dutta [112], to determine the $\mu_{\pm stage}$ values might be necessary for this application.

As an aside, work in the field of TM-based ADCs and estimating the initial input signals and μ parameters though the digital output data [2, 41, 42, 112, 114] may have potential applications in the field of chaotic encryption and decryption. Multiple encryption and decryption systems have been developed which employ the chaotic TM [115] or a combination of the TM and other chaotic maps [116, 117]. An investigation into the employment of the μCAs from this work, to create a more efficient encryption and decryption process which requires minimal computation, could be undertaken.

Overall, this work has advanced the viability of TMs being employed for analogue to digital data conversion. This in turn will enable an improvement in trade-offs between speed, power consumption and circuit area at higher resolutions, depending on the TM-based ADC structure chosen. Furthermore, this research has demonstrated the potential in employing TM-based ADCs within measurement systems which required small signal variations to be consistently detected across a relatively large signal range.

8 Conclusion and Further Work

8.1 Conclusions

Multiple advancements with compensating for loss in the output accuracy of tent map (TM) based analogue to digital converters (ADCs), caused by non-ideal TM gain (μ), have been demonstrated. This involved the development of a stand-alone TM-based ADC with an embedded μ compensation system (μCS) comprising a novel μ compensation algorithm (μCA). The μCA developed required much lower levels of computation resources, compared to a past solution proposed to compensate for non-ideal μ, enabling a compensation system to be embedded within a TM-based ADC.

The TM-based ADC and μCS, which was developed for a specific type of measurement system application, also demonstrated the viability of employing this data converter within high precision and high accuracy measurement systems. Furthermore, the refinements made to the μCA during this project has enabled the production of μCS for different TM-based ADC structures, making the notion of employing TM-based ADCs in different applications more obtainable.

The main conclusions are:

1. The viability of employing a TM-based ADC, with an embedded μCS, within a data acquisition (DAQ) system, for an ultrasonic measurement system (UMS), was investigated using MATLAB simulation and practical testing.
2. Theoretical analysis using MATLAB has demonstrated the output accuracy of a TM-based ADC is affected when the μ of each TM stage was less than two.

3. A novel μCA was proposed, which compensated for non-ideal μ through using the Gray code output data produced by the TM-based ADC, to aid the calculations for which compensation values needed to be added/subtracted from the digital output data, to produce a more accurate digital representation of the original analogue signal which was sampled and converted.
4. Three other techniques were developed to enhance and enable the novel algorithm to be adapted for employment with a variety of TM-based ADC structures. These techniques compensated for:
    - non-matching TM stage μ;
    - non-matching slope μ for each TM stage ($\mu_{\pm stage}$); and
    - non-ideal μ when the digital data produced from the TM stages was from a sub-ranging ADC rather than comparators.
5. A mathematical model was developed to assess the performance of a TM-based ADC (consisting of 7 comparator and TM stages and a 12-bit COTS ADC to digitise the final TM stage output) after the μCA processes the digital data. Results showed an increase in bit accuracy from 5.81 bits to 15.68 bits, which met the specification for the UMS DAQ system. Noise has a negative effect on the ADC output accuracy, but amplitudes below two ADC step sizes were tolerable.
6. The VHDL implementation of the μCA and practical experiments proved embedded real-time μ compensation was possible and demonstrated the concept of developing a standalone TM-based ADC with embedded compensation.
7. Challenges need to be overcome to realise physical TM-based ADC for the UMS application, but the results obtained have proven the viability and concept of employing a TM-based ADC, with an embedded μCS, to perform higher resolution data

conversion within a measurement system which requires small signal variations to be consistently detected across a relatively large signal range.

8.2 Future Work

The following is a list of suggestions of proposed further work:

- Fabricate an electronic implementation of the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC, with an embedded µCS comprising the µCA-3, onto silicon. The fabrication of this TM-based ADC with µCS design will result in better matched TM slope gains that are closer to the ideal value of 2. This will enable better ADC output accuracy after compensation and enable higher resolution TM-based ADC designs to be realised.
- Investigate the possibility of enhancing the µCA further by developing an algorithm which can establish the TM stage and slope gains from the TM-based ADC output, enabling an auto calibrating µCS to be produced. This would enable a more effective µCS to be deployed with a TM-based ADC, because the comprising µCA would be more immune to TM stage and slope gains varying over time (due to the resistors which set the µ values drifting over time and with temperature).
- Investigate the possibility of applying the µCAs developed to enhance TM-based encryption and decryption systems. Some encryption and decryption processes have employed TM functions within the process, and this work might enable a system to be produced with reduced computational requirements.

9 References

[1] P. Hazell, P. Mather, A. Longstaff, and S. Fletcher, "A Non-linear Tent Map-Based ADC Design for Sensitive Condition Monitoring Measurement Systems," presented at the COMADEM 2019, Huddersfield, England, Sept. 3-5, 2019, 2020.

[2] P. Hazell, P. Mather, A. Longstaff, and S. Fletcher, "Digital System Performance Enhancement of a Tent Map-Based ADC for Monitoring Photovoltaic Systems," *Electronics,* vol. 9, no. 9, Sept. 2020, Art no. 1554, doi: https://doi.org/10.3390/electronics9091554.

[3] A. B. Borisov and V. V. Zverev, M. Efroimsky, L. Gamberg, D. Gitman, A. Lazarian, and B. Smirnov, Eds. *Nonlinear Dynamics: Non-Integrable Systems and Chaotic Dynamics* (De Gruyter Studies in Mathematical Physics no. 36). Berlin, Germany: De Gruyter Inc., 2016.

[4] B. Razavi, *Principles of data conversion system design*. Pisataway, NJ, USA: IEEE Press, 1995.

[5] J. Park and S. Mackay, *Practical Data Acquisition for Instrumentation and Control Systems*. Oxford, UK: Elsevier Sci. & Technol., 2003.

[6] Maxim Integrated, "Tutorials 283 INL/DNL Measurements for High-Speed Analog-to-Digital Converters (ADCs)," Maxim Integrated, San Jose, CA, USA, Nov. 2001. Accessed: Jan. 5, 2021. [Online]. Available: https://www.maximintegrated.com/en/design/technical-documents/tutorials/2/283.html

[7] C. Pearson, "High-Speed, Analog-to-Digital Converter Basics," Texas Instruments, Dallas, TX, USA, Jan. 2011. Accessed: May. 24, 2021. [Online]. Available: https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/73/High_2D00_Speed_2C00_-Analog_2D00_to_2D00_Digital-Converter-Basics.pdf

[8] P. G. A. Jespers, *Integrated Converters - D to A and A to D Architectures, Analysis and Simulation*. NJ, USA: Oxford Univ. Press, 2001.

[9] N. Storey, *Electronics: a systems approach*, 5th ed. Harlow, UK: Pearson, 2013.

[10] M. M. R. Mano, *Digital Design: With an Introduction to the Verilog Hdl, Vhdl, and Systemverilog*, 6th ed. NY, USA: Pearson Educ., 2018.

[11] S. Sangwine, *Electronic components and technology*, 2nd ed. (Tutorial Guides in Electronic Engineering, no. 13). London: Chapman & Hall, 2007.

[12] W. Bolton, *Programmable Logic Controllers*, 4th ed. Jordan Hill, London, UK: Elsevier Sci. & Technol., 2006.

[13] J. Liu, X. Zhang, Z. Li, and X. Li, "A tent map based A/D conversion circuit for robot tactile sensor," *J. Sensors,* vol. 2013, Aug. 2013, doi: 10.1155/2013/624981.

[14] R. del Rio and J. M. de la Rosa, *CMOS Sigma-Delta Converters: Practical Design Guide*. Somerset, UK: John Wiley & Sons, Inc., 2013.

[15] A. M. A. Ali, *High speed data converters*. London, UK: The IET, 2016.

[16] S. Berberkic, "Measurement of small signal variations using one-dimensional chaotic maps," Ph.D. dissertation, Dept. Eng. and Technol., Univ. Huddersfield, Huddersfield, England, 2014. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/23737/

[17] P. J. Ashenden, *The student's guide to VHDL*, 2nd ed. Burlington, MA, USA: Morgan Kaufmann, 2008.

[18] T. Kapitaniak, *Chaos for Engineers: Theory, Applications, and Control*, 2nd ed. Berlin, Germany: Springer, 2000.
[19] G. L. Baker and J. P. Gollub, *Chaotic Dynamics: an introduction*, 2nd ed. Cambridge, UK: Cambridge Univ. Press, 1996.
[20] A. S. Morris, *Principles of measurement and instrumentation*, 2nd ed. Hemel Hempstead, UK: Prentice Hall, 1993.
[21] M. Parker, *Digital Signal Processing 101: Everything You Need to Know to Get Started*. Saint Louis, MO, USA: Elsevier Science & Technology, 2010.
[22] R. Gilmore and M. Lefranc, *The topology of chaos: Alice in Stretch and Squeezeland*, 2nd ed. Weinheim, Germany: Wiley-VCH, 2011.
[23] T. L. Floyd, *Electronic devices: conventional current version*, 8th ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2008.
[24] T. S. El-Ali, *Discrete systems and digital signal processing with MATLAB*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2012.
[25] R. v. d. Plassche, *Integrated analog-to-digital and digital-to-analog converters* (The Kluwer International Series in Engineering and Computer Science: Analog Circuits and Signal Processing). Dordrecht, The Netherlands: Kluwer Academic Publishers, 1994.
[26] B. Baker, "How delta-sigma ADCs work, Part 2," *Analog Appl. J.,* vol. 2011, no. 4Q, p. 5, 2011.
[27] Maxim Integrated, "Tutorials 1041 Understanding Integrating ADCs," Maxim Integrated, San Jose, CA, USA, May. 2, 2002. Accessed: Jul. 31, 2021. [Online]. Available: https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1041.html
[28] Maxim Integrated. "Tutorials 748 The ABCs of Analog to Digital Converters: How ADC Errors Affect System Performance." Maxim Integrated. https://www.maximintegrated.com/en/design/technical-documents/tutorials/7/748.html (accessed May. 31, 2021).
[29] M. P. Kennedy, "A Nonlinear Dynamics Interpretation of Algorithmic A/D Conversion," *Int. J. Bifurcation and Chaos,* vol. 5, no. 3, pp. 891 - 893, Jun. 1994.
[30] The MathWorks Inc. "MATLAB." The MathWorks, Inc. https://uk.mathworks.com/products/matlab.html (accessed Jul. 28, 2021).
[31] Intel. "ModelSim*-Intel® FPGA Edition Software " Intel. https://www.intel.co.uk/content/www/uk/en/software/programmable/quartus-prime/model-sim.html (accessed Jul. 28 2021, 2021).
[32] C.-T. Chen, *Signals and systems*, 3rd ed. NJ, USA: Oxford University Press (in English), 2004.
[33] K. C. Smith and A. Sedra, "The current conveyor—A new circuit building block," *Proc. IEEE,* vol. 56, no. 8, pp. 1368-1369, Aug. 1968, doi: 10.1109/PROC.1968.6591.
[34] T. Kapitaniak, K. Zyczkowski, U. Feudel, and C. Grebogi, "Analog to digital conversion in physical measurements," *Chaos, Solitons and Fractals,* vol. 11, no. 8, pp. 1247-1251, 2000, doi: 10.1016/S0960-0779(99)00003-X.
[35] M. E. Waltari and K. A. I. Halonen, *Circuit Techniques for Low-Voltage and High-Speed A/D Converters* (The Springer International Series in Engineering and Computer Science). NJ, USA: Kluwer Academic Publishers, 2002.
[36] W. Kester, "MT-001 TUTORIAL Taking the Mystery out of the Infamous Formula, "SNR = 6.02N + 1.76dB," and Why You Should Care," Analog Devices, 2009. Accessed: Feb.

25, 2021. [Online]. Available: https://www.analog.com/media/en/training-seminars/tutorials/MT-001.pdf

[37] Atmel Corporation, "AVR121: Enhancing ADC resolution by oversampling," Microchip, [Online], Sept. 2005. Accessed: Oct. 2, 2018. [Online]. Available: https://www.microchip.com/content/dam/mchp/documents/OTH/ApplicationNotes/ApplicationNotes/doc8003.pdf

[38] T. Ndjountche, *CMOS Analog Integrated Circuits : High-Speed and Power-Efficient Design*. Boca Raton, FL, USA: Taylor & Francis Group, 2011.

[39] S. Louwsma, E. Tuijl, and B. Nauta, *Time-interleaved analog-to-digital converters* (Analog Circuits and Signal Processing). Springer, 2010.

[40] S. Kocis and Z. Figura, *Ultrasonic measurements and technologies* (Sensor physics and technology, no. 4). London, UK: Chapman & Hall, 1996.

[41] R. Basu, "Estimation of Input Variable as Initial Condition of a Chaos Based Analogue to Digital Converter," Ph.D. dissertation, Dept. Eng. and Technol., Univ. Huddersfield, Huddersfield, England, 2018. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/34821/

[42] R. Basu, D. Dutta, S. Banerjee, V. Holmes, and P. Mather, "An Algorithmic Approach for Signal Measurement Using Symbolic Dynamics of Tent Map," *IEEE Trans. Circuits and Syst. I: Regular Papers,* vol. 65, no. 7, pp. 2221-2231, Jul. 2018, doi: 10.1109/TCSI.2017.2773202.

[43] S. Bashir, S. Ali, S. Ahmed, and V. Kakkar, "Analog-to-digital converters: A comparative study and performance analysis," in *2016 Int. Conf. Comput., Commun. and Automat. (ICCCA)*, Greater Noida, India Apr. 29-30 2016: IEEE, pp. 999-1001, doi: 10.1109/CCAA.2016.7813861.

[44] Texas Instruments, "ADC368x 18-bit 0.5 to 65-MSPS Low Noise Ultra-low Power Dual Channel ADC," Texas Instruments, Dec. 2020. Accessed: Jul. 22, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/adc3683.pdf?ts=1649495070972&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FADC3683

[45] Analog Devices, "20-Bit, 1.8 MSPS/1 MSPS/500 kSPS, Easy Drive, Differential SAR ADCs Data Sheet AD4020/AD4021/AD4022 " Analog Devices, Norwood, MA, USA, Feb. 2021. Accessed: Jul. 22, 2021. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad4020-4021-4022.pdf

[46] Texas Instruments, "ADS1212 ADS1213 22-Bit ANALOG-TO-DIGITAL CONVERTER," Texas Instruments, Dallas, TX, USA, Feb. 2004. Accessed: Jul. 22, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/ads1213.pdf?ts=1649495246037&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FADS1213

[47] Texas Instruments, "ADS1675 4MSPS, 24-Bit Analog-to-Digital Converter," Texas Instruments, Dallas, TX, USA, Aug. 2010. Accessed: Jul. 22, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/ads1675.pdf?ts=1649495305602&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FADS1675

[48] Texas Instruments, "ADS1282 High-Resolution Analog-To-Digital Converter," Texas Instruments, Dallas, TX, USA, Mar. 2015. Accessed: Jul. 22, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/ads1282.pdf?ts=1649424981595&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FADS1282

[49] Analog Devices, "LTC2500-32 32-Bit Oversampling ADC with Configurable Digital Filter," Analog Devices, USA, Jan. 2018. Accessed: Jul. 22, 2021. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/250032fb.pdf

[50] Microchip Technology Inc., "MCP3422/3/4 18-Bit, Multi-Channel ΔΣ Analog-to-Digital Converter with I2C™ Interface and On-Board Reference," Microchip Technology Inc., USA, Mar. 2009. Accessed: Jul. 22, 2021. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/22088c.pdf

[51] Linear Technology, "LTC2430/LTC2431 20-Bit No Latency ΔΣTM ADCs with Differential Input and Differential Reference," Linear Technology, Milpitas, CA, USA, 2002. Accessed: Jul. 22, 2021. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/24301f.pdf

[52] Microchip Technology Inc., "MCP3550/1/3 Low-Power, Single-Channel 22-Bit Delta-Sigma ADCs," Microchip Technology Inc., USA, Jul. 2014. Accessed: Jul. 22, 2021. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/20001950F.pdf

[53] Linear Technology, "LTC2413 24-Bit No Latency ΔΣ™ ADC, with Simultaneous 50Hz/60Hz Rejection," Linear Technology, Milpitas, CA, USA Aug. 2000. Accessed: Jul. 22, 2021. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/2413fa.pdf

[54] Texas Instruments, "ADS1287 Low-Power, 1000-SPS, Wide-Bandwidth, Analog-to-Digital Converter With Programmable Gain Amplifier," Texas Instruments, Dallas, TX, USA, Aug. 2019. Accessed: Jul. 22, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/ads1287.pdf

[55] Analog Devices Inc., "32-Bit, 10 kSPS, Sigma-Delta ADC with 100 µs Settling and True Rail-to-Rail Buffers Data Sheet AD7177-2 " Analog Devices Inc., Norwood, MA, USA, Mar. 2016. Accessed: Jul. 22, 2021. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/AD7177-2.pdf

[56] D. W. Upton *et al.*, "Gated Pipelined Folding ADC based Low Power Sensor for Large-Scale Radiometric Partial Discharge Monitoring," *IEEE Sensors J.,* vol. 20, no. 14, pp. 7826 - 7836, Mar. 2020, doi: 10.1109/JSEN.2020.2982576.

[57] D. W. Upton, "Low Power Signal Processing Techniques for Radiometric Partial Discharge Detection in Wireless Sensor Network," Ph.D. dissertation, Dept. Eng. and Technol., Univ. Huddersfield, Huddersfield, UK, 2018. [Online]. Available: https://eprints.hud.ac.uk/id/eprint/34726/

[58] G. Chen and Y. Huang, *Chaotic Maps: Dynamics, Fractals, and Rapid Fluctuations* (Synthesis Lectures on Mathematics and Statistics, no. 4). Morgan & Claypool Publishers, 2011.

[59] B. Holdsworth, and C. Woods, *Digital Logic Design*, 4 ed. Oxford, UK: Elsevier Science & Technology, 2002

[60] B. Ram, *Computer Fundamentals : Architecture & Organisation*, New Delhi, India: New Age International Ltd, 2007.

[61] W. Kester, *Data Conversion Handbook*, 1 ed. Burlington, MA, USA: Newnes, 2005, pp. 709-894.

[62] S. O'brien, *Turbo Pascal 6: The Complete Reference*. Berkeley, CA, USA: Osbourne McGraw-Hill, 1991.
[63] O. F. Olabode, S. Fletcher, A. P. Longstaff, and N. S. Mian, "Precision Core Temperature Measurement of Metals Using an Ultrasonic Phase-Shift Method," *J. Manuf. and Mater. Process.,* vol. 3, no. 3, pp. 80-91, Sept. 2019, doi: 10.3390/jmmp3030080.
[64] H. E. Kelley, "Tungsten," in *SME Mineral Processing & Extractive Metallurgy Handbook*, R. C. K. Dunne, S. Komar; Young, Courtney A. Ed. Englewood, CO, USA: Soc. for Mining, Metall., and Exploration (SME), 2019, ch. 125, pp. 2147-132.
[65] F. Cverna, *ASM Ready Reference - Thermal Properties of Metals* (ASM Materials Data). Materials Park, OH, USA: ASM Int., 2002.
[66] Analog Devices, "AD8302 (Rev. B) LF–2.7 GHz RF/IF Gain and Phase Detector," Analog Devices, Norwood, MA, USA, Oct. 2018. Accessed: Jun. 12, 2019. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad8302.pdf
[67] R. H. Walden, "Analog-to-digital converter survey and analysis," *IEEE J. Sel. Areas in Commun.,* vol. 17, no. 4, pp. 539-550, Apr. 1999, doi: 10.1109/49.761034.
[68] R. H. Walden, "Performance trends for analog to digital converters," *IEEE Commun. Mag.,* vol. 37, no. 2, pp. 96-101, Feb. 1999, doi: 10.1109/35.747256.
[69] B. Murmann. *ADC Performance Survey 1997-2021 (ISSCC & VLSI Symposium)*. [Online]. Available: https://web.stanford.edu/~murmann/adcsurvey.html
[70] B. Murmann, "The Race for the Extra Decibel: A Brief Review of Current ADC Performance Trajectories," *IEEE Solid-State Circuits Mag.,* vol. 7, no. 3, pp. 58-66, Sept. 2015, doi: 10.1109/MSSC.2015.2442393.
[71] N. N. Çikan and M. Aksoy, "Analog to Digital Converters Performance Evaluation Using Figure of Merits in Industrial Applications," presented at the 2016 Eur. Model. Symp. (EMS), Pisa, Italy, Nov. 28-30, 2016.
[72] T. Drenski and J. C. Rasmussen, "ADC & DAC - Technology Trends and Steps to Overcome Current Limitations," in *2018 Opt. Fiber Commun. Conf. and Expo. (OFC)*, San Diego, CA, USA, Mar. 11-15, 2018: IEEE.
[73] Microchip Technology Inc. "ADC Integral Non-Linearity (INL)." Microchip Technology. https://microchipdeveloper.com/adc:adc-inl (accessed May. 25, 2021).
[74] D. Tuite, "Relate ADC Topologies and Performance to Applications," *Electron. design,* vol. 60, no. 12, pp. 38-41, Sept. 2012.
[75] RS Components Ltd. "Analogue to Digital Converters." RS Components. https://uk.rs-online.com/web/c/semiconductors/data-converters/analogue-to-digital-converters/ (accessed Jul. 6, 2021).
[76] Arrow Electronics Inc. "Data Acquisition Analog to Digital Converters - ADCs " Arrow Electronics. https://www.arrow.com/en/products/search?prodline=Analog%20to%20Digital%20Converters%20-%20ADCs&selectedType=plNames (accessed Jul. 6, 2021).
[77] Premier Farnell Limited. "Analog-to-Digital Converters - ADC." Premier Farnell. https://uk.farnell.com/w/c/semiconductors-ics/data-signal-conversion/analog-to-digital-converters-adc?st=adcs (accessed Jul. 6, 2021).
[78] Digi-Key Electronics. "Data Acquisition - Analog to Digital Converters (ADC)." Digi-Key Electronics. https://www.digikey.co.uk/products/en/integrated-circuits-ics/data-acquisition-analog-to-digital-converters-adc/700?FV=-

8%7C700&quantity=0&ColumnSort=-348&page=1&pageSize=25 (accessed Jul. 6, 2021).

[79] Mouser Electronics Inc. "Analog to Digital Converters - ADC." Mouser Electronics. https://www.mouser.co.uk/Semiconductors/Data-Converter-ICs/Analog-to-Digital-Converters-ADC/_/N-4c43g (accessed Jul. 6, 2021).

[80] Maxim Integrated. "Tutorial 1870 Demystifying Delta-Sigma ADCs," Maxim Integr., Jan. 2003. Accessed: Feb. 13, 2020. [Online]. Available: https://pdfserv.maximintegrated.com/en/an/Sigma-Delta_ADCs.pdf

[81] Maxim Integrated, "Tutorial 1080 Understanding SAR ADCs: Their Architecture and Comparison with Other ADCs," Maxim Integr., [Online], Oct. 2001. Accessed: Jul. 28, 2021. [Online]. Available: https://pdfserv.maximintegrated.com/en/an/AN1080.pdf

[82] M. Clifford, "Fundamental Principles Behind the Sigma-Delta ADC Topology: Part 1," Analog Devices Inc., Norwood, MA, USA, 2016. Accessed: Jul. 28, 2021. [Online]. Available: https://www.analog.com/media/en/technical-documentation/tech-articles/Fundamental-Principles-Behind-the-Sigma-Delta-ADC-Topology-Part-1.pdf

[83] M. Furuta and T. Itakura, "Trends in the design of high-speed, low-power analog-to-digital converters," in *2015 IEEE Int. Symp. Radio-Freq. Integration Technol. (RFIT)*, Sendai, Japan Aug. 26-28 2015: IEEE, pp. 169-171, doi: 10.1109/RFIT.2015.7377923.

[84] L. F. Chaparro, *Signals and systems using MATLAB*, 2nd ed. Kidlington, Oxford, UK: Academic Press, 2011.

[85] X. Zhang and Y. Cao, "A Novel Chaotic Map and an Improved Chaos-Based Image Encryption Scheme," *The Scientific World J.,* vol. 2014, Jul. 2014, Art no. 713541, doi: 10.1155/2014/713541.

[86] P. O. Pouliquen, K. A. Boahen, and A. G. Andreou, "A Gray-code MOS current-mode analog-to-digital converter design," in *1991 IEEE Int. Symp. Circuits and Syst.*, Singapore Jun. 11-14, 1991: IEEE, pp. 1924-1927, doi: 10.1109/ISCAS.1991.176785.

[87] B. M. Wilamowski, M. E. Sinangil, and G. Dundar, "A Gray-Code Current Mode ADC Structure," in *2006 IEEE Mediterranean Electrotechnical Conf. (MELECON 2006)*, Benalmadena, Malaga, May 16-19, 2006: IEEE, pp. 35-38, doi: 10.1109/MELCON.2006.1653029.

[88] W. Kester, "MT-025 Tutorial Architectures VI: Folding ADCs," Analog Devices, [Online], Oct. 2008. [Online]. Available: https://www.analog.com/media/en/training-seminars/tutorials/mt-025.pdf

[89] A. Medio and M. Lines, *Nonlinear Dynamics: A Primer*. NJ, USA: Cambridge Univ. Press, 2001.

[90] J. Vries, *Topological Dynamical Systems: An Introduction to the Dynamics of Continuous Mappings* (De Gruyter Studies in Mathematics, no. 59). Berlin, Germany: De Gruyter, 2014.

[91] B. D. Smith, "An unusual electronic analog-digital conversion method," *IRE Trans. Instrum.,* Article vol. PGI-5, pp. 155-160, Jun. 1956, doi: 10.1109/IRE-I.1956.5007017.

[92] S. Arayawat, A. Chaikla, V. Riewruja, P. Julsereewong, and T. Trisuwannawat, "A low-voltage algorithmic ADC based on Gray coding," in *Proc. 7th Int. Conf. Signal Process. 2004 (ICSP'04)*, Beijing, China Aug. 31 - Sept. 4, 2004, vol. 1: IEEE, pp. 500-503.

[93] A. Chaikla, S. Arayawat, and V. Riewruja, "OTA-based Gray-code Algorithmic ADC," in *2006 SICE-ICASE Int. Joint Conf.*, Busan, South Korea, Oct. 18-21, 2006: IEEE, pp. 5787-5791, doi: 10.1109/SICE.2006.314677.

[94] S. Arayawat, U. Thubtong, P. Julsereewong, V. Riewruja, and A. Julsereewong, "A voltage-mode Gray-code algorithmic ADC," in *2008 Soc. Instrum. and Control Eng. Jpn. (SICE) Annu. Conf.*, Chofu, Japan Aug. 20-22, 2008: IEEE, pp. 605-608, doi: 10.1109/SICE.2008.4654728.

[95] W. Petchmaneelumka and A. Julsereewong, "A Gray-code algorithmic analog-to-digital converter based on operational conveyors," in *Int. conf. Control, Automat. and Syst. (ICCAS) 2010*, Gyeonggi-do, South Korea Oct. 27-30, 2010: IEEE, pp. 1617-1621, doi: 10.1109/ICCAS.2010.5669636.

[96] V. Litovski, M. Andrejevic, and M. Nikolic, "Chaos Based Analog-to-digital Conversion of Small Signals," in *2006 8th Seminar Neural Netw. Appl. in Elect. Eng.*, Belgrade, Serbia Sept. 25-27 2006: IEEE, pp. 173-176, doi: 10.1109/NEUREL.2006.341205.

[97] R. C. Jaeger and T. N. Blalock, *Microelectronic Circuit Design*, 5th ed. NY, USA: McGraw-Hill Educ. (in English), 2016.

[98] R. Moghimi, "Curing Comparator Instability with Hysteresis," *Analog Dialogue,* vol. 34, no. 7, Nov. 2000. [Online]. Available: https://www.analog.com/media/en/analog-dialogue/volume-34/number-1/articles/curing-comparator-instability-with-hysteresis.pdf.

[99] A. G. W. Venes and R. J. van-de-Plassche, "An 80-MHz, 80-mW, 8-b CMOS folding A/D converter with distributed track-and-hold preprocessing," *IEEE J. Solid-State Circuits,* vol. 31, no. 12, pp. 1846-1853, Dec. 1996, doi: 10.1109/4.545804.

[100] A. Dinu and A. Vlad, "The compound tent map and the connection between gray codes and the initial condition recovery," U.P.B. Sci. Bull., Series A, vol. 76, no. 1, pp. 17-28, 2014.

[101] D. Luengo and I. Santamaría, "Asymptotically optimal estimators for chaotic digital communications," in *Chaotic Signals in Digital Communications*, M. Eisencraft, R. Attux, R. Suyama, Baton Rouge, US: Taylor & Francis Group, 2014, ch. 11, pp. 297-324.

[102] L. Liu, J. Hu, Z. He, C. Han, and C. Lu, "Chaotic signal reconstruction with application to noise radar system," in *ISPACS 2010*, Dec. 2010, doi: 10.1109/ISPACS.2010.5704782.

[103] Y. Ren, H. Lin, Z. Ma, and X. Shan, "Performance analysis in general cyclic ADCs," *Int. J. Bifurcation and Chaos in Appl. Sci. and Eng.,* vol. 13, no. 8, pp. 2369 - 2376, Aug. 2003, doi: 10.1142/S0218127403007862.

[104] E. M. Bollt, T. Stanford, Y. C. Lai, and K. Zyczkowski, "What symbolic dynamics do we get with a misplaced partition? : On the validity of threshold crossings analysis of chaotic time-series," *Physica D: Nonlinear Phenomena,* Article vol. 154, no. 3-4, pp. 259-286, Jun. 2001, doi: 10.1016/S0167-2789(01)00242-1.

[105] C. Xi, G. Yong, and Y. Yuan, "A novel method for the initial condition estimation of a tent map," *Chin. Phys. Lett.,* vol. 26, no. 7, pp. 1-3, Apr. 2009, Art no. 078202, doi: 10.1088/0256-307X/26/7/078202.

[106] *Folding ADC Operation Control (FA_clock.vhd)*. (2018). University of Huddersfield. Accessed: Feb. 20, 2019. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/34742/1/Haigh%20THESIS.pdf

[107] Texas Instruments, "THS1030 3-V to 5.5-V 10-Bit, 30 MSPS CMOS Analog-to-Digital Converter datasheet (Rev. E)," Texas Instrum. , Dallas, TX, USA, 2003.

[108] The MathWorks Inc. "snr." The MathWorks, Inc. https://uk.mathworks.com/help/signal/ref/snr.html (accessed May. 22, 2021).

[109] J. Bird, *Engineering Mathematics*. 8th ed. London, UK: Routledge, 2017.

[110] P. P. Chu, *FPGA Prototyping by VHDL Examples : Xilinx Spartan-3 Version*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2008.

[111] M. Gustavsson, J. J. Wikner, and N. Nianxiong Tan, *CMOS Data Converters for Communications* (The Kluwer International Series in Engineering and Computer Science). NY, USA: Kluwer Academic Publishers, 2000.

[112] D. Dutta, "Estimation of Chaos Function for the Implementation of High-Resolution Measurement System," Ph.D. dissertation, Dept. Eng. and Technol., Univ. Huddersfield, Huddersfield, England, 2018. [Online]. Available: http://eprints.hud.ac.uk/id/eprint/34856/

[113] B. Ginetti, P. G. A. Jespers, and A. Vandemeulebroecke, "A CMOS 13-b cyclic RSD A/D converter," *IEEE J. Solid-State Circuits,* vol. 27, no. 7, pp. 957-964, Jul. 1992, doi: 10.1109/JSSC.1992.854935.

[114] D. Dutta, R. Basu, S. Banerjee, V. Holmes, and P. Mather, "Parameter estimation for 1D PWL chaotic maps using noisy dynamics," *Nonlinear Dyn.,* vol. 94, no. 4, pp. 2979 - 2993, Dec. 2018, doi: 10.1007/s11071-018-4538-x.

[115] C. G. M. Vishwas and R. S. Kunte, "An Image Cryptosystem based on Tent Map," in *2020 3rd Int. Conf. Smart Syst. and Inventive Technol. (ICSSIT)*, Tirunelveli, India, Aug. 20-22, 2020: IEEE, pp. 1069-1073, doi: 10.1109/ICSSIT48917.2020.9214291.

[116] M. Gupta, K. K. Gupta, M. R. Khosravi, P. K. Shukla, S. Kautish, and A. Shankar, "An Intelligent Session Key-Based Hybrid Lightweight Image Encryption Algorithm Using Logistic-Tent Map and Crossover Operator for Internet of Multimedia Things," *Wireless Pers. Commun.,* Aug. 2021, doi: 10.1007/s11277-021-08742-3.

[117] B. S. Shashikiran, K. Shaila, and K. R. Venugopal, "Logistic and Tent Map Encrypted Image Steganography in Transformation Domain using DWT-LSB Technique," in *2021 Int. Conf. Intell. Technol. (CONIT)*, Karnataka, India, Jun. 25-27, 2021: IEEE, pp. 1-6, doi: 10.1109/CONIT51480.2021.9498497.

[118] *Estimating initial condition through interval arithmetic*. (2018). University of Huddersfield. Accessed: Jun. 27, 2020. [Online]. Available: https://eprints.hud.ac.uk/id/eprint/34821/

Appendices

List of Appendices

Appendix A: Schematics of the practical TM-based ADC PCBs developed for the project and the respective list of components.

Appendix B: MATLAB code developed to assess the TM-based ADC structure, the TM-ARCHα-n ADC, and the fundamental TM gain compensation algorithm, µCA-1. Code relating to the VHDL µCA-1 implementation is also included.

Appendix C: MATLAB code developed to assess both TM-based ADC structures (the TM-ARCHα-n ADC and TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC) with the enhanced TM gain compensation algorithms, µCA-2 and µCA-3. Code relating to the VHDL µCA-2 implementation is also included.

Appendix D: Tables of results relating to tests performed during this research.

Appendix A

A.1 Tent Map Based ADC PCB

A.1.1 Schematics of the TM-ARCHα-7 ADC PCB

Figure A-1 to Figure A-5 present the schematic for the PCB design of the TM-ARCHα-7 ADC. Figure A-1 presents the sample and hold circuitry, while Figure A-2 and Figure A-3 show the data conversion circuitry which employed TM functions. Figure A-4 gives the voltage reference generation circuitry along with the connectors and power supplies. Figure A-5 illustrates the $\mu_{\pm stage}$ alteration resistors.

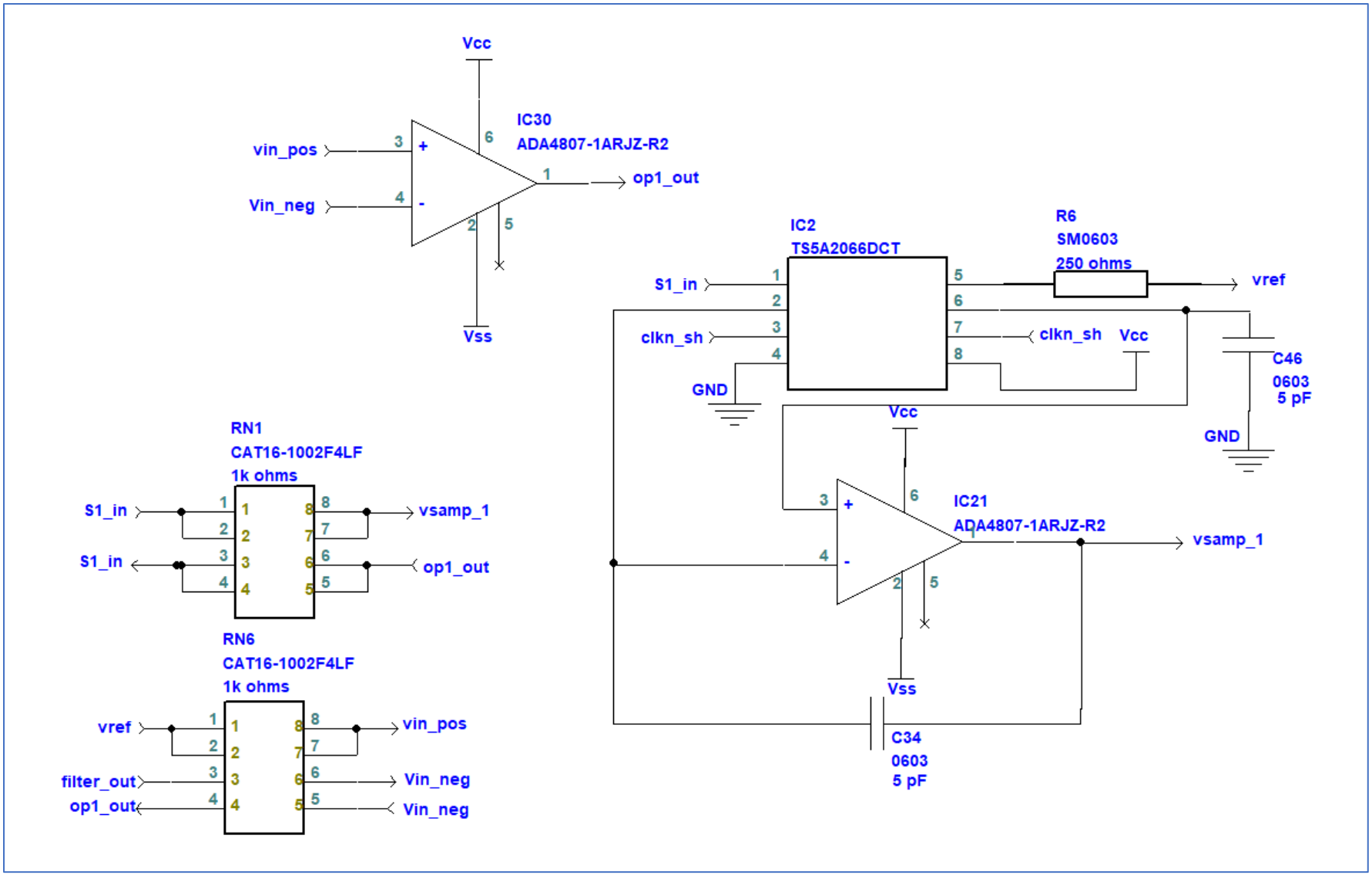


Figure A-1: Sample and hold schematic.

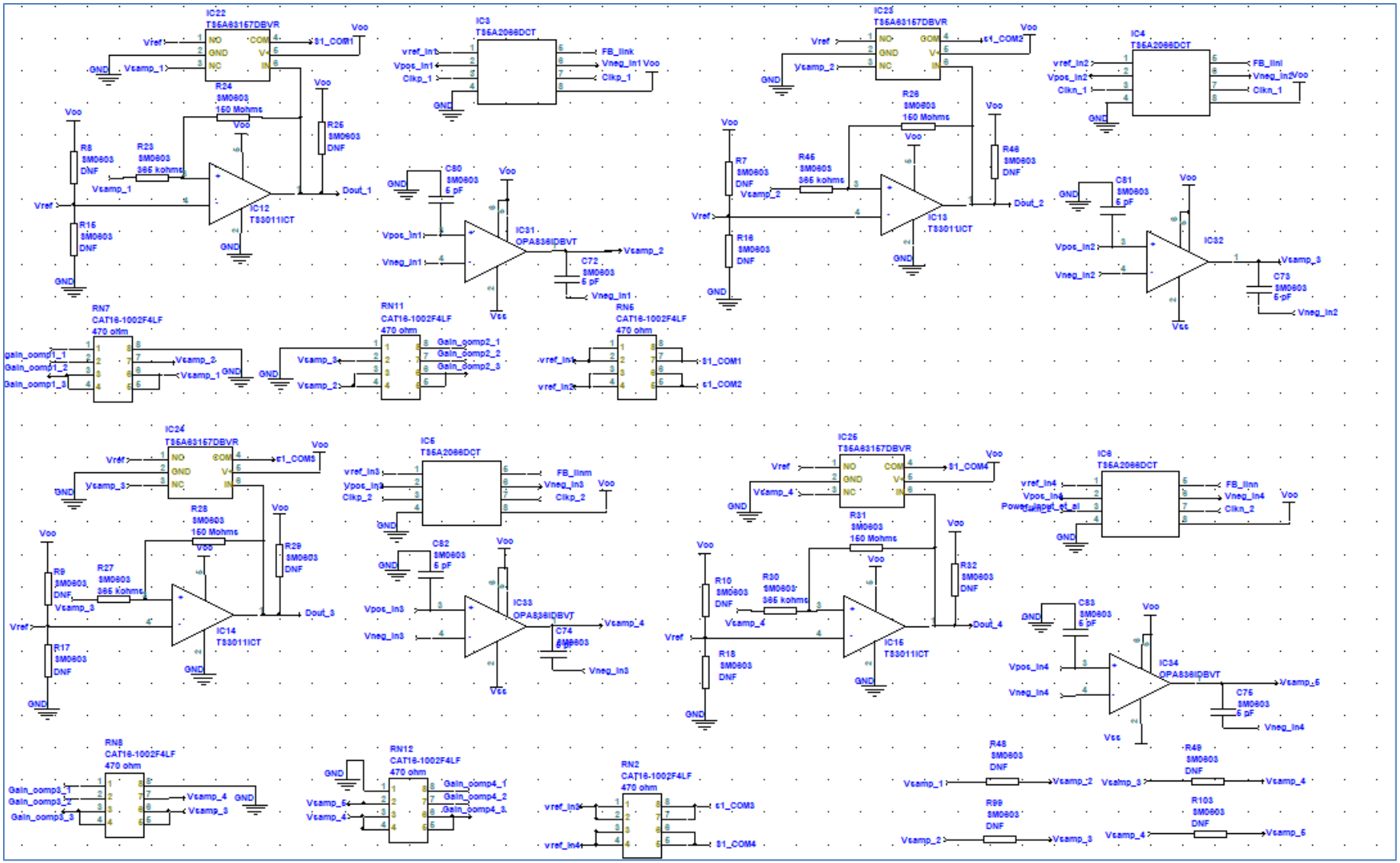


Figure A-2: TM Stages 1 to 4.

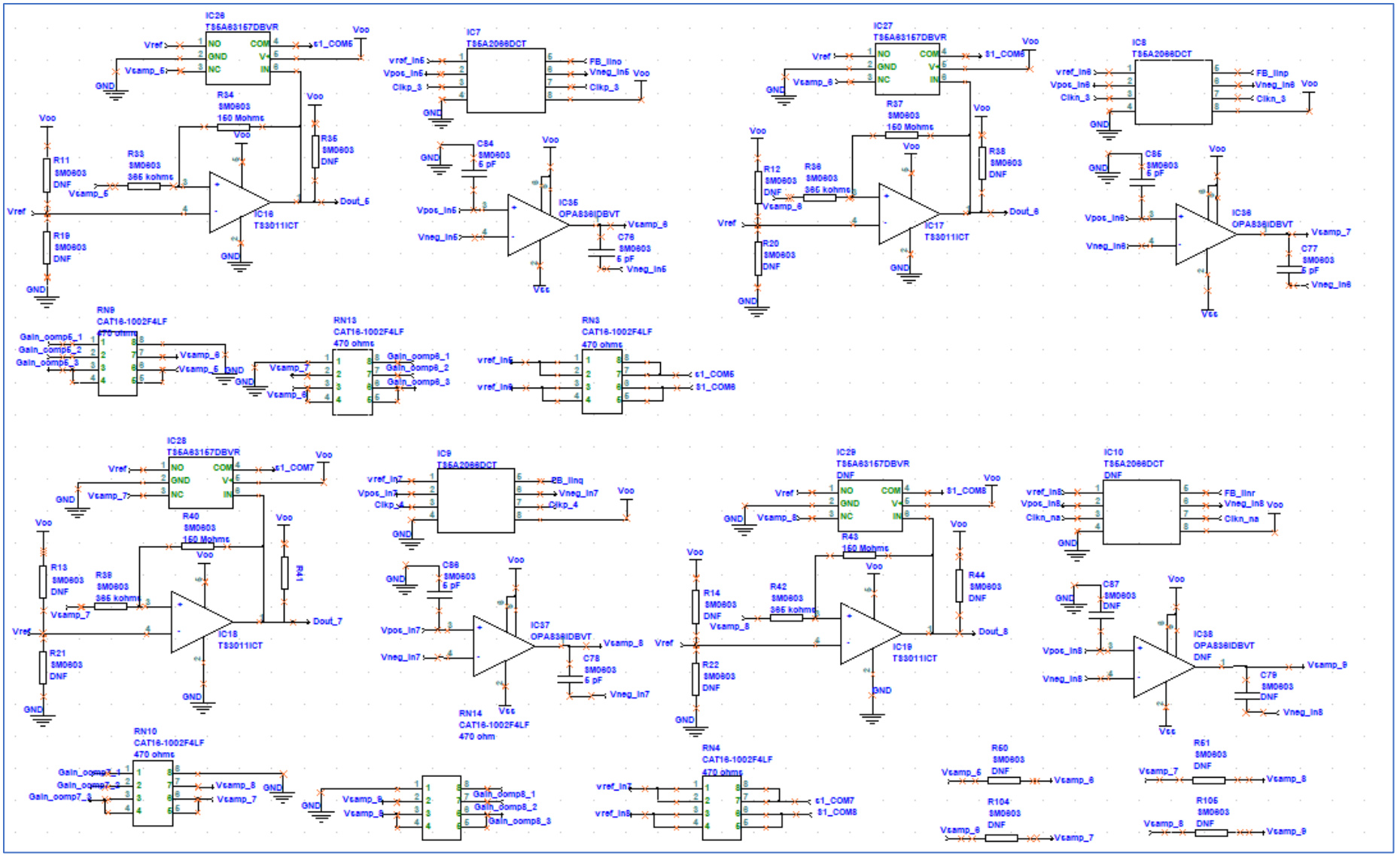


Figure A-3: TM stages 5 to 7.

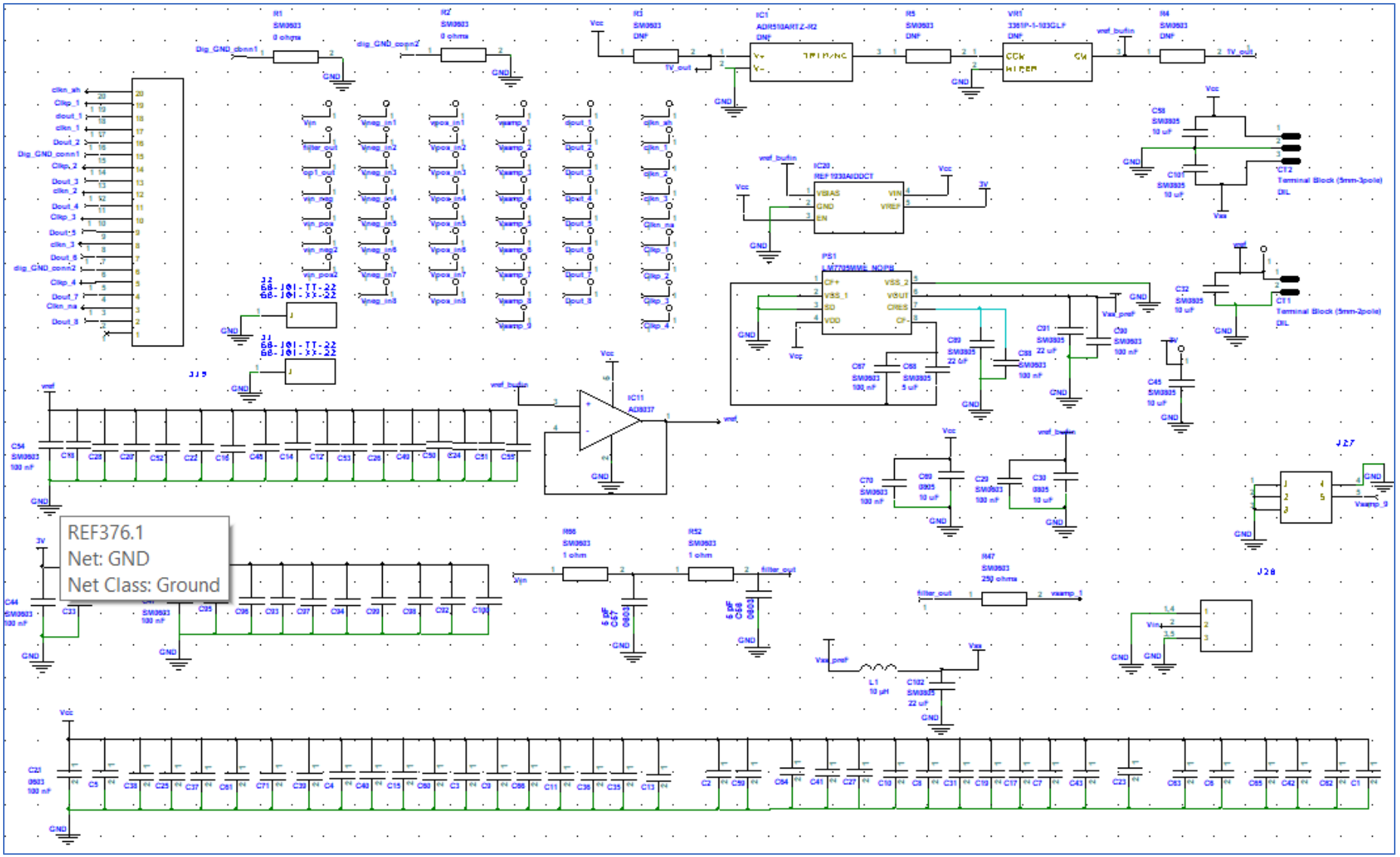


Figure A-4: Power, connectors, decoupling and filter circuitry.

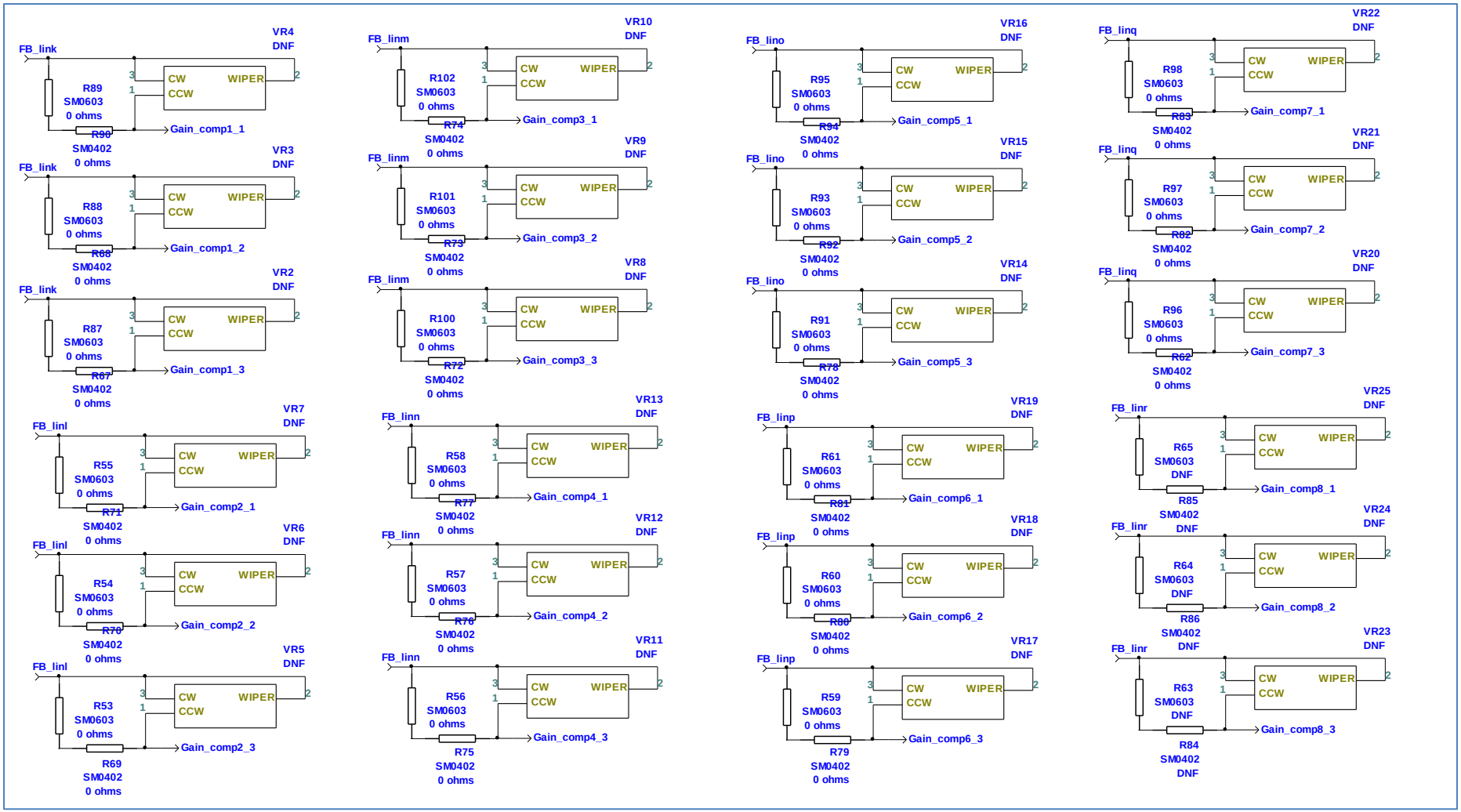


Figure A-5: $\mu_{\pm}$ alteration circuitry.

A.1.2 List of Components for the TM-ARCHα-7 ADC PCB

Table A-1 presents the list of components employed in the TM-ARCHα-7 ADC PCB design, described in the previous section.

Ref Name	Component	Value	Package	Description
C1 - C29, C31, C33, C35 - C44, C47 - C55, C59 - C67, C70, C71, C88, C90, C92 - C100	0603 SMT Capacitor	100 nF	SM0603	Capacitor, Surface Mount Multi-Layer Ceramic
C30, C32, C45, C58, C69, C101, C102	0805 SMT Capacitor	10 µF	SM0805	Capacitor, Surface Mount Multi-Layer Ceramic
C34, C46, C56, C57, C72 - C87	0603 SMT Capacitor	5 pF	SM0603	Capacitor, Surface Mount Multi-Layer Ceramic
C68	0805 SMT Capacitor	5 µF	SM0805	Capacitor, Surface Mount Multi-Layer Ceramic
C89, C91	0805 SMT Capacitor	22 µF	SM0805	Capacitor, Surface Mount Multi-Layer Ceramic
CT1	Terminal Block (5mm-2pole)		DIL	Terminal Block (5mm-2pole)
CT2	Terminal Block (5mm-3pole)		DIL	Terminal Block (5mm-3pole)
IC1	ADR510ARTZ-R2	DNF	SOT - 23 - 3	1 V Voltage Reference
IC2 - IC10	TS5A2066DCT		SM8	Dual channel SPST 10 Ω analogue switch
IC11	AD8037		SOT-23-5	Op-amp
IC12 - IC19	TS3011ICT		SC-70	Push-Pull Comparator

IC20	REF1930AIDDCT		SOT-23-5	Dual Output Vref and Vref/2 Voltage Reference
IC21, ADA4807	ADA4807		SOT-23	Op-amp
IC22 - IC29	TS5A63157DBVR		SOT-23	15-Ohm SPDT Analogue Switch
IC31 - IC38	OPA836IDBVT		SOT-23	Op-amp
J1, J2	SS-101-TT-22		1x1 PCB socket strip 2.54 mm	1x1 PCB socket strip
J3 - J18, J20 - J26, J29 - J56	Test Point			
J19	AW127-20_G-T		1 x 20 PCB socket 2.54 mm	1 x 20 PCB socket 2.54 mm
J27	SMA Connector Receptacle		Straight 50 Through Hole SMA Connector	SMA PCB mount straight socket jack
J28	SMA Connector Receptacle		Straight 50 Edge SMA Connector	SMA PCB edge mount straight jack
L1	0805 SMT Inductor	10 µH	SM0805	Inductor
PS1	LM7705MME_NOPB		MSOP	LM7705MME/NOPB Inverter, Supplies - 0.232 V
R1, R2	0603 SMT Resistor	0 Ω	SM0603	SMT Resistor
R3 - R5, R7 - R22, R25, R29, R32, R35, R38, R41, R44, R46, R48 - R51, R63 - R65, R99, R103 - R105	0603 SMT Resistor	DNF	SM0603	SMT Resistor
R6, R47	0603 SMT Resistor	250 Ω	SM0603	SMT Resistor
R23, R27, R30, R33, R36, R39, R42, R45	0603 SMT Resistor	365 kΩ	SM0603	SMT Resistor
R24, R26, R28, R31, R34, R37, R40, R43	0603 SMT Resistor	150 MΩ	SM0603	SMT Resistor
R52, R66	0603 SMT Resistor	1 Ω	SM0603	SMT Resistor

R62, R67 - R83, R90, R92, R94	0402 SMT Resistor	0 Ω[8]	SM0402	SMT Resistor
R84 - R86	0402 SMT Resistor	DNF	SM0402	SMT Resistor
R53 - R61, R87 - R89, R91, R93, R95 - R98, R100 - R102	0603 SMT Resistor	0 Ω[8]	SM0603	SMT Resistor
RN1 - RN14	1206 SMT 4 Resistor Array	470 Ω	SM1206 4 Array	SMD 1206 Bus Array 4 Resistors 1
VR1	SMT Trimpot	DNF	3361P1103GLF	SMD Single Turn Trimmer
VR2 - VR25	SMT Trimmer	DNF	PVG3G500C01R00	Trimmer Resistors - SMD

Table A-1: Bill of materials for PCB version of the TM-ARCHα-7 ADC.

[8] Initially set to 0 Ω, but later changed to bring $\mu_{\pm stage}$ below 2.

A.2 COTS ADC Breakout Board

A.2.1 Schematic of COTS ADC Breakout Board

Figure A-6 presents the schematic of the COTS THS1030 10-bit ADC breakout board for the TM-ARCHβ-n-$R_{sub\text{-}ranging}$ ADC structure.

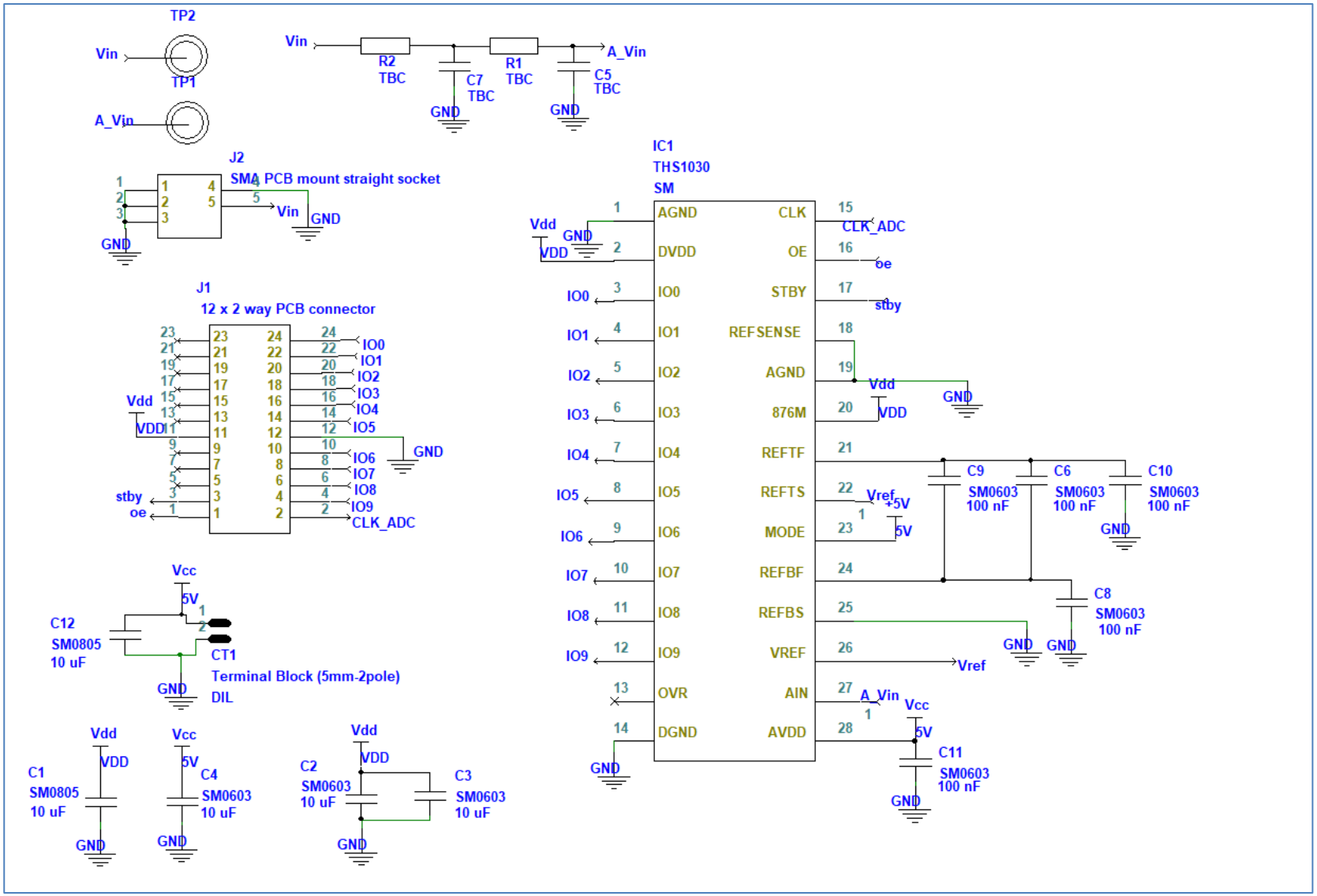


Figure A-6: Schematic of the breakout board for the THS1030 10-bit ADC.

A.2.2 List of Components for the COTS ADC Breakout Board

Table A-2 provides the list of components of the COTS THS1030 10-bit ADC breakout board presented in the previous section.

Ref Name	Component	Value	Package	Description
C1 - C4, C12	0805 Capacitor	10 µF	SM0805	Capacitor, Surface Mount Multi-Layer Ceramic
C5, C7	0603 Capacitor	TBC	SM0603	Capacitor, Surface Mount Multi-Layer Ceramic
C6, C8 - C11	0603 Capacitor	100 nF	SM0603	Capacitor, Surface Mount Multi-Layer Ceramic
CT1	Terminal Block (5mm-2pole)			Terminal Block (5mm-2pole)
IC1	THS1030		TSSOP	10 bit ADC
J1	SSW-112-01-T-D		12 x 2 way PCB connector	12 x 2 way PCB connector socket
J2	SMA connector		SMA PCB Mount Straight Socket	SMA PCB mount straight socket jack
R1, R2	0603 resistor	TBC	SM0603	Thick Film Surface Mount Resistor
TP1, TP2	Test Point			

Table A-2: Bill of Materials for breakout board.

Appendix B

The code presented in this appendix relates to the simulation results presented Sections 5.1 to 5.6.

B.1 MATLAB Scripts for Uncompensated Tent Map Based ADC Output Accuracy Analysis

B.1.1 Code for Bit Accuracy Predictions Analysis

The following code listing presents the TM-ARCHα-15 ADC mathematical model developed in MATLAB. How the quantisation error and bit accuracy of the uncompensated digital output was calculated is also shown.

```
%% Reset command and figure windows
clc;                %clears the command window and the workspace
clf;
clear;

%% Initialise key parameters for model
resolution = 16;    % number of TM stages + 1
Vmax = 3;           % valid input max.
Vmin = 0;           % valid input min.
Vref = 1.5;         % set partition point voltage
Step_size = (Vmax-Vmin)/(2^resolution); %calculating step size

%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
Fs = 25000000;                          % sample rate = 25 MHz
f_fundamental = Fs/2^(resolution+2);    % fundamental frequency
T = (1/f_fundamental);                  % number of periods times
fundamental frequency
dt = 1/Fs;
x = 0:dt:T;
y = (Vmax-Vmin)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);
y(number_samples) = Vmax;

%gain = [1.9 1.99 2];
gain = (1.9: 0.02: 2);
gain_size = length(gain);

%% Define arrays
z = zeros(resolution, number_samples); % TM input and output signals
Dout = zeros(gain_size, resolution, number_samples); % Gray Code output
```

```
bin_representation = zeros(gain_size, resolution, number_samples); %
binary representation of Gray Code
output_representation = zeros(gain_size, number_samples); % Stores
digital outputs as decimal numbers

%% Input goes through TM-based ADC %
%% Input goes through TMs %
for g = 1: 1: gain_size
    for i = 1: 1: number_samples % Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds Gray code word
of sample
            if ((z(res, i) <= Vref) && (z(res, i) >= Vmin))  % if input
to the folding stage is less than or equal to the reference voltage
                z((res+1), i) = gain(g)*z(res, i); % first TM difference
equation
                Dout(g, res, i) = 0;               % comparator output =
0
            elseif ((z(res, i) > Vref) && (z(res, i) <=
Vmax))                    % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain(g)*Vref)-(gain(g)*(z((res), i)-
Vref)); % second TM difference equation
                Dout(g, res,i) =
1;                                          % comparator output = 0
            elseif (z(res, i) > Vmax)   % if the input is greater than
the valid input range
                z((res +1), i) = Vmin;  % output = Vmax
                Dout(g, res, i) = 1;    % comparator output = 1
            else                         % if the input is less than the
valid input range
                z((res +1), i) = Vmin;  % output = Vmin
                Dout(g, res, i) = 0;    % comparator output = 0
            end
        end
        % establish final bit
        if (z(resolution, i) <= (Vref))
            Dout(g, resolution, i) = 0;
        else
            Dout(g, resolution, i) = 1;
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output

    %% Determine uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(g, :,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray codeword
to binary
        bin_representation(g,:,i) = binary ;      %save binary to an
array (verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution           % convert binary values to
the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep ;
        end
        output_representation(g, i) = decimal_rep  ; %modify decimal
value so it lies within the input voltage range
    end

    %% Calculate difference between input and output (quantisation
error) and bit accuracy
    for i = 1: 1: length(y)
        uncorrected_difference(g, i) = ((Vmax-
Vmin)*output_representation( g, i)- y(i))/Step_size; %uncompensated
difference
```

```
    end
    %calculate bit accuracy of compensated and uncompensated ADC
    UD(g, 1) = max(abs(uncorrected_difference(g, :)));
    effective_bit_accuracy_UD(g, 1) = resolution-log2(ceil(UD(g, 1)))-1;
```

The MATLAB function below was developed to convert the Gray code, stored in the Dout array, into binary code.

```
function b = gray2bin(g)
% Gray code to binary function
%   Taken from http://www.matrixlab-examples.com/gray-code.html
b(1) = g(1);
for i = 2 : length(g);
    x = xor((b(i-1)), (g(i)));
    b(i) =(x);
end
```

B.1.2 Code for Static Performance Predictions Analysis

The code below was developed to analyse the static performance of the TM-ARCHα-15 ADC model presented in Appendix B.1.1.

```
      %% Determine static performance
    max_VD = max(output_representation(g, :)); % Determine max. digital
output
    min_VD = min(output_representation(g, :)); % Determine min. digital
output
    endpoint_grad = (max_VD - min_VD)/(y(number_samples) - Vmin); %
Determine gradient of end-point transfer function
    endpoint_const = output_representation(g, 1);    % Determine digital
output axis intercept of end-point transfer function

    % Ideal ADC
    Digital_ideal = floor((endpoint_grad*y +
endpoint_const)*pow2(resolution)); %Determine end-point transfer
function

    %DNL
    Digital_output =
floor(output_representation(g, :)*pow2(resolution)); %actual Transfer
function
    Digital_monitor = 1;
    Analogue_monitor = y(1);
    k = 1;
    for i = 1: 1: number_samples   % start establishing DNL for each
digital output code
        if Digital_output(i) == Digital_output(Digital_monitor)
```

```
        else
            DNL(g, k) = ((y(i) - Analogue_monitor)/Step_size) - 1;
            Digital_monitor = i;
            Analogue_monitor = y(i);
            k = k + 1;
        end
    end

    DNL_max(g) = max(DNL(g, :)); % max. DNL
    DNL_min(g) = min(DNL(g, :)); % min. DNL
    i = 1;

    % INL
    for D = 0 : 1: pow2(resolution)-1 % start establishing INL for each
digital output code
        Dpos_act = find(Digital_output == D, 1);
        Dpos_ideal = find(Digital_ideal == D, 1);
        if isempty(Dpos_act)
            missing_codes(g, 1) = missing_codes(g, 1)+ 1; % calculate
missing codes
        else
            INL(g, i) = (y(Dpos_act) -
y(Dpos_ideal))*pow2(resolution);  % INL result
            i = i+1;
        end
    end
    INL_max(g) = max(INL(g, :)); % max. INL
    INL_min(g) = min(INL(g, :)); % min. INL

    % Offset
    offset(g) = Digital_output(1); % determines offset when input
voltage = 0 V

    % Gain
    full_scale_error(g) = max(Digital_output) - (pow2(resolution)-1); %
max digital output - max ideal output
    Gain_error(g) = full_scale_error(g) - offset(g); % calculates gain
error
end
```

B.1.3 Code for Dynamic Performance Predictions Analysis

The code below was developed to analyse the dynamic performance of the TM-ARCHα-15 ADC model presented in Appendix B.1.1.

```
%% Sine Wave Input Signal
F_samp = 25000000; % sample frequency = 25 MHz
N = pow2(resolution+2); % set N
M = 131071;             % set M
frequency = F_samp*(M/N); % set input frequency
dt = 1/F_samp;
T = 1;
x = (1:N)*dt;
y = (Vmax-Vmin)*(sin(2*pi*frequency*x)+1)/2;  % creating input
sinusoidal signal
df = F_samp*(0:(N/2)-1)/N;

                                    [...]

    %% Calculate Dynamic Performance
    output_representation(g, :) = detrend(output_representation(g, :),
'constant'); % remove DC offset
    SINAD(g) = sinad(output_representation(g, :), F_samp); % SINAD value
    SNR(g) = snr(output_representation(g, :), F_samp); % SNR value
    SFDR(g) = sfdr(output_representation(g, :), F_samp); % SFDR value
    THD(g) = thd(output_representation(g, :), F_samp,5,'aliased'); % THD
value
    ENOB(g) = (SINAD(g) - 1.76)/6.02; % ENOB value
end
```

B.2 MATLAB Scripts for Tent Map Based ADC with the Fundamental Tent Map Gain Compensation Algorithm Output Accuracy Analysis

B.2.1 Code for Bit Accuracy Predictions Analysis

The following code presents the analysis of the µCA-1 which was developed to compensate the output data produced by the TM-ARCHα-15 ADC mathematical model shown in Appendix B.1.1. The code extract also highlights the calculations performed to determine the bit accuracy of the compensated output.

```
                                    [...]
for g = 1: 1: gain_size
    %% Ideal DM values - look up table
    VHDL_bits = resolution + 8;
    for i = 1:1:(resolution - 1)
        LUT_theory(i) = (1/mpower(gain(g), i))-(1/pow2(i)); %Calculate
difference value
        LUT_VHDL(i) = floor(pow2(VHDL_bits)*((1/mpower(gain(g), i))-
(1/pow2(i)))); %Calculate difference value
    end
    %% Input goes through TM-based ADC %
    %% Input goes through TMs %
                                    [...]
    %% Sign for Difference Measure (SDM)
    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(g, 1,i); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(g,2,i), Dout(g,3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(g, res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end
    %% Difference Measure: selected for each respective gray code bit

    for i = 1: 1: length(y) %Samples of input signal
        DV_theory(1,i) = 0; %Ideal as it hasn't passed through a TM
        DV_VHDL(1,i) = 0; %Ideal as it hasn't passed through a TM
        for res = 2: 1: resolution   % gives remaining bits of DM
            if (Dout(g,res, i) > 0)
                DV_theory(res, i) = LUT_theory(res - 1);
                DV_VHDL(res, i) = LUT_VHDL(res - 1);
            else
                DV_theory (res, i) = 0;
```

```
                DV_VHDL (res, i) = 0;
            end
        end
    end
    %% Signed Difference Value

    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV_theory(res, i) = DV_theory(res, i);
                SDV_VHDL(res, i) = DV_VHDL(res, i);
            else
                SDV_theory(res, i) = -DV_theory(res, i);
                SDV_VHDL(res, i) = -DV_VHDL(res, i);
            end
        end
    end

    % Determine DV
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum_theory(i) = sum(SDV_theory(:,i));
        SDV_sum_VHDL(i) = sum(SDV_VHDL(:,i))/pow2(VHDL_bits);
    end

    %% Implement correction
    %uncompensated output
    for i = 1: 1: length(y) % converting Gray-code representation of
samples, to binary
        gray_code_vector = Dout(g,:,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution                 %convert binary values
to the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep;
        end
        output_representation(g, i) = decimal_rep  ; %modify decimal
value so it lies within the input voltage range
    end
    voltage_representation(g, :) = output_representation(g, :)*(Vmax -
Vmin);
    %compensated ADC output
    for i = 1: 1: length(y) % compensate output
        if (SDM(1,i) == 1)
            corrected_output_theory(g,i) = output_representation(g,i) -
SDV_sum_theory(i);
            corrected_output_VHDL(g,i) = output_representation(g,i) -
SDV_sum_VHDL(i);
        else
            corrected_output_theory(g,i) = output_representation(g,i) +
SDV_sum_theory(i);
            corrected_output_VHDL(g,i) = output_representation(g,i) +
SDV_sum_VHDL(i);
        end
    end

    %% Calculate difference between input and output (quantisation
error) and bit accuracy

    uncorrected_difference(g, :) = (voltage_representation(g, :)-
y)/Step_size;   %uncompensated difference
    corrected_difference_theory(g, :) =
((corrected_output_theory(g, :)*(Vmax - Vmin)) -
y)/Step_size;         %compensated difference
```

```
    corrected_difference_VHDL(g, :) =
((corrected_output_VHDL(g, :)*(Vmax - Vmin))- y)/Step_size; %compensated
difference

    %calculate bit accuracy of compensated and uncompensated ADC
    UD(g, 1) = max(abs(uncorrected_difference(g, :)));
    CD_theory(g, 1) = max(abs(corrected_difference_theory(g, :)));
    CD_VHDL(g, 1) = max(abs(corrected_difference_VHDL(g, :)));
    effective_bit_accuracy_UD(g, 1) = resolution  - log2(ceil(UD(g, 1)))
- 1;  %if UD = 1 bit accuracy should be resolution - 1
    effective_bit_accuracy_CD_theory(g, 1) = resolution -
log2(ceil(CD_theory(g, 1))) - 1;
    effective_bit_accuracy_CD_VHDL(g, 1) = resolution -
log2(ceil(CD_VHDL(g, 1))) - 1;
end
```

B.2.2 Code for Static Performance Predictions Analysis

The code below was developed to analyse the static performance of the TM-ARCHα-15 ADC model shown in Appendix B.1.1 after the digital output data had been compensated using the µCA-1 presented in Appendix B.2.1.

```
   %% Determine static performance
    max_VD = max(corrected_output_VHDL(g, :)); % Determine max. digital
output
    min_VD = min(corrected_output_VHDL(g, :)); % Determine min. digital
output
    endpoint_grad = (max_VD - min_VD)/(y(number_samples) - Vmin); %
Determine gradient of end-point transfer function
    endpoint_const = corrected_output_VHDL(g, 1); % Determine digital
output axis intercept of end-point transfer function

    % Ideal ADC
    Digital_ideal = floor((endpoint_grad*y +
endpoint_const)*pow2(resolution));  %Determine end-point transfer
function

    %DNL
    Digital_output =
floor(corrected_output_VHDL(g, :).*pow2(resolution)); %actual Transfer
function
    Digital_monitor = 1;
    Analogue_monitor = y(1);

    k = 1;
    for i = 1: 1: number_samples   % start establishing DNL for each
digital output code
        if Digital_output(i) == Digital_output(Digital_monitor)

        else
            DNL(g, k) = ((y(i) - Analogue_monitor)/Step_size) - 1;
            Digital_monitor = i;
            Analogue_monitor = y(i);
            k = k + 1;
```

```
        end
    end

    DNL_max_comp(g) = max(DNL(g, :)); % max. DNL
    DNL_min_comp(g) = min(DNL(g, :)); % min. DNL
    i = 1;

    % INL
    for D = 0 : 1: pow2(resolution)-1 % start establishing INL for each
digital output code
        Dpos_act = find(Digital_output == D, 1);
        Dpos_ideal = find(Digital_ideal == D, 1);
        if isempty(Dpos_act)
            missing_codes(g, 1) = missing_codes(g, 1)+ 1; % calculate
missing codes
        else
            INL(g, i) = (y(Dpos_act) -
y(Dpos_ideal))*pow2(resolution); % INL result
            i = i+1;
        end
    end
    INL_max_comp(g) = max(INL(g, :)); % max. INL
    INL_min_comp(g) = min(INL(g, :)); % min. INL

    % Offset
    offset_comp(g) = Digital_output(1); % determines offset when input
voltage = 0 V

    % Gain
    full_scale_error_comp(g) = max(Digital_output) - (pow2(resolution)-
1); % max digital output - max ideal output
    Gain_error_comp(g) = full_scale_error_comp(g) - offset_comp(g); %
calculates gain error

end
```

B.2.3 Code for Dynamic Performance Predictions Analysis

The code below was adapted from the code extract presented in Appendix B.1.3 to determine the dynamic performance of the TM-ARCHα-15 ADC model after the digital output data had been compensated using the µCA-1 presented in Appendix B.2.1.

```
%% Dynamic Performance
corrected_output_VHDL(g, :) = detrend(corrected_output_VHDL(g, :),
'constant'); % remove DC
SINAD(g) = sinad(corrected_output_VHDL(g, :)*(Vmax - Vmin),
F_samp);    % SINAD value
SNR(g) = snr(corrected_output_VHDL(g, :)*(Vmax - Vmin), F_samp);    %
SNR value
SFDR(g) = sfdr(corrected_output_VHDL(g, :)*(Vmax - Vmin), F_samp);  %
SFDR value
THD(g) = thd(corrected_output_VHDL(g, :)*(Vmax - Vmin), F_samp, 5,
'aliased');  % THD value
ENOB(g) = (SINAD(g) - 1.76)/6.02;   % ENOB value
```

B.3 MATLAB Script for of the Fundamental Tent Map Gain Compensation Algorithm

The following MATLAB script was employed to perform a sensitivity analysis on a TM-ARCHα-15 ADC and TM-ARCHα-7 ADC as discussed in Section 5.3.

```
%% Characteristics for Tent-Map Based ADC
resolution = 8;              %number of TM stages + 1
gain = [1.9 1.99]; % TM gain
Vmax = 3; % valid input max.
Vmin = 0; % valid input min.
Vref = 1.5;              %set partition point voltage
Step_size = (Vmax-Vmin)/(2^resolution); %calculating step size

%% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/pow2(resolution + 2); % fundamental frequency
T = (1/f_fundamental); %number of periods times fundamental frequency
dt = 1/Fs;
x = 0:dt:T;
y = (Vmax-Vmin)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);
y(number_samples) = Vmax;
gain_size = length(gain);
%% deviation from gain
deviation_percent = 5;   % set deviation interval in terms of percentage
lower_percentage = -(deviation_percent/2);
upper_percentage = (deviation_percent/2);
percentage_inc = 0.1;
deviations = [lower_percentage: percentage_inc : upper_percentage];

% establish percentage
lower_limit = (100 + lower_percentage)/100; % set lower limit
upper_limit = (100 + upper_percentage)/100; % set upper limit
increment = percentage_inc/100;
number_deviations = round(1+(upper_limit - lower_limit)/increment);

%% Pre-allocate vectors
% This was done to reduce simulation time
Dout = zeros(gain_size, resolution, number_samples);
corrected_difference = zeros(number_samples, number_deviations);
SDM = zeros(resolution, number_samples);
bin_representation = zeros(resolution, number_samples);
corrected_output = zeros(number_samples, number_deviations);
SDV = zeros(resolution, number_samples, number_deviations);
SDV_sum = zeros(number_samples, number_deviations);
SDV_dec = zeros(number_samples, number_deviations);
DV = zeros(resolution, number_samples, number_deviations);
LUT = zeros(resolution - 1, number_deviations);

%% Start Sensitivity analysis
for g = 1: 1: gain_size  % Chose μADC
    Estimated_gain_range =
[gain(g)*(lower_limit):(gain(g)*increment):gain(g)*(upper_limit)]; %rang
e of μ to be employed by the compensation algorithm
    for i = 1: 1: number_samples %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
```

```
            if ((z(res, i) <= Vref) && (z(res, i) >= Vmin))  % if input
to the folding stage is less than or equal to the reference voltage
                z((res+1), i) = gain(g)*z(res, i);  % first TM
difference equation
                Dout(g, res, i) = 0;              % comparator output = 0
            elseif ((z(res, i) > Vref) && (z(res, i) <= Vmax)) % if
input to the folding stage is more than the reference voltage
                z((res +1), i) = (gain(g)*Vref)-(gain(g)*(z((res), i)-
Vref));
                Dout(g, res,i) = 1;
            elseif (z(res, i) > Vmax)
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 1;
            else
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 0;
            end
        end

        if (z(resolution, i) <= (Vref))
            Dout(g, resolution, i) = 0;
        else
            Dout(g, resolution, i) = 1;
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output

    %% Ideal DM values - look up table
    %Calculate all the DM values for each μ being employed by the
compensation algorithm
    for i = 1:1:(resolution - 1)
        for j = 1:1:number_deviations
            LUT(i,j) = (1/mpower(Estimated_gain_range(j), i))-
(1/pow2(i)); %Calculate difference value
        end
    end

    %% Sign for Difference Measure (SDM)
    for i = 1: 1: number_samples %Samples of input signal
        SDM(1, i) = Dout(g,1,i); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(g,2,i), Dout(g,3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(g,res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end

    %% Difference Measure: selected for each respective gray code bit
    %Select the DM values determined using each μ being employed by the
compensation algorithm
    for i = 1: 1: number_samples %Samples of input signal
        for j = 1:1:number_deviations
            DV(1,i,j) = 0; %Ideal as it hasn't passed through a TM
            for res = 2: 1: resolution   % gives remaining bits of DM
                if (Dout(g,res, i) > 0)
                    DV(res, i,j) = LUT(res - 1,j);
                else
                    DV (res, i,j) = 0;
                end
```

```
            end
        end
    end

    %% Signed Difference Value
    %Calculate the DV values determined using each μ being employed by
the compensation algorithm
    for i = 1: 1: number_samples %Samples of input signal
        for j = 1:1:number_deviations
            for res = 1: 1: resolution   % gives remaining bits of DV
                if (SDM(res, i) > 0)
                    SDV(res, i, j) = DV(res, i, j);
                else
                    SDV(res, i, j) = -DV(res, i, j);
                end
            end
        end
    end

    % Determine DV
    for i = 1: 1: number_samples %decimal of SDV
        for j = 1:1:number_deviations
            SDV_sum(i, j) = sum(SDV(:,i, j));
            SDV_dec(i, j) = (Vmax-Vmin)*SDV_sum(i, j);
        end
    end

    %% Implement correction
    %Implement the compensation employing the DV values determined using
each μ being employed by the compensation algorithm

    %uncompensated output
    for i = 1: 1: number_samples % converting Gray code representation
of samples, to binary
        gray_code_vector = Dout(g,:,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution           %convert binary values to the
equivalent voltage
            decimal_rep = (Vmax-Vmin)*(binary(j)/(2^j))+ decimal_rep ;
        end
        output_representation(i) = decimal_rep; %modify decimal value so
it lies within the input voltage range
    end

    %compensated ADC output
    for i = 1: 1: number_samples
        for j = 1:1:number_deviations
            if (SDM(1,i) == 1)
                corrected_output(i,j) = output_representation(i) -
SDV_dec(i,j);
            else
                corrected_output(i,j) = output_representation(i) +
SDV_dec(i,j);
            end
        end
    end

    %% Calculate quantisation error and bit accuracy
    uncorrected_difference = (output_representation-
y)/Step_size;  %uncompensated difference
    for j = 1:1:number_deviations
        corrected_difference(:,j) = (corrected_output(:,j)-
y(:))/Step_size;         %compensated difference
    end
```

```
    %calculate bit accuracy of compensated and uncompensated ADC
    UD(g) = max(abs(uncorrected_difference));
    for j = 1:1:number_deviations
        CD(g,j) = max(abs(corrected_difference(:,j)));
    end
    effective_bit_accuracy_UD = resolution - (log2(ceil(UD)) +1);  %if
UD = 1 bit accuracy should be resolution - 1

    for j = 1:1:number_deviations
        effective_bit_accuracy_CD(g,j) = resolution -
(log2(ceil(CD(g,j))) +1);
    end
end
```

B.4 MATLAB Script for Comparison with the Tent Map Gain Compensation Algorithm by Basu

The following script was developed to compare the µCA-1 assessed in Section 5.2 with the µCA developed by Basu [41, 42]. The code highlighted blue was obtained from [118].

```
clc;                %clears the command window and the workspace
clf;
clear;

%% Characteristics for Tent Map
resolution = 16;                %number of TM stages - 1
gain = [1.9: 0.005: 2]; % TM gain
Vmax = 3; % valid input max.
Vmin = 0; % valid input min.
Vref = 1.5;             %set partition point voltage
Step_size = (Vmax-Vmin)/(2^resolution); %calculating step size
%% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref

F_samp = 25000000;      %sample rate = 25 MHz
f_fundamental = F_samp/pow2(resolution + 2); % fundamental frequency
T = (1/f_fundamental); %number of periods times fundamental frequency
dt = 1/F_samp;
x = 0:dt:T;
y = (Vmax-Vmin)*(sawtooth(2*pi*f_fundamental*x)+1)/2;
gain_size = length(gain);
number_samples = length(x);
sample_number = (1: 1: number_samples);

for g = 1: 1: gain_size

    for i = 1: 1: number_samples %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            if ((z(res, i) <= Vref) && (z(res, i) >= Vmin))  % if input
to the folding stage is less than or equal to the reference voltage
                z((res+1), i) = gain(g)*z(res, i);
                Dout(g, res, i) = 0;
            elseif ((z(res, i) > Vref) && (z(res, i) <= Vmax))   % if
input to the folding stage is more than the reference voltage

                z((res +1), i) = (gain(g)*Vref)-(gain(g)*(z((res), i)-
Vref));
                Dout(g, res,i) = 1;
            elseif (z(res, i) > Vmax)
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 1;
            else
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 0;
            end
        end

        if (z(resolution, i) <= (Vref))
            Dout(g, resolution, i) = 0;
```

```
        else
            Dout(g, resolution, i) = 1;
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output

    %% Output before correction
    %uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(g,:,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution                 %convert binary values
to the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep;
        end
        output_representation(g, i) = decimal_rep  ; %modify decimal
value so it lies within the input voltage range
    end

    %% Basu et al algorithm - Code obtained from
https://eprints.hud.ac.uk/id/eprint/34821/
    format long
    iteration = resolution;                % setting number of iterations
    Parameter_Mu = gain(g)/2;    % setting parameter for estimation
    A = 0;                    % initialising lower bound
    B = 0;                     % initialising upper bound
    Delta = 0;                 % scaled interval size initialised
    l = 0;                     % size of the interval initialised
    alpha = 0;                 % odd even counter variable initialised
    N = length(y);
    Symbols_Gray = transpose(squeeze(Dout(g, :, :)));  % copying
generated grey code for estimation
    X0_Dash_Array = zeros(N,1); % estimated initial condition array
initialised
    Diff= zeros(N,1);   % error or difference between the actual and
    % estimated initial condition
    X0_Dash = 0;       % single initial condition estimate variable
initialised
    for j = 1:N      % for N initial conditions
        for i = 1:iteration % for i iteration of each initial condition
            alpha = alpha + Symbols_Gray(j,i); % count number of 1s
odd/even
            if i == 1                  % if the first symbol
                if Symbols_Gray(j,1) == 1 % is 1 then the primary half
interval
                    A = 0.5;           % is mirrored with lower bound =
0.5
                    B = 0;              % and upper bound = 0
                else
                    A = 0;         % other wise keeping primary half
                    B = 0.5;           % unmirrored
                end
            else
                if rem(alpha,2) == 0 % if no. of '1's in the sequence is
even
                    A = A;             % lower bound unchanged
                    B = A + Delta;     % upper bound shifted to lower
bound +%scaled interval size
                else                  % if no. of '1's in the sequence is
odd
                    A = B -Delta;     % lower bound is shifted to
upperbound -% delta
```

```
                    B = B;              % upper bound is unchanged
end
                end
            end
            l = B - A;      % determine the length of newly formed
interval
            Delta = l/(2*Parameter_Mu); % size of the interval scaled %
proportional to mu
            % first symbol is not due to the result of TM iteration
therefore orienting
            % the final estimated point is necessary and therefore
scaled accordingly
            % and again the interval is unmirrored for the range 0.5-1
(with first% symbol as 1)
        end
        if rem(alpha,2) == 0    % if no. of '1's in the sequence is even
            if Symbols_Gray(j,1) == 1 % if the first symbol is 1
                X0_Dash = 1 -(A/Parameter_Mu);  % unmirror the interval
                %  and scale down by mu
            else                        % if the first symbol is 0
                X0_Dash = (A/Parameter_Mu); % leave the orientation
unhanged
                % scale down by mu
            end
        else
            if Symbols_Gray(j,1) == 1
                X0_Dash = 1 -(B/Parameter_Mu);
            else
                X0_Dash = (B/Parameter_Mu);
            end
        end
        X0_Dash_Array(j,1) = X0_Dash;     % store the estimated result
        A = 0;                            % reset all variables for the
        B = 0;                              % for the next new
estimation
        Delta= 0;
        l = 0;
        alpha = 0;
        X0_Dash = 0;
    end

    %% Algorithm from this research
    %% Ideal DM values - look up table
    VHDL_bits = resolution + 8;
    for i = 1:1:(resolution - 1)
        LUT(i) = (1/mpower(gain(g), i))-(1/pow2(i)); %Calculate
difference value
    end

    %% Sign for Difference Measure (SDM)
    for i = 1: 1: number_samples %Samples of input signal
        SDM(1, i) = Dout(g,1,i); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(g,2,i), Dout(g,3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(g,res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end

    %% Difference Measure: selected for each respective gray code bit
```

```
    %Select the DM values determined using each μ being employed by the
compensation algorithm
    for i = 1: 1: number_samples %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
        for res = 2: 1: resolution   % gives remaining bits of DM
            if (Dout(g,res, i) > 0)
                DV(res, i) = LUT(res - 1);
            else
                DV (res, i) = 0;
            end
        end
    end

    %% Signed Difference Value
    %Calculate the DV values determined using each μ being employed by
the compensation algorithm
    for i = 1: 1: number_samples %Samples of input signal

        for res = 1: 1: resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV(res, i) = DV(res, i);
            else
                SDV(res, i) = -DV(res, i);
            end
        end
    end

    % Determine DV
    for i = 1: 1: number_samples %decimal of SDV
        SDV_sum(i) = sum(SDV(:,i));
        SDV_dec(i) = SDV_sum(i);
    end

    %% Implement correction
    %Implement the compensation employing the DV values determined using
each μ being employed by the compensation algorithm
    %compensated ADC output
    for i = 1: 1: number_samples

        if (SDM(1,i) == 1)
            corrected_output(g,i) = output_representation(g, i) -
SDV_dec(i);
        else
            corrected_output(g,i) = output_representation(g, i) +
SDV_dec(i);
        end
    end
    %% Output after correction
    X0_Dash_Array_T(g, :) = transpose(X0_Dash_Array);
    uncorrected_difference(g, :) = (output_representation(g, :)*(Vmax -
Vmin)- y)/Step_size;   %uncompensated difference
    Basu_corrected_difference(g, :) = ((X0_Dash_Array_T(g, :)*(Vmax -
Vmin)) - y)/Step_size; % compensated difference for Basu's Method
    Research_corrected_difference(g, :) = ((corrected_output(g, :)*(Vmax
- Vmin)) - y)/Step_size;% compensated difference for this research's
method

    % establishing bit accuracy before and after compensation
    UD(g, 1) = max(abs(uncorrected_difference(g, :)));
    Basu_CD(g, 1) = max(abs(Basu_corrected_difference(g, :)));
    Research_CD(g, 1) = max(abs(Research_corrected_difference(g, :)));
    effective_bit_accuracy_UD(g, 1) = resolution  - log2(ceil(UD(g, 1)))
- 1;  %if UD = 1 bit accuracy should be resolution - 1
    Basu_effective_bit_accuracy_CD(g, 1) = resolution -
log2(ceil(Basu_CD(g, 1))) - 1;
    Research_effective_bit_accuracy_CD(g, 1) = resolution -
log2(ceil(Research_CD(g, 1))) - 1;
end
```

B.5 Code for VHDL Implementation of the Fundamental Tent Map Gain Compensation Algorithm

B.5.1 VHDL Code to Control the TM-ARCHα-7 ADC

The following code is an adaption of the VHDL code developed by Richard Haigh [56, 106] to acquire and process the TM-ARCHα-7 output. Additional lines of code added during this research to the original code listing have been highlighted blue. The original source code can be found in [106].

```
----------------------------------------------------------------------
--Title: Folding ADC Operation Control (FA_clock.vhd)
--Author: Richard Haigh
--Date: 12/03/17
--Availability:
http://eprints.hud.ac.uk/id/eprint/34742/1/Haigh%20THESIS.pdf
-- Date Edited: 20/10/2019
----------------------------------------------------------------------

-- declare libraries--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

-- define entity of FA_CLOCK module --
ENTITY FA_CLOCK IS
PORT  (clkp1, clkp2, clkp3, clkp4 : out std_logic; --clks to drive ADC
       clkn1, clkn2, clkn3, clkn4 : out std_logic;
       PLL_IN : in std_logic;         --clk to PLL module
       PLL_RST: in std_logic;
       PLL_LOCK : OUT std_logic;
       data_out : out std_logic_vector (7 downto 0);  -- parallel data
output pins
       CE0, CF0, CG0, CH0, CI0, CJ0, CK0, CL0 : in std_logic; --
Comparator outputs
      -- O   E    O    E    O    E    O    E Comparator Group: O = odd;
E = even
       GorB : in  std_logic;
       just_3V3 : out std_logic;
       just_0V : out std_logic);
END ENTITY;

-- Define Architecture of FA_CLOCK
ARCHITECTURE behav OF FA_CLOCK IS

component PLLTEST1 is -- declare component PLLTEST1 (produced 250 MHz
clock)
      PORT
      (areset     : IN STD_LOGIC  := '0';
       inclk0     : IN STD_LOGIC  := '0';
       c0         : OUT STD_LOGIC ;
       locked     : OUT STD_LOGIC);
      END component PLLTEST1;
```

```
component gain_correction is -- declare compensation gain_correction (µ
compensation algorithm)
      generic (n: positive:= 8);
      PORT  (PLL_OUT : in std_logic;  -- PLL clock (feb 17: 250 MHz)
             res : in std_logic;  -- resets all Difference registers
             Input_Gray : in std_logic_vector (n-1 downto 0); -- TM-
based ADC Gray Code output
             correct_en : in std_logic; -- enable compensation
             correct_fin : out std_logic; -- finish compensation
             UncorrectedBINARY      : in std_logic_vector (n-1 downto
0);   -- uncompensated Binary Code
             CorrectedBINARY : out std_logic_vector (n-1 downto 0));    -
- compensated Binary Code
END component gain_correction;

-- Define signals for FA_CLOCK
SIGNAL count : integer := 0; -- to track number of PLL clock cycles
signal clk  : std_logic := '0'; -- PLLTEST1 output (PLL clock)
signal CA0 , CB0 , CC0, CD0 : std_logic:= '0'; -- 4 MSB bits of TM-based
ADC output set to 0
--This VHDL module was originally designed for a 12-bit TM-based ADC,
but was adapted for a 8-bit version.
-- For this reason, the 4 MSBs are set to 0.

-- 5 arrays to store the TM-based ADC output and align the Gray code
values
signal A0 : std_logic_vector (1 downto 0)       := "00";
signal A1 : std_logic_vector (3 downto 0)       := "0000";
signal A2 : std_logic_vector (5 downto 0)       := "000000";
signal A3 : std_logic_vector (7 downto 0)       := "00000000";
SIGNAL A4 : STD_LOGIC_VECTOR (9 DOWNTO 0)       := "0000000000";
signal A5 : std_logic_vector (11 downto 0)      := "000000000000";

-- 1 bit arrays to store and transfer the comparator outputs
signal CA1 , CA2 , CB1 , CB2 , CC1 , CC2 , CD1 , CD2 , CE1 , CE2 , CF1 ,
CF2 , CG1, CG2, CH1, CH2 , CI1, CI2 , CJ1 , CJ2 , CK1, CK2 , CL1 , CL2 :
std_logic := '0';
signal bin_out : Std_logic_vector(11 downto 0):= "000000000000"; --
array to store the uncompensated binary output
signal gray_out: std_logic_vector(11 downto 0):= "000000000000"; --
array to store the Gray code output
signal clkp, clkn : std_logic:='0';  -- two types of non-overlapping
clocks

-- additional signals for gain correction module
signal buff_in : std_logic:='0'; -- ready to acquire compensated data
from µ compensation module
signal buff_out   : std_logic:='0'; -- ready to output data to µ
compensation module
signal cor_res : std_logic:='0';
signal cor_bin_out : std_logic_vector (7 downto 0) := (others => '0');

BEGIN

-- assign signals to PLLTEST1
      PLL0: PLLTEST1
      port map ( --PLL module creates 250 MHz
                areset=> PLL_RST ,
                inclk0 => PLL_IN , -- 50 MHz in
                c0 => clk ,           -- 250 MHz out
                locked => PLL_LOCK); -- portmap PLL

-- assign signals to gain_correction
      GC_Correct: gain_correction
      port map (              --correct ADC output
            PLL_OUT => clk,    -- 250 MHz clk
            res   => cor_res ,
```

```
            Input_Gray => A5(7 downto 0),
            correct_en => buff_out,
            correct_fin => buff_in,
            UncorrectedBINARY => bin_out(7 downto 0),
            CorrectedBINARY => cor_bin_out);
      just_3v3 <= '1'; -- for jump GorB connector
      just_0v <= '0';

      -- 4 MSBs set as zero.
      CA0 <= '0';
      CB0 <= '0';
      CC0 <= '0';
      CD0 <= '0';

      -- assign clock signals (clkp1 is the s/h clock)
      clkp1 <= clkn;
      clkp2 <= clkp;
      clkp3 <= clkp;
      clkp4 <= clkp;
      clkn1 <= clkn;
      clkn2 <= clkn;
      clkn3 <= clkn;
      clkn4 <= clkn;

      ADC_OPERATION: PROCESS

      BEGIN
      WAIT UNTIL RISING_EDGE(clk);   --clk is the PLL output
      -- Non over lapping clock generation statements
      IF count < 4 THEN
            clkp <= '0';
            clkn <= '1';
            count <= count + 1;

      ELSIF count = 4 THEN
            clkp <= '0';
            clkn <= '0';
            count <= count + 1;

      ELSIF count > 4 AND count < 9 THEN
            clkp <= '1';
            clkn <= '0';
            count <= count + 1;

      ELSIF count = 9 THEN
            clkp <= '0';
            clkn <= '0';
            count <= 0 ;
      END IF;
      --dave counts up 0 to 9
      --data control

      IF count = 9 THEN----------------------------------------------
--------------------------------0
      --sync odds
            CA2 <= CA1;
            CC2 <= CC1;
            CE2 <= CE1;
            CG2 <= CG1;
            CI2 <= CI1;
            CK2 <= CK1;

      ELSIF count = 0 THEN-------------------------------------------
--------------------------------1
      --shift array
            A5(11 downto 2) <= A4 (9 downto 0);
            A4(9 downto 2) <= A3 (7 downto 0);
            A3(7 downto 2) <= A2 (5 downto 0);
```

```
            A2(5 downto 2) <= A1 (3 downto 0);
            A1(3 downto 2) <= A0 (1 downto 0);

      ELSIF count = 1 THEN-------------------------------------------------
---------------------------------2
      --load odds in array
            A0(1) <= CA2;
            A1(1) <= CC2;
            A2(1) <= CE2;
            A3(1) <= CG2;
            A4(1) <= CI2;
            A5(1) <= CK2;

      ELSIF count = 2 THEN-------------------------------------------------
-----------------------------------3
            buff_out <= '0'; -- end correction
      ELSIF count = 3 THEN-------------------------------------------------
---------------------------------4
      --sample_evens <= '1';
            CB1 <= CB0;
            CD1 <= CD0;
            CF1 <= CF0;
            CH1 <= CH0;
            CJ1 <= CJ0;
            CL1 <= CL0;

      ELSIF count = 4 THEN-------------------------------------------------
---------------------------------5
      -- sync evens 1
            CB2 <= CB1;
            CD2 <= CD1;
            CF2 <= CF1;
            CH2 <= CH1;
            CJ2 <= CJ1;
            CL2 <= CL1;

      ELSIF count = 5 THEN-------------------------------------------------
---------------------------------6
      ELSIF count = 6 THEN-------------------------------------------------
---------------------------------7
      --load evens in array
            A0(0) <= CB2;
            A1(0) <= CD2;
            A2(0) <= CF2;
            A3(0) <= CH2;
            A4(0) <= CJ2;
            A5(0) <= CL2;

      ELSIF count = 7 THEN-------------------------------------------------
---------------------------------8
      --convert Gray code to binary
            bin_out (11) <= A5(11);
            bin_out (10) <= A5(11) xor A5(10);
            bin_out (9) <= (A5(11) xor A5(10)) xor A5(9);
            bin_out (8) <= ((A5(11) xor A5(10)) xor A5(9)) xor A5(8);
            bin_out (7) <= (((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7);
            bin_out (6) <= ((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6);
            bin_out (5) <= (((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6)) xor A5(5);
            bin_out (4) <= ((((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6)) xor A5(5))xor A5(4);
            bin_out (3) <= (((((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6)) xor A5(5))xor A5(4)) xor A5(3);
            bin_out (2) <= ((((((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6)) xor A5(5))xor A5(4)) xor A5(3)) xor A5(2);
```

```
            bin_out (1) <= (((((((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6)) xor A5(5))xor A5(4)) xor A5(3)) xor A5(2))
xor A5(1);
            bin_out (0) <= ((((((((((A5(11) xor A5(10)) xor A5(9)) xor
A5(8))xor A5(7)) xor A5(6)) xor A5(5))xor A5(4)) xor A5(3)) xor A5(2))
xor A5(1)) xor A5(0);
            buff_out <= '1'; --start compensation
      ELSIF count = 8 THEN-----------------------------------------------
-----------------------------------9
            --sample odds
            CA1 <= CA0;
            CC1 <= CC0;
            CE1 <= CE0;
            CG1 <= CG0;
            CI1 <= CI0;
            CK1 <= CK0;
      End if;

END PROCESS ADC_OPERATION;
--data_out <= bin_out(7 downto 0);  --output ADC output
data_out <= cor_bin_out;      --output corrected ADC output
END behav;
```

B.5.2 VHDL Implementation of the µCA-1

The following code is the VHDL implementation of the µCA-1 from this research.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

ENTITY gain_correction is
generic (n: positive:= 8); -- define resolution of TM-based ADC
PORT  (PLL_OUT: in std_logic;  -- PLL clock (feb 17: 250 MHz)
       res: in std_logic;  -- resets all Difference registers
       Input_Gray: in std_logic_vector(n-1 downto 0); --ADC output in Gray
code
       correct_en: in std_logic;
       correct_fin: out std_logic;
       UncorrectedBINARY: in std_logic_vector(n-1 downto 0); --ADC output
in binary
       CorrectedBINARY: out std_logic_vector(n-1 downto 0)); --Corrected
ADC output
END ENTITY;

ARCHITECTURE behav OF gain_correction IS

-- define signals to be employed within the module
signal Graycode: std_logic_vector(n-1 downto 0):= (others => '0');
signal SDM_reg: std_logic_vector(n-1 downto 0):= (others => '0');
signal UC_binary: std_logic_vector((n + 2)-1 downto 0):= (others => '0');
signal C_binary: std_logic_vector((n + 2)-1 downto 0):= (others => '0');
signal SDV_reg: std_logic_vector((n + 2)-1 downto 0):= (others => '0');
signal correct: std_logic_vector(n-1 downto 0):= (others => '0');
signal counter: integer := 0;
signal C_fin: std_logic := '0';
signal do_correct: std_logic := '0';
```

```
-- LUT for gain = 1.9; DM resolution = 10 bits
type LUT is array (n-2 downto 0) of std_logic_vector((n + 2)-1 downto 0);
--precalculated DM values
signal diff_bit: LUT :=  --binary values multiplied by 2^(8+2) = 1024
(need to shift back by 2)
(     "0000011010",
      "0000011011",
      "0000010101",
      "0000001110",
      "0000001001",
      "0000000101",
      "0000000011");

type Diff is array (n-1 downto 0) of std_logic_vector((n + 2)-1 downto 0);
-- array to store relevant DM values
signal assign_diff: Diff;

type sum_Diff is array (n-2 downto 0) of std_logic_vector((n + 2)-1 downto
0); -- array to aid the calculation of the DV value
signal polarity_diff: sum_Diff;

BEGIN
      -- start compensation
      Gain_compensate: PROCESS
      BEGIN
      WAIT UNTIL RISING_EDGE(PLL_OUT);   --PLL_IN the PLL clock 250MHz

      -- obtain Gray code and binary code from the TM-based ADC
      Graycode(n-1 downto 0) <= Input_Gray;
      UC_binary((n + 2)-1 downto (n + 2)-n)     <= UncorrectedBINARY;  --
binary code from ADC form the 8 MSBs of this array
      UC_binary((n + 2)-(n+1) downto 0)   <= (others => '0');         --
remaining LSBs are set to zero.

      --Should the correction process be applied?
            IF correct_en = '1' AND counter < 3 THEN --yes
                  C_fin <= '0';           -- compensation is in process
                  counter <= counter + 1; -- increment counter
                  -- Find sign of difference measure (3 MSBs)
                  SDM_reg(n-1) <= Graycode(n-1);
                  SDM_reg(n-2) <= '1';
                  SDM_reg(n-3) <= (Graycode(n-2) XOR Graycode(n-3));

--                -- Find Difference measure--
                  assign_diff(n-1) <= (others => '0');
--
                  --Do correction
                  polarity_diff(n-2) <= assign_diff(n-2);
--
                  if SDM_reg(n-1) = '1' then -- is MSB of Gray code = 1?
                        C_binary <= UC_binary - SDV_reg;

                else         -- is MSB of Gray code = 0?
--
                        C_binary <= UC_binary + SDV_reg;
                  end if;
            ELSIF counter = 3 AND   correct_en   =   '1'   THEN   --   no,
compensation finished
                  C_fin <= '1';
            ELSIF counter = 3 AND correct_en = '0' THEN --no
                  counter <= 0;
            ELSE
                  null;
            END IF;
      END process;

      -- Sign for Difference measure
```

```
      SDM: for i in n-4 downto 0 generate
      begin
            SDM_reg(i) <= (SDM_reg(i+1) XOR Graycode(i));
      end generate SDM;

      -- Assign difference measure
      DV: for i in n-2 downto 0 generate
      begin
            with Graycode(i) select
            assign_diff(i)  <= diff_bit(i) when '1',
            (others => '0') when others;
      end generate DV;

      -- Do correction/ calculate difference value
      SDV: for i in n-3 downto 0 generate
      begin
            with SDM_reg(i) select
            polarity_diff(i) <= polarity_diff(i+1) + assign_diff(i) when
'1',
            polarity_diff(i+1) - assign_diff(i) when others;
            SDV_reg <= polarity_diff(0);
      end generate SDV;

CorrectedBINARY <= C_binary((n + 2)-1 downto (n + 2)-n); -- Transmit
corrected binary code
correct_fin <= C_fin;
END behav;
```

B.5.3 MATLAB Script to Aid Creation of TM-ARCHα-7 ADC Signal Emulator

The MATLAB script below was developed to establish a sequence of values to be added to the sum of the previous value. These values created the Gray code that a TM-ARCHα-7 ADC, with a μ = 1.9 and supplied a full-scale ramp input signal, would produce if the comparator outputs had been aligned. The sequence of values was then employed by the TM-ARCHα-7 ADC Signal Emulator (see Section B.5.4) and enabled this component to produce a similar output to a TM-based ADC before the comparator outputs are aligned by the control logic within the FPGA.

```
%% Characteristics for Tent Map
gain = [1.9];
Vmax = 3.0;
Vmin = 0;
Vref = 1.5;                         %set reference voltage
Step_size = (Vmax-Vmin)/(2^resolution); %calculating step size
%% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
samples = (2^(resolution+1));       %number of samples
dt = 1/samples;
x = 0:dt:1;
y = (Vmax-Vmin)*x;

%% Gain look up table
for i = 1:1:(resolution - 1)
    LUT(i) = (1/mpower(gain, i))-(1/pow2(i));
end

%% Input goes through TMs %
for i = 1: 1: length(y) %Samples of input signal
    z(1,i) = y(i);
    for res = 1: 1: resolution % Folds and finds gray code word of
sample
        if (z(res, i) <= Vref) % if input to the folding stage is less
than or equal to the reference voltage
            z((res+1), i) = gain*z(res, i);
            Dout(res, i) = 0;
        elseif (z(res, i) > Vref)                    % if input to the
folding stage is more than the reference voltage
            z((res +1), i) = gain*(2*Vref - z((res), i));
            Dout(res, i) = 1;
        end
    end
end
%z array gives the inputs to each Tent map stage
%Dout array provides the Gray code output

%% Implement correction
%uncorrected output
for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
    gray_code_vector = Dout(:,i);           %get Gray code word
    gray_rep = transpose(gray_code_vector);
    gray_int(i) = bi2de(gray_rep, 'left-msb');

    %looking at binary equivalent
    binary = gray2bin(gray_code_vector);    %convert Gray code word to
binary
    bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
    bin_rep = transpose(bin_representation(:,i));
    bin_int(i) = bi2de(bin_rep, 'left-msb');
end
%% Determine Ramp add values (to be employed by the signal emulator)
ramp_diff_shift = [gray_int(2 : length(gray_int) ), gray_int(1)]; %
shift gray_int values right by on in the array. Bring gray_int(1) to the
leftmost index in the array.
ramp_add = ramp_diff_shift - gray_int; % determine the next value to be
added to the output to create a ramp
```

B.5.4 VHDL code of the TM-ARCHα-7 ADC Signal Emulator

The following code was developed to imitate the output of a TM-ARCHα-7 ADC, with a μ = 1.9, when supplied with a ramp input signal.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

ENTITY Signal_pipelined_generator is
generic (n: integer:= 8);
PORT  (clk_p: in std_logic;  -- s/h clock       (25 MHz)
       res: in std_logic;  -- resets all Difference registers
       Output_signal_piped:  out  std_logic_vector(n-1  downto  0));  --
equivalent output of the TM-ADC
END ENTITY;

ARCHITECTURE behav OF Signal_pipelined_generator IS
signal Gen_counter: integer := 0;
signal in_value: integer := 0;          --value into the "TM-ADC"
signal in_valueV: std_logic_vector(7 downto 0):= (others => '0'); --binary
equivalent of in_value
signal out_value: std_logic_vector(7 downto 0):= (others => '0'); --output
signal S0: std_logic_vector(1 downto 0):= (others => '0');
signal S1: std_logic_vector(3 downto 0):= (others => '0');
signal S2: std_logic_vector(5 downto 0):= (others => '0');
signal S3: std_logic_vector(7 downto 0):= (others => '0');

type add2prev is array (0 to 2**(n+1)) of integer;
signal diff: add2prev :=            --difference values for ramp
(
-- ramp signal emulation array--
-- contents generated by MATLAB Script presented in B.5.3--
);
begin
      form_signal_odds: PROCESS
      BEGIN
      WAIT UNTIL RISING_EDGE(clk_p);   --PLL_IN the PLL clock 250MHz

      in_value  <=  in_value  +  diff(Gen_counter);   --Add  difference  to
previous input amplitude
      in_valueV  <= std_logic_vector(to_unsigned(in_value, n)); --convert
to binary
      S3(7 downto 6) <= in_valueV(7 downto 6); -- 2 MSBs go to the 2 MSBs
of array S3
      S2(5 downto 4) <= in_valueV(5 downto 4); -- 3rd and 4th MSBs go to
the 2 MSBs of array S2
      S1(3 downto 2) <= in_valueV(3 downto 2); -- 3rd and 4th LSBs go to
the 2 MSBs of array S1
      S0(1 downto 0) <= in_valueV(1 downto 0); -- 2 LSBs go to the 2 MSBs
of array S0

      out_value <= S3;   --ADC output would be the same as S3
      if Gen_counter < 2**(n+1) then
            Gen_counter <= Gen_counter + 1;     --increment counter
            S3(5  downto  0)  <=  S2(5  downto  0);  --shift  S2  values  into
S3[5:0]
            S2(3  downto  0)  <=  S1(3  downto  0);  --shift  S1  values  into
S2[3:0]
```

```
            S1(1 downto 0) <= S0(1 downto 0); --shift S0 values into
S1[1:0]
      else
            Gen_counter <= 0;                                        --
reset counter
            S3(5 downto 0) <= S2(5 downto 0); --shift S2 values into
S3[5:0]
            S2(3 downto 0) <= S1(3 downto 0); --shift S1 values into
S2[3:0]
            S1(1 downto 0) <= S0(1 downto 0); -- 2 LSBs go to the 2 MSBs
of array S0
      end if;

end process;
Output_signal_piped     <= out_value;

END behav;
```

B.5.5 Combining Components for Test

Figure B-1 is a schematic connecting the FA_clock and Signal_pipelined_generator components (see Appendix B.5.1 and Appendix B.5.4 respectively) for testing.

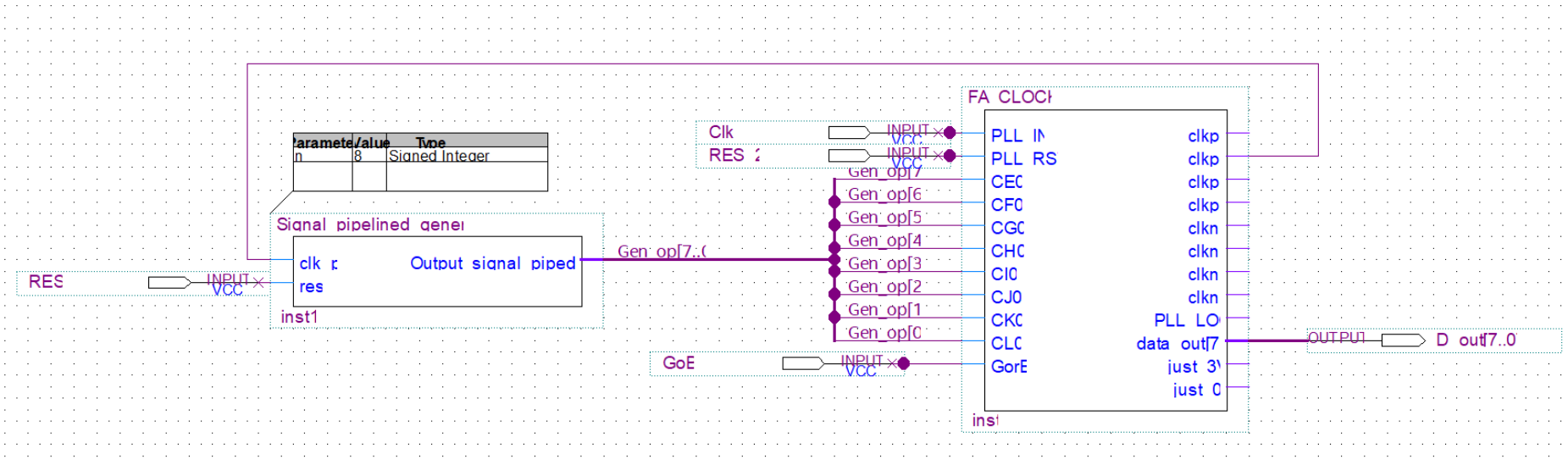


Figure B-1: Components combined using a schematic within Quartus.

B.5.6 Test Bench for Testing the µCA-1 VHDL Implementation

The following code is the test bench developed to test the implemented µCA-1 via simulation.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

ENTITY ramp_to_sys_tb IS
END ENTITY;

ARCHITECTURE behav OF ramp_to_sys_tb IS

signal clk_50MHz: std_logic := '0'; --50 MHz signal to PLL module
signal R1:  std_logic := '0';  -- reset 1
signal R2:  std_logic := '0';  -- reset 2
signal G_B: std_logic := '0';
signal DATA: std_logic_vector(7 downto 0) := (others => '0'); -- output
data from  main component
signal num: integer := 0;  -- data converted to integer
file Results: text; -- converted data to be saved to a text file

component repeat_of_orginial_algorithm_experiment is -- signal generator
and FA_clock combined
      PORT  (GoB:  IN  STD_LOGIC;
             RES_2:  IN  STD_LOGIC;
             Clk:  IN  STD_LOGIC;
             RES:  IN  STD_LOGIC;
             D_out:     OUT STD_LOGIC_VECTOR(7 downto 0));
      end component  repeat_of_orginial_algorithm_experiment;

Begin

 main_component_INST : repeat_of_orginial_algorithm_experiment
 port map
 (GoB => G_B,
  RES_2 => R2,
  CLK => clk_50MHz,
  RES =>    R1,
  D_out => DATA);

clking_50: Process  -- generate 50 MHz clock
begin
      number1: for i in 1 to 100000 loop
      clk_50MHz <= '1';
      wait for 10 ns;
      clk_50MHz <= '0';
      wait for 10 ns;
      end loop number1;
      end process clking_50;

      save_results: Process -- save data into text files
      variable O_line   : line;
      begin

      --file_open(Results, "Before_correction.txt", write_mode);  --
before correction
      file_open(Results, "After_correction.txt", write_mode); --after
correction
```

```
      number2: for i in 1 to 100000 loop
      num <= to_integer(unsigned(DATA)); --convert binary ADC output to
integers
      write(O_line, num, right, 8);      --write data to file
      writeline(Results, O_line);

      wait for 40 ns;
      end loop number2;
      file_close(Results);
      end process save_results;

end behav;
```

B.6 MATLAB Scripts for Approximating Difference Measure Values for the Fundamental Tent Map Gain Compensation Algorithm

B.6.1 Code for Creating the SLE&A Equations

The code below was used to develop the straight-line approximation of the DM versus μ plots for all bits (except the MSB). Straight line approximations of the error versus μ plots were also determined. The equations determined from these straight-line approximations were employed in the SLE&A method discussed in Section 5.6 and presented in Appendix B.6.2.

```
%% Ideal DM values -  look up table
resolution = 16;   % number of TM stages - 1
gain = [1.9: 0.0001: 2];  % gain range
gain_t = transpose(gain);
for g = 1:1:length(gain)
    for i = 1:1:(resolution)-1
        LUT(g,i) = (1/mpower(gain(g), i))-(1/pow2(i));
    end
end

G = length(gain);  % number of gains to be plotted
%% Do straight-line approximation

n = 1: 1: resolution - 1;
Results = zeros(2, resolution - 1);
for i = 1:1:resolution - 1 % calculate gradient
    a = (LUT(G,i) - LUT(1,i))/(gain(G) - gain(1));
    b = LUT(G,i) - a*gain(G);
    Results(1, i) = a;
    Results(2, i) = b;
end

for g = 1:1:length(gain)
    for i = 1:1:(resolution - 1)
        a = Results(1,i);
        b = Results(2,i);
        LUT_approx(g, i) = a*gain(g) + b;
    end
end

%% straight line approxs for error approx
DIFF = LUT - LUT_approx; % calculate error

for i = 1:1:resolution - 1

  %approximation for first half
    y_grad = (DIFF((1+ length(gain))/2 ,i) - DIFF(1,i));
    x_grad = gain(1,(1+ length(gain))/2) - gain(1,1);
    a = y_grad/x_grad;
    b = DIFF((1+ length(gain))/2,i) - a*gain(1,(1+ length(gain))/2); %
constant calculated when u = 1.95
```

```
    Errors1(1, i) = a;
    Errors1(2, i) = b;

   %approximation for second half
    c = (DIFF(G,i) - DIFF((1+ length(gain))/2,i))/((gain(1,G) -
gain(1,(1+ length(gain))/2)));
    d = DIFF((1+ length(gain))/2,i) - c*gain(1,(1+ length(gain))/2); %
constant calculated when u = 1.95
    Errors2(1, i) = c;
    Errors2(2, i) = d;
 end
```

B.6.2 Code for Simulating SLE&A Method

The code below presents how the SLE&A method was implemented to approximate the DM values employed within the µCA-1.

```
%% load data obtained from curve_fitting_for_gain.m
resolution = 16;              %Number of TM stages
%curvefittingforgain =
load('Straight_line_approx_1.9_to_2_0.0001inc.mat');
curvefittingforgain =
load('Straight_line_approx_min_point_1.9_to_2_0.0001inc.mat');
Indexs = load('error_grad_change.mat');

%% Characteristics for Tent Map
gain = [1.9: 0.01:1.99];  %TM gain
gain_max = 2;   % Min and max values used in straight-line approximation
gain_min = 1.9;
gain_mid = 1.95; %mid-point of gain values approximated
Vmax = 3;
Vmin = 0;
Vref = 1.5;                      %set partition point voltage
Step_size = (Vmax-Vmin)/(2^resolution); %calculating step size
%% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html

Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/pow2(resolution + 2); %380 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T-dt;
y = (Vmax-Vmin)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2;  % 0 to 3 V at
380 Hz ramp input signal
number_samples = length(x);
%% DM values -  look up table

for g = 1:1:length(gain)
    %ideal DM values
    for i = 1:1:(resolution - 1)
        LUT(i) = (1/mpower(gain(g), i))-(1/pow2(i));
    end

    %approximated DM values
    % straight line approx
    for i = 1:1:(resolution - 1)
```

```
        a = curvefittingforgain.Results(1,i);
        b = curvefittingforgain.Results(2,i);

        if gain(g) <= gain_mid
            c = curvefittingforgain.Errors1(1,i);
            d = curvefittingforgain.Errors1(2,i);
        else
            c = curvefittingforgain.Errors2(1,i);
            d = curvefittingforgain.Errors2(2,i);
        end
        Error_approx(i) = (c*gain(g) + d);
        LUT_approx(i) = (a*gain(g) + b) + Error_approx(i);
    end

    for i = 1: 1: number_samples %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            if ((z(res, i) <= Vref) && (z(res, i) >= Vmin))  % if input
to the folding stage is less than or equal to the reference voltage
                z((res+1), i) = gain(g)*z(res, i);
                Dout(g, res, i) = 0;
            elseif ((z(res, i) > Vref) && (z(res, i) <=
Vmax))                      % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain(g)*Vref)-(gain(g)*(z((res), i)-
Vref));
                Dout(g, res,i) = 1;
            elseif (z(res, i) > Vmax)
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 1;
            else
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 0;
            end
        end

        if (z(resolution, i) <= (Vref))
            Dout(g, resolution, i) = 0;
        else
            Dout(g, resolution, i) = 1;
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output


    %% Sign for Difference Measure (SDM)

    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(g, 1,i); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(g, 2,i), Dout(g, 3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(g, res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end
    %% Difference Measure: selected for each respective gray code bit
    for i = 1: 1: length(y) %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
```

```
        for res = 2: 1: resolution   % gives remaining bits of DV
            if (Dout(g, res, i) > 0)
                DV(res, i) = LUT(res - 1); %ideal
                DV_approx(res, i) = LUT_approx(res - 1); %approx
            else DV(res, i) = 0;
                DV_approx(res, i) = 0;
            end
        end
    end
    %% Signed Difference Value

    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV(res, i) = DV(res, i); %ideal
                SDV_approx(res, i) = DV_approx(res, i); %approx
            else SDV(res, i) = -DV(res, i);  %ideal
                SDV_approx(res, i) = -DV_approx(res, i); %approx
            end
        end
    end

    %DV values
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum(i) = sum(SDV(:,i));
        SDV_dec(i) = (Vmax-Vmin)*SDV_sum(i);  %DV ideal
        SDV_sum_approx(i) = sum(SDV_approx(:,i));
        SDV_dec_approx(i) = (Vmax-Vmin)*SDV_sum_approx(i); %DV approx.
    end


    %% Implement compensated

    %uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(g, :,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution
            decimal_rep = (Vmax-Vmin)*(binary(j)/(2^j))+ decimal_rep ;
        end
        output_representation(i) = decimal_rep  ; %modify decimal value
so it lies within the input voltage range
    end

    %compensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        if (SDM(1,i) == 1)
            corrected_output(i) = output_representation(i) -
SDV_dec(i);       %ideal
            corrected_output_approx(i) = output_representation(i) -
SDV_dec_approx(i);  %approx
        else
            corrected_output(i) = output_representation(i) +
SDV_dec(i);         %ideal
            corrected_output_approx(i) = output_representation(i) +
SDV_dec_approx(i); %approx
        end
    end

    uncorrected_difference = (output_representation - y)/Step_size;
    corrected_difference = (corrected_output - y)/Step_size;
```

```
    corrected_difference_approx = (corrected_output_approx -
y)/Step_size;

    %% display key information (Effective resolution)

    UD = max(abs(uncorrected_difference))
    CD = max(abs(corrected_difference_approx))
    CD_ideal = max(abs(corrected_difference))
    effective_bit_accuracy_UD(g) = resolution - (log2(ceil(UD))
+1);  %if UD = 1 bit acc should be resolution - 1
    effective_bit_accuracy_CD(g) = resolution - (log2(ceil(CD)) +1);
    effective_bit_accuracy_CD_ideal(g) = resolution -
(log2(ceil(CD_ideal)) +1);

    Ideal_and_approx_difference = corrected_difference -
corrected_difference_approx;
end
```

B.6.3 Code for Simulating SA Method

The code below presents how the SA method was implemented to approximate the DM values employed within the µCA-1.

```
%% Characteristics for Tent Map
resolution = 16;              %number of TM stages - 1
approx_gain = [1.9]; %u_o
gain = [1.9: 0.01: 1.99];      %u_c
Vmax = 3;             %valid input signal max.
Vmin = 0;             %valid input signal min.
Vref = 1.5;                    %set partition point voltage
Step_size = (Vmax-Vmin)/(2^resolution); %calculating step size
%% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
f_fundamental = 380; % 380 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
Fs = 25000000;      %sample rate = 25 MHz
dt = 1/Fs;
x = 0:dt:T-dt;
y = (Vmax-Vmin)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2;  % 0 to 3 V at
380 Hz ramp input signal.
number_samples = length(x);

for g = 1: 1: length(gain)
    %% DM values -  look up table
    scalar(g) = (1-(gain(g) - approx_gain)/(2-approx_gain));
    %approx DM values
    for i = 1:1:(resolution - 1)
        LUT(i) = ((1/mpower(approx_gain, i))-(1/pow2(i)));
    end
    %ideal DM values
    for i = 1:1:(resolution - 1)
        LUT_ideal(i) = ((1/mpower(gain(g), i))-(1/pow2(i)));
    end

    LUT_approx = scalar(g)*LUT;
    %% Input goes through TM-based ADC %
    for i = 1: 1: number_samples %Samples of input signal
        z(1,i) = y(i);
```

```
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            if ((z(res, i) <= Vref) && (z(res, i) >= Vmin))  % if input
to the folding stage is less than or equal to the reference voltage
                z((res+1), i) = gain(g)*z(res, i);
                Dout(g, res, i) = 0;
            elseif ((z(res, i) > Vref) && (z(res, i) <=
Vmax))                    % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain(g)*Vref)-(gain(g)*(z((res), i)-
Vref));
                Dout(g, res,i) = 1;
            elseif (z(res, i) > Vmax)
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 1;
            else
                z((res +1), i) = Vmin;
                Dout(g, res, i) = 0;
            end
        end

        if (z(resolution, i) <= (Vref))
            Dout(g, resolution, i) = 0;
        else
            Dout(g, resolution, i) = 1;
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output

    %% Sign for Difference Measure (SDM)
    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(g, 1,i); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(g, 2,i), Dout(g, 3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(g, res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end
    %% Difference Measure: selected for each respective gray code bit
    for i = 1: 1: length(y) %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
        for res = 2: 1: resolution   % gives remaining bits of DV
            if (Dout(g, res, i) > 0)
                DV(res, i) = LUT_ideal(res - 1); %ideal
                DV_approx(res, i) = LUT_approx(res - 1); %approx
            else DV(res, i) = 0;
                DV_approx(res, i) = 0;

            end
        end
    end
    %% Signed Difference Value

    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV(res, i) = DV(res, i); %ideal
                SDV_approx(res, i) = DV_approx(res, i); %approx
            else SDV(res, i) = -DV(res, i);  %ideal
```

```
                SDV_approx(res, i) = -DV_approx(res, i); %approx
            end
        end
    end

    %DV values
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum(i) = sum(SDV(:,i));
        SDV_dec(i) = (Vmax-Vmin)*SDV_sum(i);  %DV ideal
        SDV_sum_approx(i) = sum(SDV_approx(:,i));
        SDV_dec_approx(i) = (Vmax-Vmin)*SDV_sum_approx(i); %DV approx.
    end

    %% Implement compensated
    %uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(g, :,i);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: resolution
            decimal_rep = (Vmax-Vmin)*(binary(j)/(2^j))+ decimal_rep ;
        end
        output_representation(i) = decimal_rep  ; %modify decimal value
so it lies within the input voltage range
    end

    %compensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        if (SDM(1,i) == 1)
            corrected_output(i) = output_representation(i) -
SDV_dec(i);       %ideal
            corrected_output_approx(i) = output_representation(i) -
SDV_dec_approx(i);  %approx
        else
            corrected_output(i) = output_representation(i) +
SDV_dec(i);         %ideal
            corrected_output_approx(i) = output_representation(i) +
SDV_dec_approx(i); %approx
        end
    end

    uncorrected_difference = (output_representation - y)/Step_size;
    corrected_difference = (corrected_output - y)/Step_size;
    corrected_difference_approx = (corrected_output_approx -
y)/Step_size;

    %% display key information (Effective resolution)

    UD = max(abs(uncorrected_difference))
    CD = max(abs(corrected_difference_approx))
    CD_ideal = max(abs(corrected_difference))
    effective_bit_accuracy_UD(g) = resolution - (log2(ceil(UD))
+1);  %if UD = 1 bit acc should be resolution - 1
    effective_bit_accuracy_CD(g) = resolution - (log2(ceil(CD)) +1);
    effective_bit_accuracy_CD_ideal(g) = resolution -
(log2(ceil(CD_ideal)) +1);

    Ideal_and_approx_difference = corrected_difference -
corrected_difference_approx;
end
```

Appendix C

The code presented in this appendix relates to the simulation and practical results presented in Chapter 6.

C.1 MATLAB Scripts for Initial Bit Accuracy Predictions of the Enhanced Tent Map Gain Compensation Algorithms

C.1.1 Code for the µCA-2 Analysis

The code below shows how the µCA-2 was tested with a more complex TM-ARCHα-15 ADC mathematical model developed in MATLAB.

```
%% Initialise key parameters for model
resolution = 16;                %number of TM stages + 1
Vcc = 5; % valid input max.
Vee = -0.25; % valid input min.
vin_range = 3 - 0;
Vref = 1.5;                %set partition point voltage
Step_size = (vin_range)/(2^resolution); %calculating step size
%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = Total_resolution + 2;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T;
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);

%% Gain and Vref Parameters
number_tests = 100;
rng(0, 'twister');   % The code in this script section is based on
https://uk.mathworks.com/help/matlab/math/floating-point-numbers-within-
specific-range.html
a = 1.9;
b = 1.99;

gain_pos = (b-a).*rand(number_tests, resolution-1) + a; % different µ+
for each TM stage
gain_neg =  (b-a).*rand(number_tests, resolution-1) + a; % different µ-
for each TM stage
%% Input goes through TM-based ADC %
```

```
for test = 1: 1: number_tests
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        z(1,1) = y(1);
        %% first sample - assume higher than previous
        if ((z(res, 1) <= VH_pos(res)) && (z(res, 1) >= Vee))  % if
input to the folding stage is less than or equal to the reference
voltage
            z((res+1), 1) = gain_pos(test,res)*z(res, 1);
            Dout(res, 1) = 0;
        elseif ((z(res, 1) > VH_pos(res)) && (z(res, 1) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
            z((res +1), 1) = (gain_neg(test,res)*Vref)-
(gain_neg(test,res)*(z((res), 1)-Vref));
            Dout(res,1) = 1;
        elseif (z(res, 1) > Vcc)
            z((res +1), 1) = Vee;
            Dout(res, 1) = 1;
        else
            z((res +1), 1) = Vee;
            Dout(res, 1) = 0;
        end
    end
    %% positive ramp goes through TMs %
    for i = 2: 1: number_samples %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            %% Higher Hysteresis voltages
            if (z(res, i) > z(res, i - 1))
                if ((z(res, i) <= VH_pos(res)) && (z(res, i) >= Vee))  %
if input to the folding stage is less than or equal to the reference
voltage
                    z((res+1), i) = gain_pos(test,res)*z(res, i);
                    Dout(res, i) = 0;
                elseif ((z(res, i) > VH_pos(res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_neg(test,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref));
                    Dout(res,i) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(res, i) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(res, i) = 0;
                end
                %% Lower Hysteresis voltages
            else
                if ((z(res, i) <= VH_neg(res)) && (z(res, i) >= Vee))  %
if input to the folding stage is less than or equal to the reference
voltage
                    z((res+1), i) = gain_pos(test,res)*z(res, i);
                    Dout(res, i) = 0;
                elseif ((z(res, i) > VH_neg(res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_neg(test,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref));
                    Dout(res,i) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(res, i) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(res, i) = 0;
```

```
                end
            end
        end
        %% last comparator
        if ((z(resolution, i) > z(resolution, i - 1))||(z(resolution,
1)))
            if (z(resolution, i) <= VH_pos(resolution))
                Dout(resolution, i) = 0;
            else
                Dout(resolution, i) = 1;
            end
        else
            if (z(resolution, i) <= VH_neg(resolution))
                Dout(resolution, i) = 0;
            else
                Dout(resolution, i) = 1;
            end
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output

    %% Sign for Difference Measure (SDM)
    %   edited for non-matching gains
    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(1,i); %MSB of Gray code output
        SDM(2, i) = 1;         %1 shows adding function
        if xor(Dout(2,i), Dout(3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end
    %% Difference Measure: selected for each respective Gray code bit
    %Select the DM values determined using each μ being employed by the
compensation algorithm
    %Calculate all the DM values for each μ being employed by the
compensation algorithm

    for i = 1: 1: length(y) %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
        gain_factor = 1;
        for res = 2: 1: resolution   % gives remaining bits of DM
            %% calculate deviations from preferred implementation
            if  Dout(res-1, i) == 1
                gain_factor = gain_factor * gain_neg(test,res-1);
            else
                gain_factor = gain_factor * gain_pos(test,res-1);
            end
            %% calculate and add difference measure values
            if (Dout(res, i) == 1)
                DV(res, i) = ((1/gain_factor)-(1/pow2(res - 1 )));
            else
                DV(res, i) = 0;
            end
        end
    end
```

[…]

C.1.2 Code for the µCA-3 Analysis

The code below shows how the TM-ARCHβ-7-12 ADC was implemented as a mathematical model and tested with µCA-3.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
COTS_ADC_res = 12;
Total_resolution = resolution + COTS_ADC_res - 1;
Vcc = 5; % valid input max.
Vee = -0.232; % valid input min.
vin_range = 3 - 0;
Vref = 1.5;              %set partition point voltage
TM_Step_size = (vin_range)/(2^resolution); %calculating step size
ADC_Step_size = (vin_range)/(2^Total_resolution); %calculating step size
%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = Total_resolution + 2;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % 95000 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T-dt;
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x) + 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);

%% Preallocate size of arrays
number_tests = 10;
Dout = zeros(number_tests, Total_resolution, number_samples);

%% Gain and Vref Parameters
rng(0,
'twister');   %%https://uk.mathworks.com/help/matlab/math/floating-
point-numbers-within-specific-range.html
a = 1.9;
b = 1.99;
% produced µ+ and µ- for TM stages
gain_pos = (b-a).*rand(number_tests, resolution-1) + a;
%gain_neg = gain_pos;
gain_neg =  (b-a).*rand(number_tests, resolution-1) + a;
VH_pos = (Vref)*(ones(1, resolution));
VH_neg = (Vref)*(ones(1, resolution));

%% Input goes through TM-based ADC %
for test = 1: 1: number_tests
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        z(1,1) = y(1);
        %% first sample - assume higher than previous
        if ((z(res, 1) <= VH_pos(1, res)) && (z(res, 1) >= Vee))  % if
input to the folding stage is less than or equal to the reference
voltage
            z((res+1), 1) = gain_pos(test, res)*z(res, 1);
            Dout(test,res, 1) = 0;
        elseif ((z(res, 1) > VH_pos(1, res)) && (z(res, 1) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
```

```
            z((res +1), 1) = (gain_neg(test, res)*Vref)-
(gain_neg(test,res)*(z((res), 1)-Vref)); % TM implementation considered
for second adaptation
            %z((res +1), 1) = (gain_pos(test, res)*Vref)-
(gain_neg(test,res)*(z((res), 1)-Vref)); %electronic implementation of
TM
            Dout(test,res,1) = 1;
        elseif (z(res, 1) > Vcc)
            z((res +1), 1) = Vee;
            Dout(test,res, 1) = 1;
        else
            z((res +1), 1) = Vee;
            Dout(test,res, 1) = 0;
        end
    end
    %% positive ramp goes through TMs %
    for i = 2: 1: length(y) %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            %% Higher Hysteresis voltages
            if (z(res, i) > z(res, i - 1))
                if ((z(res, i) <= VH_pos(1, res)) && (z(res, i) >=
Vee))  % if input to the folding stage is less than or equal to the
reference voltage
                    z((res+1), i) = gain_pos(test,res)*z(res, i);
                    Dout(test,res, i) = 0;
                elseif ((z(res, i) > VH_pos(1, res)) && (z(res, i) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_neg(test,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref)); % TM implementation considered
for second adaptation
                    % z((res +1), i) = (gain_pos(test,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref)); % electronic implementation of
TM
                    Dout(test,res,i) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(test,res, i) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(test,res, i) = 0;
                end
                %% Lower Hysteresis voltages
            else
                if ((z(res, i) <= VH_neg(1, res)) && (z(res, i) >=
Vee))  % if input to the folding stage is less than or equal to the
reference voltage
                    z((res+1), i) = gain_pos(test,res)*z(res, i);
                    Dout(test,res, i) = 0;
                elseif ((z(res, i) > VH_neg(1, res)) && (z(res, i) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_neg(test,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref)); % TM implementation considered
for second adaptation
                    %z((res +1), i) = (gain_pos(test,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref)); % electronic implementation of
TM
                    Dout(test,res,i) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(test,res, i) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(test,res, i) = 0;
                end
```

```
            end
        end
        z_final(test, i) = z(resolution, i);
    end

    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output
end
%% clear unrequired variables
clear a
clear b
clear c
clear d
clear test
clear i
clear dt
clear f_fundamental
clear Fs
clear res
clear Vcc
clear Vee
clear z

COTS_ADC_input = zeros(number_samples, 1);
LSBs_vect = zeros(COTS_ADC_res, number_samples);
%% start Test
for test = 1: 1: number_tests
    for i = 1: 1: number_samples
        %% nth TM stage output goes through the COTS ADC
        COTS_ADC_input(i, 1) = z_final(test, i); %final TM output goes
through a commerical ADC
        %Use Bernoulli Map
        for res = 1: 1: COTS_ADC_res
            if COTS_ADC_input(i, res) <= Vref
                COTS_ADC_input(i, res + 1) = 2*COTS_ADC_input(i, res);
                LSBs_vect(res,i) = 0;
            else
                COTS_ADC_input(i, res + 1) = (2*COTS_ADC_input(i, res))
- 2*Vref;
                LSBs_vect(res,i) = 1;
            end
        end
        %convert to Gray Code
        for j = 1: 1: COTS_ADC_res
            if j ==1
                Dout(test, (j+resolution) - 1, i) = LSBs_vect(j,i);
            else
                Dout(test, (j+resolution) - 1, i) = xor(LSBs_vect(j-
1,i) , LSBs_vect(j,i));
            end
        end
    end

    %% Sign for Difference Measure (SDM)
    %   edited for non-matching gains
    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(test,1,i); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(test,2,i), Dout(test,3, i)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: Total_resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(test,res, i))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
```

```
            end
        end
    end

%% Difference Measure: selected for each respective gray code bit
        %Select the DM values determined using each µ being employed by
the compensation algorithm
    %Calculate all the DM values for each µ being employed by the
compensation algorithm

    for i = 1: 1: length(y) %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
        gain_factor = 1;
        for res = 2: 1: Total_resolution   % gives remaining bits of DM
            %% calculate deviations from preferred implementation
            if  Dout(test,res-1, i) == 1
                if res <= resolution
                    gain_factor = gain_factor * gain_neg(test,res-1);
                else
                    gain_factor = gain_factor*2;
                end

            else
                if res <= resolution
                    gain_factor = gain_factor * gain_pos(test,res-1);
                else
                    gain_factor = gain_factor*2;
                end
            end
            %% calculate and add difference measure values
            if (Dout(test,res, i) == 1)
                DV(res, i) = ((1/gain_factor)-(1/pow2(res - 1 )));
            else
                DV(res, i) = 0;
            end
        end
    end

    %% Signed Difference Value

    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: Total_resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV(res, i) = DV(res, i);
            else SDV (res, i) = -DV(res, i);
            end
        end

    end
    %% Determine DV
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum(i) = sum(SDV(:,i));

    end

    %% Determine uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(test,:,i);          %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: Total_resolution              %convert binary
values to the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep ;
        end
```

```
        output_representation(i) = decimal_rep  ; %modify decimal value
so it lies within the input voltage range
    end

    %% Implement Compensation
    for i = 1: 1: length(y)
        if (SDM(1,i) == 1)
            corrected_output(i) = vin_range*((output_representation(i) -
SDV_sum(i)));

        else
            corrected_output(i) = vin_range*((output_representation(i) +
SDV_sum(i)));

        end
    end

    voltage_representation = vin_range*output_representation;

    %% Calculate difference between input and output and bit accuracy

    for i = 1: 1: length(y)
        uncorrected_difference(i) = (voltage_representation(i)-
y(i))/ADC_Step_size; %uncompensated difference
        corrected_difference( i) = (corrected_output(i)-
y(i))/ADC_Step_size;         %compensated difference

    end
    %calculate bit accuracy of compensated and uncompensated ADC
    UD(test) = max(abs(uncorrected_difference(:)));
    CD(test) = max(abs(corrected_difference(:)));

    effective_bit_accuracy_UD(test) = Total_resolution -
log2(ceil(UD(test))) - 1;
    effective_bit_accuracy_CD(test) = Total_resolution -
log2(ceil(CD(test))) - 1;
end
```

C.2 MATLAB Scripts for Sensitivity Analysis of the Enhanced Tent Map Gain Compensation Algorithms

C.2.1 MATLAB Script for Assessing the Sensitivity of μ+ Deviating from μ-

The code below was developed to perform a sensitivity analysis on mathematical models of the TM-ARCHβ-7-12 ADC and TM-ARCHα-7 ADC, as discussed in Section 6.2.1.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
COTS_ADC_res = 12;
%COTS_ADC_res = 1;  % acts as a comparator on the final stage
Total_resolution = resolution + COTS_ADC_res - 1;
Vcc = 5; % valid input max.
Vee = -0.232; % valid input min.
vin_range = 3 - 0;
Vref = 1.5;              %set partition point voltage
TM_Step_size = (vin_range)/(2^resolution); %calculating step size
ADC_Step_size = (vin_range)/(2^Total_resolution); %calculating step size

%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = Total_resolution+2;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % 95000 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T-dt;
%y = vin_range-(vin_range)*(sawtooth(2*pi*f_fundamental*x) + 1)/2; %ramp
input signal - 0 to 3 V
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);
y(number_samples) = 3;

%% Preallocate size of arrays
Dout = zeros(number_samples, Total_resolution);
COTS_ADC_input = zeros(number_samples, COTS_ADC_res + 1);
LSBs_vect = zeros(COTS_ADC_res, number_samples);
uncorrected_difference = zeros(1, number_samples);
corrected_difference = zeros(1, number_samples);
output_representation = zeros(1, number_samples);
corrected_output = zeros(1, number_samples);
bin_representation = zeros(Total_resolution, number_samples);
%% Gain and Vref Parameters
rng(0,
'twister');   %%https://uk.mathworks.com/help/matlab/math/floating-
point-numbers-within-specific-range.html
gain_deviations = (-0.0002:0.00001:0.0002); %% non-percentage
number_tests = length(gain_deviations);
effective_bit_accuracy_UD = zeros(1, number_tests);
effective_bit_accuracy_CD = zeros(1, number_tests);

a = 1.9;
b = 1.99;
gain_pos = (b-a).*rand(1, resolution-1) + a;
```

```
gain_neg = ones(number_tests, resolution-1);
for i = 1: 1: number_tests
    gain_neg(i, :) = gain_pos + gain_deviations(i);
end

VH_pos = (Vref)*(ones(1, resolution));
VH_neg = (Vref)*(ones(1, resolution));

%% Input goes through TM-based ADC %
for test = 1: 1: number_tests
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        z(1,1) = y(1);
        %% first sample - assume higher than previous
        if ((z(res, 1) <= VH_pos(1, res)) && (z(res, 1) >= Vee))  % if
input to the folding stage is less than or equal to the reference
voltage
            z((res+1), 1) = gain_pos(1, res)*z(res, 1);
            Dout(1, res) = 0;
        elseif ((z(res, 1) > VH_pos(1, res)) && (z(res, 1) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
            z((res +1), 1) = (gain_pos(1, res)*Vref)-
(gain_neg(test,res)*(z((res), 1)-Vref)); % electronic implementation of
TM
            Dout(1, res) = 1;
        elseif (z(res, 1) > Vcc)
            z((res +1), 1) = Vee;
            Dout(1, res) = 1;
        else
            z((res +1), 1) = Vee;
            Dout(1, res) = 0;
        end
    end

    %% positive ramp goes through TMs %
    for i = 2: 1: length(y) %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            %% Higher Hysteresis voltages
            if (z(res, i) > z(res, i - 1))
                if ((z(res, i) <= VH_pos(1, res)) && (z(res, i) >=
Vee))  % if input to the folding stage is less than or equal to the
reference voltage
                    z((res+1), i) = gain_pos(1,res)*z(res, i);
                    Dout(i, res) = 0;
                elseif ((z(res, i) > VH_pos(1, res)) && (z(res, i) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref)); % electronic implementation of
TM
                    Dout(i, res) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(i, res) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(i, res) = 0;
                end
                %% Lower Hysteresis voltages
            else
                if ((z(res, i) <= VH_neg(1, res)) && (z(res, i) >=
Vee))  % if input to the folding stage is less than or equal to the
reference voltage
                    z((res+1), i) = gain_pos(1,res)*z(res, i);
                    Dout(i, res) = 0;
```

```
                elseif ((z(res, i) > VH_neg(1, res)) && (z(res, i) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(test,res)*(z((res), i)-Vref)); % electronic implementation of
TM
                    Dout(i, res) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(i, res) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(i, res) = 0;
                end
            end
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output

    for i = 1: 1: number_samples
        %% nth TM stage output goes through the COTS ADC
        COTS_ADC_input(i, 1) = z(resolution, i); %final TM output goes
through a commerical ADC
        %Use Bernoulli Map
        for res = 1: 1: COTS_ADC_res
            if COTS_ADC_input(i, res) <= Vref
                COTS_ADC_input(i, res + 1) = 2*COTS_ADC_input(i, res);
                LSBs_vect(res,i) = 0;
            else
                COTS_ADC_input(i, res + 1) = (2*COTS_ADC_input(i, res))
- 2*Vref;
                LSBs_vect(res,i) = 1;
            end
        end
        %convert to Gray Code
        for j = 1: 1: COTS_ADC_res
            if j ==1
                Dout(i, j+resolution - 1) = LSBs_vect(j,i);
            else
                Dout(i, j+resolution - 1) = xor(LSBs_vect(j-1,i) ,
LSBs_vect(j,i));
            end
        end
    end

    %% Sign for Difference Measure (SDM)
    %   edited for non-matching gains
    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(i, 1); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(i, 2), Dout(i, 3)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: Total_resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(i, res))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end

    %% Difference Measure: selected for each respective gray code bit
    %Select the DM values determined using each μ being employed by the
compensation algorithm
```

```
    %Calculate all the DM values for each μ being employed by the
compensation algorithm
    for i = 1: 1: length(y) %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
        gain_factor = 1;
        for res = 2: 1: Total_resolution   % gives remaining bits of DM
            %% calculate deviations from preferred implementation
            if  Dout(i, res - 1) == 1
                if res <= resolution
                    gain_factor = gain_factor * gain_neg(test, res-1);
                else
                    gain_factor = gain_factor*2;
                end

            else
                if res <= resolution
                    gain_factor = gain_factor * gain_pos(1, res-1);
                else
                    gain_factor = gain_factor*2;
                end

            end
            %% calculate and add difference measure values
            if (Dout(i, res) == 1)
                DV(res, i) = ((1/gain_factor)-(1/pow2(res - 1 )));
            else
                DV(res, i) = 0;
            end
        end
    end

    %% Signed Difference Value
    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: Total_resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV(res, i) = DV(res, i);
            else SDV (res, i) = -DV(res, i);
            end
        end
    end
    %% Determine DV
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum(i) = sum(SDV(:,i));
    end

    %% Determine uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(i, :);            %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: Total_resolution               %convert binary
values to the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep;
        end
        output_representation(i) = decimal_rep; %modify decimal value so
it lies within the input voltage range
    end

    %% Implement Compensation
    for i = 1: 1: length(y)
        if (SDM(1,i) == 1)
            corrected_output(i) = vin_range*((output_representation(i) -
SDV_sum(i)));
        else
```

```
            corrected_output(i) = vin_range*((output_representation(i) +
SDV_sum(i)));
        end
    end

    voltage_representation = vin_range*output_representation;
    %% Calculate difference between input and output and ENOB

    for i = 1: 1: length(y)
        uncorrected_difference(i) = (voltage_representation(i)-
y(i))/ADC_Step_size; %uncompensated difference
        corrected_difference( i) = (corrected_output(i)-
y(i))/ADC_Step_size;         %compensated difference

    end
    %calculate bit accuracy of compensated and uncompensated ADC
    UD = max(abs(uncorrected_difference(:)));
    CD = max(abs(corrected_difference(:)));

    effective_bit_accuracy_UD(test) = Total_resolution - log2(ceil(UD))
- 1;
    effective_bit_accuracy_CD(test) = Total_resolution - log2(ceil(CD))
- 1;

end
```

C.2.2 MATLAB Script for Assessing the Sensitivity Between $\mu_{\pm algorithm}$ and $\mu_{\pm ADC}$

The code below was developed to perform a sensitivity analysis on mathematical models of the TM-ARCHβ-7-12 ADC and TM-ARCHα-7 ADC, as discussed in Section 6.2.2.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
%COTS_ADC_res = 12;
COTS_ADC_res = 1;           % acts as a comparator
Total_resolution = resolution + COTS_ADC_res - 1;
Vcc = 5; % valid input max.
Vee = -0.232; % valid input min.
vin_range = 3 - 0;
Vref = 1.5;              %set partition point voltage
TM_Step_size = (vin_range)/(2^resolution); %calculating step size
ADC_Step_size = (vin_range)/(2^Total_resolution); %calculating step size
%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = Total_resolution+2;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % 95000 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T-dt;
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);
y(number_samples) = 3;
```

```
%% Preallocate size of arrays
Dout = zeros(number_samples, Total_resolution);
COTS_ADC_input = zeros(number_samples, COTS_ADC_res + 1);
LSBs_vect = zeros(COTS_ADC_res, number_samples);
uncorrected_difference = zeros(1, number_samples);
corrected_difference = zeros(1, number_samples);
output_representation = zeros(1, number_samples);
corrected_output = zeros(1, number_samples);
bin_representation = zeros(Total_resolution, number_samples);
%% Gain and Vref Parameters

% random µ+ and µ- adapted from
https://uk.mathworks.com/help/matlab/math/floating-point-numbers-within-
specific-range.html
rng(0, 'twister');
%gain_deviations = (-0.1:0.001:0.1); %% percentage
gain_deviations = (-2:0.001:2); %% percentage
number_tests = length(gain_deviations);
effective_bit_accuracy_UD = zeros(1, number_tests);
effective_bit_accuracy_CD = zeros(1, number_tests);

a = 1.9;
b = 1.99;
%d_max = 50*(10^-6);
%d_min = -50*(10^-6);
d_max = 0.0283;
d_min = -0.0133;
gain_pos = (b-a).*rand(1, resolution-1) + a;
gain_neg = ((d_max-d_min).*rand(1, resolution-1) + d_min) + gain_pos;

VH_pos = (Vref)*(ones(1, resolution));
VH_neg = (Vref)*(ones(1, resolution));

%% Input goes through TM-based ADC %
for test = 1: 1: number_tests
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        z(1,1) = y(1);
        %% first sample - assume higher than previous
        if ((z(res, 1) <= VH_pos(1, res)) && (z(res, 1) >= Vee))  % if
input to the folding stage is less than or equal to the reference
voltage
            z((res+1), 1) = gain_pos(1, res)*z(res, 1);
            Dout(1, res) = 0;
        elseif ((z(res, 1) > VH_pos(1, res)) && (z(res, 1) <=
Vcc))                          % if input to the folding stage is more than
the reference voltage
            z((res +1), 1) = (gain_pos(1, res)*Vref)-
(gain_neg(1,res)*(z((res), 1)-Vref)); % electronic implementation of TM
            Dout(1, res) = 1;
        elseif (z(res, 1) > Vcc)
            z((res +1), 1) = Vee;
            Dout(1, res) = 1;
        else
            z((res +1), 1) = Vee;
            Dout(1, res) = 0;
        end
    end
    %% positive ramp goes through TMs %
    for i = 2: 1: length(y) %Samples of input signal
        z(1,i) = y(i);
        for res = 1: 1: resolution - 1 % Folds and finds gray code word
of sample
            %% Higher Hysteresis voltages
            if (z(res, i) > z(res, i - 1))
                if ((z(res, i) <= VH_pos(1, res)) && (z(res, i) >=
Vee))  % if input to the folding stage is less than or equal to the
reference voltage
```

```
                    z((res+1), i) = gain_pos(1,res)*z(res, i);
                    Dout(i, res) = 0;
                elseif ((z(res, i) > VH_pos(1, res)) && (z(res, i) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(1,res)*(z((res), i)-Vref)); % electronic implementation of TM
                    Dout(i, res) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(i, res) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(i, res) = 0;
                end
                %% Lower Hysteresis voltages
            else
                if ((z(res, i) <= VH_neg(1, res)) && (z(res, i) >=
Vee))  % if input to the folding stage is less than or equal to the
reference voltage
                    z((res+1), i) = gain_pos(1,res)*z(res, i);
                    Dout(i, res) = 0;
                elseif ((z(res, i) > VH_neg(1, res)) && (z(res, i) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
                    z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(1,res)*(z((res), i)-Vref)); % electronic implementation of TM
                    Dout(i, res) = 1;
                elseif (z(res, i) > Vcc)
                    z((res +1), i) = Vee;
                    Dout(i, res) = 1;
                else
                    z((res +1), i) = Vee;
                    Dout(i, res) = 0;
                end
            end
        end
    end
    %z array gives the inputs to each Tent map stage
    %Dout array provides the Gray code output
    for i = 1: 1: number_samples
        %% nth TM stage output goes through the COTS ADC
        COTS_ADC_input(i, 1) = z(resolution, i); %final TM output goes
through a commerical ADC
        %Use Bernoulli Map
        for res = 1: 1: COTS_ADC_res
            if COTS_ADC_input(i, res) <= Vref
                COTS_ADC_input(i, res + 1) = 2*COTS_ADC_input(i, res);
                LSBs_vect(res,i) = 0;
            else
                COTS_ADC_input(i, res + 1) = (2*COTS_ADC_input(i, res))
- 2*Vref;
                LSBs_vect(res,i) = 1;
            end
        end
        %convert to Gray Code
        for j = 1: 1: COTS_ADC_res
            if j ==1
                Dout(i, j+resolution - 1) = LSBs_vect(j,i);
            else
                Dout(i, j+resolution - 1) = xor(LSBs_vect(j-1,i) ,
LSBs_vect(j,i));
            end
        end
    end

    %% Sign for Difference Measure (SDM)
    %   edited for non-matching gains
```

```
    for i = 1: 1: length(y) %Samples of input signal
        SDM(1, i) = Dout(i, 1); %MSB of Gray code output
        SDM(2, i) = 1;          %1 shows adding function
        if xor(Dout(i, 2), Dout(i, 3)) % find 3rd bit of SDM
            SDM(3,i) = 1;
        else
            SDM(3,i) = 0;
        end
        for res = 4: 1: Total_resolution   % gives remaining bits of SDM
            if xor(SDM(res-1,i), Dout(i, res))
                SDM(res,i) = 1;
            else
                SDM(res,i) = 0;
            end
        end
    end

    %% Difference Measure: selected for each respective gray code bit
    %Select the DM values determined using each μ being employed by the
compensation algorithm
    %Calculate all the DM values for each μ being employed by the
compensation algorithm

    for i = 1: 1: length(y) %Samples of input signal
        DV(1,i) = 0; %Ideal as it hasn't passed through a TM
        gain_factor = 1;
        for res = 2: 1: Total_resolution   % gives remaining bits of DM
            %% calculate deviations from preferred implementation
            if  Dout(i, res - 1) == 1
                if res <= resolution
                    gain_factor = gain_factor * (gain_neg(1, res-1)*(1+
gain_deviations(test)/100)) ;
                else
                    gain_factor = gain_factor*2;
                end
            else
                if res <= resolution
                    gain_factor = gain_factor * (gain_pos(1, res-1)*(1+
gain_deviations(test)/100)) ;
                else
                                gain_factor = gain_factor*2;
                end

            end
            %% calculate and add difference measure values
            if (Dout(i, res) == 1)
                DV(res, i) = ((1/gain_factor)-(1/pow2(res - 1 )));
            else
                DV(res, i) = 0;
            end
        end
    end

    %% Signed Difference Value

    for i = 1: 1: length(y) %Samples of input signal
        for res = 1: 1: Total_resolution   % gives remaining bits of DV
            if (SDM(res, i) > 0)
                SDV(res, i) = DV(res, i);
            else SDV (res, i) = -DV(res, i);
            end
        end
    end
    %% Determine DV
    for i = 1: 1: length(y) %decimal of SDV
        SDV_sum(i) = sum(SDV(:,i));

    end
```

```
    %% Determine uncompensated output
    for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
        gray_code_vector = Dout(i, :);           %get Gray code word
        binary = gray2bin(gray_code_vector);    %convert Gray code word
to binary
        bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
        decimal_rep = 0;
        for j = 1: 1: Total_resolution               %convert binary
values to the equivalent voltage
            decimal_rep = (binary(j)/(2^j))+ decimal_rep ;
        end
        output_representation(i) = decimal_rep  ; %modify decimal value
so it lies within the input voltage range
    end

    %% Implement Compensation
    for i = 1: 1: length(y)
        if (SDM(1,i) == 1)
            corrected_output(i) = vin_range*((output_representation(i) -
SDV_sum(i)));

        else
            corrected_output(i) = vin_range*((output_representation(i) +
SDV_sum(i)));

        end
    end

    voltage_representation = vin_range*output_representation;

    %% Calculate difference between input and output and bit accuracy
    for i = 1: 1: length(y)
        uncorrected_difference(i) = (voltage_representation(i)-
y(i))/ADC_Step_size; %uncompensated difference
        corrected_difference( i) = (corrected_output(i)-
y(i))/ADC_Step_size;         %compensated difference

    end
    %calculate bit accuracy of compensated and uncompensated ADC
    UD = max(abs(uncorrected_difference(:)));
    CD = max(abs(corrected_difference(:)));

    effective_bit_accuracy_UD(test) = Total_resolution - log2(ceil(UD))
- 1;
    effective_bit_accuracy_CD(test) = Total_resolution - log2(ceil(CD))
- 1;

end
cutoff = 15*ones(1, number_tests);
```

C.3 MATLAB Script for Analysing Final TM-ARCHβ-7-12 ADC Model and µCA-3

The code below presents the final mathematical model developed of the TM-ARCHβ-7-12 ADC and highlights how the bit accuracy was determined before and after the µCA-3 was applied. The code employed to assess the static and dynamic performance was the same as that shown in Appendix B.1.2, Appendix B.1.3, Appendix B.2.2 and Appendix B.2.3.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
COTS_ADC_res = 12;
Total_resolution = resolution + COTS_ADC_res - 1;
Vcc = 5; % valid input max.
Vee = -0.232; % valid input min.
vin_range = 3 - 0;
Vref = 1.5;              %set partition point voltage
TM_Step_size = (vin_range)/(2^resolution); %calculating step size
ADC_Step_size = (vin_range)/(2^Total_resolution); %calculating step size

%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = Total_resolution+2;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T;
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);
y(number_samples) = 3;

%% Preallocate size of arrays
Dout = zeros(number_samples, Total_resolution);
COTS_ADC_input = zeros(number_samples, COTS_ADC_res + 1);
LSBs_vect = zeros(COTS_ADC_res, number_samples);
%% Gain and Vref Parameters
% Comparator Hysteresis - threshold voltages 1/2 step size above and
below
% the partition point voltage
VTH = Vref + ADC_Step_size/2;
VTL = Vref - ADC_Step_size/2;
VH_pos =  VTH*ones(1, resolution);
VH_neg = VTL*ones(1, resolution);
% adapted code from
% https://uk.mathworks.com/help/matlab/math/floating-point-numbers-
within-specific-range.html
% to select random µ+ and µ- values
rng(0, 'twister');
a = 1.9;
b = 1.99;
d_max = 50*(10^-6);
d_min = -50*(10^-6);
gain_pos = (b-a).*rand(1, resolution-1) + a; % µ+ for each TM stage
```

```
gain_neg = ((d_max-d_min).*rand(1, resolution-1) + d_min)+gain_pos; % µ-
for each TM stage
%% Input goes through TM-based ADC %
for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
    z(1,1) = y(1);
    %% first sample - assume higher than previous
    if ((z(res, 1) <= VH_pos(1, res)) && (z(res, 1) >= Vee))  % if input
to the folding stage is less than or equal to the reference voltage
        z((res+1), 1) = gain_pos(1, res)*z(res, 1);
        Dout(1, res) = 0;
    elseif ((z(res, 1) > VH_pos(1, res)) && (z(res, 1) <=
Vcc))                       % if input to the folding stage is more than
the reference voltage
        z((res +1), 1) = (gain_pos(1, res)*Vref)-
(gain_neg(1,res)*(z((res), 1)-Vref)); % Electronic TM implementation
        Dout(1, res) = 1;
    elseif (z(res, 1) > Vcc)
        z((res +1), 1) = Vee;
        Dout(1, res) = 1;
    else
        z((res +1), 1) = Vee;
        Dout(1, res) = 0;
    end
end
%% positive ramp goes through TMs %
for i = 2: 1: length(y) %Samples of input signal
    z(1,i) = y(i);
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        %% Higher Hysteresis voltages
        if (z(res, i) > z(res, i - 1))
            if ((z(res, i) <= VH_pos(1, res)) && (z(res, i) >= Vee))  %
if input to the folding stage is less than or equal to the reference
voltage
                z((res+1), i) = gain_pos(1,res)*z(res, i);
                Dout(i, res) = 0;
            elseif ((z(res, i) > VH_pos(1, res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(1,res)*(z((res), i)-Vref)); % Electronic TM implementation
                Dout(i, res) = 1;
            elseif (z(res, i) > Vcc)
                z((res +1), i) = Vee;
                Dout(i, res) = 1;
            else
                z((res +1), i) = Vee;
                Dout(i, res) = 0;
            end
            %% Lower Hysteresis voltages
        else
            if ((z(res, i) <= VH_neg(1, res)) && (z(res, i) >= Vee))  %
if input to the folding stage is less than or equal to the reference
voltage
                z((res+1), i) = gain_pos(1,res)*z(res, i);
                Dout(i, res) = 0;
            elseif ((z(res, i) > VH_neg(1, res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(1,res)*(z((res), i)-Vref)); % Electronic TM implementation
                Dout(i, res) = 1;
            elseif (z(res, i) > Vcc)
                z((res +1), i) = Vee;
                Dout(i, res) = 1;
            else
                z((res +1), i) = Vee;
```

```
                Dout(i, res) = 0;
            end
        end
    end
end
%z array gives the inputs to each Tent map stage
%Dout array provides the Gray code output
for i = 1: 1: number_samples
    %% nth TM stage output goes through the ideal COTS ADC
    COTS_ADC_input(i, 1) = z(resolution, i); %final TM output goes
through a commerical ADC
    for res = 1: 1: COTS_ADC_res
        if COTS_ADC_input(i, res) <= Vref
            COTS_ADC_input(i, res + 1) = 2*COTS_ADC_input(i, res);
            LSBs_vect(res,i) = 0;
        else
            COTS_ADC_input(i, res + 1) = (2*COTS_ADC_input(i, res)) -
2*Vref;
            LSBs_vect(res,i) = 1;
        end
    end
    %convert to Gray Code
    for j = 1: 1: COTS_ADC_res
        if j ==1
            Dout(i, j+resolution - 1) = LSBs_vect(j,i);
        else
            Dout(i, j+resolution - 1) = xor(LSBs_vect(j-1,i) ,
LSBs_vect(j,i));
        end
    end
end
%% Sign for Difference Measure (SDM)
%   edited for non-matching gains
for i = 1: 1: length(y) %Samples of input signal
    SDM(1, i) = Dout(i, 1); %MSB of Gray code output
    SDM(2, i) = 1;          %1 shows adding function
    if xor(Dout(i, 2), Dout(i, 3)) % find 3rd bit of SDM
        SDM(3,i) = 1;
    else
        SDM(3,i) = 0;
    end
    for res = 4: 1: Total_resolution   % gives remaining bits of SDM
        if xor(SDM(res-1,i), Dout(i, res))
            SDM(res,i) = 1;
        else
            SDM(res,i) = 0;
        end
    end
end
%% Difference Measure: selected for each respective gray code bit
%Select the DM values determined using each μ being employed by the
compensation algorithm
%Calculate all the DM values for each μ being employed by the
compensation algorithm
for i = 1: 1: length(y) %Samples of input signal
    DV(1,i) = 0; %Ideal as it hasn't passed through a TM
    gain_factor = 1;
    for res = 2: 1: Total_resolution   % gives remaining bits of DM
        %% calculate deviations from preferred implementation
        if  Dout(i, res - 1) == 1
            if res <= resolution
                gain_factor = gain_factor * gain_neg(res-1);
            else
                gain_factor = gain_factor*2; % COTS ADC - modelled as
ideal TM stages
            end
        else
            if res <= resolution
```

```
                gain_factor = gain_factor * gain_pos(res-1);
            else
                gain_factor = gain_factor*2; % COTS ADC - modelled as
ideal TM stages
            end
        end
        %% calculate and add difference measure values
        if (Dout(i, res) == 1)
            DV(res, i) = ((1/gain_factor)-(1/pow2(res - 1 )));
        else
            DV(res, i) = 0;
        end
    end
end
%% Signed Difference Value
for i = 1: 1: length(y) %Samples of input signal
    for res = 1: 1: Total_resolution   % gives remaining bits of DV
        if (SDM(res, i) > 0)
            SDV(res, i) = DV(res, i);
        else SDV (res, i) = -DV(res, i);
        end
    end
end
%% Determine DV
for i = 1: 1: length(y) %decimal of SDV
    SDV_sum(i) = sum(SDV(:,i));
end
%% Determine uncompensated output
for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
    gray_code_vector = Dout(i, :);            %get Gray code word
    binary = gray2bin(gray_code_vector);    %convert Gray code word to
binary
    bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
    decimal_rep = 0;
    for j = 1: 1: Total_resolution                %convert binary values
to the equivalent voltage
        decimal_rep = (binary(j)/(2^j))+ decimal_rep;
    end
    output_representation(i) = decimal_rep  ; %modify decimal value so
it lies within the input voltage range
end
%% Implement Compensation
for i = 1: 1: length(y)
    if (SDM(1,i) == 1)
        corrected_output(i) = vin_range*((output_representation(i) -
SDV_sum(i)));
    else
        corrected_output(i) = vin_range*((output_representation(i) +
SDV_sum(i)));
    end
end
voltage_representation = vin_range*output_representation;
%% Calculate quantisation error and bit accuracy
for i = 1: 1: length(y)
    uncorrected_difference(i) = (voltage_representation(i)-
y(i))/ADC_Step_size; %uncompensated difference
    corrected_difference( i) = (corrected_output(i)-
y(i))/ADC_Step_size;         %compensated difference

end
%calculate bit accuracy of compensated and uncompensated ADC
UD = max(abs(uncorrected_difference(:)));
CD = max(abs(corrected_difference(:)));
effective_bit_accuracy_UD = Total_resolution - log2(ceil(UD)) - 1;
effective_bit_accuracy_CD = Total_resolution - log2(ceil(CD)) - 1;
                                    […]
```

C.4 MATLAB Script for Noise Analysis Simulation

The following MATLAB script shows how the noise analysis discussed in Section 6.4 was performed.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
COTS_ADC_res = 12;
Total_resolution = resolution + COTS_ADC_res - 1;
Vcc = 5; % valid input max.
Vee = -0.232; % valid input min.
Vmax = 3;
Vmin = 0;
vin_range = 3 - 0;
Vref = 1.5;              %set partition point voltage
TM_Step_size = (vin_range)/(2^resolution); %calculating step size
ADC_Step_size = (vin_range)/(2^Total_resolution); %calculating step size
%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = Total_resolution+2;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % 95000 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamantal frequency
dt = 1/Fs;
x = 0:dt:T;
%y = vin_range-(vin_range)*(sawtooth(2*pi*f_fundamental*x) + 1)/2; %ramp
input signal - 0 to 3 V
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
number_samples = length(x);
y(number_samples) = 3;

%% added noise
noise_pp  = 2*ADC_Step_size; % Set noise as a multiple of step size
SNR_not_dB = (Vmax-Vmin)/noise_pp;   % signal to noise ratio
snr = 20*log10(SNR_not_dB);          % calculate SNR to dB

internal_noise = zeros(resolution, number_samples);
internal_noise = awgn(internal_noise,snr); %superimpose white gaussian
noise on the input signal

%% Preallocate size of arrays
Dout = zeros(number_samples, Total_resolution);
COTS_ADC_input = zeros(number_samples, COTS_ADC_res + 1);
LSBs_vect = zeros(COTS_ADC_res, number_samples);
%% Gain and Vref Parameters
% Comparator Hysteresis - threshold voltages 1/2 step size above and
below
% The partition point voltage
VTH = Vref + ADC_Step_size/2;
VTL = Vref - ADC_Step_size/2;

VH_pos =  VTH*ones(1, resolution);
VH_neg = VTL*ones(1, resolution);

% adapted code from
% https://uk.mathworks.com/help/matlab/math/floating-point-numbers-
within-specific-range.html
% to select random µ+ and µ- values
```

```
rng(0, 'twister');
a = 1.9;
b = 1.99;
d_max = 50*(10^-6);
d_min = -50*(10^-6);
gain_pos = (b-a).*rand(1, resolution-1) + a; % μ+ for each TM stage
gain_neg =  ((d_max-d_min).*rand(1, resolution-1) + d_min)+gain_pos; %
μ- for each TM stage

%% Input goes through TM-based ADC %

for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
    z(1,1) = y(1) + internal_noise(res,1);      % superimpose noise onto
input signal
    %% first sample - assume higher than previous
    if ((z(res, 1) <= VH_pos(1, res)) && (z(res, 1) >= Vee))  % if input
to the folding stage is less than or equal to the reference voltage
        z((res+1), 1) = gain_pos(1, res)*z(res, 1)+
internal_noise(res+1,1);  % superimpose noise onto TM output signal
        Dout(1, res) = 0;
    elseif ((z(res, 1) > VH_pos(1, res)) && (z(res, 1) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
        z((res +1), 1) = (gain_pos(1, res)*Vref)-
(gain_neg(1,res)*(z((res), 1)-Vref))+ internal_noise(res+1,1); %
superimpose noise onto TM output signal
        Dout(1, res) = 1;
    elseif (z(res, 1) > Vcc)
        z((res +1), 1) = Vee;
        Dout(1, res) = 1;
    else
        z((res +1), 1) = Vee;
        Dout(1, res) = 0;
    end
end
%% positive ramp goes through TMs %
for i = 2: 1: length(y) %Samples of input signal
    z(1,i) = y(i)+ internal_noise(1,i); % superimpose noise onto input
signal
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        %% Higher Hysteresis voltages
        if (z(res, i) > z(res, i - 1))
            if ((z(res, i) <= VH_pos(1, res)) && (z(res, i) >= Vee))  %
if input to the folding stage is less than or equal to the reference
voltage
                z((res+1), i) = gain_pos(1,res)*z(res, i)+
internal_noise(res+1,i); % superimpose noise onto TM output signal
                Dout(i, res) = 0;
            elseif ((z(res, i) > VH_pos(1, res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(1,res)*(z((res), i)-Vref))+ internal_noise(res+1,i); %
superimpose noise onto TM output signal
                Dout(i, res) = 1;
            elseif (z(res, i) > Vcc)
                z((res +1), i) = Vee;
                Dout(i, res) = 1;
            else
                z((res +1), i) = Vee;
                Dout(i, res) = 0;
            end
            %% Lower Hysteresis voltages
        else
```

```
            if ((z(res, i) <= VH_neg(1, res)) && (z(res, i) >= Vee))  %
if input to the folding stage is less than or equal to the reference
voltage
                z((res+1), i) = gain_pos(1,res)*z(res, i)+
internal_noise(res+1,i); % superimpose noise onto TM output signal
                Dout(i, res) = 0;
            elseif ((z(res, i) > VH_neg(1, res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain_pos(1,res)*Vref)-
(gain_neg(1,res)*(z((res), i)-Vref))+ internal_noise(res+1,i); %
superimpose noise onto TM output signal
                Dout(i, res) = 1;
            elseif (z(res, i) > Vcc)
                z((res +1), i) = Vee;
                Dout(i, res) = 1;
            else
                z((res +1), i) = Vee;
                Dout(i, res) = 0;
            end
        end
    end
end
%z array gives the inputs to each Tent map stage
%Dout array provides the Gray code output
for i = 1: 1: number_samples
    %% nth TM stage output goes through the COTS ADC
    % No noise superimposed within COTS ADC, as it is being modelled as
an
    % Ideal ADC
    COTS_ADC_input(i, 1) = z(resolution, i); %final TM output goes
through a commerical ADC
    %Use Bernoulli Map
    for res = 1: 1: COTS_ADC_res
        if COTS_ADC_input(i, res) <= Vref
            COTS_ADC_input(i, res + 1) = 2*COTS_ADC_input(i, res);
            LSBs_vect(res,i) = 0;
        else
            COTS_ADC_input(i, res + 1) = (2*COTS_ADC_input(i, res)) -
2*Vref;
            LSBs_vect(res,i) = 1;
        end
    end
    %convert to Gray Code
    for j = 1: 1: COTS_ADC_res
        if j ==1
            Dout(i, j+resolution - 1) = LSBs_vect(j,i);
        else
            Dout(i, j+resolution - 1) = xor(LSBs_vect(j-1,i) ,
LSBs_vect(j,i));
        end
    end
end

%% Sign for Difference Measure (SDM)
%   Edited for non-matching gains
for i = 1: 1: length(y) %Samples of input signal
    SDM(1, i) = Dout(i, 1); %MSB of Gray code output
    SDM(2, i) = 1;          %1 shows adding function
    if xor(Dout(i, 2), Dout(i, 3)) % find 3rd bit of SDM
        SDM(3,i) = 1;
    else
        SDM(3,i) = 0;
    end
    for res = 4: 1: Total_resolution   % gives remaining bits of SDM
        if xor(SDM(res-1,i), Dout(i, res))
            SDM(res,i) = 1;
        else
```

```
            SDM(res,i) = 0;
        end
    end
end


%% Difference Measure: selected for each respective gray code bit

%Select the DM values determined using each μ being employed by the
compensation algorithm
%Calculate all the DM values for each μ being employed by the
compensation algorithm

for i = 1: 1: length(y) %Samples of input signal
    DV(1,i) = 0; %Ideal as it hasn't passed through a TM
    gain_factor = 1;
    for res = 2: 1: Total_resolution   % gives remaining bits of DM
        %% calculate deviations from preferred implementation
        if  Dout(i, res - 1) == 1
            if res <= resolution
                gain_factor = gain_factor * gain_neg(res-1);
            else
                gain_factor = gain_factor*2;
            end


        else
            if res <= resolution
                gain_factor = gain_factor * gain_pos(res-1);
            else
                gain_factor = gain_factor*2;
            end

        end
        %% calculate and add difference measure values
        if (Dout(i, res) == 1)
            DV(res, i) = ((1/gain_factor)-(1/pow2(res - 1 )));
        else
            DV(res, i) = 0;
        end
    end
end

%% Signed Difference Value

for i = 1: 1: length(y) %Samples of input signal
    for res = 1: 1: Total_resolution   % gives remaining bits of DV
        if (SDM(res, i) > 0)
            SDV(res, i) = DV(res, i);
        else SDV (res, i) = -DV(res, i);
        end
    end

end
%% Determine DV
for i = 1: 1: length(y) %decimal of SDV
    SDV_sum(i) = sum(SDV(:,i));

end

%% Determine uncompensated output
for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
    gray_code_vector = Dout(i, :);            %get Gray code word
    binary = gray2bin(gray_code_vector);    %convert Gray code word to
binary
    bin_representation(:,i) = binary ;      %save binary to an array
(verification of results in MATLAB workspace)
```

```
    decimal_rep = 0;
    for j = 1: 1: Total_resolution                %convert binary values
to the equivalent voltage
        decimal_rep = (binary(j)/(2^j))+ decimal_rep ;
    end
    output_representation(i) = decimal_rep  ; %modify decimal value so
it lies within the input voltage range
end

%% Implement Compensation
for i = 1: 1: length(y)
    if (SDM(1,i) == 1)
        corrected_output(i) = vin_range*((output_representation(i) -
SDV_sum(i)));

    else
        corrected_output(i) = vin_range*((output_representation(i) +
SDV_sum(i)));

    end
end
voltage_representation = vin_range*output_representation;

%% Calculate difference between input and output and bit accuracy
for i = 1: 1: length(y)
    uncorrected_difference(i) = (voltage_representation(i)-
y(i))/ADC_Step_size; %uncompensated difference
    corrected_difference( i) = (corrected_output(i)-
y(i))/ADC_Step_size;         %compensated difference

end
%calculate bit accuracy of compensated and uncompensated ADC
UD = max(abs(uncorrected_difference(:)));
CD = max(abs(corrected_difference(:)));

effective_bit_accuracy_UD = Total_resolution - log2(ceil(UD)) - 1;
effective_bit_accuracy_CD = Total_resolution - log2(ceil(CD)) - 1;
```

C.5 Code for VHDL Implementation of an Enhanced Tent Map Gain Compensation Algorithm

The majority of the VHDL code employed for the tests discussed in Section 6.3 was identical to that employed in Section 5.5 and thus presented in Appendix B.5. Therefore, only code listings which were modified are presented in this section.

C.5.1 MATLAB Script to Aid the Creation of an TM-ARCHα-7 ADC Signal Emulator

The MATLAB script below was developed to choose a set of $\mu_{\pm}$ values at random, then to establish a sequence of values to be added to the sum of the previous value, in order to create the Gray code that an TM-ARCHα-7 ADC might produce if supplied with a full-scale ramp input signal. The sequence of values was then employed by the TM-ARCHα-7 ADC Signal Emulator (see Appendix C.3.3) and enabled this component to produce a similar output to a TM-based ADC before the comparator outputs are aligned by the control logic within the FPGA. The DM values required by the VHDL implementation of the μCA-2 were also established using this MATLAB script.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
Vcc = 5; % valid input max.
Vee = 0; % valid input min.
vin_range = 3 - 0;
Vref = 1.5;              %set partition point voltage
Step_size = (vin_range)/(2^resolution); %calculating step size
%% Generate input signal
% input - sawtooth wave adapted from
https://uk.mathworks.com/help/signal/ref/sawtooth.html
pwr_val = resolution + 1;
Fs = 25000000;      %sample rate = 25 MHz
f_fundamental = Fs/2^(pwr_val); % 95000 Hz fundamental frequency
T = (1/f_fundamental); %number of periods times fundamental frequency
dt = 1/Fs;
x = 0:dt:T-dt;
y = (vin_range)*(sawtooth(2*pi*f_fundamental*x)+ 1)/2; %ramp input
signal - 0 to 3 V
%y = (vin_range)*(sin(2*pi*f_fundamental*x)+ 1)/2; %ramp input signal -
0 to 3 V
number_samples = length(x);

%% Gain and Vref Parameters
VHDL_res = resolution + 2;
rng(0, 'twister');  %%
https://uk.mathworks.com/help/matlab/math/floating-point-numbers-within-
specific-range.html
a = 1.9;
b = 1.99;
d_max =0.0283;
d_min = -0.0133;
gain_pos = (b-a).*rand(1, resolution-1) + a;
gain_neg =  ((d_max-d_min).*rand(1, resolution-1) + d_min)+gain_pos;
inv_pos = 1./gain_pos;
inv_neg = 1./gain_neg;
for i = 1: 1: resolution -1
    inv_two(i) = pow2(-i);
end
VH_pos = (Vref)*ones(1, resolution);
VH_neg = (Vref)*ones(1, resolution);

%% determine DM values
% start calculating all the different gain factors
% resolution - 1 DM values could be required to compensate each ADC
output
% there are a maximum of 2^(resolution -1) gain factors
gain_combinations = ones(resolution - 1, pow2(resolution - 1));

% half the possible gain factors will have TM1 µ+, and the rest TM1 µ-
gain_combinations(1,1:2^(resolution-2)) = inv_neg(1,1);
gain_combinations(1,1+2^(resolution-2): 2^(resolution-1)) =
inv_pos(1,1);

% Determine all the different gain factors
for res = 2: 1: resolution - 1
    count = 1;
    for combo =  1: 1: 2^(resolution-1)
        if count <= 2^(resolution - (1 + res))
            gain_combinations(res, combo) =
inv_neg(1,res)*gain_combinations(res - 1, combo);
        else
            gain_combinations(res, combo) =
inv_pos(1,res)*gain_combinations(res - 1, combo);
        end
        if count == 2^(resolution - res)
            count = 1;
        else
            count = count + 1;
```

```
        end
    end
end

% calculate the different DM values
for res = 1:1:resolution - 1
    for combo = 1:1:  2^(resolution-1)
        DM_array(res, combo) = gain_combinations(res, combo)-
inv_two(res);
    end
end

% convert the DM values to binary format, and separate each DM with a
comma
for res = 1:1:resolution - 1
    for combo = 1:1: 2^(resolution-1)
        DM_array_bin(res, combo) =
join(string(de2bi(floor(pow2(VHDL_res)*DM_array(res, combo)), VHDL_res,
'left-msb')),"");
    end
    DM_array_VHDL(res, 1) = join(DM_array_bin(res, :), '", "');
end

%% Input goes through TM-based ADC %%
for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
    z(1,1) = y(1);
    %% first sample - assume higher than previous
    if ((z(res, 1) <= VH_pos(res)) && (z(res, 1) >= Vee))  % if input to
the folding stage is less than or equal to the reference voltage
        z((res+1), 1) = gain_pos(res)*z(res, 1);
        Dout(res, 1) = 0;
    elseif ((z(res, 1) > VH_pos(res)) && (z(res, 1) <= Vcc)) % if input
to the folding stage is more than the reference voltage
        z((res +1), 1) = (gain_pos(res)*Vref)-
(gain_neg(test,res)*(z((res), 1)-Vref));
        Dout(res,1) = 1;
    elseif (z(res, 1) > Vcc)
        z((res +1), 1) = Vee;
        Dout(res, 1) = 1;
    else
        z((res +1), 1) = Vee;
        Dout(res, 1) = 0;
    end
end
%% positive ramp goes through TMs %
for i = 2: 1: length(y) %Samples of input signal
    z(1,i) = y(i);
    for res = 1: 1: resolution - 1 % Folds and finds gray code word of
sample
        %% Higher Hysteresis voltages
        if (z(res, i) > z(res, i - 1))
            if ((z(res, i) <= VH_pos(res)) && (z(res, i) >= Vee))  % if
input to the folding stage is less than or equal to the reference
voltage
                z((res+1), i) = gain_pos(res)*z(res, i);
                Dout(res, i) = 0;
            elseif ((z(res, i) > VH_pos(res)) && (z(res, i) <=
Vcc))                      % if input to the folding stage is more than
the reference voltage
                z((res +1), i) = (gain_pos(res)*Vref)-
(gain_neg(res)*(z((res), i)-Vref));
                Dout(res,i) = 1;
            elseif (z(res, i) > Vcc)
                z((res +1), i) = Vee;
                Dout(res, i) = 1;
            else
                z((res +1), i) = Vee;
```

```
                Dout(res, i) = 0;
            end
            %% Lower Hysteresis voltages
        else
            if ((z(res, i) <= VH_neg(res)) && (z(res, i) >= Vee))  % if
input to the folding stage is less than or equal to the reference
voltage
                z((res+1), i) = gain_pos(res)*z(res, i);
                Dout(res, i) = 0;
            elseif ((z(res, i) > VH_neg(res)) && (z(res, i) <= Vcc)) %
if input to the folding stage is more than the reference voltage
                z((res +1), i) = (gain_pos(res)*Vref)-
(gain_neg(res)*(z((res), i)-Vref));
                Dout(res,i) = 1;
            elseif (z(res, i) > Vcc)
                z((res +1), i) = Vee;
                Dout(res, i) = 1;
            else
                z((res +1), i) = Vee;
                Dout(res, i) = 0;
            end
        end
    end
    %% last comparator
    if ((z(resolution, i) > z(resolution, i - 1))||(z(resolution, 1)))
        if (z(resolution, i) <= VH_pos(resolution))
            Dout(resolution, i) = 0;
        else
            Dout(resolution, i) = 1;
        end
    else
        if (z(resolution, i) <= VH_neg(resolution))
            Dout(resolution, i) = 0;
        else
            Dout(resolution, i) = 1;
        end
    end
end
%z array gives the inputs to each Tent map stage
%Dout array provides the Gray code output

%% create ramp
%uncorrected output
for i = 1: 1: length(y) % converting Gray code representation of
samples, to binary
    gray_code_vector = Dout(:,i);           %get Gray code word
    gray_rep = transpose(gray_code_vector);
    gray_int(i) = bi2de(gray_rep, 'left-msb');

    %looking at binary equivalent
    binary = gray2bin(gray_code_vector); %convert Gray code word to
binary
    bin_representation(:,i) = binary; % save binary to an array
(verification of results in MATLAB workspace)
    bin_rep = transpose(bin_representation(:,i));
    bin_int(i) = bi2de(bin_rep, 'left-msb');
end

%% Determine Ramp add values (to be employed by the signal emulator)
ramp_diff_shift = [gray_int(2 : length(gray_int) ), gray_int(1)]; %
shift gray_int values right by on in the array. Bring gray_int(1) to the
leftmost index in the array.
ramp_add = ramp_diff_shift - gray_int; % determine the next value to be
added to the output to create a ramp
ramp_add_string = join(string(ramp_add), ","); % convert to string and
separate the binary words with a comma. This speeds up the process of
inserting the values into the VHDL code.
```

C.5.2 VHDL Implementation of the µCA-2

The code listing below presents the VHDL implementation of the µCA-2.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

ENTITY gain_correction is
generic (n: positive:= 8);  -- define resolution of TM-based ADC
      PORT  (PLL_OUT: in std_logic;  -- PLL clock (feb 17: 250 MHz)
             res: in std_logic;  -- resets all Difference registers
             Input_Gray: in std_logic_vector(n-1 downto 0); --ADC output
in Gray code
             correct_en: in std_logic;
             correct_fin: out std_logic;
             UncorrectedBINARY: in std_logic_vector(n-1 downto 0); --ADC
output in binary
             CorrectedBINARY: out std_logic_vector(n-1 downto 0)); --
Corrected ADC output
END ENTITY;

ARCHITECTURE behav OF gain_correction IS
-- define signals to be employed within the module
signal Graycode: std_logic_vector (n-1 downto 0):= (others => '0');
signal SDM_reg:  std_logic_vector (n-1 downto 0):= (others => '0');
signal UC_binary:   std_logic_vector((n+2)-1 downto 0):= (others =>
'0');
signal C_binary:    std_logic_vector((n+2)-1 downto 0):= (others =>
'0');
signal SDV_reg:  std_logic_vector((n+2)-1 downto 0):= (others => '0');
signal correct:  std_logic_vector(n-1 downto 0):= (others => '0');
signal counter: integer := 0;
signal C_fin:  std_logic := '0';
signal do_correct : std_logic := '0';

-- LUT for random µ+ and µ- values determined by MATLAB script
type LUT is array (n-2 downto 0, 2**(n-1) -1 downto 0) of
std_logic_vector((n+2)-1 downto 0); --precalculated different measure
values for the adapted algorithm
-- each row represents the possible DMs for a different TM output

signal diff_bit : LUT :=            --binary values multiplied by 2^10 =
1024 (need to shift back by 2)
(
-- DM values array--
-- contents generated by MATLAB Script presented in C.5.1--
);

type Diff is array (n-1 downto 0) of std_logic_vector((n+2)-1 downto 0);
-- array to store relevant DM values
signal assign_diff: Diff;

type sum_Diff is array (n-2 downto 0) of std_logic_vector((n+2)-1 downto
0); -- array to aid the calculation of the DV value
signal polarity_diff: sum_Diff;

BEGIN
      Gain_compensate: PROCESS
      BEGIN
```

```
      WAIT UNTIL RISING_EDGE(PLL_OUT);   --PLL_IN the PLL clock 250MHz
      Graycode(n-1 downto 0) <= Input_Gray;
      UC_binary((n+2)-1 downto (n+2)-n) <= UncorrectedBINARY(n-1 downto
0);
      UC_binary((n+2)-(n+1) downto 0) <= (others => '0');

      --Should the correction process be applied?
      IF correct_en = '1' AND counter < 3 THEN --yes
            C_fin <= '0';
            counter <= counter + 1;
            -- Find sign of difference measure
            SDM_reg(n-1) <= Graycode(n-1);
            SDM_reg(n-2) <= '1';
            SDM_reg(n-3) <= (Graycode(n-2) XOR Graycode(n-3));

            -- Find Difference measure
            assign_diff(n-1) <= (others => '0');

            --Do correction
            polarity_diff(n-2) <= assign_diff(n-2);

            if SDM_reg(n-1) = '1' then -- is MSB of Gray code = 1?
                  C_binary((n+2)-1 downto 0) <= UC_binary((n+2)-1 downto
0) - SDV_reg((n+2)-1 downto 0);
            else -- is MSB of Gray code = 0?
                  C_binary((n+2)-1 downto 0) <= UC_binary((n+2)-1 downto
0) + SDV_reg((n+2)-1 downto 0);
            end if;

      ELSIF counter = 3 AND correct_en = '1' THEN --no
            C_fin <= '1';
      ELSIF counter = 3 AND correct_en = '0' THEN --no
            counter <= 0;
      ELSE
            null;
      END IF;
END process;

-- Sign for Difference measure
SDM: for i in n-4 downto 0 generate
begin
      SDM_reg(i) <= (SDM_reg(i+1) XOR Graycode(i));
end generate SDM;

-- Assign difference measure
DV: for i in n-2 downto 0 generate
begin
      with Graycode(i) select
      assign_diff(i)  <= diff_bit(i, to_integer(unsigned(Graycode(n-1
downto 1)))) when '1', -- use Gray code to establish which DM value is
required
      (others => '0') when others;
end generate DV;

-- Do correction/ calculate difference value
SDV: for i in n-3 downto 0 generate
begin
      with SDM_reg(i) select
      polarity_diff(i) <= polarity_diff(i+1) + assign_diff(i) when '1',
      polarity_diff(i+1) - assign_diff(i) when others;
      SDV_reg <= polarity_diff(0);
end generate SDV;

CorrectedBINARY((n-1) downto 0) <= C_binary((n+2)-1 downto (n+2)-n);
correct_fin <= C_fin;
END behav;
```

C.5.3 VHDL Code of the TM-ARCHα-7 ADC Signal Emulator

The following code was developed to imitate the output of the TM-ARCHα-7 ADC modelled in the MATLAB script (see Appendix C.5.1), when supplied with a ramp input signal.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use IEEE.numeric_std.all;

ENTITY Signal_pipelined_generator is
generic (n: integer:= 8);
PORT  (clk_p: in std_logic;  -- s/h clock       (25 MHz)
       res: in std_logic;  -- resets all Difference registers
       Output_signal_piped: out std_logic_vector(n-1 downto 0)); --
equivalent output of the TM-ADC
END ENTITY;

ARCHITECTURE behav OF Signal_pipelined_generator IS
signal Gen_counter: integer := 0;
signal in_value: integer := 0;         --value into the "TM-ADC"
signal in_valueV: std_logic_vector(7 downto 0):= (others => '0'); --
binary equivalent of in_value
signal out_value: std_logic_vector(7 downto 0):=(others => '0'); --
output
signal S0: std_logic_vector(1 downto 0):= (others => '0');
signal S1: std_logic_vector(3 downto 0):= (others => '0');
signal S2: std_logic_vector(5 downto 0):= (others => '0');
signal S3: std_logic_vector(7 downto 0):= (others => '0');

type add2prev is array (0 to 2**(n+1) - 1) of integer;
signal diff: add2prev :=            --difference values for ramp
(
-- ramp signal emulation array--
-- contents generated by MATLAB Script presented in C.5.1--
);

begin
form_signal_odds: PROCESS
      BEGIN
      WAIT UNTIL RISING_EDGE(clk_p);   --PLL_IN the PLL clock 250MHz
      in_value <= in_value + diff(Gen_counter);  --Add difference to
previous input amplitude
      in_valueV  <= std_logic_vector(to_unsigned(in_value, n)); --
convert to binary
      S3(7 downto 6) <= in_valueV (7 downto 6); -- 2 MSBs go to the 2
MSBs of array S3
      S2(5 downto 4) <= in_valueV (5 downto 4); -- 3rd and 4th MSBs go
to the 2 MSBs of array S2
      S1(3 downto 2) <= in_valueV (3 downto 2); -- 3rd and 4th LSBs go
to the 2 MSBs of array S1
      S0(1 downto 0) <= in_valueV (1 downto 0); -- 2 LSBs go to the 2
MSBs of array S0
      out_value <= S3;  --ADC output would be the same as S3
      if Gen_counter < 2**(n+1) - 1 then
            Gen_counter <= Gen_counter + 1;     --increment counter
            S3(5 downto 0) <= S2(5 downto 0); --shift S2 values into
S3[5:0]
            S2(3 downto 0) <= S1(3 downto 0); --shift S1 values into
S2[3:0]
```

```
            S1(1 downto 0) <= S0(1 downto 0); --shift S0 values into
S1[1:0]
      else
            Gen_counter <= 0;                              ----reset counter
            S3(5 downto 0) <= S2(5 downto 0); --shift S2 values into
S3[5:0]
            S2(3 downto 0) <= S1(3 downto 0); --shift S1 values into
S2[3:0]
            S1(1 downto 0) <= S0(1 downto 0); -- 2 LSBs go to the 2 MSBs
of array S0
      end if;
end process;

Output_signal_piped     <= out_value;
END behav;
```

C.6 Code for the Practical Implementation of a Tent Map Based ADC with an Embedded Tent Map Gain Compensation System

The MATLAB code extract below was adapted to model the practical implementation of the TM-ARCHα-7 ADC with the μCA-2. The extract below shows how key parameters were configured, for example, the input signal, comparator hysteresis (threshold voltages are based on the measured Vref and resistor values implemented on the PCB in order to apply external hysteresis on the comparators) and $\mu_{\pm}$ values. The rest of the MATLAB script was similar to code presented in Appendix C.5.1, so is not presented.

The VHDL code and MATLAB script employed in the practical experiment was identical to the code presented in Appendix B.5.1, Appendix C.5.1 and Appendix C.5.2. The only difference were the $\mu_{\pm}$ values (see Appendix D.2) used in the MATLAB script, which altered the DM values being generated to be employed by the VHDL.

```
%% Initialise key parameters for model
resolution = 8;              %number of TM stages + 1
Vcc = 5; % valid input max.
Vee = -0.232; % valid input min.
Vref = 1.505;   %set partition point voltage based on measurement
vin_range = 2*Vref - 0;
VHDL_res = 10;  % resolution of DM values

% resistors used to set threshold voltages for comparator hysteresis
hys_R1 = 364000;
hys_R2 = 150000000;
VTH = (((hys_R1 + hys_R2)*Vref)-(hys_R1*0.09))/hys_R2;
VTL = (((hys_R1 + hys_R2)*Vref)-(hys_R1*4.89))/hys_R2;
Step_size = (2*Vref)/(2^resolution); %calculating step size
VH_pos =  VTH*ones(1, resolution);
VH_neg = VTL*ones(1, resolution);

%% Generate input signal
pwr_val = resolution + 1;
y = (0:0.1:3);  % measured input signals
number_samples = length(y);
x = 1:1:number_samples;

%% Gain and Vref Parameters
% spreadsheet contains midrange μ+ and μ- calculated for each TM stage
```

```
readtable("C:\Users\pkhaz\Documents\Data\Work\Uni -
PhD\Tests_year3\PCBv_3\31st July 2021\DMM gain
test.xlsm",'Sheet','Sheet3', 'range', 'K2:Q7');
gains = table2array(ans);

gain_pos = gains(5,:); % μ+
gain_neg = gains(6,:); % μ-
                                    […]
```

Appendix D

D.1 Effects the Resolution of the Difference Measure values has on Bit Accuracy for Different TM Gains

Table D-1 and Table D-2 presents the results from the experiment described in Section 5.2.1, where different r values were tested to determine the minimum DM resolution required in order for the TM-based ADC to have the same bit accuracy after compensation over a range of μ, when compared to the theoretical DM values being employed.

	Bit accuracy (bits)							
μ value	**Prior Compensation**	**Theoretical DM values**	**DM values to 17-bit resolution**	**DM values to 18-bit resolution**	**DM values to 20-bit resolution**	**DM values to 22-bit resolution**	**DM values to 23-bit resolution**	**DM values to 24-bit resolution**
1.9	4.41	13.42	13	13.42	13.42	13.42	13.42	13.42
1.91	4.55	14	13.42	13.42	13.42	13.42	13.42	14
1.92	4.71	14	13.42	13.42	14	14	14	14
1.93	4.89	14	13.42	13.42	14	14	14	14
1.94	5.12	14	13.42	13.42	14	14	14	14
1.95	5.38	14	13.42	13.42	14	14	14	14
1.96	5.69	14	13.42	14	14	14	14	14
1.97	6.10	14	13.42	14	14	14	14	14
1.98	6.67	14	13.42	14	14	14	14	14
1.99	7.65	14	13.42	14	14	14	14	14
2	15	15	15	15	15	15	15	15

Table D-1: Summary of bit accuracy before and after compensation for a TM-ARCHα-15 ADC.

	Bit accuracy (bits)			
μ value	**Prior Compensation**	**Theoretical DM values**	**DM values to 9-bit resolution**	**DM values to 10-bit resolution**
1.9	4.19	6	6	6
1.91	4.19	6	6	6
1.92	4.42	6	6	6
1.93	4.42	6	5.42	6
1.94	4.68	6	6	6
1.95	5	6	6	6
1.96	5	6	6	6
1.97	5.42	6	6	6
1.98	6	6	6	6
1.99	6	6	6	6
2	7	7	7	7

Table D-2: Summary of bit accuracy before and after compensation for a TM-ARCHα-7 ADC.

D.2 TM Slope Gains Calculated from Electronic Implementation of the TM-based ADC

Table D-3 presents the calculated $\mu_{\pm}$ values for each TM stage of the electronic implementation of the TM-ARCHα-7 ADC.

	TM1	TM2	TM3	TM4	TM5	TM6	TM7
μ+ midrange	1.9220	1.9900	1.9744	1.9383	1.9186	1.8881	1.8446
μ- midrange	1.9309	1.9614	1.9756	1.9065	1.8179	1.8111	1.8483

Table D-3: $\mu_{\pm}$ values determined for each TM stage of the electronic implementation of the TM-ARCHα-7 ADC.