NTNU
Norwegian University of Science and Technology

NORDIC SEMICONDUCTOR

Investigation of High Accuracy Mixed-Signal Time-to-Digital Converter

Ghassan Al-Omari

Master of Science
Department of Electronic Systems
Norwegian University of Science and Technology

February, 2020

Contents

Preface

This thesis fulfills the requirements for the degree in Master of Science in Engineering for Electronic Systems Design at the Norwegian University of Science and Technology (NTNU), at the Department of Electronic Systems.

The subject of this master thesis is to measure the time between the rising edges of two separate signals. The initial suggestion for this time measurement is a Vernier Delay Line (VDL) however, other options are explored. A big concern with Vernier delay line is the consistency of the accuracy, as delay lines will have production variations. A specification for the required accuracy is established and used as the basis for the suggested solution. Even though it falls outside the scope of the thesis, calibration of the delay line or other solutions can be used as a way of further increasing the accuracy of the time measurement. The selected solution is only required to produce an unsigned logic output representing the time difference between the two separate input signals. The work done in this master thesis is not a continuation on a specialization project from previous semester and was completed in the standard time period.

Sammendrag

I denne masteroppgaven er hovedmålet å måle tiden mellom stigende flanker av to separate signaler. Det ble foreslått å bruke en såkalt ”Vernier Delay Line” (VDL), men problemet med dem er at nøyaktigheten er inkonsekvent på grunn av produksjonsvariasjoner. Derfor er det motivasjon til å prøve ut andre løsninger for å måle tidsforskjell. Det blir utviklet et krav til nøyaktigheten av målingen, som brukes som veiledning til den foreslåtte løsningen. Selvom det er utenfor omfanget av oppgaven kan kalibrering av forsinkelseslinjen eller andre løsninger øke nøyaktigheten ytterligere. Den valgte løsningen trenger kun å produsere en ”unsigned logic” på utgangen for å representere tidsforskjellen mellom de to inngangsignalene. Denne masteroppgaven er ikke en fortsettelse av en prosjektoppgave utført semesteret før, og den ble fullført på normert tid.

Oppgaven tar for seg en TDC arkitektur som bruker flere VDLer i parallell for å måle tidsforskjell. TDC arkitekturen oversetter ”thermometer”-kode til binærkode uten å bruke konvensjonelle enkodere. I oppgaven brukes både digitale og analoge designmetoder. De analoge designene er implementert med 28nm FDSOI CMOS teknologi, og de digitale designene er implementert ved hjelp av SystemVerilog.

Den foreslåtte TDC arkitekturen er en 6-bits monotonisk TDC med 4.62 lineære bits. TDCen sampler i en hastighet av 110 MS/s over et dynamisk område av 630ps. Prosess og ”mismatch” variasjoner er ekstrahert fra det analoge designet, og ikke-lineæriteter er ekstrahert fra det digitale designet. Den foreslåtte arkitekturen har en DNL av +1.4/-1.0 og INL av +1.6/-1.0 som viser fordelen av å bruke parallelle forsinkelseslinjer. Arkitekturen forbruker 0.887 mW fra en 0.9V spenningskilde, og opptar 0.0039 mm^2. Resultatene i denne oppgaven er fra målinger tatt fra et ”pre-layout” design.

Abstract

A TDC architecture that uses a multiple VDLs operating in parallel to measure time is discussed in this thesis. The TDC converts the thermometer code to a binary code without using conventional encoders. The thesis took advantage of both analog and digital workspaces, the analog workspace is implemented using 28nm FDSOI CMOS technology, while the digital workspace is implemented in SystemVerilog.

The proposed TDC is 6 bits monotonic TDC, with 4.62 linear bits. The TDC samples at a rate of 110MS/s over 630ps dynamic range. Process and mismatch variation are extracted from the analog workspace, while nonlinearities are extracted from the digital workspace. The proposed TDC has a DNL of +1.4/-1.0 and INL of +1.6/-1.0, which shows the benefit of using parallel delay lines. The TDC consumes a 0.877 mW from a 0.9V voltage supply and occupies a 0.0039 mm^2. The results in this thesis are reported from pre-layout measurements.

Acknowledgements

I want to express my sincere gratitude to Professor Morten Olavsbråten, Engineer Svein Henninen, and Engineer Carolina Velezmoro. Their motivation, support and technical insight have guided me during my master thesis. I would like to take a moment of silence for my late advisor professor Kjetil Arnt Svarstad, who has always been a source of motivation for me.

I wish to show my appreciation for the Norwegian University of Science and Technology (NTNU), and for Nordic Semiconductor to give me the opportunity to use the available tools and for their continuous support. I would like to thanks Aslak Holen, Khalil Gammoh, Somayeh Zadeh, and Harri Kinnunen, whose assistance was a milestone in the completion of this thesis. A special thanks to Yohan Sandvik for assisting me in writing the preface in Norwegian.

Thanks to my parents Ali and Khawla, my dear sisters Farah, Raghad, Duha, Shatha, Gheed, and to my dearest friends Aaliyah, Elias, Giovanna, Khalid, Milla, Musa, Omar, and Saed, for their support and encouragement. They have always been part of my life and part of this thesis.

In the end, I want to thanks everyone and every institution that helped and supported me. I could not have completed my master's degree successfully without it.

List of Figures

List of Tables

Abbreviations

ADC	Analog to Digital Converter. 4, 8
AI	Artificial Intelligence. 1
CMOS	Complementary Metal Oxide Semiconductor. 1
CV	Coefficient of Variation. 37
DL	Delay Line. 3
DLL	Digital Phase-Looked Loop. 1, 2
DNL	Differential Nonlinearity. 5
DUT	Design Under Test. 31
ENOB	Effective Number of Bits. 5
FDSOI	Fully Depleted Silicon on Insulator. 20, 49
GRO	Gated Ring Oscillator. 12
INL	Integral Nonlinearity. 5
LIDAR	Light Detection and Ranging. 1
LSB	Least Significant Bit. 4
MOM	Metal Oxide Metal. 25
NMOS	N-Channel MOSFET. 26, 31

PET Positron Emission Tomography. 2

PLL Phase Lock Loop. 2

PMOS P-Channel MOSFET. 26, 31

PVT Power, Voltage and Process Variation. 1

RO Ring Oscillator. 12

SR Set-Reset. 22

TDC Time to Digital Converter. 1, 2, 4

TOF Time of Flight. 2

VCO Voltage Controlled Oscillator. 8

VDL Vernier Delay Line. 11

Chapter 1

Introduction

The industry today is pushing for more tasks to be done autonomously (i.e. using robots and Artificial Intelligence (AI)), such as Light Detection and Ranging (LIDAR) used in autonomous cars, Digital Phase-Looked Loop (DLL), and various biomedical applications. Many of these applications require a highly precise time measurement to achieve the expected performance.

Designing a circuit that can achieve a nanosecond or even a picosecond precision can elevate the performance of these systems dramatically [1]. A circuit with such performance should also be compact and power-efficient. Analog building blocks could achieve such a performance, but they often have large area and high power consumption. Digital building block can overcome these drawbacks and offers a more compact, and power-efficient design. Moreover, digital circuits benefit substantially from the advancing of the Complementary Metal Oxide Semiconductor (CMOS) technology.

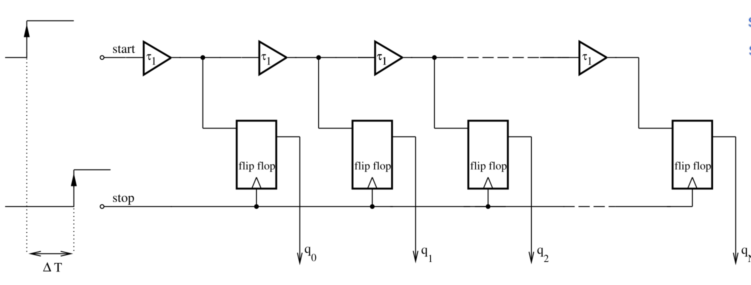
Time to Digital Converters (TDC) are circuits that convert a time measurement into a digital code. TDCs have become very popular in recent years due to their performance and scaling of CMOS process. The latter one is important since the TDC can be designed using all-digital blocks, which means that the TDC can have the advantages of digital circuits while being tolerant to Power, Voltage and Process Variation (PVT).

The scaling down of CMOS technology combined with architectural improvements made TDC a very attractive choice for systems that depend on time measurement. For example, in communication and mixed-signal systems, recent works [2], [3] have shown a significant increase in the performance by replacing the phase detector in the Phase Lock Loop (PLL)/Digital Phase-Looked Loop (DLL) with a TDC because it is more accurate and area/power efficient. Other applications have adopted TDC to take advantage of their superior performance, such as biomedical imaging using Positron Emission Tomography (PET) technique [4]. Furthermore, the automotive industry took these advantages as well by developing laser rangefinders using Time of Flight (TOF) [5]. Measurement instruments implemented TDC to achieve state of the art instrument accuracy, such as oscilloscopes, logic analyzers, and timing jitter measurement [6] [7].

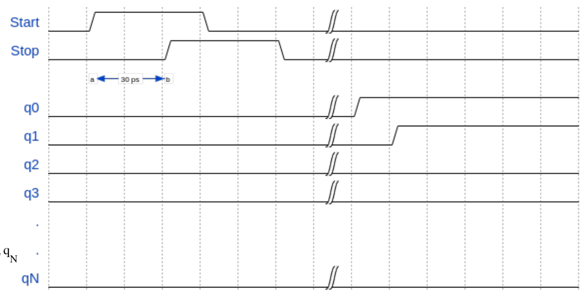
This work focuses on studying and comparing different TDC architectures and find a suitable implementation for the given specifications. Moreover, study the effects of process variations on the implemented TDC and overcome these variations. The rest of this work is organized as follows; Chapter 2 gives a background on TDC, their classification, and a comparison between different implementations. Based on that, Chapter 3 describes the TDC design and simulation methods. Chapter 4 presents simulation results, Chapter 5 discusses these results and deliver conclusions which are represented in chapter 6.

Chapter 2

Theory



(a) Delay line TDC Architecture Block Diagram



(b) 100ps Measurement Timing Diagram

Figure 2.1: Illustration for Simple TDC

A simple TDC architecture consists of a Delay Line (DL) with a D-flip flop, as shown in figure 2.1 (a). The TDC measures the time difference between start and stop signals (ΔT) by sampling the state of delayed start signal through flip-flops. The output of TDC is a digital word in Unary coding (thermometer code), where the output is represented with n ones followed by zeros. For example, 4 will be represented as (111100...). Figure 2.1 (b) shows the timing diagrams of measuring a 100ps (ΔT) using the previous TDC. The thermometer code output q is (11000..), which is defined by the TDC metrics.

2.1 Performance Metrics

There are several numbers of TDC architectures, each of which has its advantages and disadvantages. In order to study these architectures some performance metrics should be defined. These metrics will allow a systematic comparison between TDC architectures since these metrics can be applied to any TDC.

2.1.1 Resolution (LSB)

The smallest time difference that a TDC can detect is called the resolution. This difference is converted to a digital output word defined by the term Least Significant Bit (LSB). Since TDC operates similarly to Analog to Digital Converter (ADC), one can look at this metric as the step width of the input-output transfer curve (quantization curve) of TDC shown in figure 2.2. Ideally, step width is defined by the application and it should be constant along the curve. Each step (1 LSB) represents a digital output word increment to a time input increment [8].

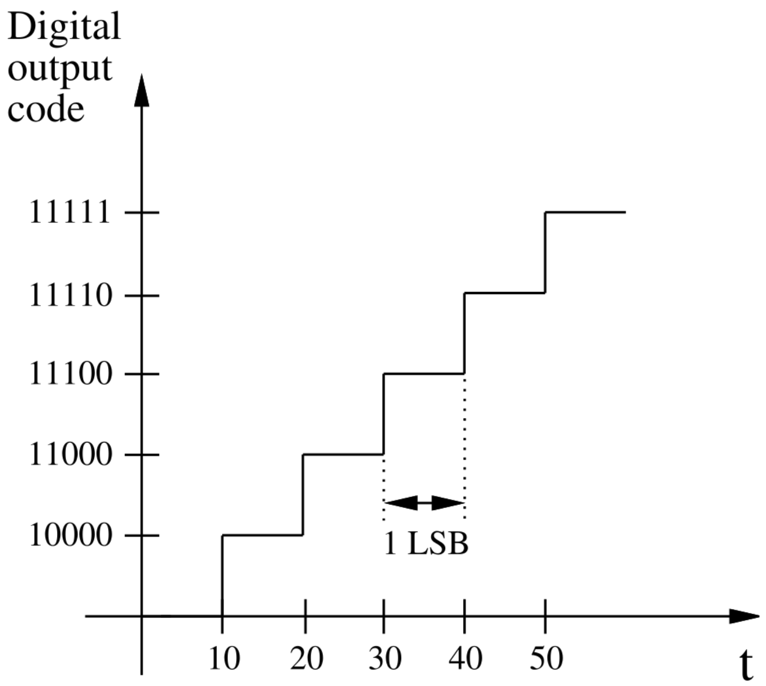


Figure 2.2: Quantization Curve for TDC

2.1.2 Non-linearities

Various noise sources affect the performance of TDC, causing the input-output transfer curve to deviate from its ideal values, resulting in a non-linear curve. These errors can come from delay time mismatch of delay cells and PVT variations. Mainly these nonlinearities can be represented as two parameters, Differential Nonlinearity (DNL) and Integral Nonlinearity (INL). Since different noise sources will cause a deviation on the ideal transfer curve, DNL parameter represents the difference between actual and ideal curves. INL, on the other hand, represents the integration of that difference along the transfer curve (DNL) up to the calculation position. In other words, the INL represents the accumulations of nonlinearities along the time conversion. There are multiple techniques to overcome these nonlinearities in TDC, such as architectural manipulation or calibration circuits.

Nonlinearities are measured in LSB, and can predict the behavior of TDC output as in ADC output [8]. Monotonicity means the output of TDC will always increase as input increases. To guarantee that a TDC is monotonic, the maximum DNL has to less than 1 LSB, or the maximum INL has to less than 0.5 LSB. In many cases, the TDC might have a DNL error greater than 1 LSB and still consider monotonic [8]. Monotonicity is essential to some applications where TDC is used in a feedback loop, where a decrease in the digital output code could make the system oscillate. Missing Codes is also an essential behavior for TDC, where if INL is larger or equal to 1 LSB can indicate a missing digital output code from the TDC [8].

2.1.3 The Linear Bits

The Effective Number of Bits (ENOB) is a metric used in ADCs to represents the number of linear bits. Since TDC and ADC are similar in performance metrics, the N_{linear} is used to represents the effective number of linear bits in a given TDC. This metric is introduced to TDC since it is challenging to generate a pure sinusoidal signal for the inputs of TDC (start and stop signals) [9] [10]. N_{linear} can be calculated using equation 2.1, where N is the number of bits for TDC.

$$N_{linear} = N - log_2(INL + 1) \tag{2.1}$$

2.1.4 Conversion Speed

The process of measuring time intervals and transform it into a digital word requires time; this time is known as the Dead Time, in which TDC cannot perform another measurement during it. In other words, the dead time measures how fast a TDC can perform one measurement. The speed in which is usually called the conversion speed, and it is given as a sample per second (S/s). This metric might be considered essential and that TDC should be designed to operate as fast as possible. However, in reality, the conversion speed depends on the application utilizing the TDC. Some architectures separate the sampling rate from the conversion speed to achieve higher speeds. By separating both operations (i.e. pipelining), TDC could convert a measured time interval into a digital word while sampling a new one at the same time [11].

2.1.5 Dynamic Range

Since TDC measure time, the maximum interval that can be measured is defined as the dynamic range. This parameter depends on the LSB and TDC architecture. In basic concepts, a higher dynamic range means that TDC can detect a broader range of measurements. This will reflect, on the LSB and conversion speed for some architectures, thus causing TDC to become slower since the measured signal has to propagate longer in TDC before it gets converted into a digital code. As with conversion speed, the wight of this parameter in designing a TDC is application dependent as well.

2.1.6 Area and Power

As for most CMOS designs, area and power consumption are considered essential parameters. Especially with the advancing of technology and the need for more efficient chips in recent years. Different architectures will have better performance metrics than others but at the cost of area and power. In order to have a more compact design, the area and power for the desired circuit must be considered before implementation because the architectural implementation has a significant effect on them.

2.1.7 Noise

Noise sources in TDC come from transistors changing the signal delay time in each stage due to jitter and PVT variations. This will cause the output of TDC to vary around the expected value for the same input. The variations of TDC output generally follow Gaussian distribution [1], which can be estimated by calculating the standard deviation (sigma) for each source.

2.1.7.1 Jitter

Jitter is a random noise added to signal from each stage by its transistors. This noise is represented as Gaussian distribution. Usually, for TDC, the jitter is less than 1% of the nominal delay; thus, it can be ignored.

2.1.7.2 Process and Mismatch

The delay times of different cells cannot be the same since no two transistors can be alike. These changes are caused by process and mismatch variations, where process variations represent changes between two transistors on different silicon wafers for the same circuit. Mismatch, on the other hand, is the difference between transistors on the same silicon wafer. These variations come from imperfections in transistors fabrications and can be modeled as Gaussian distribution using Monte Carlo simulations [12]. These effects will be studied further in this work.

2.1.7.3 Supply Voltage Variation

Supply variations affect the speed of the transistors in the circuit, thus changing the overall performance. These variations are ignored in this work by assuming a well designed band-gap voltage supply is available.

2.1.7.4 Temperature Variation

The temperature change affects the threshold (V_{th}) of transistors, which will reflect on their speed and the overall performance of TDC. However, in this work, the temperature is considered fixed at 27°

2.2 TDC Architectures

TDC measures time intervals between two different signals (i.e. START and STOP); for some applications such as DLL/PLL, TDC is used to lock the loop by comparing the output frequency with a reference frequency to use it as a control signal for the Voltage Controlled Oscillator (VCO). However, in this work, the input signals (START and STOP) will be provided from the same source. Since there are various architectures, there is no general operation for TDCs, and every architecture performs the time measurement differently. These variations mean that performance metrics of a TDC are defined by the time measurement technique (architecture of the TDC). In other words, the overall performance of a system that uses time measurements depends significantly on the TDC architecture. In the following section, popular architectures will be presented along with how they operate and their pros and cons. Finally, a comparison will be made to discuss and select a suitable architecture for this work.

2.2.1 Analog-type TDCs

Analog architectures are considered to be the first generation of TDCs. The most straightforward approach is to change the charge in a capacitor with respect to time differences (i.e. convert the time to voltage level), as shown in figure 2.3. Then convert this voltage into a digital code using either an ADC or a voltage comparator. Although this method is a straight forward, the resolution and the dynamic range depends on the number of bits for the ADC (the resolution of the ADC), which limits TDC performance [13].

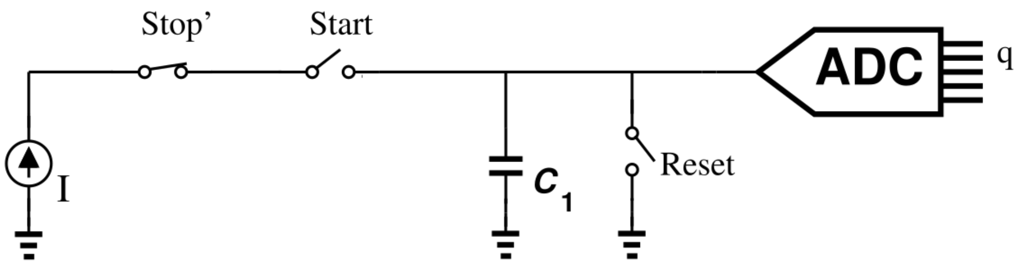


Figure 2.3: Block Diagram of Basic Analog TDC

To improve the resolution, dual-slope (pulse stretching) architecture is developed, where a voltage comparator and a second capacitor (C_2) are introduced, as shown in figure 2.4. During the measurement of the time interval, the voltage on the positive node of the comparator is larger than the negative node. This difference will make the output of the comparator to be +1. When the stop signal comes, the current source i_2 will start to return the output voltage

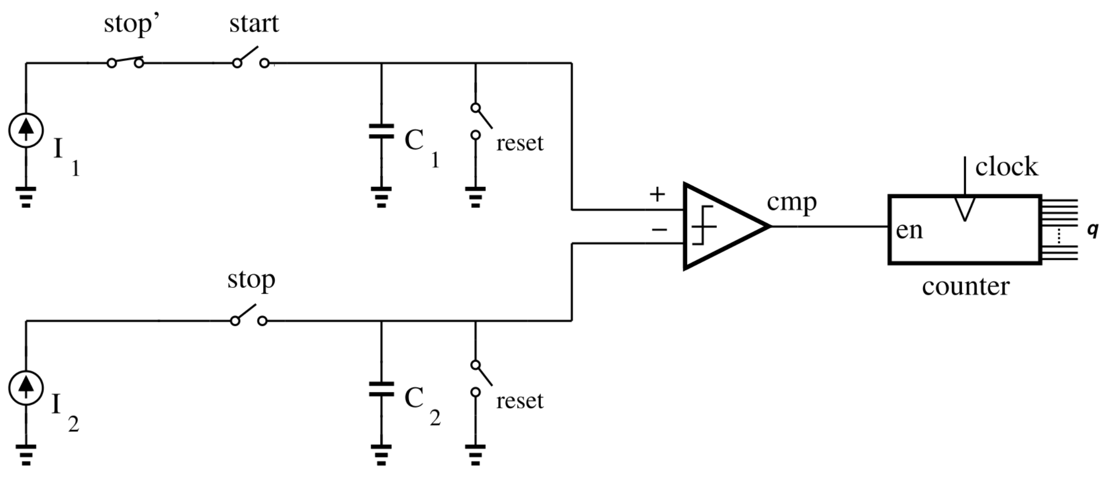


Figure 2.4: Block Diagram of Dual Slope Analog TDC

of the comparator to zero. Using a counter with a known reference clock, the time needed to compensate for the voltage difference on the node can be measured, which represents the time interval. This method can achieve high time resolution and high dynamic range [14].

Time-Amplifier architecture (TA) is introduced to improve the resolution of TDC by having coarse-fine measuring paths. TA TDC takes the residue of time measurement from the coarse path and amplified it to be measured again. Architecture that uses TAs has an excellent resolution, such as [15]. However, the main disadvantage is that time-amplifiers have a small linear region in which they can operate, which results in a very narrow dynamic range [1].

Although analog TDCs offer good metrics, they still suffer from PVT variations, high INL for long-dynamic range, require careful design and take large area [1], which makes them unpopular in modern CMOS design.

2.2.2 Digital-Type TDCs

Digitally implemented TDCs overcome the drawbacks of the analog implementations and achieve the same or even better metrics as CMOS technology advances. There are many techniques to implement digital TDC; the simplest is to count the number of ticks for a reference clock during the time interval. However, to achieve high resolution, a very accurate Giga-Hz reference clock and a compatible counter are needed, which is not practical in real-life applications.

2.2.2.1 Delay Line TDC

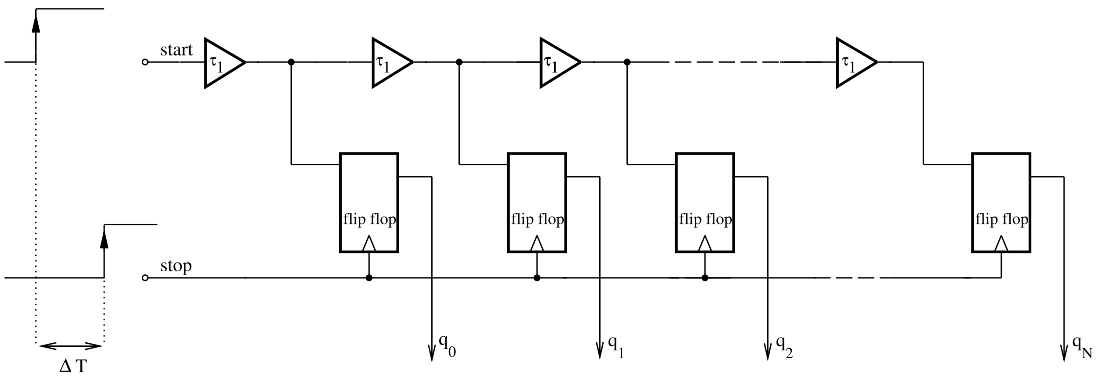


Figure 2.5: Block Diagram of Basic DL TDC

In order to obtain a high-resolution TDC without the need for an accurate clock signal, a delay line can be placed in the start signal path with the output of every delay element is connected to a flip-flop as shown in figure 2.5. To measure a time interval, the start signal has to propagate through the delay line within a specific delay time (τ_1) added by each delay cell, and when stop signal toggles to one, it will latch all the outputs of the delay line. The final output word will be a thermometer code with every one represents a single time measurement (LSB = τ_1). For example, if ΔT is 30ps, and the resolution (LSB) is 15ps, each delay cell will add 15ps delay to the start signal until stop signal becomes one. The output q, in this case, will be (11000...). The drawback of delay line TDCs is the limitation on their resolution (LSB), which is limited by the minimum propagation delay of transistors. Moreover, it requires calibration against PVT, and it is only suitable for a short dynamic range [1].

2.2.2.2 Vernier Delay Line TDC

The Vernier Delay Line (VDL) overcome technical limitations on the resolution by introducing a faster second delay line for the stop signal, as shown in figure 2.6. A sub-gate resolution is now achievable through the difference between delay lines cells $LSB = \tau_1 - \tau_2$, where τ_1 and τ_2 are the delay for cells in the start line and stop line, respectively. For example, if ΔT is 15, τ_1 and τ_2 are 20ps and 15ps, respectively. The resolution (LSB) will be 5ps; each signal will experience different delay times while propagating. Since the stop signal will propagate faster, it will catch up with the start signal at q_2 and sample the outputs of the flip-flops. The output q, in this case, will be (11100...).

Furthermore, since the resolution is the difference in the delay time, the VDL architecture can compensate first order PVT variation if the two delay lines are well-matched [16]. Although the VDL implementation solved the resolution issue, it still requires an exponential increment in the number of stages to increase the dynamic range (2^N).

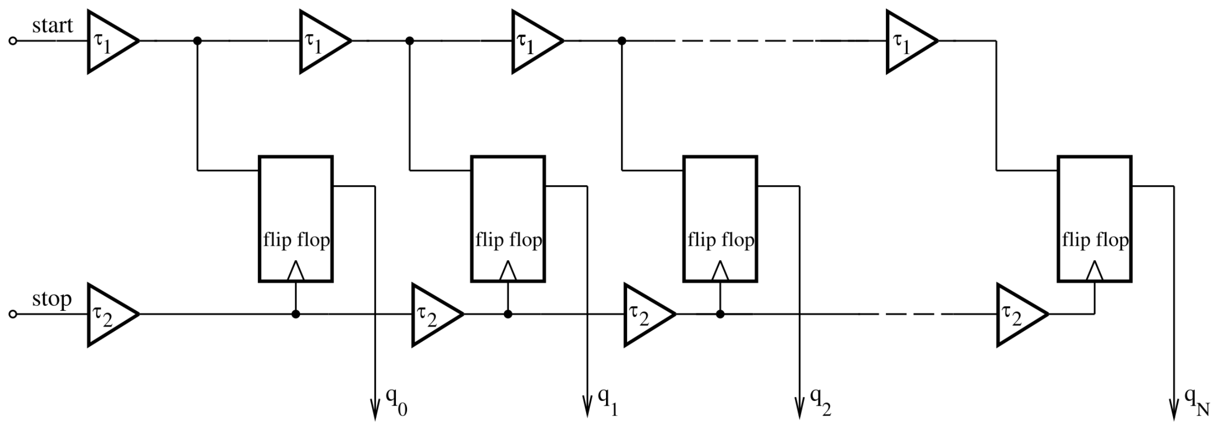


Figure 2.6: Block Diagram of Vernier Delay Line TDC

2.2.2.3 Ring Oscillator Vernier TDC

Ring Oscillator (RO) VDL solves the dynamic range issue of the normal VDL while maintaining a high resolution. By looping back the input signals (i.e start and stop) around and connects the end of the loop to a counter, as shown in figure 2.7. The counter output will indicate how many times the signals have passed, thus increasing the dynamic range dramatically, which will depend on the counter output and the number of delay stages. However, this architecture has worse nonlinearities, since noise and jitter will be accumulated throughout the ring [1].

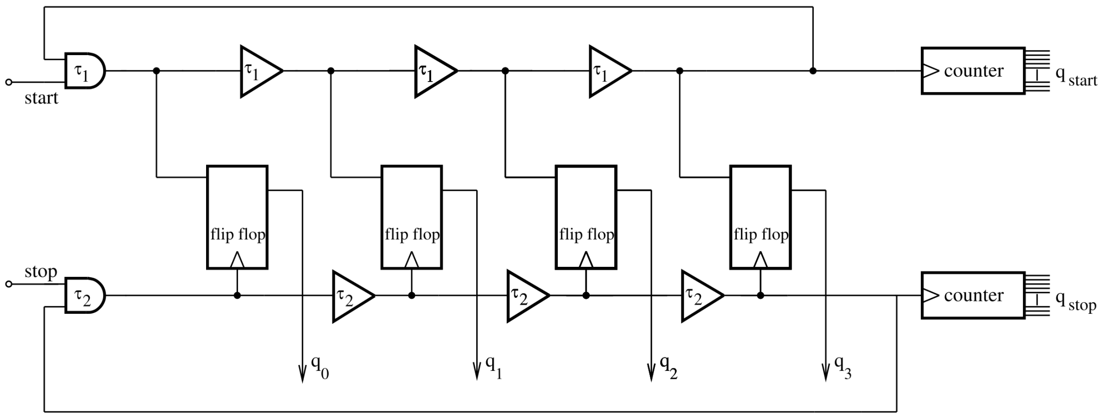


Figure 2.7: Block Diagram of RO Vernier Delay Line TDC

Gated Ring Oscillator (GRO) VDL architecture [17] is introduced to eliminate the noise accumulation problem, by adding an enable signal to the delay cells in the fast and slow delay lines for the TDC in figure 2.7. The enable signal will be on at the beginning of measurement and will make the delay lines work as regular oscillators. When stop signal catches up with start signal, the enable will be off, thus saving the current values of delay cell nodes. Any new measurement starting point to be the ending point of the previous one, which will make the residue at the end of the previous measurement transfer to the new one. This transfer will provide first-order noise shaping and reduce the mismatch between delay cells [1], [18]. The drawback of GRO is the need for holding previous values until the next measurement starts. This means the need for a capacitor with low leakage to keep old values valid, which is difficult to obtain in modern CMOS process. Moreover, even if the first-order noise is eliminated, the GRO architecture still requires calibration against PVT variations.

2.2.2.4 Two Dimensional TDC

Another method for improving the narrow dynamic range for VDL is 2-D architectures VDL [19], which solves the dynamic range issue by constructing a two-dimensional matrix from the output of GRO vernier delay lines as shown in figure 2.8. This architecture makes the growth in the delay line elements to the $\sqrt{N}$ instead of exponentially (2^N) while maintaining a high resolution (LSB). Figure 2.8 shows the output matrix of the 2-D GRO VDL, where the diagonal line represents the output expected from the normal VDL (i.e.$\Delta = LSB$) and the lower-half part represents the extended delay using the slow chain. The upper-half part, however, represents invalid measurements, where the stop signal will arrive before the start signal.

For example, if ΔT is 60ps, τ_x and τ_y are 50ps and 40ps, respectively. The resolution will be 10ps. In the beginning, for the time measurement, the control circuit will enable both GROs. Both signals will circulate since ΔT is larger than 50ps; at $X = 2$, and $Y = 2$ stop signal will pass start signal, which will set the enable to off. An encoding circuit will detect the signals circulation and modify the coordinates with respect to the number of circulations, decrementing Y by one in this example. The final coordinates for the Vernier plane are $X = 2$ and $Y = 1$, which indicate 6 LSB (60 ps). Although this architecture improves the dynamic range, the overall complexity of TDC increases, plus the need for advanced control and decoding circuitry.

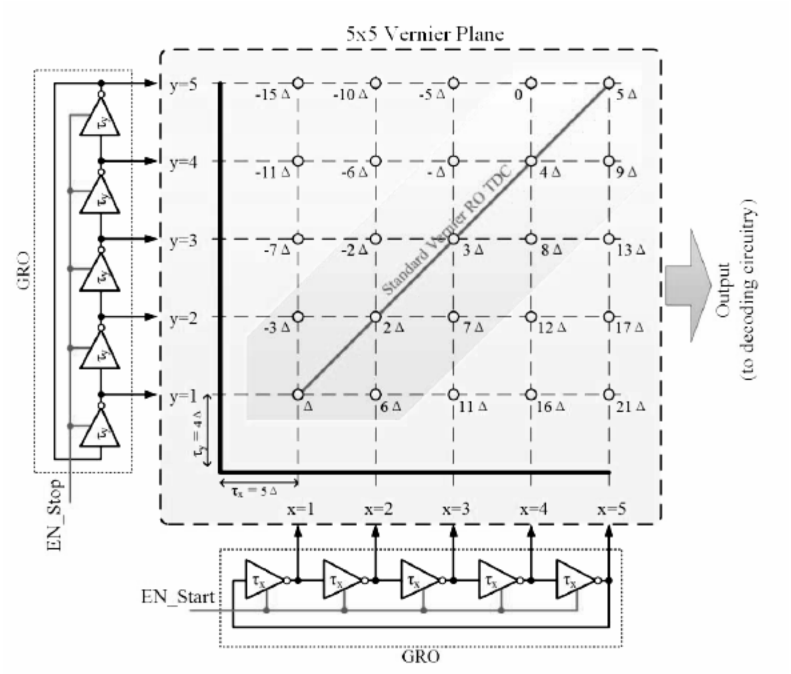


Figure 2.8: Block Diagram of 5x5 2-D GRO-VDL TDC [1]

2.2.2.5 Stochastic TDCs

All TDCs will have mismatch and jitter between different stages, which affect the performance of most TDCs. Nevertheless, stochastic architecture takes this drawback and uses stochastic properties to achieve high resolution and stability. The stochastic TDC implemented in [20] uses a set of latches known as arbiters ($q_0 - q_n$) shown in figure 2.9, with different offset voltages (V_{th}) which represents process variations. When the start signal crosses the voltage (V_{th}) of a latch before the stop signal, the output of that latch will be +1; otherwise, it will be -1. Since all latches are identical, their cumulative summation of input voltage offsets follows a normal or Gaussian distribution. This will produce a resolution equal to $\frac{\sqrt{2\pi}*\sigma_{vth}}{2^N-1}$, where σ_{vth} is the standard deviation of the offset voltage (V_{th}) and the $2^N - 1$ represents the number of latches. Even though this TDC takes advantage of process variations, it stills suffers from PVT variations and requires calibration to overcome it[21].

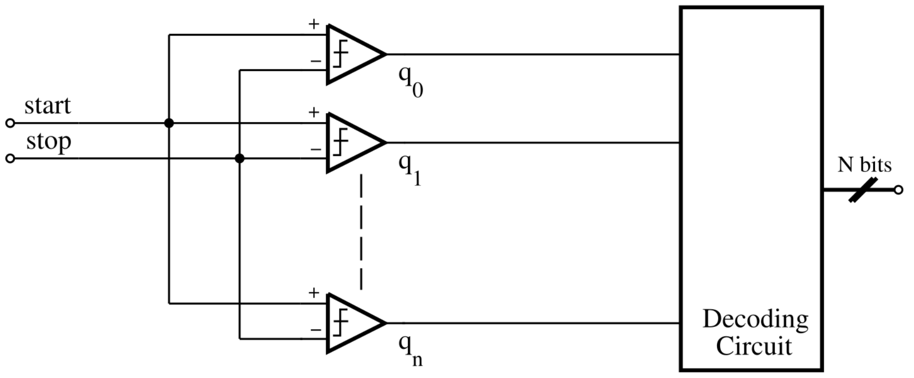


Figure 2.9: Block Diagram of Stochastic TDC Based on Latches

Recent developments in the stochastic TDC, such as [22], has overcome the PVT variations without the need for calibration. Figure 2.10, shows the implementation of the architecture, which counts the number of edges (d_n) produced by a fast refrence clock (i.e. 850MHz) in the delay line during the start and stop signals. Since the random mismatch and jitter will have a Gaussian distribution across all edges and delay cells, the summation of the output from the delay line will have a uniform distribution. Thus PVT variations will only change the delay for delay cell, not the distribution of the edges. This ensures that the total number of edges will stay the same across different corners. Furthermore, to achieve higher resolution and eliminate irregularities, an LSB bits truncation circuit is also implemented since most PVT variations will affect the lower bits of the TDC output code (the fine measurement). This architecture has shown excellent performance and resilient to variations. However, to achieve high characteristic

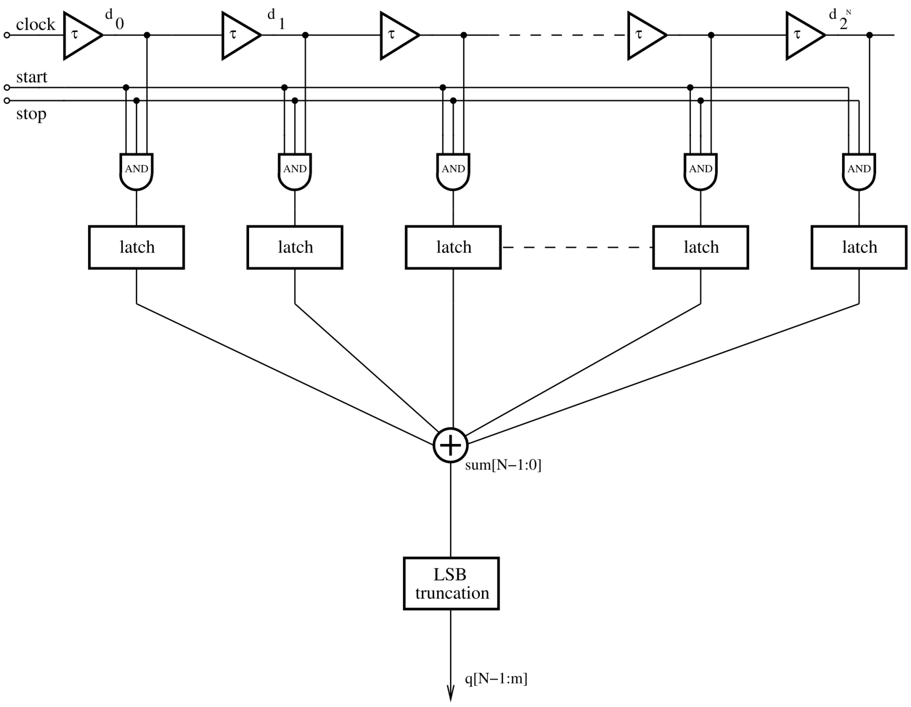


Figure 2.10: Block Diagram of Stochastic TDC Based on DL

distributions of PVT variations and jitter, the delay line has to be extremely long. For this TDC to achieve these characteristics, the delay line has to have 2^{14} delay cells. Furthermore, the authors of [22] did not prove the distribution of variations mathematically, and the LSB truncation shows that they have designed a TDC with femtosecond resolution to obtain measurements in picoseconds (i.e. using a stopwatch to count years).

2.3 TDC calibration

Since most TDC architectures suffer from PVT variations, which can degrade TDC performance, calibration circuits or methods are necessary to maintain a consistent TDC performance over PVT variations. The need for such circuitry is considered one of the most significant drawbacks to TDC because calibration circuits are often complex and require a big area [23].

Calibration schemes vary widely depending on TDC architecture. Often, TDC is used along PLL/DLL to replace the phase detector found in them. Since the clock signal of the PLL/DLL is always running and experience the same PVT variations, it can be used to calibrate the TDC using a control voltage delay cells, thus improve the overall accuracy. The feedback loop in [24] uses a DLL to improved the overall resolution and linearity dramatically of TDC to become 60 ps with +/- 1 LSB accuracy.

PVT variations affect the delay time of each cell in TDC differently; authors in [18] uses a different approach to calibrate TDC, as seen in figure 2.11. Using a circuit which can output the difference between the ideal delay and actual delay of the cells, by measuring the differences from outside the chip or by having a very accurate cell (ruler cell) to compare with, as in [25]. This difference is then used in a lookup table to get a digital calibration code that corresponds to that difference. This code will be substituted to a capacitor bank connected to each delay cell, which adjusts the total delay of the cell.

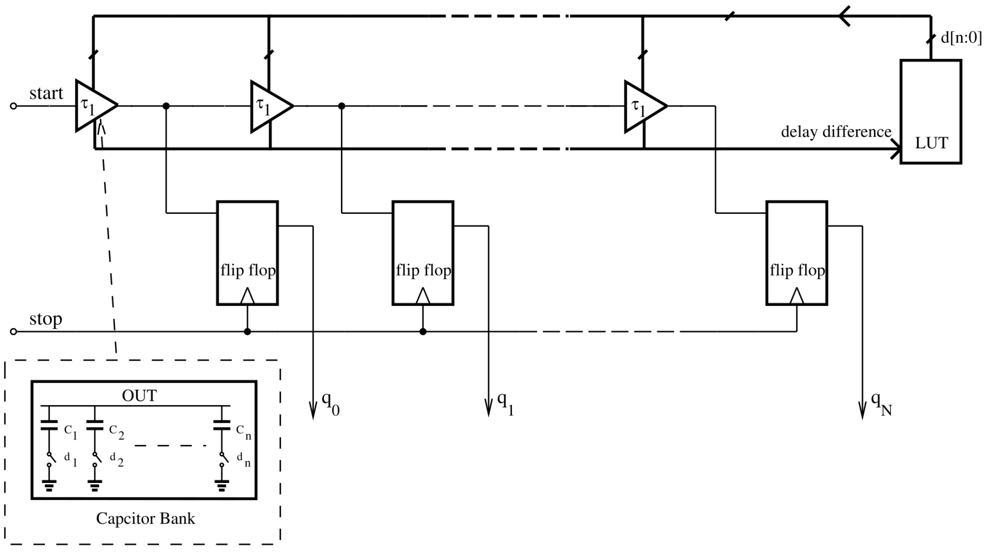


Figure 2.11: Block Diagram for Calibration of a DL TDC

2.4 Thermometer-to-Binary Encoder

Most TDC produces the output of the time measured as a thermometer code; an encoder is necessary to convert the time measurement to a binary-weighted code. The method for encoding the output of the TDC can affect the performance due to Bubble-Error, where some invalid transitions in TDC could cause the thermometer code output to have unsystematic 1s or 0s. For example, if a TDC measured a 40ps period with a resolution (LSB) of 10ps, the ideal thermometer code output should be (111100000...). However, the realistic (actual) code will be like (111100100...) or (11010000...) because of meta-stability errors, mismatch, and cross-talk [26].

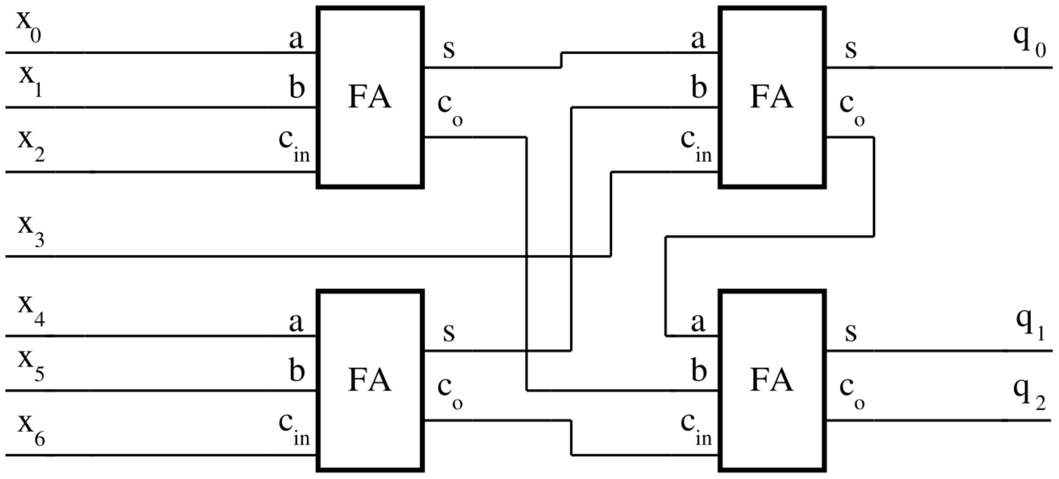


Figure 2.12: 3 Bit Wallace-Tree Encoder Block Diagram

The main contrasts between different encoders are speed, power consumption, and circuit complexity. The most straightforward architecture is the Wallace tree encoder, which counts the number of ones in the thermometer code using a $[2^N - (N+1)]$ full-adders, as shown in figure 2.12, where N is the number of encoded bits. This architecture is used widely in TDC due to its ability to correct bubble-error [26].

Another approach for encoding thermometer codes is a multiplexer based encoder, which uses $2^{N-1}th$ bit of the output code (pivot bit) to obtain binary-weighted code, as shown in figure 2.13. The number of multiplexers used in this approach is similar to the number of full-adders used in Wallace-Tree encoder. The work is done in [27] shows that Mux-Based encoders are less-complex and have lower power consumption than Wallace-Tree. However, it cannot correct bubble-error.

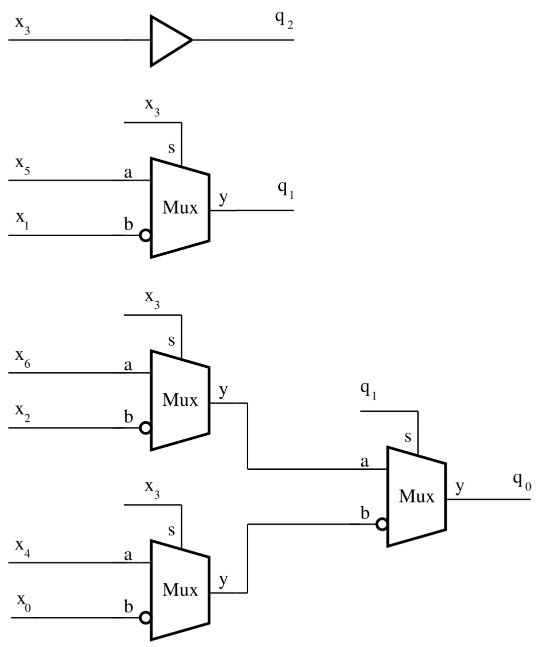


Figure 2.13: 3 Bit Mux-Based Encoder Block Diagram

Fat-Tree encoder is also a simplified approach used in TDC, where NOR gates are used to generate an intermediate code known as a one-hot code that contains no more than one logical one. For example, if the thermometer code of a TDC is (1110000), the one-hot code will be (000100), where the position of the logical one will be converter using NOR and NAND gates to represent the binary word which is (0011) in this case. One advantage of using Fat-Tree encoder is Bubble-Error correction but not to the level of Wallace-Tree [28].

Table 2.1 shows a detailed compression between different encoders architecture conducted by [28]. From the table, it can be seen that even if some encoders offer error correction to the output, using such a circuit will increase the size, power consummation while decreasing the speed of a TDC.

Table 2.1: Comparative Analysis of 5-bits Encoder Architectures

Architecture	**Number of transistors**	**Average dynamic current current, μA**	**Maximum delay time, ps**
Wallace-Tree encoder	624	585	1276
Mux-Based encoder	114	261	817
Fat-Tree encoder	392	286	471

2.5 Motivation and Selection

The previous sections sum up the available architectures for TDC today. The selection methodology is based on the nature of this work, which focuses on the higher-level implementation and investigates the existed TDC architectures. Furthermore, experimenting with different ideas to reduce the overall complexity and improve the performance of TDC. From the previous, the most applicable architecture for this work is Vernier Delay Line (VDL) TDC. The VDL is a digital-based TDC with high resolution, usually in 10th ps and +/-1 LSB DNL [29]. Moreover, being a digital-based decreases development time, power consumption, and circuit area. Nevertheless, VDL TDC uses complex components such as flip-flops and encoders, which reduce its efficiency in measuring large time intervals.

Chapter 3

Design and Simulation Methodologies

The ongoing investigation in this work suggests in order to reduce the complexity of VDL TDC, it is desirable to investigate further down from the higher-level implementation of the TDC. This implies to explore various delay segments at transistor level and find a way to reduce the overhead complexity added by the delay segments, encoders, and calibration circuits. Specifications for the TDC are shown in table 3.1; these specifications are chosen arbitrarily to have a start point for the design process and guide the workflow of the project. The design and simulations for TDC at the transistor level are performed using a commercially available 28*nm* Fully Depleted Silicon on Insulator (FDSOI) CMOS process, with 0.9V supply voltage.

Table 3.1: TDC Initial Specification

Property	**Goal**
Architecture	VDL
Resolution (LSB)	10ps
Conversion Speed	100MS/s
Dynamic Range	630
Number of Bits	6
DNL/INL	+/- 1 LSB
Power Supply	0.9 V
Area & Power	minimum

3.1 Delay Segment

Most implementations of VDL TDC use two delay cells and a D-flip flop to sample the time difference as shown in figure 3.1. In general, all flip-flops have delay, setup, and hold times known as timing parameters, [30] and [31] define these parameters as:

- **Delay Time:** (i.e. propagation time); the required time for a signal to propagate from the input (D) to the output (q) when a leading edge of a clock triggers the (clk) terminal.
- **Setup Time:** the required time for a signal to be stable at the input (D) **before** a leading edge of a clock triggers the (clk) terminal.
- **Hold Time:** the required time for a signal to be stable at the input (D) **after** a leading edge of a clock triggers the (clk) terminal.

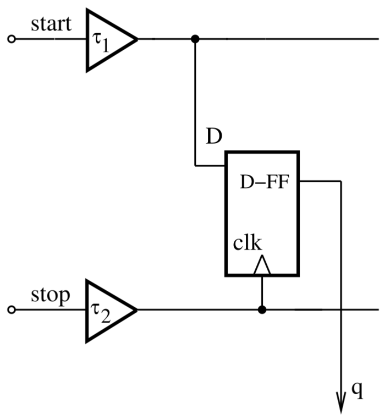


Figure 3.1: Typical Vernier TDC Segment

Using a D-flip-flop in the delay segment can cause drawbacks to TDC, such as increasing area and power consumption while decreasing TDC speed. Nevertheless, violating setup and hold times will make the flip-flop output to be in meta-stability status, where the output (q) will have an undefined value (X). This can be very problematic for TDC design since setup times for flip-flops are in hundreds of picoseconds. For example, the D-flip-flops reported in [32] have a setup time of 137ps and 197ps, if a 10ps time interval to be measured using a Vernier TDC with 10ps resolution (LSB); which implies that the time difference between the start and stop delay elements should be 10ps. When the time signals (i.e. start and stop) propagate through the TDC, the output of the flip-flop in figure 3.1 will be zero instead of one after the time sampling.

This can be illustrated through the timing diagram shown in figure 3.2. Since the stop signal propagates faster than the start signal, both signals will catch-up at the end of the time measurement which can cause an error in the time reading as in this case or causing the flip-flop to enter a meta-stability status if the start signal does not fulfill the setup and hold times. Thus using a standard flip-flop can deteriorate TDC performance. The work done in [19] acknowledged the limitation and utilized a Time-Comparator based on a Set-Reset (SR) latch to resolve this issue. However, even with improved architectures, the setup and hold time of flip-flops will still vary with PVT variations, which adds more varying parameters to consider for TDC.

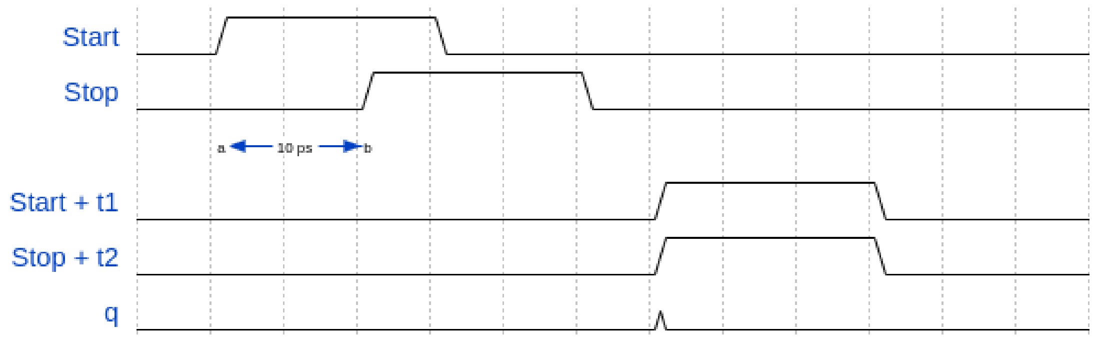


Figure 3.2: Vernier TDC Segment Timing Diagram

3.1.1 Proposed Vernier Segment

The proposed vernier segment is based on work done in [33], which uses a standard 65nm CMOS technology. The proposed segment consists of a latch integrated within the delay lines, as shown in figure 3.3. The delay latch is modeled as a zero delay multiplexer and a delay cell with the start signal directly connected to the input (a). When the select terminal (s) is low, the latch is transparent (output y = a), and when (s) is high, the latch will hold its output value.

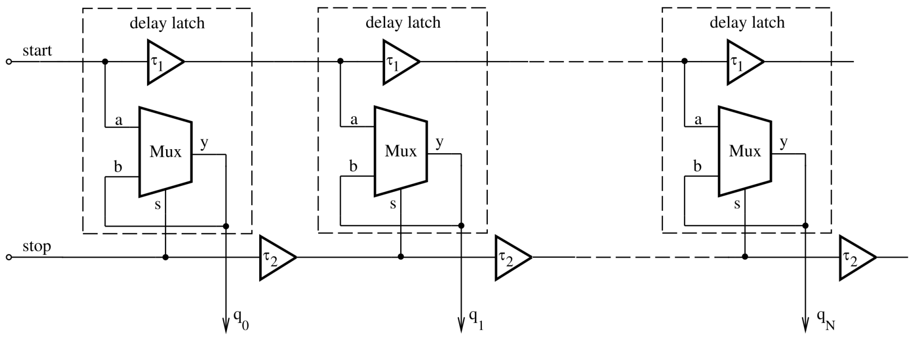


Figure 3.3: Block Diagram for the Proposed VDL TDC

Assuming ($\tau_1 > \tau_2$), in the begin of a time measurement cycle, both signals (start and stop) are zeros, and all delay latches are transparent with zero outputs when the start signal begins to propagate through the delay latch, it will increase the thermometer code (q). The stop signal begins to propagate after time ΔT, catching up with the start signal and setting the delay latches into their hold status. At the delay stage where the stop signal passes the start signal, it will set that delay latch to non-transparent status, holding zero output value, thus holding the start signal and finishing the time measurement. Therefore, the output thermometer code (q) is linearly dependent on the time difference of ΔT (linear thermometer code versus delay).

The integration of the latch within the delay cells eliminates the metastability issue, since output of latch and output of start delay cell will be one at the same time. Recall the previous example, where ΔT is 10ps, the TDC has an LSB of 10ps, and that both delay cells have a propagation delay of 210ps and 200ps delay time for (τ_1 and τ_2), respectively. Since the start signal is connected directly to the multiplexer, the output (q_0) will be set to high after 210ps (i.e. setup time), which is within time stop signal propagates through the delay cell as shown in figure 3.4. For the next stages, the stop signal will be ahead of the start signal causing the rest of latches outputs to be zeros, thus terminating the time measurement.

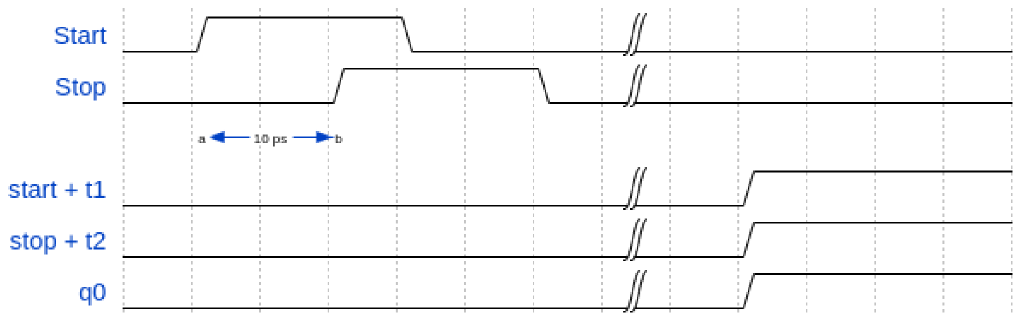


Figure 3.4: Timing Diagram for the Proposed Delay Segment

The implementation of the proposed delay segment at the transistor level and its layout are shown in figures 3.5 and 3.6, respectively.

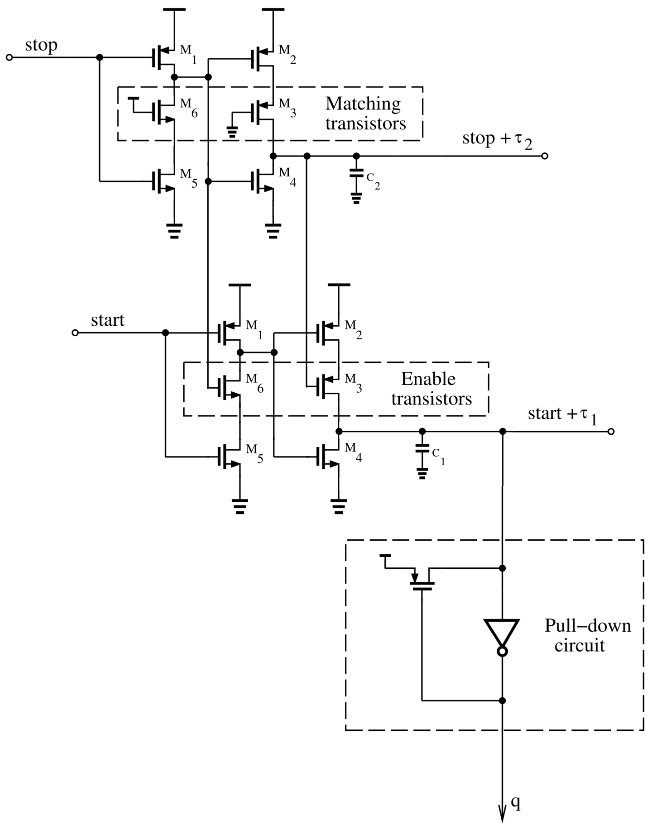


Figure 3.5: Proposed Delay Segment Schematic

The delay cell is implemented by cascading two dynamic inverters plus connecting a Metal Oxide Metal (MOM) capacitor to the output node; the purpose of the capacitor is to control the delay time. The latching mechanism is performed by enable transistors, which are controlled by the delayed stop signal. When stop signal is zero, gate voltages for the enable transistors (M_6 and M_3) are logical high and logical low, respectively. Hence, making the start delay latch works as an ordinary non-inverting delay element. After stop signal becomes high and propagates through the delay cell, gate voltages for the enable transistors in the delay latch toggles removing the paths to supply and ground for the inverters. The output of the delay latch (q) becomes a floating node, thus holding its current value. Note that the output thermometer code (q) in the proposed delay segment is inverted, hence the pull-down circuit. Nevertheless, this can be adjusted easily using an inverter.

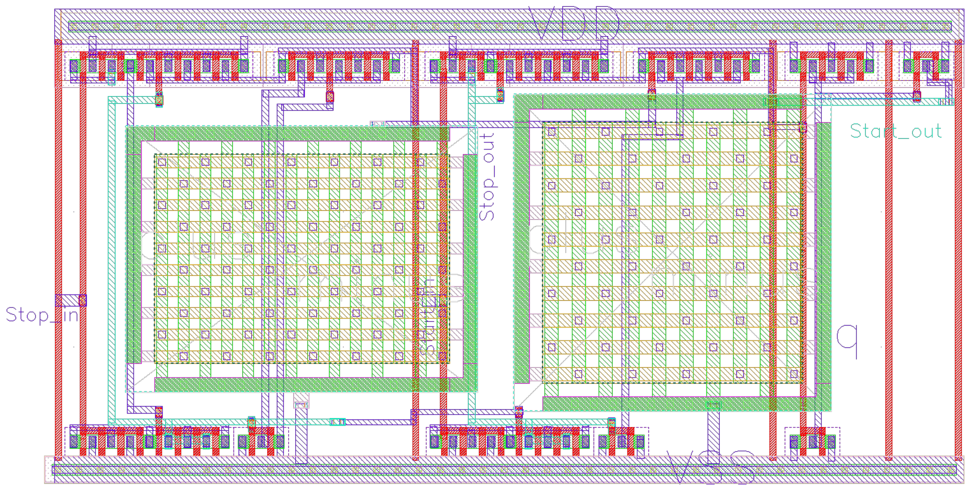


Figure 3.6: Layout for the Proposed Delay Segment with 10ps LSB

The pull-down circuit in figure 3.5 acts as a buffer driving the thermometer-to-binary encoder [33]. As for matching transistors (M_6 and M_3), their purpose is to assure that the two delay line are well-matched, thus compensating for PVT variations [16]. The proposed delay latch requires four transistors (two identical sets of M_6 and M_3) in delay cells, to perform the latching mechanism of the thermometer output code, which is 18 transistors less than the D-flip-flop used in [32], and eight transistors less than the Time Comparator implemented in [19]. This results in a more power-efficient and smaller size TDC [33].

Transistors sizes for the proposed delay segment are summarized in table 3.2. Initially, they are taken from [33] and modified to match the differences in technologies used. Throughout the designing process, sizes is increased to reduce the effect of process variations. These increments are based on, the square root of the transistor area (i.e. $A = W * L$) is inversely proportional to the process variations [34], and personal observation of various simulation runs. The length (L) of all transistors is set to $45nm$ (1.5 times the minimum length) to reduce the leakage current [35], which necessary for this implementation since the output of the delay latch (q) becomes a floating node and must maintain its value. Nevertheless, these increments will create a trade-off between process variations and cell leakage against the area, power, and speed [34] [35]. The values for capacitors (C_1 and C_2) are set to achieve the required resolution (LSB) of 10ps.

Table 3.2: Delay Segment Component Values and Sizes

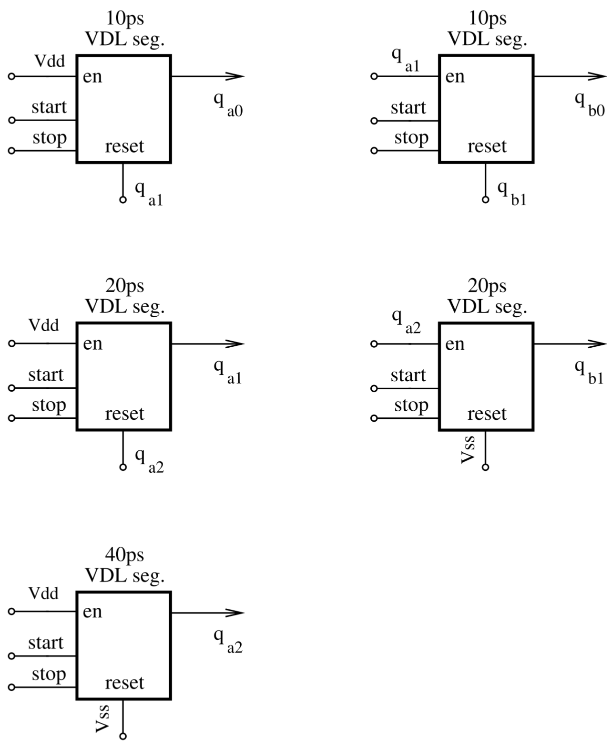
Element	**Type**	**W (μm)**
M_1	PMOS	0.3
M_2, M_3	PMOS	0.6
M_4	NMOS	0.2
M_5, M_6	NMOS	0.4
C_1	Capacitor	4.4 fF
C_2	Capacitor	4.1 fF

3.2 Less Complex Approach of Encoding

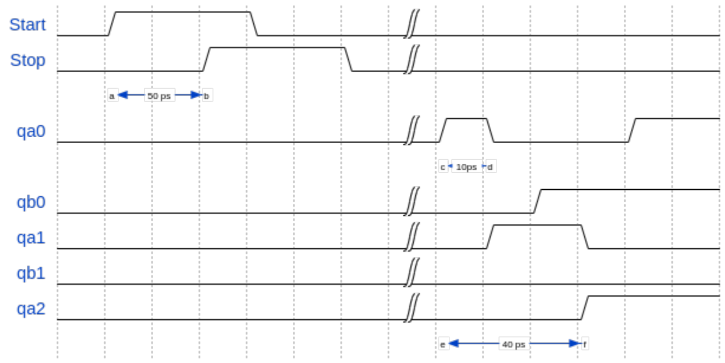
An essential building block for any VDL TDC is the thermometer to binary encoder, but as mentioned earlier, these encoders often add an overhead complexity and increase size and power consumption. The authors of [33] reported that the multiplexer based encoder used in their TDC accounts for approximately 50% and 60% of the total area and the dynamic power, respectively. These figures raise an important question; is there any way to design a more efficient encoder? Or is this block necessary for TDCs? Can it be replaced by a smarter design?

The original need for the encoder block in a TDC comes from the usage of delay lines with sampling latches. These lines produce the output as a thermometer code with the weight of the resolution (LSB) is represented for every logical one in the code, not as a binary-weighted digital code. In other words, if a TDC has an LSB of 10ps and the measured time difference (ΔT) is 320ps, by using a thermometer code, the output will be a 32 bits code (111.....111) since every bit represents a 10ps measurement. Whereas, if a binary-weighted code is used, it will only require 6 bits code (100000) to represents the same measurement. From that, it is evident that by modifying the architecture of the VDL TDC, one can overcome this issue. Three main architectures are implemented using Verilog-A, where only one architecture proves prudent to be implemented at the transistor level.

Figure 3.7 (a) shows a high-level implementation of an early attempt for solving the encoder issue. It consists of a modified VDL segment with different resolutions running in parallel. The segments have an enable pin (en), and a reset pin to control the start signal, a simple delay cell for the stop signal, and a latch for sampling the measured time. The configuration in figure 3.7 (a) gives a 3 bits TDC with 10ps LSB using only 5 Vernier segments.



(a) TDC Architecture Block Diagram



(b) 50ps ΔT Measurement Timing diagram

Figure 3.7: Early Attempt Encoder Solution

The simplicity of this design comes from the ability to control the start signal propagation (enabling and resetting), while the stop signal propagates. A 50ps time difference (ΔT) measurement timing diagram is shown in figure 3.7 (b), where the output of the q_{a0} segment (i.e. 10ps LSB), is controlled by the output of q_{a1} segment (i.e. 20ps LSB) and so on. When q_{a1} becomes high (which indicates a 20ps measurements), it will enable the q_{b0} segment to measure the last 10ps and reset q_{a0} segment. The output code can be extracted as a binary code by using simple logic gates. However, it is found out that designing such a Vernier delay segment to produce an accurate enable and reset signal is complicated at the transistor level. Furthermore, since TDC has 10ps resolution, the Vernier segment must be capable of resetting within this time to be able to measure the next 10ps.

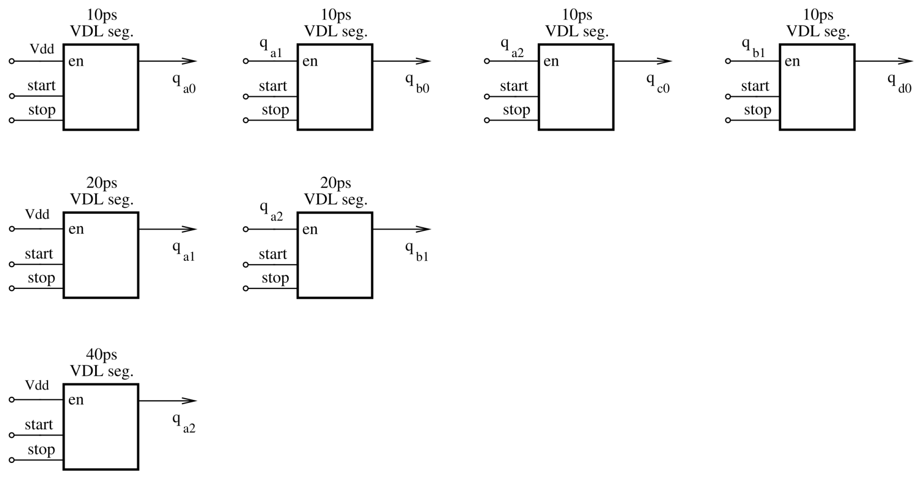


Figure 3.8: Modification for the Encoder Solution

Since matching between the reset and enable signal is difficult, a modification to the previous implementation is done by removing the reset pin for segments and adding two 10ps segments, as shown in figure 3.8. These modifications result in a simpler design for the Vernier segment and remove the requirement for resetting within 10ps. The implementation works in similar way as figure 3.7 (a), but with a difference that the 40ps segment (q_{a2}) will enable q_{b1} and q_{c0}. Another way to look at it, that the 40ps segment (q_{a0}) will start the fine measurement using the q_{b1}, q_{c0} and q_{d0}. Nevertheless, the main problem for this implementation on the transistor level is relying on the enable signal to start the next measurement. For example, if the time difference is 30ps, q_{a1} will become high indicating a 20ps measurement which will enable the 10ps (q_{b0}) segment. However, q_{a1} will not be logic one until the start signal propagates through it. This propagation delay will result in the start and stop losses of the time difference between them (both of the signals will be high at the time the enable signal comes), thus losing time information.

3.2.1 Proposed method for encoding

Figure 3.9 shows the proposed TDC implementation, which solves all of the previous issues by implementing multiple (parallel) VDLs with different resolutions measuring the time difference in parallel. The resolutions for the lines have to be in the power of $[(2^n) * LSB]$; this will simplify the encoding producer for thermometer codes.

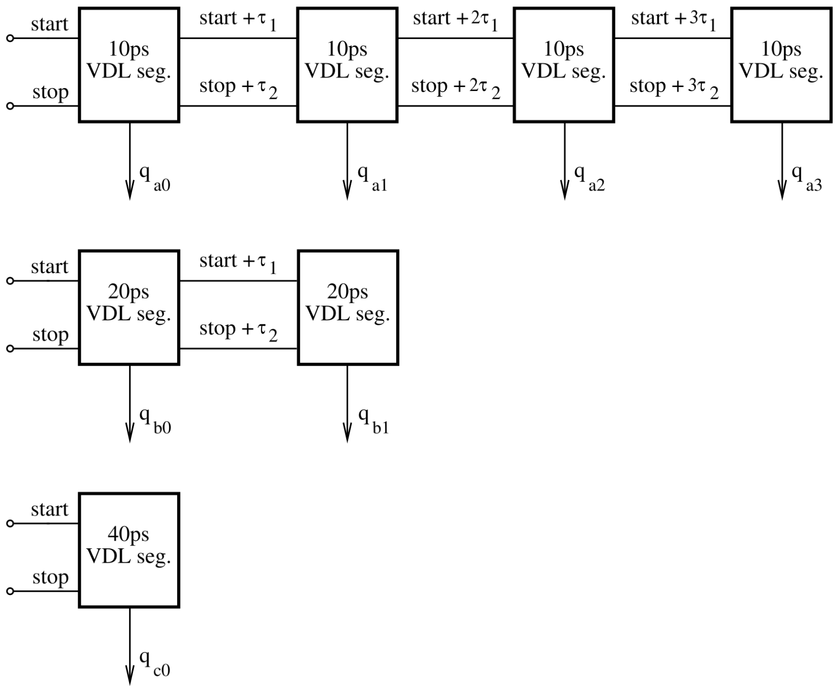


Figure 3.9: Proposed TDC Implementation

The TDC in figure 3.9 has a 3 bits binary output code, 40ps dynamic range with 10ps LSB. The 3 bits output can be encoded from 7 thermometer bits using standard logic gates, as demonstrated in the boolean equations 3.1, where n and q represent the binary and thermometer codes, respectively.

$$
\begin{aligned}
n_0 &= \bar{n}_2.\bar{n}_1.q_{a0} \quad + \quad \bar{n}_2.n_1.q_{a2} \\
n_1 &= \bar{n}_2.q_{b0}.q_{a1} \\
n_2 &= q_{c0}.q_{b1} \quad + \quad q_{b1}.q_{a3} \quad + \quad q_{c0}.q_{a3}
\end{aligned}
\tag{3.1}
$$

An example of measuring a 30ps time difference (ΔT) is shown in the timing diagram shown in figure 3.10, the parallel VDLs independently generate thermometer codes with different resolutions. The thermometer code from the first delay line ($q_{a0} - q_{a3}$) is 1110, since it has an LSB of 10ps. As for the second delay line ($q_{b0} - q_{b1}$) the thermometer code will be 10, correspond to its 20ps LSB. The last line (q_{c0}) thermometer code will be 0 since (ΔT) is less than its resolution (i.e. 40ps).

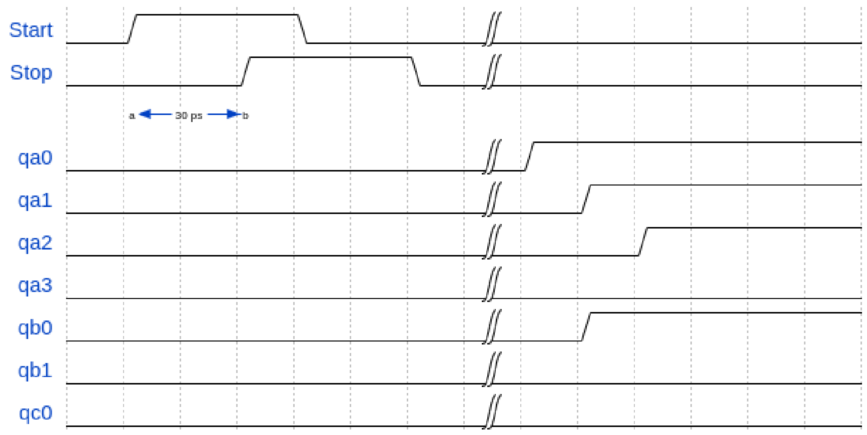


Figure 3.10: Timing Diagram of the Proposed TDC Implementation

Substituting the values of delay lines from figure 3.10 into equations 3.1 will produce the output word as a binary-weighted code, which is in this example (011). Theoretically, adding more lines to compute the final binary output should add more robustness to TDC output since every delay line will experience different PVT variations, as the binary output code for TDC is a combination of all lines. Moreover, the usage of simple logic gates can offer a new method for encoding the thermometer code, thus reducing the complexity of TDC encoders. To the author's best knowledge, the proposed parallel Vernier delay lines architecture has not been implemented in previous papers nor patents.

The proposed TDC architecture uses the delay segment shown in figure 3.5. The values for the capacitors C_1 used to achieve the different resolutions is shown in table A.1 in Appendix A. Layout for different resolution segments are shown in Appendix A, the only difference between figures is the capacitor size.

3.3 Test and Verification

Simulations performed in this work are divided into main sections; analog simulations at the transistor level and digital simulations in SystemVerilog. The analog simulations are performed to extract the behavior of the proposed delay cells before it is passed to the digital workspace to implemented the TDC.

3.3.1 Simulation Assumptions

To simplify the designed procedure, start and stop signals are assumed to be independent and jitter-free. Furthermore, the rise time is 20ps for both signals, and the pulse width is long enough for transistors to pass values in the TDC.

3.3.2 Corners and Temperature

The proposed implementation is designed and simulated at 27° c, typical process corner. Other corners and temperature simulations could be considered to analyze their impact on TDC performance.

3.3.3 Layout

The layout for the Vernier-Delay segment is shown in 3.6. Multiple measures are performed to have a uniform layout such as using the number of fingers with same widths (W) for all transistors (NMOS and PMOS), and having the same lengths (L) for all transistors as well. Unfortunately, mismatch and process variations are taken from the transistor level only (pre-layout), since it is not possible to extract any post-layout results due to technical limitations on the available workspace.

3.3.4 The Proposed Delay Segment

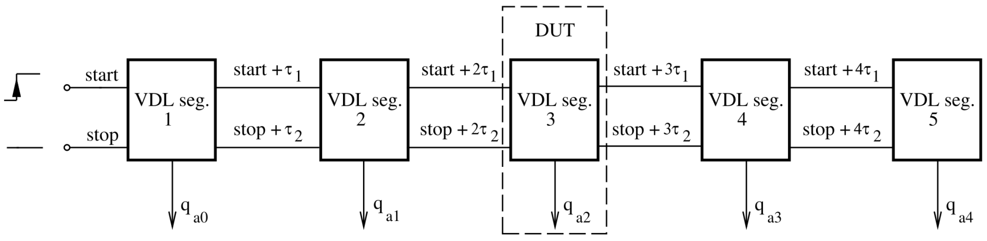


Figure 3.11: Proposed TDC Segment TestBench

For the proposed delay segment (fig.3.5), analog simulations are performed first to find the propagation delay of each line and the resolution (LSB) for each segment in the TDC. The setup in figure 3.11 shows that delay times for start and stop signals are extracted from the Design Under Test (DUT) segment (i.e. seg.3). The usage of dummies gives more realistic behavior for signals and takes into account loading effects. The extraction of the delay time for the start signal is done by setting the stop signal to zero and measuring the time between when the input signal becomes 0.8V and when the output signal becomes 0.8 since only positive edges of signals are considered. Same procedure for measuring the delay time for the stop signal, expect to set the start signal to zero.

Afterward, a Monte Carlo simulation is performed to find the effects of mismatch and process variations and extract a realistic mean value and standard deviation (sigma) for delay times. The proposed architecture in this work implies different delay lines with different resolutions (i.e. 10ps, 20ps, 40ps, 80ps, 160ps, 320ps), this is achieved by increasing the size of the capacitor for the start delay line (c_1) and implement previous procedures to find the delays, resolutions, and variations (sigma). Finally, delay times mean values and variations (sigma) are substituted in the SystemVerilog model along with logic gates, as shown in figure 3.12 (this figure is just a 3 bits demonstration of what the actual 6 bits TDC will look like).

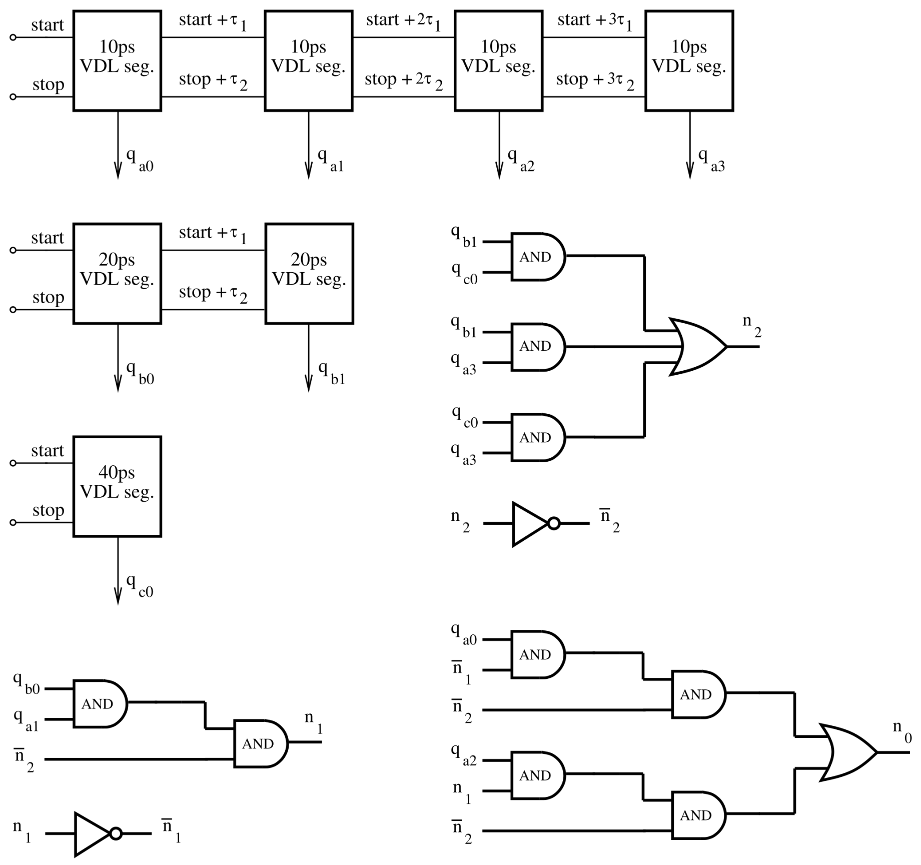


Figure 3.12: Verilog TestBench of the Proposed TDC

The top-level view of the SystemVerilog model is shown in figure 3.13. The variations in the delay time are generated in the testbench using "$dist\_normal(seed, mean, standard\_deviation)$" function, which produces random numbers that follow Gaussian distribution with a standard deviation (sigma); these variations are sent to the Vernier segment in the DUT. The segment is listed in C.1, Appendix C, which has a delay function with an ideal flip-flop to sample the time (since the proposed segment does not has timing constraints). The TDC core listed in C.2, Appendix C, which defines the corresponding delay lines by instantiating multiple segments with different resolutions.

All of the previous, ensures the SystemVerilog model will have the transistor level behavior for delay cells in the TDC implemented in SystemVerilog. The logic gates encode the time measurement (n) as binary-weighted from the thermometer code (q) to generate the quantization curve for TDC in the DUT. The non-linearities curves (DNL and INL) are obtained by generating a fixed time sequence to produce the corresponded time measurements. These measurements are subtracted from ideal measurements (ideal quantization curve) to plot DNL curve and integrated to obtain the INL curve. Codes of the SystemVerilog models and testbench is shown in Appendix C.

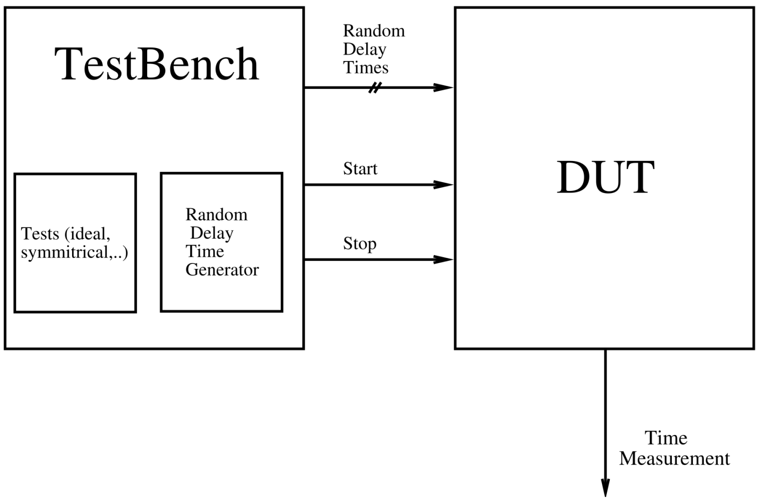


Figure 3.13: SystemVerilog Top Level View

Chapter 4

Simulation Results

The achieved simulation results and methods from both transistor model and SystemVerilog model of the designed TDC are presented in this chapter. The proof of concept for the simplified TDC designs is performed using Verilog-A models, but no simulation results from these designs are reported, since the models reported timing diagrams similar to the illustration diagrams in Chapter 3. Results of the final designed TDC are represented in this chapter, where all simulations at the transistor models are performed in typical process corner at 27° C, unless it is mentioned otherwise.

The transistor-level of the proposed delay segment is simulated several times in order to verify a systematic behavior and meet up with the required specifications. Simulation of circuits is performed using *Spectre Circuit* Simulator in *Cadence Virtuoso*. Tests are performed in ADE L/XL using Transient Analysis with moderate accuracy, and calculations are performed by the Visualization & Analysis XL calculator. Monte Carlo Analysis modeled the delay variations of the segments with 300 points for each delay segment with different resolutions (LSB). These variations are then implemented and simulated in a SystemVerilog model to obtain the output curve for TDC using *Questa* Simulator in *MentorGraphics*. Finally, all of these results are record and presented using *Matlab*.

4.1 Proposed TDC Implementation

The implementation of the proposed TDC in this work is shown in figure 4.1, which has 6 VDLs with different resolutions (LSB). The proposed TDC has a 6 bits (n) output binary-weighted code (after the encoding circuit) with 10ps resolution (LSB). The dynamic range for the VDL TDCs is $(2^n - 1) * LSB$, which gives a 630 ps for this configuration.

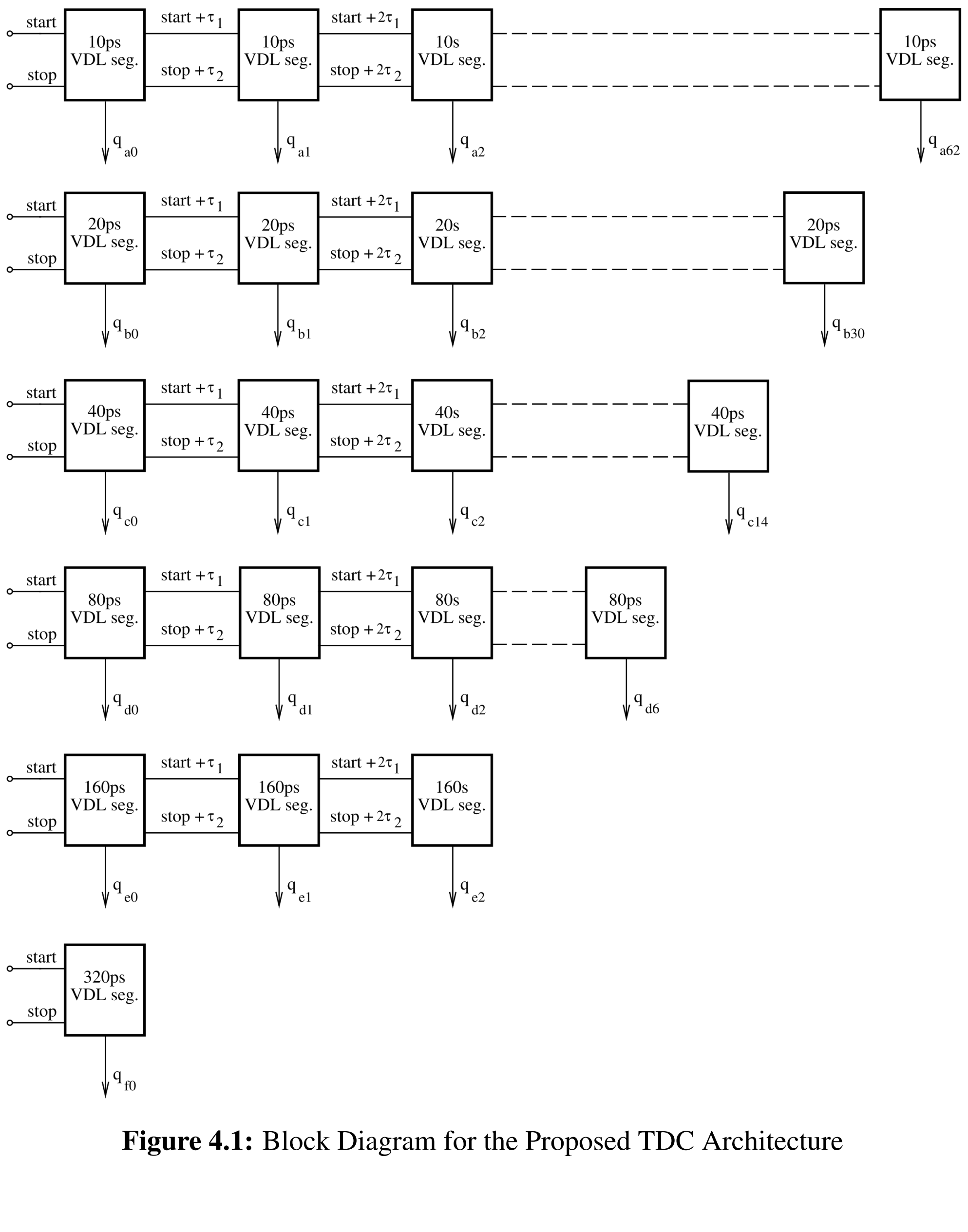


Figure 4.1: Block Diagram for the Proposed TDC Architecture

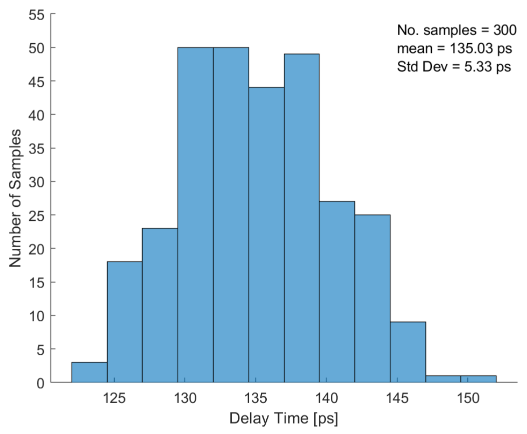
Table 4.1 illustrates the total number of delay segments along with their resolutions (LSB) and their corresponded thermometer output bits. Propagation delay times for the start cells are illustrated as well, where the propagation delay for stop delay cells is $125ps$ for the entire design.

Table 4.1: Delay Segments Parameters

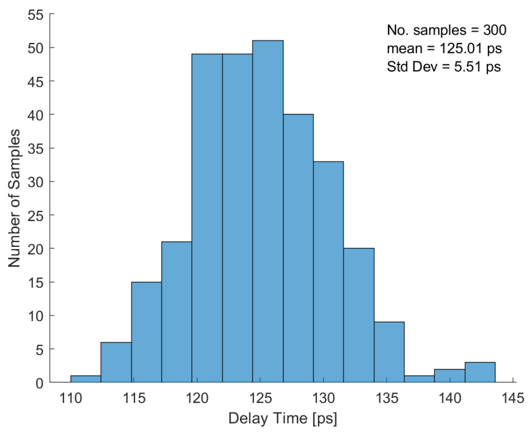
Delay Line	**LSB (ps)**	**Start Signal Delay (ps)**	**Thermometer Output**	**Number of Delay Segments**
1	10	135	63 bits, $a_0 \rightarrow a_{63}$	67
2	20	145	31 bits, $b_0 \rightarrow b_{30}$	35
3	40	165	15 bits, $c_0 \rightarrow c_{14}$	19
4	80	205	7 bits, $d_0 \rightarrow d_6$	11
5	160	285	3 bits, $e_0 \rightarrow e_2$	7
6	320	445	1 bit, f_0	5

4.2 Process and Mismatch Variations for Delay Segments

Monte Carlo simulation is performed on each delay cell for every delay line resolution. Figure 4.2 predicts the how the start and stop delay cells (i.e. τ_1 and τ_2) will vary. Both delay cells have a standard deviation (σ) less than 5% from the nominal value of the propagation delay.



(**a**) Start Delay cell Histogram



(**b**) Stop Delay cell Histogram

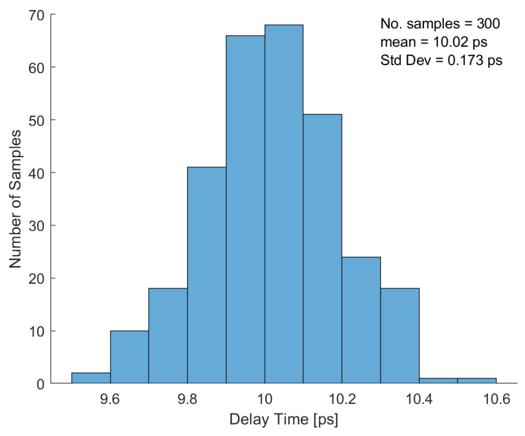
Figure 4.2: Monte Carlo Histogram for 10ps Delay Segment

Table 4.2 shows variations for each each delay line based on Monte Carlo histograms in Appendix B. The Coefficient of Variation (CV) is the ratio of the standard deviation to the mean value, which represents the percentage of variations a data set has [36]. For example, delay segment one will have lower probability (less variations) to deviate from its mean value than segment number 6. Since the stop delay cell in each segment is not modified to generate different resolutions, it has a mean value of $125.01ps$, a standard deviation(σ) $5.51ps$, and a CV of 4.4%, for the entire TDC.

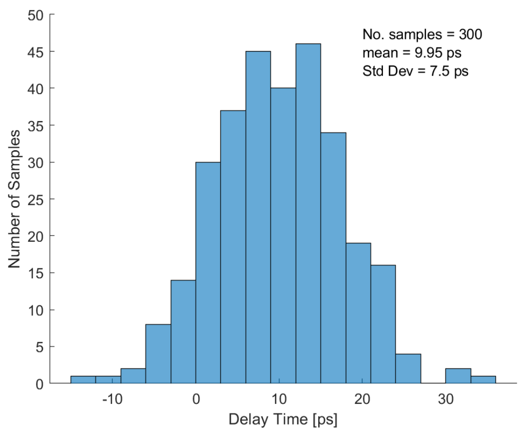
Table 4.2: Delay Segments Variations

Delay Segment	**Start Nominal Delay(ps)**	**Start Mean Delay(ps)**	**Standard Deviation σ (ps)**	**Coefficient of Variation**
1	135	135.03	5.33	3.9%
2	145	145.50	5.80	4.0%
3	165	165.10	6.79	4.1%
4	205	205.46	8.79	4.3%
5	285	284.97	12.8	4.5%
6	445	446.54	21.0	4.7%

Figure 4.3 (a) predicts the variations of the resolution (LSB) for the 10ps segment (LSB = $\tau_1 - \tau_2$) caused by process variations and transistors mismatch. The resolution has a mean value of $10.02ps$ and standard deviation (σ) of $0.173ps$, which a CV less than 2% from the LSB mean value.



(a) Same seeding for the random function



(b) Different seeding for the random function

Figure 4.3: Histogram for Simulated 10ps LSB Segment

Variation values are based on sets of data generated from the SystemVerilog Testbench using the (*dist_normal*) function, which uses a seed to generate different random numbers. The usage of the same seed number simulates the condition where both delay cells are well matched, which can be seen in figure 4.3 (a). However, if a different seed number is used for both cells, the standard deviation (σ) of the resolution is 7.50 with a CV of 75.3%, which indicates that both cells are not matched, as shown in figure 4.3 (b).

Tables 4.3 and 4.4 shows variation differences for the rest of delay segments with fixed and different seeds, respectively. These numbers are based on simulation histograms in Appendix B.

Table 4.3: Segments Resolution Variations (with fixed seed)

Delay Segment	**Nominal LSB Value(ps)**	**Mean LSB Value(ps)**	**Standard Deviation σ (ps)**	**Coefficient of Variation**
1	10	10.02	0.17	1.7%
2	20	20.49	0.26	1.2%
3	40	40.09	1.18	2.9%
4	80	80.45	3.03	3.7%
5	160	159.9	6.73	4.2%
6	320	321.5	14.3	4.4%

Table 4.4: Segments Resolution Variations (with different seed)

Delay Segment	**Nominal LSB Value(ps)**	**Mean LSB Value(ps)**	**Standard Deviation σ (ps)**	**Coefficient of Variation**
1	10	9.95	7.50	75.3%
2	20	20.42	7.78	38.1%
3	40	40.02	8.41	21.0%
4	80	80.38	9.84	12.2%
5	160	159.9	13.0	8.13%
6	320	321.5	20.1	6.25%

4.3 Encoding Circuit

The implementation of the encoding circuit is done using SystemVerilog code listed in C.2, Appendix C, which is based on Boolean equations similar to the equations in Chapter 3. The circuit takes the thermometer outputs from different delay lines and generates a binary code using AND, OR, and NOT gates. Table 4.5 shows details for the standard logic gates used in the encoding circuit, where the number in front of the gate name represents number of inputs. The delay and power are measured on pre-layout standalone cells with no load effects.

The circuit uses 179 gates with 1564 transistors, with a maximum delay time of 433ps, which is measured through the critical path of 22 gates depth. These numbers are an approximation, derived directly from the Boolean Equations since no actual synthesizer data are available to publish. Furthermore, since the output of the proposed delay segment in figure 3.5 is inverted, an additional 120 inverters are required to correct the thermometer code before passing it to the encoding circuit. These inverters will not affect the total area, power, or propagation time for the encoding circuit and are not accounted in table 4.5, since their effects are insignificant and any encoding circuit will share them.

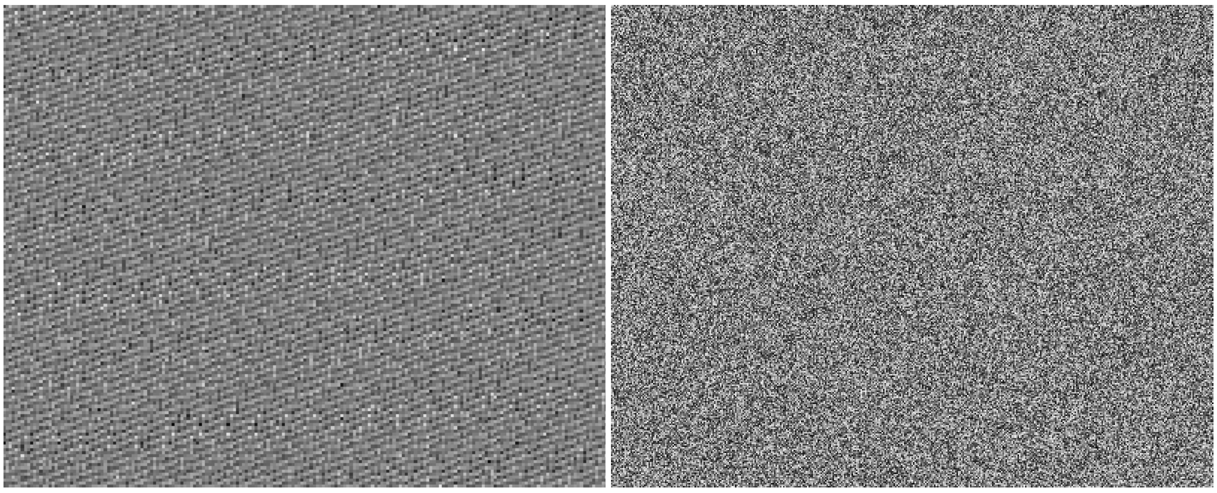
Table 4.5: Proposed Encoding Circuit Gate Count

Gate	**Propagation Delay (ps)**	**Average Power (nW)**	**Gates Count**	**Transistors Count**
Inverter	5	0.01	15	30
2_AND	19	0.19	43	258
3_AND	29	0.20	4	32
4_AND	42	0.21	8	80
6_AND	74	0.22	48	672
2_OR	12	0.23	25	150
3_OR	13	0.25	11	88
4_OR	14	0.27	23	230
5_OR	15	0.29	2	24
		Total Count	179	1564

4.4 Nonlinearities (DNL and INL)

The nonlinearities are measured by implementing various delay cells using the ($dist\_normal$) function, recalling from previous subsection the effects of seed numbers on the resolution of delay segments, which can change nonlinearities for TDC. Initially, the seeding for the function is performed in a $for-loop$ with different but consecutive numbers, which deteriorated nonlinearities and added a constant offset to the DNL and INL over various runs. A further look at the ($dist\_normal$) function showed that the output is following a strangely skewed pattern for the random numbers generated.

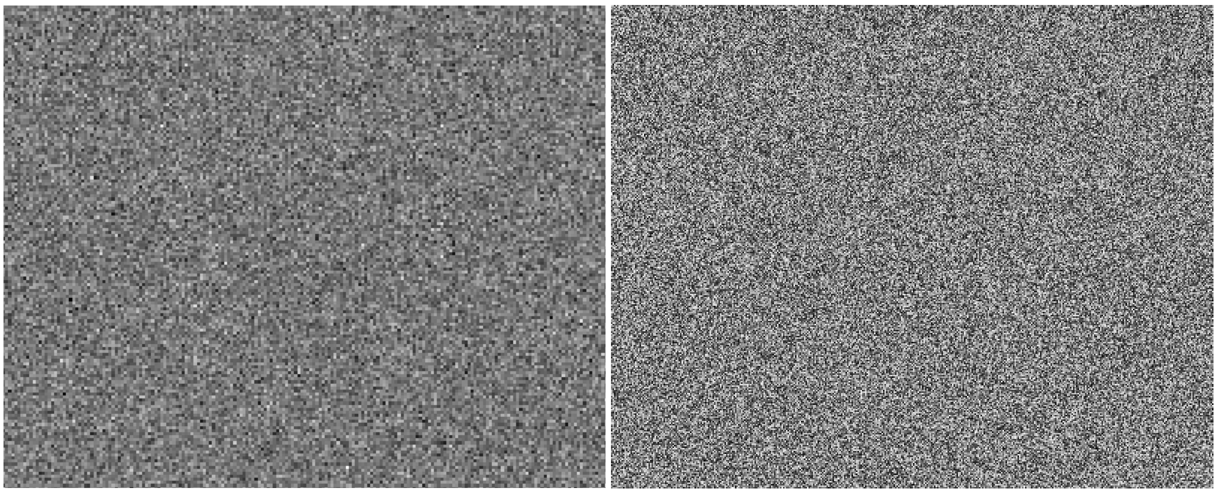
This pattern is confirmed by generating 65536 numbers and representing each number as a pixel with brightness ranging between black (minimum value) and white (maximum value) using $matlab$ function $imshow$. The generated random numbers are displayed as a grayscale image, which can indicate the quality of the random number generation. Figure 4.4 (a) shows grayscale image obtain from numbers generated by ($dist\_normal$) function, where a strange pattern (stripes and waves) can be seen versus the ideal grayscale image for random numbers in figure 4.4 (b).



(a) *dist_normal* Function with correlated seed **(b)** Ideal Random Function

Figure 4.4: Grayscale Images for Random Functions

The elimination of the pattern is done by seeding the *dist_normal* function with uncorrelated random numbers, which produce an ideal behavior of the function, as seen in figure 4.5.



(a) *dist_normal* Function with Uncorrelated Seed **(b)** Ideal Random Function

Figure 4.5: Grayscale Images for Random Functions

The DNL and INL are then extracted from SystemVerilog TestBench using uncorrelated seeding for both the proposed TDC and a conventional VDL with Wallace-Tree encoder. The Wallace encoder is presented for comparison purposes, and it is connected to the thermometer output code of the first delay line (i.e. 10ps line) since this line can work as a stand-alone TDC with the same specifications as the proposed TDC. Three scenarios are taken into consideration, a **Well-Matched** delay cells scenario (same uncorrelated seeds for both delay cells) for the proposed TDC and Wallace TDC. **Un-Matched** delay cells scenario (different uncorrelated seeds for both delay cells) for the proposed TDC and Wallace TDC. **Small Variations** scenario (almost ideal) for the proposed TDC, where all delay lines have lower variation percentages (mean values for LSBs are closer to nominal values).

Nonlinearities are measured by ramping an input time sequence from 0 to 639ps with 1ps time step. The small timestep gives precise, realistic, and constant DNL and INL figures over multiple runs. After removing the offset and gain errors, the DNL is measured using equation 4.1, which expressed DNL in terms of LSB. equation 4.1 measure DNL from time transition points (T_{n+1} and T_n) of the output code in the quantization curve through the entire time sequence from $0 \leq n \leq 639$.

$$DNL = \frac{T_{n+1} - T_n}{LSB} - 1 \tag{4.1}$$

The INL is measured by passing a straight line through the endpoints of TDC quantization curve from zero output code to all ones output code. This method is known as End-Point, which gives precise results for INL. Figure 4.10 shows DNL and INL of the first scenario (well-matched cells) for both the proposed TDC and Wallace tree TDC. It can be noticed that both TDCs have similar variations for the DNL but with lower amplitude for the Wallace TDC, thus causing the proposed TDC to have higher INL than Wallace TDC.

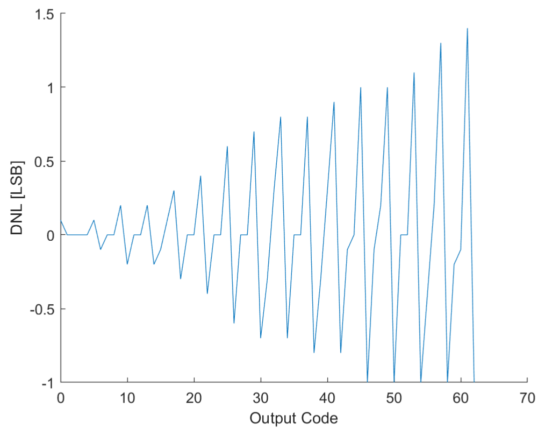


Figure 4.6: Proposed TDC DNL

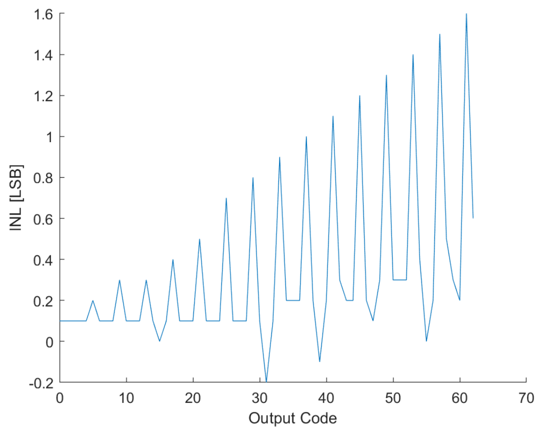


Figure 4.7: Proposed TDC INL

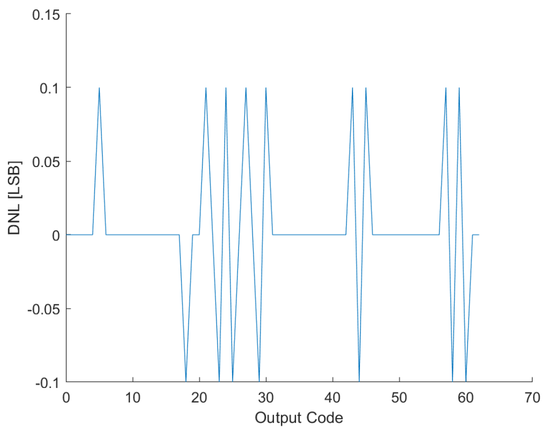


Figure 4.8: Wallace TDC DNL

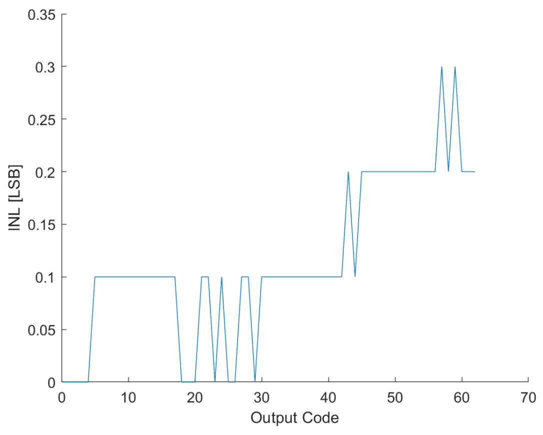


Figure 4.9: Wallace TDC INL

Figure 4.10: First Case Scenario DNL and INL

The second case scenario (un-matched delay cells) is shown in figure 4.15, where it can be seen that both TDCs have a significant variations with high amplitude in their DNL and INL, which shows the effect of high CVs in table 4.4.

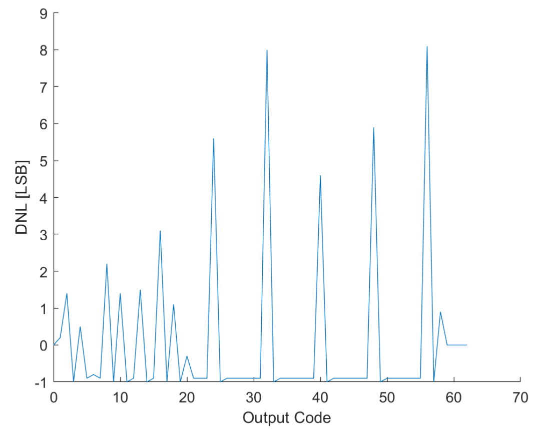


Figure 4.11: Proposed TDC DNL

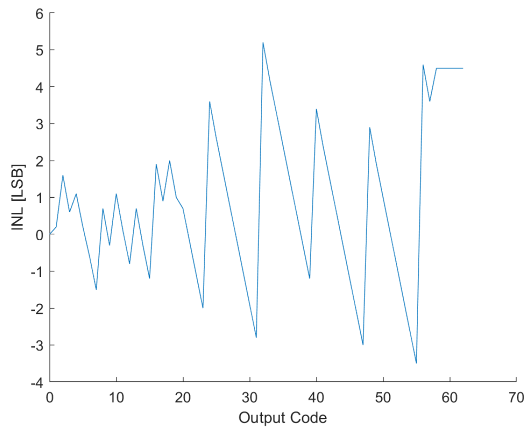


Figure 4.12: Proposed TDC INL

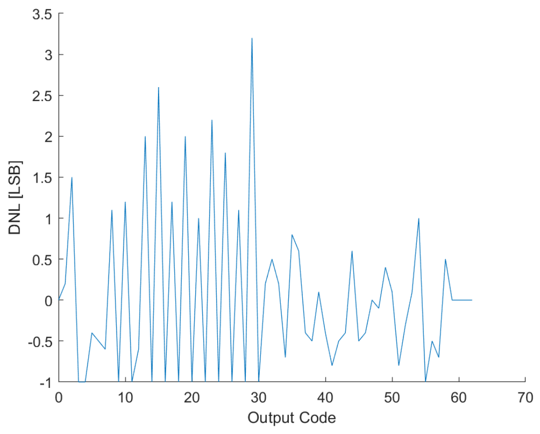


Figure 4.13: Wallace TDC DNL

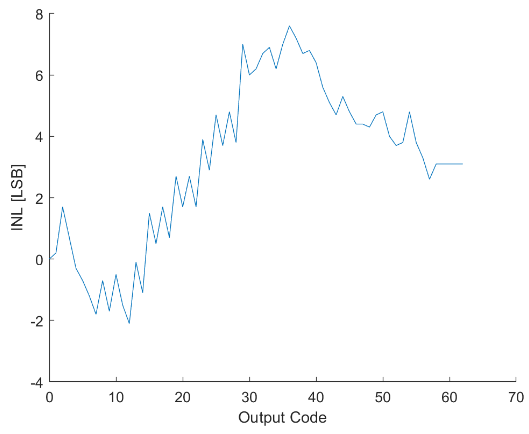
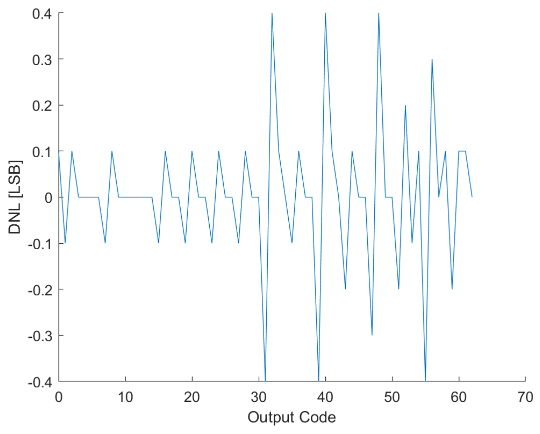


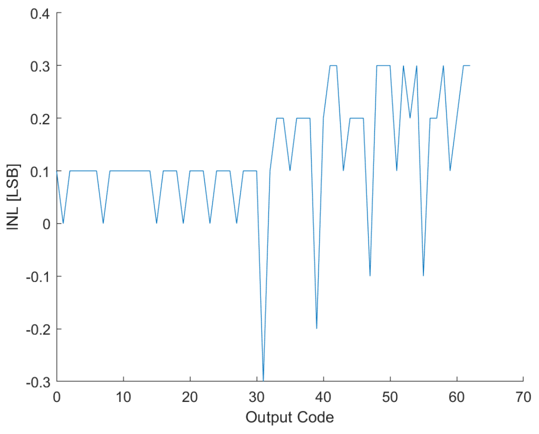
Figure 4.14: Wallace TDC INL

Figure 4.15: Second Case Scenario DNL and INL

The last case scenario is only related to the proposed TDC since assuming lower variations in other delay lines (i.e. 20ps to 320ps lines) will not affect the Wallace TDC because it is connected to the output of the first delay line (i.e. 10ps line). The DNL and INL show that the proposed TDC noise performance got improved and, DNL figure does not have significant LSB values at higher output codes.



(a) Proposed TDC DNL



(b) Proposed TDC INL

Figure 4.16: Third Case Scenario

Table 4.6 lists all case scenarios for the different simulated TDCs with the maximum and minimum nonlinearities.

Table 4.6: Simulated Nonlinearities

Case Scenario	**TDC Architecture**	**DNL (LSB)**	**INL (LSB)**
Well Matched	Proposed	+1.4/-1.0	+1.6/-0.2
	Wallace	+0.1/-0.1	+0.3/0.0
Un-Matched	Proposed	+7.8/-1.0	+5.1/-3.4
	Wallace	+3.4/-1.0	+7.7/-2.1
Small Variations	Proposed	+0.4/-0.4	+0.3/-0.3

A further investigation is conducted to find out what causes the nonlinearities to differ from the almost ideal (well-match lines) and the first case scenario for the proposed TDC. Figure 4.17 shows the actual output code of all six Vernier lines in the proposed TDC versus the ideal output code; it is clear that every line is behaving differently and produce a skewed output. For example, at time 160ps, the ideal output code should be 16 for all lines, but instead, only two lines are correspondent with ideal output (i.e. lines 3 and 5); the other lines are skewed to the right by one measurement.

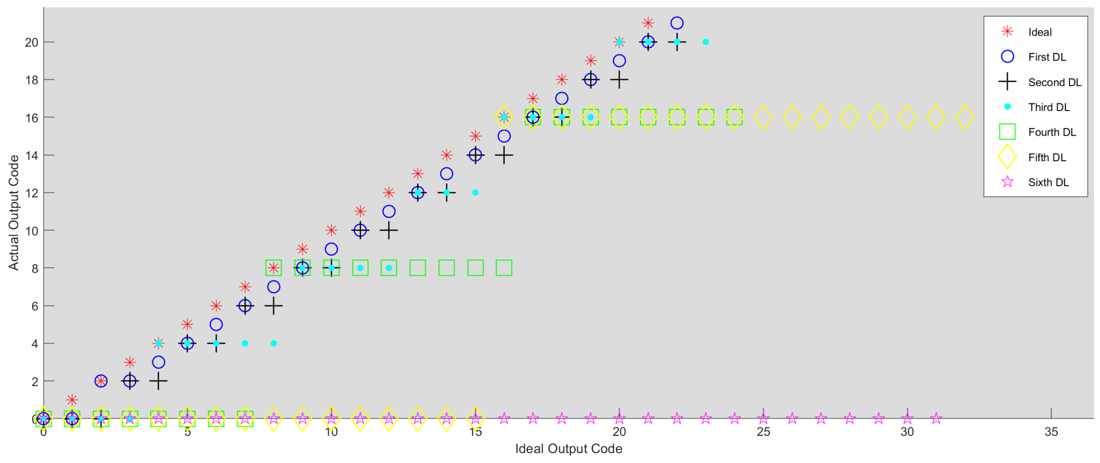


Figure 4.17: Output From All Delay Lines

Table 4.7 shows the effects of INL on the final output code for the TDC through the N_{linear} metric. The highest absolute value for INL is used in equation 2.1 for all case scenarios and simulated architectures.

Table 4.7: Number of Linear Bits (N_{linear})

Case Scenario	TDC Architecture	$\mathbf{N_{linear}}$
Well Matched	Proposed	4.62
	Wallace	5.62
Un-Matched	Proposed	3.34
	Wallace	2.87
Small Variation	Proposed	5.51

4.5 Area

The total area of the TDC is divided between the area of VDLs and encoding circuit. Table 4.8 shows the total area of the Vernier lines, which is extracted from the layout of each delay segment multiplied by the total number of segments.

Table 4.8: Vernier Delay Lines Area

Delay Segment	**Area of Seg. (μm^2)**	**Number of Seg.**	**Total area (μm^2)**
1	23.8	67	1594
2	23.8	35	833
3	23.8	19	452
4	25.2	11	277
5	33.6	7	235
6	48.4	5	242

The area occupied by the encoding circuit is 235 (μm^2), which is an approximation based on the total number of transistors multiplied by the area of a single transistor from a standard cell reported in [37], which use a 28nm CMOS working in the near-threshold region. Combining the two area figure provides an approximation area of 3868 (μm^2), for the proposed TDC.

4.6 TDC Speed

The conversion speed for TDC is $110MS/s$ correspond to the total propagation delay time of the longest Vernier line (i.e. the 10ps line) plus the propagation time of the critical path of the encoding circuit. The 10ps VDL has 67 delay cells for the start signal, where each cell has a 135ps propagation delay. The total propagation time for these cells plus the encoding circuit gives the period of the TDC, which is approximately equal 9.05ns.

4.7 Power Consumption

The total average power of TDC is divided between VDLs and the encoding circuit. The power measurement is performed on the schematic level (pre-layout) of both the VDLs and for the encoding circuit, by measuring the average current and multiplying it by the supply voltage. Table 4.9 shows an average power of $876\mu W$ for the VDLs, calculated by multiplying the power for each segment by the total number of segments in the specific line and adding the total power for each line together.

Table 4.9: Vernier Delay Lines Power Consumption

Delay Segment	**Power of Seg. (μW)**	**Number of Seg.**	**Total Power (μW)**
1	5.8	67	392
2	6.1	35	211
3	6.2	19	118
4	6.5	11	72
5	6.8	7	48
6	7.1	5	35

The average power consumption for the encoding circuit is $0.037\mu W$, which calculated from table 4.5. In a similar approach, by multiplying the total number of each gate with the corresponded power consumption and adding the power consumption of all gates together. Leakage power is also considered, by setting all inputs to zero and observing the current through circuits. The VDLs and the encoding circuit have a $0.017\mu W$ and $0.043\mu W$ leakage power, respectively. Combining all the power figures of VDL and the encoding circuit gives an $877\mu W$ total power consumption for the proposed TDC.

4.8 TDC Survey

The proposed TDC performance metrics are summarized in table 4.10 along with other published Vernier TDCs for compression, as well the case where the Wallace Tree encoder is used in this work. The proposed TDC uses custom-designed (not synthesized) delay segments to construct a parallel VDL to achieve decent linearity, area, and power consumption.

Table 4.10: Performance Summary and Comparison

	[19]	**[38]**	**[33]***	**This Work****	**This Work****
Technology	65nm CMOS	130nm CMOS	65nm CMOS	28nm CMOS	28nm CMOS
Supply(V)	1.2	1.2	1.2	0.9	0.9
Architecture	2-D VGRO	GVRO	VDL	VDL	Parallel VDL
Encoder	NA	NA	Multiplexer	Wallace***	Proposed
Resolution(ps)	4.8	7.3	5.7	10	10
Number of Bits	7	7	7	6	6
Conversion Speed(MS/s)	50	2.4	100	110	110
Dynamic Range(ns)	0.6	9	0.73	0.630	0.630
DNL(LSB)****	1	3.2	1.2	0.1	1.4
INL(LSB)****	3.3	1.2	9	0.3	1.6
N_{linear}^{*****}	4.89	5.86	3.67	5.62	4.62
Area (mm^2)	0.02	0.03	0.004	0.0019	0.0039
Power(mW)	1.7	1.2	1.75	0.393	0.877

* Original Work

** Pre-layout results

*** Connected to first delay line

**** Maximum absolute value

***** $N_{linear} = N - log_2(INL + 1)$

Chapter 5

Discussion

This project aims to investigate different TDC architectures, choose an architecture, and make improvements. The usage of both digital and analog workspaces gave the author more flexibility to experiment with different ideas to improve on existing architectures while having a more realistic behavior for the models. Furthermore, the author gained more knowledge about both world analog and digital, which would be beneficial for future mixed-signal projects. The Vernier Delay Line TDC is chosen because it showed the potential of improving the already existing topology and simplifying it.

The work considers only typical conditions and normal operating temperatures for the transistors since any other conditions will drag the project out from the main scope and will require more time and calibration. The 0.9V supply voltage is considered to be ideal since other people in any given project will handle it. A 20ps time is added to the start and stop signals to have a more realistic behavior. No jitter is considered in the signals since the work in [21] and [22] reported a 0.005 change in the measured delay by the jitter, which is insignificant.

This chapter will discuss the simulated results in the same order given in Chapter 4. Finally, a performance summary for the designed TDC, along with a comparison between initial specifications versus final results. Due to technical difficulties, all results for mismatch/process variations, area, power, and propagation delay are pre-layout results. Even though the layout for the proposed VDL segments is drawn, no post-layout results are extracted because neither DRC or LVS checks are available despite many attempts to get them running. The lack of post-layout extraction is also applied to the encoding circuit results. It must be pointed out that post layout results will different since 28nm FDSOI CMOS technology is used, which will affect the performance metrics of the proposed TDC [37] [34].

5.1 Proposed Implementation

Initially, the proposed VDL TDC in figure 4.1 showed a significant improvement in the area and power consumption. Furthermore, since multiple delay lines are used to generate the final binary output code, it was thought it would add a significant robustness to the output code (i.e. calibration) against process and mismatch variations.

The proposed TDC uses six parallel Vernier delay lines, as in figure 4.1. The TDC has a total number of 144 delay segments, with different resolutions (LSB) for each line, as detailed in table 4.1. The segments are constructed from the same delay segment in figure 3.5 with different values for C_1 to achieve different resolutions by changing the delay time for the start signal, the values for the capacitors for different resolution are shown in Appendix A, table A.1. As for the C_2, it is kept the same to have the same variations and consistent behavior for segments. This made the design to be more time-efficient since one capacitor size has to be changed.

Dynamic range and resolution (LSB) of the proposed TDC are dominated by the first delay line (i.e. 10ps line). The TDC cannot detect any resolution lower then 10ps since no line combination can detect any time different (ΔT) less than the first line. The dynamic range for the proposed TDC is 630ps, which corresponds to 63 segments multiplied by the resolution. Table 4.1 shows that the first line has 67 segments, since each line needs a two dummy segment at the beginning and the at the end to make signals (i.e. start and stop) stable. To increase the dynamic range for the proposed TDC, the designer must increase the lengths (number of segments) for all lines with respect to the line resolution. For example, if the dynamic range to become 730ps, ten segments must be added to the first line, four segments to the second line, two segments to the third line, one segment to the fourth line and no changes for the last two lines, since their resolution is higher than the dynamic range increment.

5.2 Process and Mismatch Variations

The process and mismatch variations for delay cells with the different resolutions are simulated using Monte Carlo with 300 points as in [33]. The standard deviation (σ) and the mean value for the propagation delay of the stop delay cell (τ_2) are constant throughout the entire design since only the start delay cell is modified to achieve different resolutions.

Standard deviations (σ) for the propagation delay of start and stop delay cells (τ_1) in table 4.2, shows CVs less than 5% for different resolutions, which results from increasing the sizes of the transistors. The mean value, on the other hand, varies from 0.03ps to 1.54ps from the nominal values, which affects the matching of the Vernier lines (i.e. start and stop delay cells must have a small deviation from the nominal values), since stop delay cell is uniformed for all lines. As seen from table 4.2, that first and fifth delay lines have the best match since the mean value for start delay cells has the smallest deviations from nominal values. The other lines will vary more, which affects the nonlinearities for the proposed TDC since the output binary code depends on all of them.

Matching of the Vernier line is confirmed in the digital workspace, by generating the proposed TDC model in SystemVerilog language and substitute the extracted transistor behavior from the analog workspace. The function "$dist\_normal(seed, mean, standard\_deviation)$" returns a probabilistic distributed number that is an average approach to the mean argument, and seed controls the numbers that the function return. A well-matched VDL will have the same seed for both start and stop delay cells since changing the seed will affect the resolution. Figure 4.3 shows the histograms for 300 point extracted from SystemVerilog to represents the 10ps delay segment. The same seed for both delay cells will give a well-matched segment, which has a mean value of 10.02ps and σ of 0.173ps, as shown in figure 4.3 (a). However, a different seed will cause the VDL segment to have a similar mean value but a large σ of 7.5ps, as shown in figure 4.3 (b), which deteriorate TDC performance.

Tables 4.3 and 4.4 illustrate the effects of seeding on the resolution of the segments for the enter proposed design. Although the mean values for the LSB do not vary that much, the differences will affect the performance of the proposed TDC as it will become clear in the next sections. It is noticed that in order to well-matched Vernier lines, three conditions have to be met. First, the delay cells must have an average value of the delay as close to the nominal. Second, the standard deviation (σ) for cell delay has to as small as possible. Finally, both delay cells (i.e. start and stop) have to experience the same variations, which in this work represented by the seed number, that simulates the condition where both delay cells experienced the same variations on the chip. However, if a different seed is used for both cells, the standard deviation (σ) of the resolution will be much larger, which confirms that both cells experienced different variations, as shown in table 4.4.

5.3 Encoding Circuit

Listing C.2, Appendix C shows the SystemVerilog TDC core, which includes the encoding circuit at the end of the code. The circuit was assumed to be simpler to implement since it will be expressed as a boolean equation and use standard gates. This assumption is valid for small TDC (i.e. TDC with a small number of bits), but as TDC gets larger, the number of inputs increases, and the complexity of the boolean equation increase dramatically as seen in the encoder code. Table 4.5 lists all the number of all used gates along with their propagation delay and average power consumption. The power and delay figures are an estimation since they are based on schematic simulation with no loading effects (gates are tested as a stand-alone with no other gates connected).

Table 5.1 shows a comparison between the proposed encoding circuit and Wallace Encoder. Results for the Wallace encoder are estimated similarly to the proposed encoding circuit to have a fair comparison. Gate counts for the Wallace encoder are estimated from the number of full adders, by using $[2^N - (N+1)]$ and assuming every full adder consists of eight 2_input NAND gate, and every NAND gate has four transistors. The average power, area, and propagation delay for the NAND gate are estimated in a similar way to the proposed encoder as well. The total propagation delay for the Wallace Tree encoder is calculated from the critical path, by estimating the number of full adders the signal has to propagate through, [21] demonstrate a figure where a critical path of 7 full adders is estimated.

Table 5.1: Proposed Encoding Circuit Vs Wallace Encoder

Encoding Circuit	**Propagation Delay (ps)**	**Average Power (nW)**	**Area Size (μm^2)**	**Gates Count**	**Transistors Count**
Proposed	433	80	235	179	1564
Wallace Tree	420	28	275	456	1824

The Wallace encoder has a similar propagation delay to the proposed encoder. However, the number of gates, transistors, and area are higher for the Wallace since the proposed encoder use a multiple inputs standard gate, which helps reduce the total count. The proposed encoding circuit did not offer any drastic improvements over Wallace as initially thought. The power for both encode is consider to be small since loading effects are not considered; the real number should be higher. Simplification of the proposed encoding circuit by reviewing the boolean equations or by using a synthesizer is possible, which could benefit the proposed TDC.

5.4 Nonlinearities

The first time nonlinearities were measured, they showed unexpected bad results. Initially, the TDC model and encoding circuit were considered to cause these results, but a further investigation through SystemVerilog models reveal the cause of this problem is the seeding for the (*dist_normal*) function. The random function will generate a random number with a specific pattern if the function is seeded with correlated numbers. Figure 4.4 confirms the observations. While figure 4.5 shows the behavior of the random function if uncorrelated numbers are used, this is used to have nonlinearities results for the TDC depended on other factors.

After fixing the pattern in the random function, nonlinearities are plotted for the proposed TDC and VDL TDC using only the first delay line from the proposed TDC connected to a Wallace Tree encoder. This is done to find out how good or bad the proposed TDC in comparison with other popular encoders. The first scenario simulates both TDC with the same seed for delay cells. Figure 4.10 showed DNL for the proposed TDC has a high value at the end of the output code. This is caused by accumulations of variations from all different lines. The DNL for the proposed TDC has a maximum value of +1.4 LSB and a minimum value of -1 LSB, which indicates an uneven distribution of the noise. The INL figure reflected the DNL distributions and shows big accumulations at higher output codes (i.e. bigger ΔT); the proposed TDC will lose more code since the maximum INL is 1.6 LSB, which confirms noise accumulations. The TDC is considered to be monotonic, even with these values of nonlinearities since the output is always increasing with time.

On the other hand, DNL for the Wallace TDC has a maximum value of +0.1 LSB and a minimum value of -0.1 LSB and shows an even distribution of the noise. The INL figure shows a maximum of 0.3 LSB, which indicates no missing codes. Furthermore, since the INL maximum value is less than 0.5 LSB, the TDC is guaranteed to be monotonic [8]. The Wallace TDC showed better noise performance over the proposed TDC since it can suppress bubble errors [26], and no noise from other Verier lines is added up through the encoding process.

The second scenario is performed to simulate the effects of changing the seed for the delay cells to see the effects of unmatched cells on the TDC nonlinearities. The DNL and INL plots confirmed the previous assumptions that unmatched delay cells would deteriorate TDC noise performance. From figure 4.15, it is observed that multiple codes are missing from the TDC output code, and both TDCs have a DNL and INL greater than 3 LSB.

The third scenario is simulated to check if nonlinearities errors for the proposed TDC comes from other Vernier lines or not, by assuming well-matched segments for all Vernier lines. The results for DNL and INL in figure 4.16 confirm the previous assumptions since the proposed TDC shows a significant improvement in nonlinearities error. The DNL and INL have become well distributed and went from +1.4/-1.0 LSB to +0.4/-0.4 LSB for the DNL, and +1.6/-0.2 LSB to +0.3/-0.3 LSB, for the INL. Figure 4.17 shows the exact reason behind these improvements, where outputs from all Vernier lines are plotted from the first scenario. It is evident that the output from each delay line is skewed to the right and does not match the ideal output over various output codes. The skewing affects higher output codes since it gets accumulated gradually from previous stages through long time measurement (i.e. big ΔT). Furthermore, the skewing is the reason behind the proposed TDC monotonicity since the output is always skewed to the right, which indicates an increase in the time measurement. However, even without this skewing, the proposed TDC in third scenarios is still monotonic, since the INL maximum value is less than 0.5 LSB.

Table 4.6 summarise the nonlinearities for all scenarios. The DNL and INL for the Wallace case outperformed the proposed TDC in the first scenario. Table 4.7 shows the linearity of the output code (N_{linear}) for the simulated TDC based on their nonlinearities performance. The Wallace Tree TDC shows better performance over the proposed TDC under normal conditions (first scenario). However, if Vernier lines are designed to have better matched with respect to other delay lines, the linearity of the output will be similar.

5.5 Conversion Speed

The proposed TDC achieves a conversion speed of 110MS/s, which is dominated by the total propagation delay of the longest line. The speed indicates that the proposed TDC is considered fast among other TDC designs according to the comparison table of recent published TDCs in [1]. The conversion speed depends on the dynamic range in this architecture; a bigger dynamic range means more delay lines, which means longer time for the signal to propagate. However, this can be solved by decreasing the delay time for delay cells (i.e. τ_1 and τ_2). For example, the proposed TDC has a propagation delay of 135ps and 125ps for start and stop cells, respectively, in the longest delay line. If it is desired to double the dynamic range and maintaining the same conversion speed, the delay time for the cells has to be decreased by half while maintaining the difference between the delays to keep the LSB unchanged.

The encoding circuits will add an overhead delay and decrease the conversion speed of the TDC. Table 5.1 shows that both encoding circuits have an average delay of $425ps$, which is significantly small compared with $9.05ns$ period time for the first delay line. Nevertheless, even if the encoding circuits have larger propagation delay time, for example, $1ns$ as suggested in [28], the conversion speed will become 100MS/s, which is not a drastic change. This issue can be resolved by decreasing the propagation delay for the cells, as mentioned before.

5.6 Area

The area occupied by the proposed TDC is 3868 μm^2 with the encoding circuit taking just 235 μm^2, which shows that the Vernier lines dominants the area unlike the work done in [33]. Table 4.8 shows the total area for each delay line, where the first delay line has the most significant contribution since it utilizes a large number of segments. The area for the proposed TDC can be reduced by looking back at the layout for the proposed delay segment in figure 3.6. The segment uses two relatively big capacitors, which utilized more than 40% of the total area of the segment. These capacitors are used to achieve the required resolution (LSB) and stability for the segments.

However, these capacitors can be omitted from the delay segments of the first and second lines in the proposed TDC since 10ps, and 20ps LSB can be achieved through the intrinsic propagation delay provided by the CMOS technology. This will have the potential to reduce the overall area by 1000 μm^2 since the first two lines have the highest contribution to the TDC area according to table 4.8. Furthermore, since the layout in figure 3.6 is not drawn using minimum spacing, the actual area should be smaller than approximated. Nevertheless, in order to eliminate the usage for capacitors, a change to the architecture should be done, and the new lines should match the other lines in variations behavior. The estimations for the encoding circuits are based on the layout for standard cells in the near-threshold region, which have a bigger area than standard cells operating at normal voltages [37].

5.7 Power

The average power consumption by TDC is $876\mu W$, where the Vernier delay lines consume most of this power. Each segment consumes a different power figure according to table 4.9, which indicates the effects of using different capacitor sizes. The total power consumption will decrease if the suggestion in the previous section is implemented (removing the capacitors in first and second lines). Overall, the power consumption for VDL is higher than expected since bigger capacitors are used to implements the delay cells with low variations. The post-layout power consumption is expected to be more significant. The average power for the encoding circuits in table 5.1 is minimal because the estimation is based at the schematic level, and the logic gates are simulated without loading other gates. However, in reality, the average power consumption will be higher than this in terms of hundreds of μW due to parasitic resistance and capacitance [34], but it would still be insignificant compared with the power of Vernier lines.

5.8 Summary

The proposed TDC showed the potential to improve the performance of the work done in [33] at the beginning of this work. Table 4.10 compares the proposed TDC with the work done in [33], among other TDC implementations. The proposed TDC achieves high resolution and speed among other TDCs due to the usage of the proposed delay segment. The dynamic range is subjective and can be increased if the application requires so. The drawback of VDL dynamic range is resolved through CMOS technology advancement, where even if a longer VDL is designed, it will not utilize a significant area or power. Furthermore, longer line results in a better distribution of PVT variations [22]. The segments resolution (LSB) behavior in table 4.3 is extracted from the digital workspace and should be verified from the transistor behavior after post-layout extraction.

The proposed work has a better noise performance over the base design, and it is comparable to the performance of other TDCs . However, the initial area reduction assumption turned out to be not entirely valid since the area utilized by the capacitors in the delay cells will overcome the saved area by not using the conventional encoders. Furthermore, using capacitors make this TDC, not synthesizable, which takes an essential advantage from digital circuit design.

The proposed encoder has a similar number of transistors as a Wallace Tree encode; this means that both encoding circuits will utilize the same area on-chip. Furthermore, Wallace encoder is easier to design and parameterize over the proposed encoder and provide bubble error correction, which improves noise performance. Table 4.10 does not illustrate that the proposed TDC uses more area or power over [33] since different CMOS technology was used.

The first delay line in the TDC with Wallace encoder showed an excellent performance in overall metrics, which indicates that a long, well-matched VDL line with Wallace Tree encoder, can suppress first order variations [22] [16]. Furthermore, VDL can be synthesizable if standard delay cells are used, and Wallace Tree encoder offers more advantages over its area and power consumption.

Nonetheless, The proposed TDC has fulfilled nearly every initial specification in table 3.1, even if nonlinearities, area, and power metrics are not as initially assumed.

Chapter 6

Conclusion

The proposed TDC uses a multiple Vernier Delay lines operating in parallel to measure time and converted it to a thermometer code. This approach converts the thermometer code to a binary code without using conventional encoders. The project took advantages from both analog and digital workspaces to overcome the technical limitations. The analog workspace is implemented using 28nm FDSOI CMOS technology, while the digital workspace is implemented in SystemVerilog.

The proposed TDC is monotonic with 6 bits output code, a 630ps dynamic range, and 110MS/s conversion speed, which considered good metrics compared with other TDCs. The nonlinearities got improved over the original work, especially in the INL figure, which got improved by 7 LSB, which shows the advantages of using the parallel VDLs. The proposed TDC has a DNL of +1.4/-1.0 and INL of +1.6/-1.0, with 4.62 output linear bits (N_{Linear}). However, the area and the power consumption did not improve as initially thought due to unforeseen things. Overall, the current performance of the proposed TDC is considered to be good over other implemented TDCs metrics. The third scenario shows the possibility to improve noise performance dramatically and make nonlinearities less than 1 LSB. Furthermore, this scenario proves that in order to have well-match VDLs, both delay cells must have minimum CV percentage and small difference between nominal and mean values for the delay times.

The comparison with Wallace Tree encoder showed by using it with a single VDL gives better performance metrics, less complicated, and more design efficient TDC. However, in order to have good metrics for any VDL TDC design, three main criteria should be considered. First, the VDL has to be as long as possible to have a Gaussian distribution of noise. Second, the delay cells must be well matched. Third, the encoding circuit should have Bubble-Error correction.

6.1 Future Work

Nonetheless, the results from the proposed TDC are considered for publishing after layout extraction is done to verify the behavior of the TDC and implement the third scenario. Since this architecture, according to the author's best knowledge, is not mentioned anywhere and worth looking more into it.

The encoding circuit could be simplified using boolean algebra or using a synthesizer. The layouts for the proposed delay segments can be improved, as discussed earlier. Noise from jitter can be considered to verify the initial assumptions. Other process corners and temperatures must be considered to study the effects of them on the TDC performance; calibration circuits can be considered if necessary.

Bibliography

[1] Zeng Cheng; Xiaoqing Zheng; M. Jamal Deen; Hao Peng. Recent developments and design challenges of high-performance ring oscillator cmos time-to-digital converters. *IEEE Transactions on Electron Devices*, 36(1):235 – 251, 2016.

[2] Amer Samarah; Anthony Chan Carusone. A digital phase-locked loop with calibrated coarse and stochastic fine tdc. *IEEE Journal of Solid-State Circuits*, 48(8):1829 – 1841, 2013.

[3] Su Pin-en; P.Madoglio; W.Y. Li Kim Hyung Seok; e. Ornelas; K. Chandrashekar; D. Shi and A Ravi. A digital fractional-n pll with a pvt and mismatch insensitive tdc utilizing equivalent time sampling technique. *IEEE Journal of Solid-State Circuits*, 48(7):1721 – 1729, 2013.

[4] Abdel S. Yousif; James W. Haslett. A fine resolution tdc architecture for next generation pet imaging. *IEEE Transactions on Nuclear Science*, 54(5):1574 – 1582, 2007.

[5] Jan Nissinen; Ilkka Nissinen; Juha Kostamovaara. Integrated receiver including both receiver channel and tdc for a pulsed time-of-flight laser rangefinder with cm-level accuracy. *IEEE Journal of Solid-State Circuits*, 44(5):1486 – 1497, 2009.

[6] Keunoh Park; Jaehong Park. 20 ps resolution time-to-digital converter for digital storage oscilloscopes. *IEEE Nuclear Science Symposium Conference Record*, 1998.

[7] E. Raisanen-Ruotsalainen; T. Rahkonen; J. Kostamovaara. An integrated time-to-digital converter with 30-ps single-shot precision. *IEEE Journal of Solid-State Circuits*, 41(12):2911–2920, 2006.

[8] Tony Chan Carusone; David Johns; Kenneth Martin. *Analog Integrated Circuit Design*. Wiley & Sons, second edition, 2011.

[9] KwangSeok Kim; WonSik Yu; SeongHwan Cho. A 9b, 1.12ps resolution 2.5b/stage pipelined time-to-digital converter in 65nm cmos using time-register. *Symposium on VLSI Circuits*, 2013.

[10] Sung-Jin Kim; Taeik Kim; Hojin Park. A 0.63ps, 12b, synchronous cyclic tdc using a time adder for on-chip jitter measurement of a soc in 28nm cmos technology. *Symposium on VLSI Circuits Digest of Technical Papers*, 2014.

[11] Jun-Seok Kim; Young-Hun Seo; Yunjae Suh; Hong-June Park; Jae-Yoon Sim. A 300-ms/s, 1.76-ps-resolution, 10-b asynchronous pipelined time-to-digital converter with on-chip digital background calibration in 0.13-μm cmos. *IEEE Journal of Solid-State Circuits*, 48(2):516–526, 2012.

[12] Asen Asenov. *Advanced Monte Carlo Techniques in the Simulation of CMOS Devices and Circuits*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[13] Stephan Henzler. *Time-to-Digital Converters*. Springer, Dordrecht, first edition, 2010.

[14] M. Kajita; M. Mizuno K. Nose. A 1-ps resolution jittermeasurement macro using interpolated jitter oversampling. *IEEE Journal of Solid-State Circuits*, 35(10):1507 – 1510, 2000.

[15] KwangSeok Kim; YoungHwa Kim; WonSik Yu; SeongHwan Cho. A 7b, 3.75ps resolution two-step time-to-digital converter in 65nm cmos using pulse-train time amplifier. *Symposium on VLSI Circuits (VLSIC)*, 2012.

[16] Jianjun Yu; Fa Foster Dai; Richard C. Jaeger. A 12-bit vernier ring time-to-digital converter in 0.13μm cmos technology. *IEEE Journal of Solid-State Circuits*, 45(4):830–842, 2010.

[17] Ping Lu; Antonio Liscidini; Pietro Andreani. A 3.6 mw, 90 nm cmos gated-vernier time-to-digital converter with an equivalent resolution of 3.2 ps. *IEEE Journal of Solid-State Circuits*, 47(7):1626–1635, 2012.

[18] Ping Lu; Ying Wu; Pietro Andreani. A 2.2-ps two-dimensional gated-vernier time-to-digital converter with digital calibration. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(11):1019–1023, 2016.

[19] Luca Vercesi; Antonio Liscidini; Rinaldo Castello. Two-dimensions vernier time-to-digital converter. *IEEE Journal of Solid-State Circuits*, 45(8):1504–1512, 2010.

[20] Volodymyr Kratyuk; avan Kumar Hanumolu; Kerem Ok; Un-Ku Moon; Kartikeya Mayaram. A digital pll with a stochastic time-to-digital converter. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 56(8):1612–1621, 2009.

[21] Khalil Jacob Gammoh. Pvt-tolerant stochastic time-to-digital converters. Master's thesis, Brigham Young University, Utah, USA, 2018.

[22] Sung-Jin Kim; Wooseok Kim; Minyoung Song; Jihyun Kim; Taeik Kim; Hojin Park. 15.5 a 0.6v 1.17ps pvt-tolerant and synthesizable time-to-digital converter using stochastic phase interpolation with 16x spatial redundancy in 14nm finfet technology. *IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015.

[23] Kentaroh Katoh; Kazuteru Namba. A low area calibration technique of tdc using variable clock generator for accurate on-line delay measurement. *Sixteenth International Symposium on Quality Electronic Design*, 2015.

[24] P. Dudek; S. Szczepanski; J.V. Hatfield. A high-resolution cmos time-to-digital converter utilizing a vernier delay line. *IEEE Journal of Solid-State Circuits*, 35(2):240–247, 2000.

[25] Tingbing Ouyang; Bo Wang; Lizhao Gao; Jiangtao Gu; Chao Zhang. A high resolution time-to-digital converter (tdc) based on self-calibrated digital-to-time converter (dtc). *IEEE 60th International Midwest Symposium on Circuits and Systems*, 2017.

[26] Sandeep Kumar; M. K. Suman; K. L. Baishnab. A novel approach to thermometer-to-binary encoder of flash adcs-bubble error correction circuit. *International Conference on Devices, Circuits and Systems (ICDCS)*, 2, 2014.

[27] E. Säll. Implementation of flash analog-to-digital converters in silicon- insulator cmos technology. Master's thesis, Linköping Studies Sci. Technology, Linköping, Sweden, 2007.

[28] M. M. Pilipko; D. V. Morozov; D. O. Budanov. Comparative analysis of cmos circuits of a thermometer-to-binary encoder for integrated flash analog-to-digital converters. *International Conference on Devices, Circuits and Systems (ICDCS)*, 46(1):45–54, 2017.

[29] Robert Bogdan Staszewski; Sudheer Vemulapalli; Prasant Vallur; John Wallberg; Poras T. Balsara. 1.3 v 20 ps time-to-digital converter for frequency synthesis in 90-nm cmos. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(3):220–224, 2006.

[30] STEPHEN H. UNGER; CHUNG-JEN TAN. Clocking schemes for high-speed digital systems. *IEEE Transactions on Computers*, c-35(10):880–895, 1986.

[31] Vladimir Stojanovic; Vojin G. Oklobdzija. Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems. *IEEE Journal of Solid-State Circuits*, 34(4):536–548, 1999.

[32] Chen Kong Teh; Tetsuya Fujita; Hiroyuki Hara; Mototsugu Hamada. A 77% energy-saving 22-transistor single-phase-clocking d-flip-flop with adaptive-coupling configuration in 40nm cmos. *IEEE International Solid-State Circuits Conference*, 2011.

[33] Niklas U. Andersson; Mark Vesterbacka. A vernier time-to-digital converter with delay latch chain architecture. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(10):773–777, 2014.

[34] Somayeh Hossein Zadeh; Trond Ytterdal; Snorre Aunet. Comparison of ultra low power full adder cells in 22 nm fdsoi technology. *IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2018.

[35] Even Låte; Trond Ytterdal; Snorre Aunet. A loadless 6t sram cell for sub- & near- threshold operation implemented in 28 nm fd-soi cmos technology. *Integration, the VLSI jounral*, 63:56–63, 2018.

[36] B. S. Everitt; Anders Skrondal. *The Cambridge Dictionary of Statistics*. Cambridge University Press, Cambridge, United Kingdom, 2010.

[37] Aslak Lykre Holen. Implementation and comparison of digital arithmetics for low voltage / low energy operation. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, 2015.

[38] Zeng Cheng; M. Jamal Deen; Hao Peng. A low-power gateable vernier ring oscillator time-to-digital converter for biomedical imaging applications. *IEEE Transactions on Biomedical Circuits and Systems*, 10(2):445–454, 2016.

Appendix A

TDC Segments Parameters and Layouts

Table A.1: Capacitor C_1 Values for Different Resolutions

Delay Segment	**Resolution LSB(ps)**	C_1 **Value(fF)**
1	10	4.4
2	20	5.0
3	40	6.2
4	80	9.2
5	160	15.0
6	320	27.4

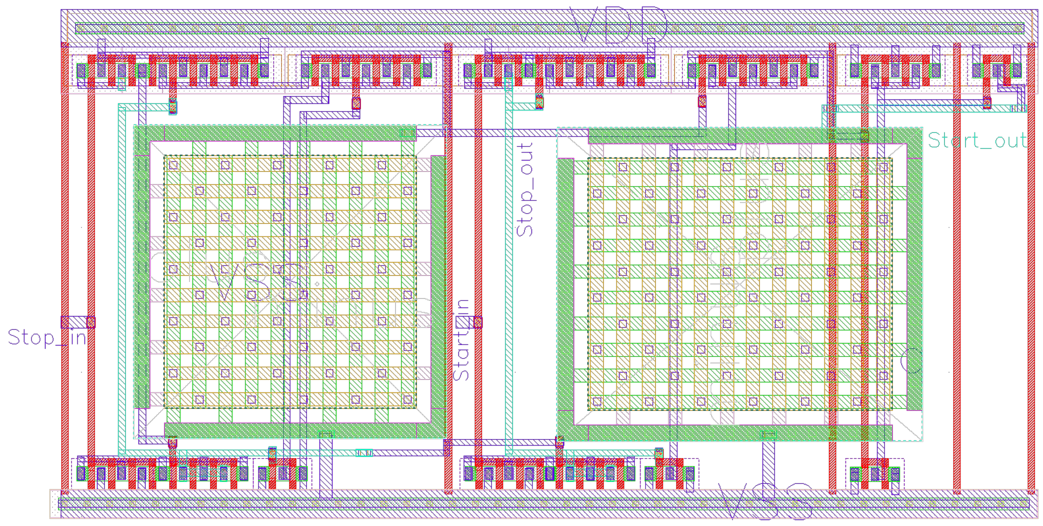


Figure A.1: Layout for the proposed delay segment with 20ps LSB

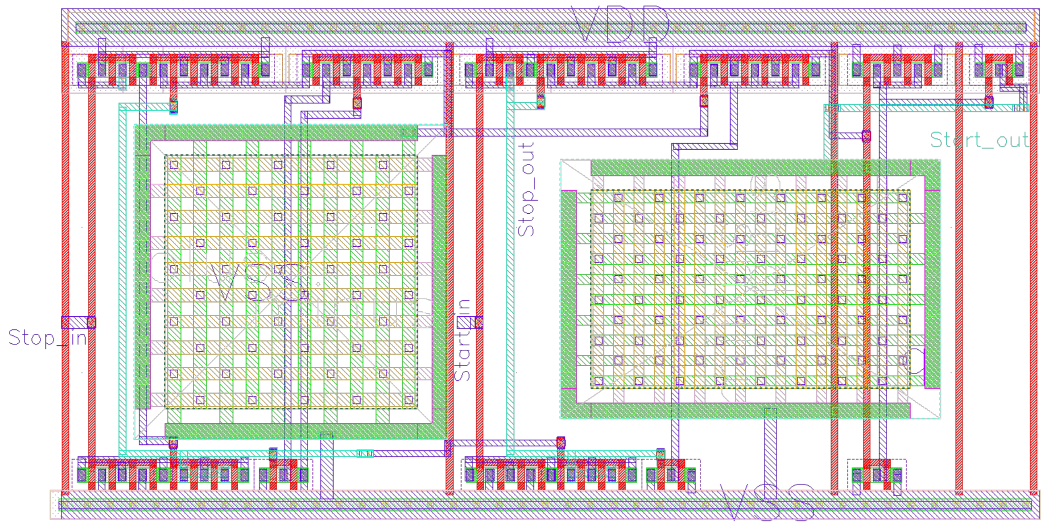


Figure A.2: Layout for the proposed delay segment with 40ps LSB

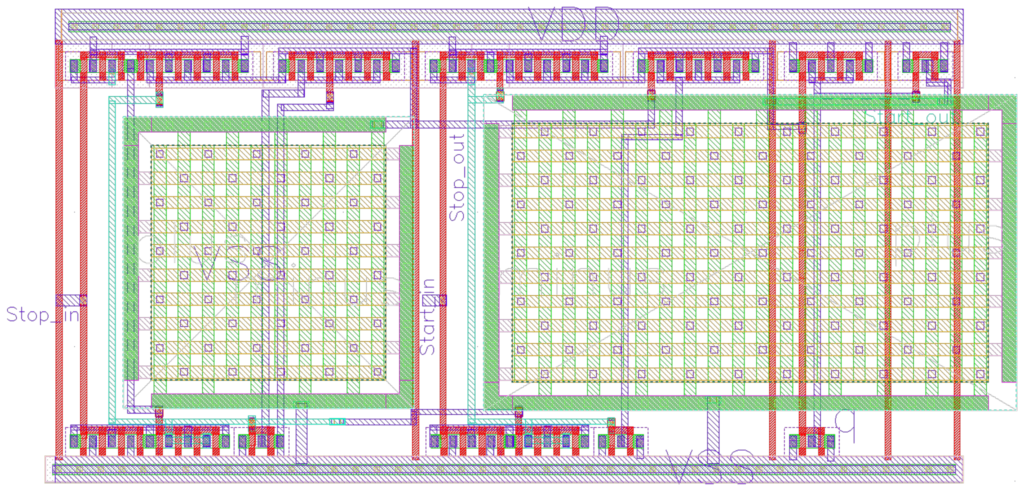


Figure A.3: Layout for the proposed delay segment with 80ps LSB

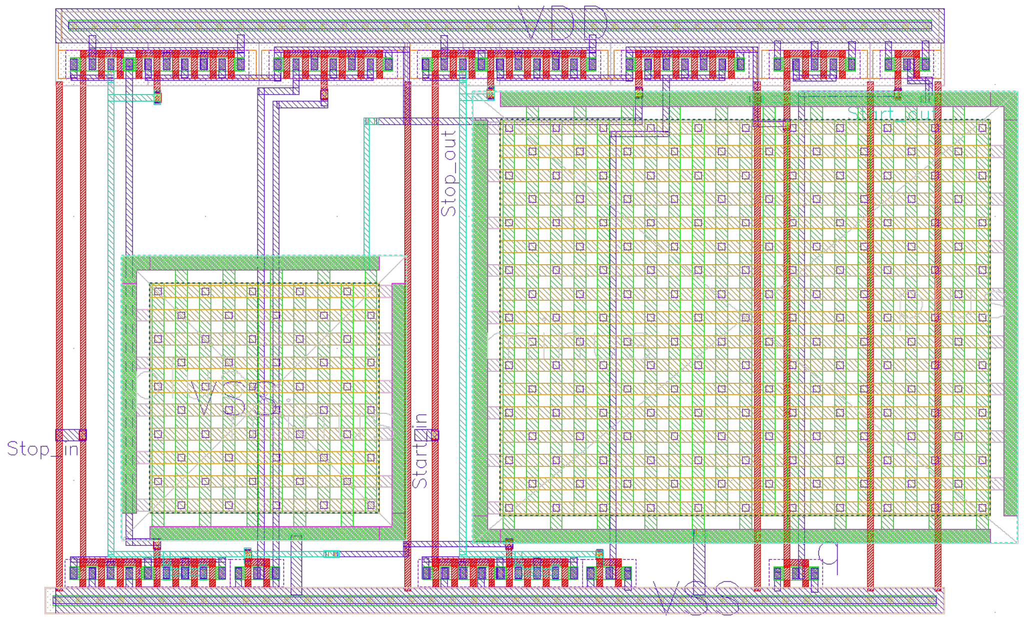


Figure A.4: Layout for the proposed delay segment with 160ps LSB

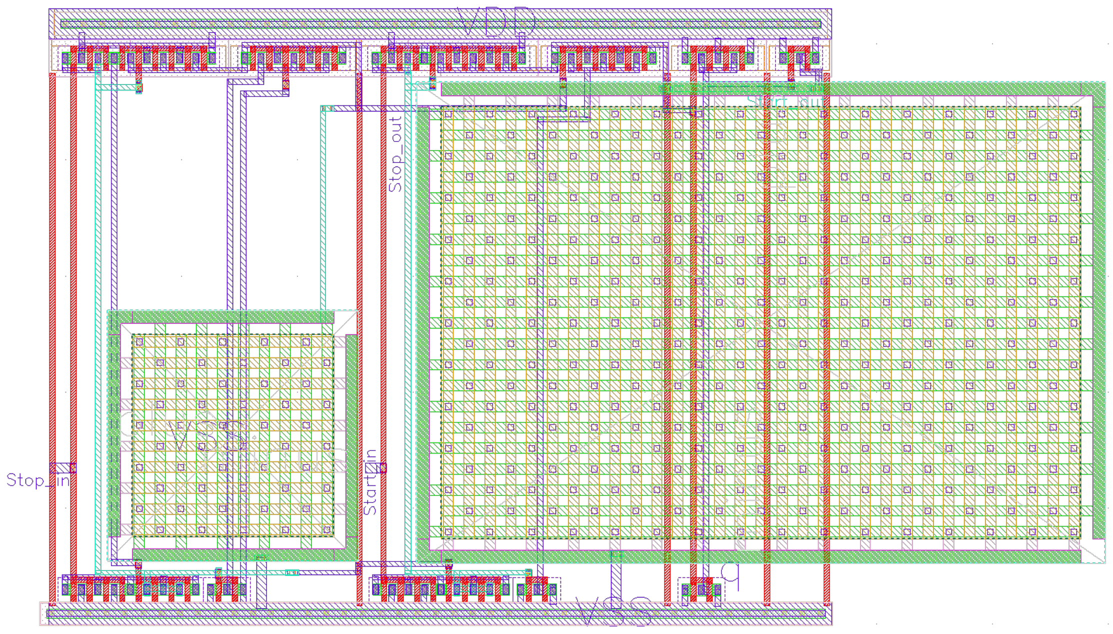
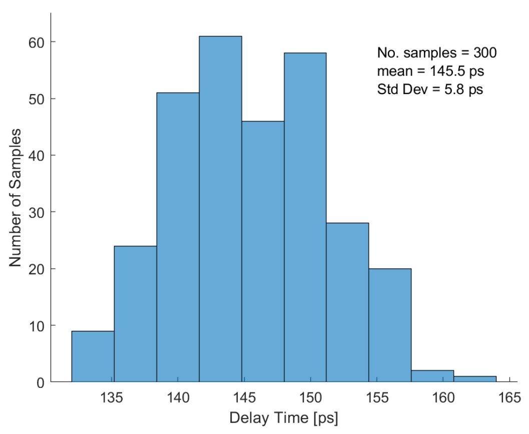


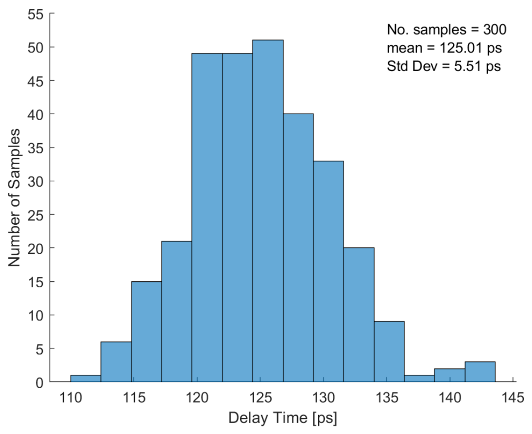
Figure A.5: Layout for the proposed delay segment with 320ps LSB

Appendix B

Simulation Results for TDC Segments

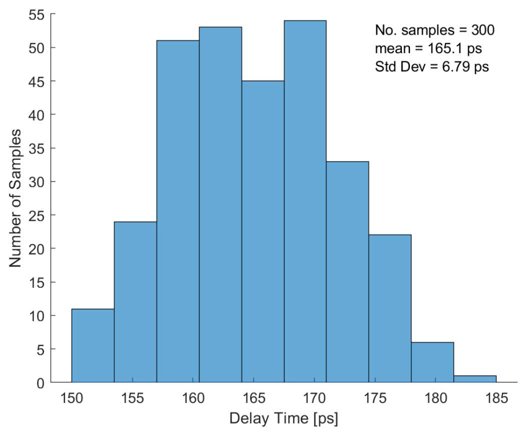


(a) Start Delay cell Histogram

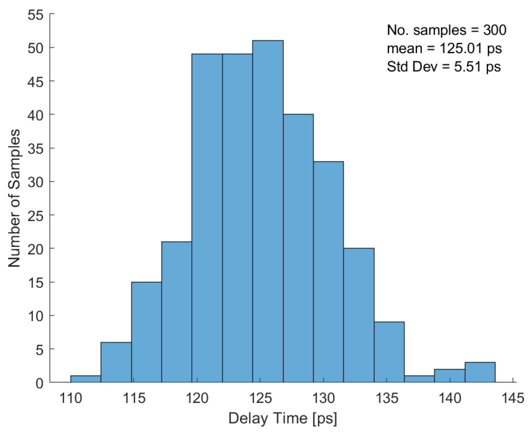


(b) Stop Delay cell Histogram

Figure B.1: Monte Carlo Histogram for 20ps Delay Segment

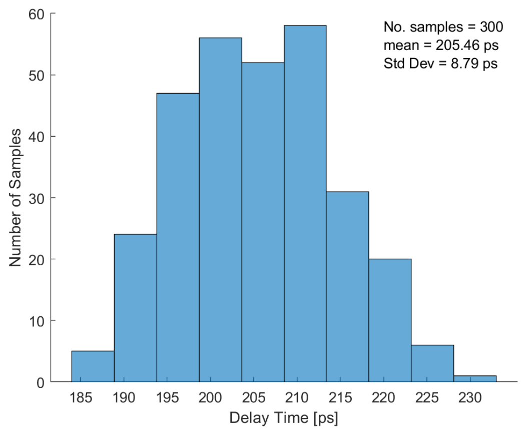


(a) Start Delay cell Histogram

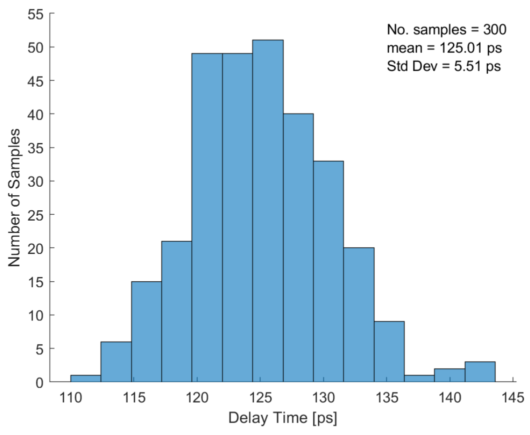


(b) Stop Delay cell Histogram

Figure B.2: Monte Carlo Histogram for 40ps Delay Segment

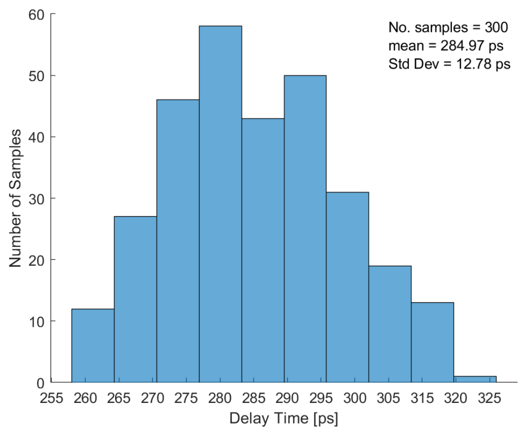


(a) Start Delay cell Histogram

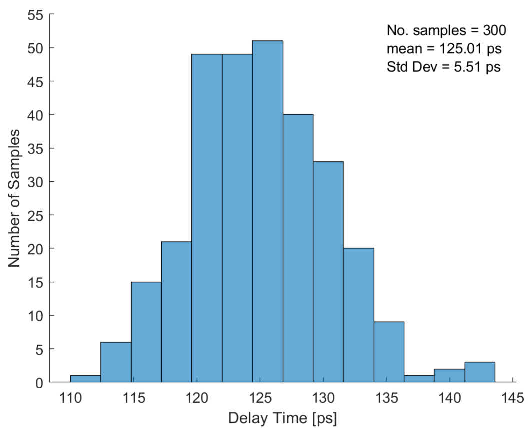


(b) Stop Delay cell Histogram

Figure B.3: Monte Carlo Histogram for 80ps Delay Segment

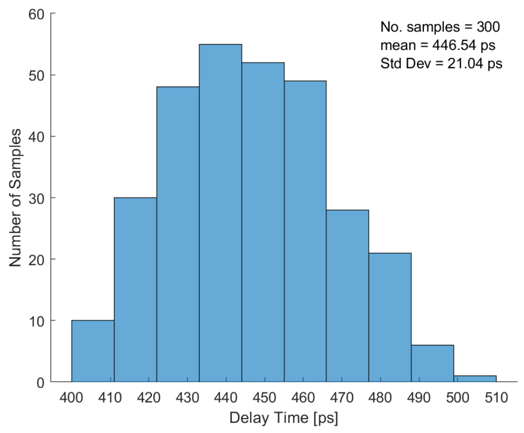


(a) Start Delay cell Histogram

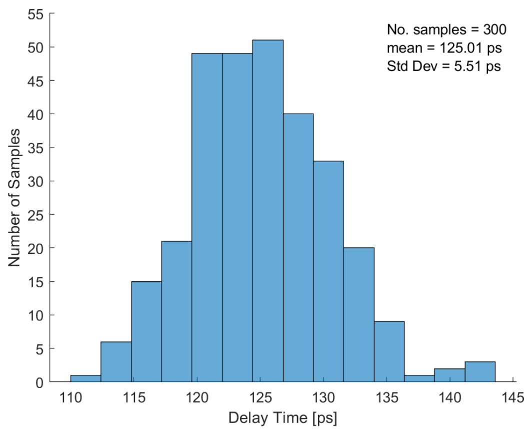


(b) Stop Delay cell Histogram

Figure B.4: Monte Carlo Histogram for 160ps Delay Segment

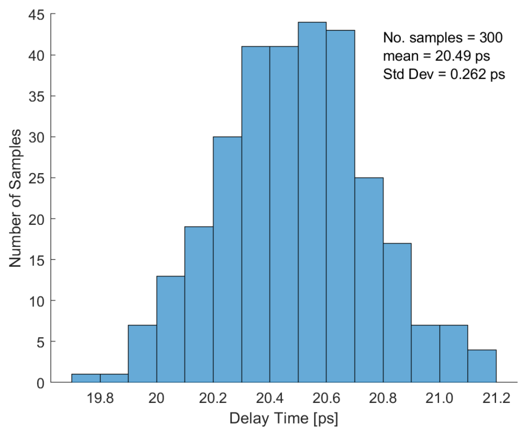


(a) Start Delay cell Histogram

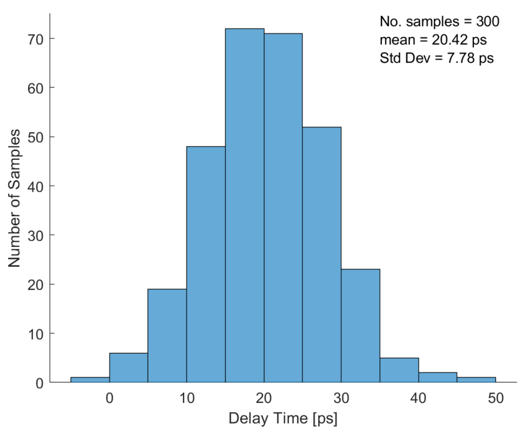


(b) Stop Delay cell Histogram

Figure B.5: Monte Carlo Histogram for 320ps Delay Segment

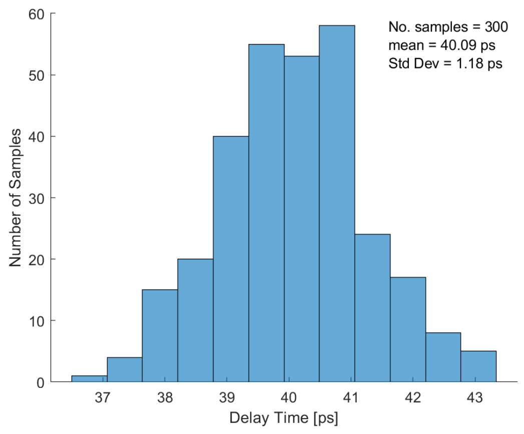


(a) Same seeding for random function

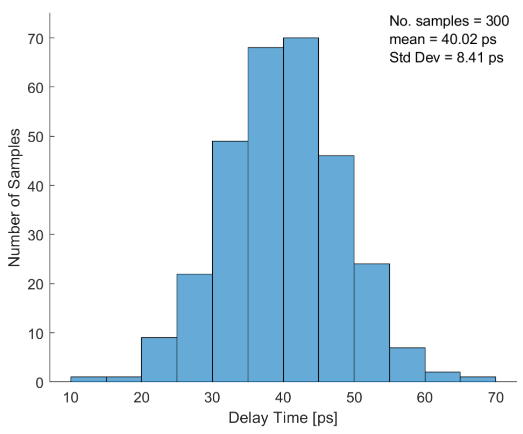


(b) Different seeding for random function

Figure B.6: Histogram for Simulated 20ps LSB Segment

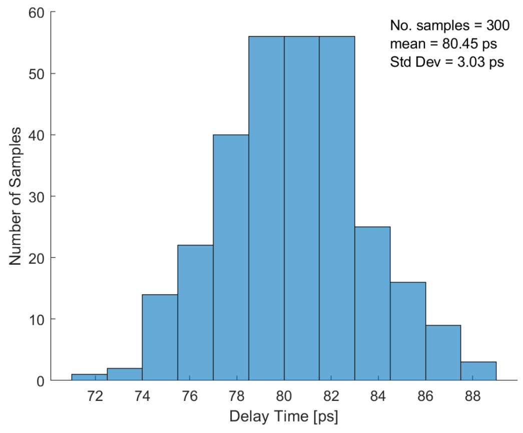


(a) Same seeding for random function

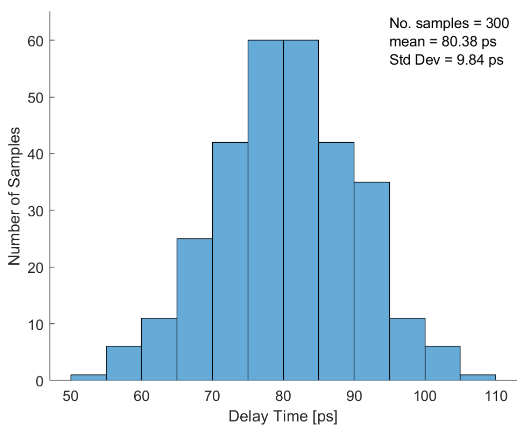


(b) Different seeding for random function

Figure B.7: Histogram for Simulated 40ps LSB Segment

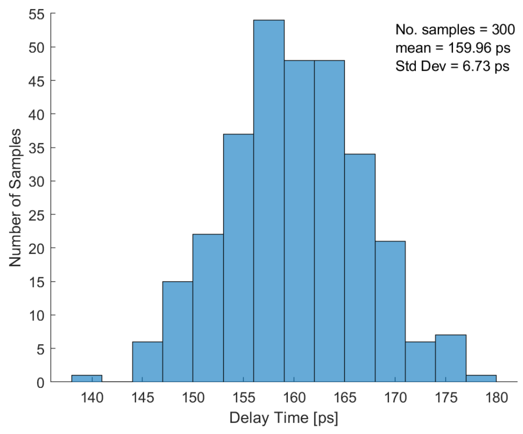


(a) Same seeding for random function

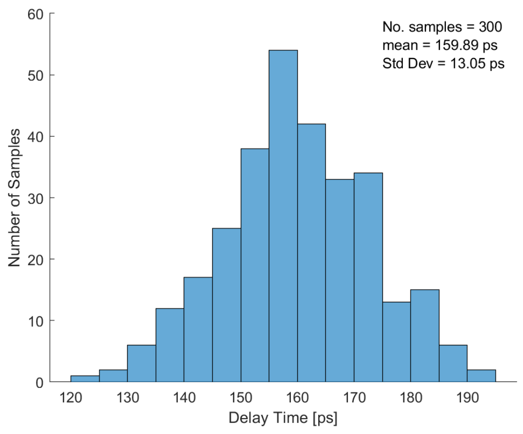


(b) Different seeding for random function

Figure B.8: Histogram for Simulated 80ps LSB Segment

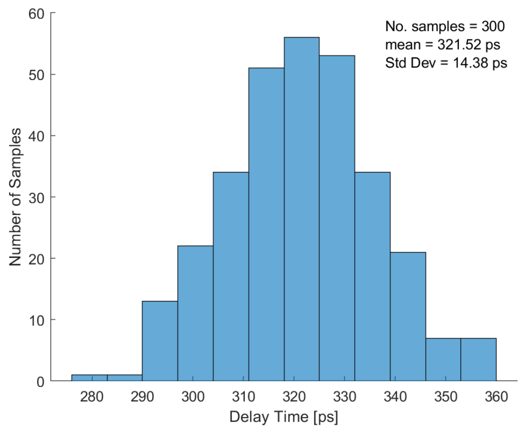


(a) Same seeding for random function

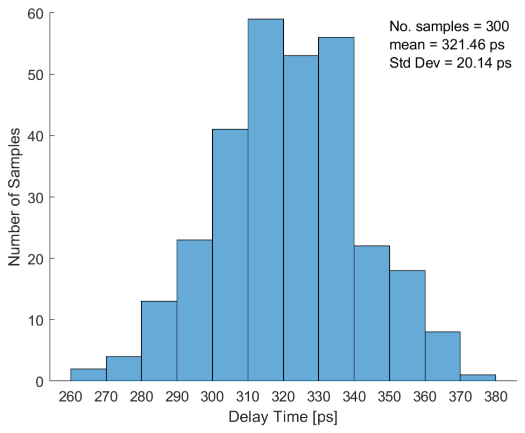


(b) Different seeding for random function

Figure B.9: Histogram for Simulated 160ps LSB Segment



(a) Same seeding for random function



(b) Different seeding for random function

Figure B.10: Histogram for Simulated 320ps LSB Segment

Appendix C

SystemVerilog Models

Listing C.1: Vernier Delay Segment

```

module VDL_variations #( // The Vernier delay segment consist of a start delay
    ↪ lien and at stop delay line with an ideal flip flop to hold the data

 )(

  input wire start,
  input wire stop,
  input real start_delay,// the delay time for the start delay cell with
     ↪ variations from the TestBecnh
  input real stop_delay, //the delay time for the stop delay cell with
     ↪ variations from the TestBecnh
  input logic reset,

  output wire start_d,
  output wire stop_d,
  output logic q_out
);
  wire start_temp ;
  wire stop_temp ;

  assign #(start_delay*1ps) start_temp = start ; // Moduling the varitions for
     ↪ the star signal
  assign start_d = start_temp;

  assign #(stop_delay*1ps) stop_temp = stop; // Moduling the varitions for the stop signal
  assign stop_d = stop_temp;

  always_ff @(posedge stop_temp or posedge reset) begin
    if(reset) begin
      q_out <= 0;
    end else begin
      q_out <= start_temp;
    end
  end

endmodule
```

Listing C.2: TDC Core Model

```
// The TDC core consist of 6 Vernier delay line + the encoding circuit

module MasterprjTDCCore #(
)(
  input logic ck,
  input logic arst,
  input logic start,
  input logic stop,

  input real start_chain_10ps[62:0],
  input real stop_chain_10ps[62:0],

  input real start_chain_20ps[30:0],
  input real stop_chain_20ps[30:0],

  input real start_chain_40ps[14:0],
  input real stop_chain_40ps[14:0],

  input real start_chain_80ps[6:0],
  input real stop_chain_80ps[6:0],

  input real start_chain_160ps[2:0],
  input real stop_chain_160ps[2:0],

  input real start_chain_320ps,
  input real stop_chain_320ps,

  output logic [5:0] q_out,
  output logic [62:0] out_thermo
  );

  logic [62:0] start_d_10 = '0;
  logic [62:0] stop_d_10 = '0;
  logic [62:0] q_a = '0;

  logic [30:0] start_d_20 = '0;
  logic [30:0] stop_d_20 = '0;
  logic [30:0] q_b = '0;

  logic [14:0] start_d_40 = '0;
  logic [14:0] stop_d_40 = '0;
  logic [14:0] q_c = '0;

  logic [6:0] start_d_80 = '0;
  logic [6:0] stop_d_80 = '0;
  logic [6:0] q_d = '0;

  logic [2:0] start_d_160 = '0;
  logic [2:0] stop_d_160 = '0;
  logic [2:0] q_e = '0;

  logic start_d_320 = '0;
  logic stop_d_320 = '0;
  logic q_f = '0;

  wire [5:0] binary_out;

// creating the Vernier Delay line with different resolutions
```

```

////////////////////////////////////////////////////////////

  VDL_variations #( // 10ps LSB Vernier delay line
    )
  u_VDL_10_seg_0(
    .start (start),
    .stop (stop),
    .start_delay (start_chain_10ps[0]),
    .stop_delay (stop_chain_10ps[0]),
    .reset (arst),
    .start_d (start_d_10[0]),
    .stop_d (stop_d_10[0]),
    .q_out (q_a[0])
  );

  genvar i;
  generate
    begin :la_VDL_10_segs
      for (i=1 ; i<63 ; i++) begin :la_VDL_10_seg
        VDL_variations #(
          )
        u_VDL_10_seg(
          .start (start_d_10[i-1]),
          .stop (stop_d_10[i-1]),
          .start_delay (start_chain_10ps[i]),
          .stop_delay (stop_chain_10ps[i]),
          .reset (arst),
          .start_d (start_d_10[i]),
          .stop_d (stop_d_10[i]),
          .q_out (q_a[i])
        );
      end
    end
  endgenerate
////////////////////////////////////////////////////////////////

```

```
VDL_variations #(  // 20ps LSB Vernier delay line
  )
u_VDL_20_seg_0(
  .start (start),
  .stop (stop),
  .start_delay (start_chain_20ps[0]),
  .stop_delay (stop_chain_20ps[0]),
  .reset (arst),
  .start_d (start_d_20[0]),
  .stop_d (stop_d_20[0]),
  .q_out (q_b[0])
);

genvar j;
generate
  begin :la_VDL_20_segs
    for (j=1 ; j<31 ; j++) begin :la_VDL_20_seg
      VDL_variations #(
        )
      u_VDL_20_seg(
        .start (start_d_20[j-1]),
        .stop (stop_d_20[j-1]),
        .start_delay (start_chain_20ps[j]),
        .stop_delay (stop_chain_20ps[j]),
        .reset (arst),
        .start_d (start_d_20[j]),
        .stop_d (stop_d_20[j]),
        .q_out (q_b[j])
      );
    end
  end
endgenerate
////////////////////////////////////////////////////////////

VDL_variations #(  // 40ps LSB Vernier delay line
  )
u_VDL_40_seg_0(
    .start (start),
    .stop (stop),
    .start_delay (start_chain_40ps[0]),
    .stop_delay (stop_chain_40ps[0]),
    .reset (arst),
    .start_d (start_d_40[0]),
    .stop_d (stop_d_40[0]),
    .q_out (q_c[0])
  );

  genvar k;
  generate
    begin :la_VDL_40_segs
      for (k=1 ; k<15 ; k++) begin :la_VDL_40_seg
        VDL_variations #(
          )
        u_VDL_40_seg(
          .start (start_d_40[k-1]),
          .stop (stop_d_40[k-1]),
          .start_delay (start_chain_40ps[k]),
          .stop_delay (stop_chain_40ps[k]),
          .reset (arst),
          .start_d (start_d_40[k]),
          .stop_d (stop_d_40[k]),
          .q_out (q_c[k])
        );
      end
    end
  endgenerate
//////////////////////////////////////////////////////////////////

  VDL_variations #(  // 80ps LSB Vernier delay line
    )
  u_VDL_80_seg_0(
    .start (start),
    .stop (stop),
    .start_delay (start_chain_80ps[0]),
    .stop_delay (stop_chain_80ps[0]),
    .reset (arst),
    .start_d (start_d_80[0]),
    .stop_d (stop_d_80[0]),
    .q_out (q_d[0])
  );

  genvar w;
  generate
    begin :la_VDL_80_segs
      for (w=1 ; w<7 ; w++) begin :la_VDL_80_seg
        VDL_variations #(
          )
        u_VDL_80_seg(
          .start (start_d_80[w-1]),
          .stop (stop_d_80[w-1]),
          .start_delay (start_chain_80ps[w]),
          .stop_delay (stop_chain_80ps[w]),
          .reset (arst),
          .start_d (start_d_80[w]),
          .stop_d (stop_d_80[w]),
          .q_out (q_d[w])
        );
      end
    end
  endgenerate
//////////////////////////////////////////////////////////////

  VDL_variations #(// 160ps LSB Vernier delay line
    )
  u_VDL_160_seg_0(
    .start (start),
    .stop (stop),
    .start_delay (start_chain_160ps[0]),
    .stop_delay (stop_chain_160ps[0]),
    .reset (arst),
    .start_d (start_d_160[0]),
    .stop_d (stop_d_160[0]),
    .q_out (q_e[0])
  );

  genvar y;
  generate
    begin :la_VDL_160_segs
      for (y=1 ; y<3 ; y++) begin :la_VDL_160_seg
        VDL_variations #(
          )
        u_VDL_160_seg(
          .start (start_d_160[y-1]),
          .stop (stop_d_160[y-1]),
          .start_delay (start_chain_160ps[y]),
          .stop_delay (stop_chain_160ps[y]),
          .reset (arst),
          .start_d (start_d_160[y]),
          .stop_d (stop_d_160[y]),
          .q_out (q_e[y])
        );
      end
    end
  endgenerate
////////////////////////////////////////////////////////////////////

  VDL_variations #( // 320ps LSB Vernier delay line
    )
  u_VDL_320_seg_0(
    .start (start),
    .stop (stop),
    .start_delay (start_chain_320ps),
    .stop_delay (stop_chain_320ps),
    .reset (arst),
    .start_d (start_d_320),
    .stop_d (stop_d_320),
    .q_out (q_f)

  );


          // Encoding Circuit

////////////////////////////////////////////////////
////////////////////////////////////////////////////

  assign binary_out[5] =
( q_a[31] & (q_b[15] | q_c[7] | q_d[3] | q_e[1] | q_f)) |
 (q_b[15] & (q_c[7] | q_d[3] | q_e[1] | q_f)) | (q_c[7] & (q_d[3] | q_e[1] | q_f))
| (q_d[3] & (q_e[1] | q_f)) | (q_e[1] & q_f);



  assign binary_out[4] =
( !binary_out[5] & ( (q_a[15] & (q_b[7] | q_c[3] | q_d[1] | q_e[0]))
| (q_b[7] & (q_c[3] | q_d[1] | q_e[0])) | (q_c[3] & (q_d[1] | q_e[0])) |
  (q_d[1] & q_e[0]) ) ) | ( binary_out[5] & ( (q_a[47] & (q_b[23] | q_c[11] | q_d[5] | q_e[2]))
| (q_b[23] & (q_c[11] | q_d[5] | q_e[2])) | (q_c[11] & (q_d[5] | q_e[2])) | (q_d[5] & q_e[2]) ) );



  assign binary_out[3] =
( !binary_out[5] & !binary_out[4] & ( (q_a[7] & (q_b[3] | q_c[1] | q_d[0])) | ( q_b[3] & (q_c[1] | q_d[0])) | (q_c[1] | q_d[0])))
| ( !binary_out[5] & binary_out[4] & ( (q_a[23] & (q_b[11] | q_c[5] | q_d[2])) | (q_b[11] & (q_c[5] | q_d[2])) | (q_c[5] | q_d[2])))
| ( binary_out[5] & !binary_out[4] & ( (q_a[39] & (q_b[19] | q_c[9] | q_d[4])) | (q_b[19] & (q_c[9] | q_d[4])) | (q_c[9] | q_d[4])))
| ( binary_out[5] & binary_out[4] & ( (q_a[55] & (q_b[27] | q_c[13] | q_d[6])) | (q_b[27] & (q_c[13] | q_d[6])) | (q_c[13] | q_d[6])));


```

```

  assign binary_out[2] =
( !binary_out[5] & !binary_out[4] & !binary_out[3] & ( ( q_a[3] & (q_b[1] | q_c[0]) ) | (q_b[1] & q_c[0])))
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & ( ( q_a[11] & (q_b[5] | q_c[2]) ) | (q_b[5] & q_c[2])))
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & ( ( q_a[19] & (q_b[9] | q_c[4]) ) | (q_b[9] & q_c[4])))
| ( !binary_out[5] & binary_out[4] & binary_out[3] & ( ( q_a[27] & (q_b[13] | q_c[6]) ) | (q_b[13] & q_c[6])))
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & ( ( q_a[35] & (q_b[17] | q_c[8]) ) | (q_b[17] & q_c[8])))
| ( binary_out[5] & !binary_out[4] & binary_out[3] & ( ( q_a[43] & (q_b[21] | q_c[10]) ) | (q_b[21] & q_c[10])))
| ( binary_out[5] & binary_out[4] & !binary_out[3] & ( ( q_a[51] & (q_b[25] | q_c[12]) ) | (q_b[25] & q_c[12])))
| ( binary_out[5] & binary_out[4] & binary_out[3] & ( ( q_a[59] & (q_b[29] | q_c[14]) ) | (q_b[29] & q_c[14])));



  assign binary_out[1] =
( !binary_out[5] & !binary_out[4] & !binary_out[3] & !binary_out[2] & ( q_a[1] &  q_b[0]))
| ( !binary_out[5] & !binary_out[4] & !binary_out[3] & binary_out[2] & ( q_a[5] & q_b[2]))
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & !binary_out[2] & ( q_a[9] & q_b[4]))
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & binary_out[2] & ( q_a[13] & q_b[6]))
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & !binary_out[2] & ( q_a[17]  & q_b[8]))
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & binary_out[2] & ( q_a[21] & q_b[10]))
| ( !binary_out[5] & binary_out[4] & binary_out[3] & !binary_out[2] & ( q_a[25] & q_b[12]))
| ( !binary_out[5] & binary_out[4] & binary_out[3] & binary_out[2] & ( q_a[29] &
   ↪  q_b[14]))
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & !binary_out[2] & ( q_a[33]
   ↪  & q_b[16]))
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & binary_out[2] & ( q_a[37]
   ↪ & q_b[18]))
| ( binary_out[5] & !binary_out[4] & binary_out[3] & !binary_out[2] & ( q_a[41]
   ↪ & q_b[20]))
| ( binary_out[5] & !binary_out[4] & binary_out[3] & binary_out[2] & ( q_a[45] &
   ↪  q_b[22]))
| ( binary_out[5] & binary_out[4] & !binary_out[3] & !binary_out[2] & ( q_a[49]
   ↪ & q_b[24]))
| ( binary_out[5] & binary_out[4] & !binary_out[3] & binary_out[2] & ( q_a[53] &
   ↪  q_b[26]))
| ( binary_out[5] & binary_out[4] & binary_out[3] & !binary_out[2] & ( q_a[57] &
   ↪  q_b[28]))
| ( binary_out[5] & binary_out[4] & binary_out[3] & binary_out[2] & ( q_a[61] &
   ↪ q_b[30]));



  assign binary_out[0] =
( !binary_out[5] & !binary_out[4] & !binary_out[3] & !binary_out[2] & !
   ↪ binary_out[1] & q_a[0])
| ( !binary_out[5] & !binary_out[4] & !binary_out[3] & !binary_out[2] &
   ↪ binary_out[1] & q_a[2])
| ( !binary_out[5] & !binary_out[4] & !binary_out[3] & binary_out[2] & !
   ↪ binary_out[1] & q_a[4])
| ( !binary_out[5] & !binary_out[4] & !binary_out[3] & binary_out[2] &
   ↪ binary_out[1] & q_a[6])
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & !binary_out[2] & !
   ↪ binary_out[1] & q_a[8])
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & !binary_out[2] &
   ↪ binary_out[1] & q_a[10])
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & binary_out[2] & !
   ↪ binary_out[1] & q_a[12])
| ( !binary_out[5] & !binary_out[4] & binary_out[3] & binary_out[2] & binary_out
   ↪ [1] & q_a[14])
```

```
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & !binary_out[2] & !
  ↪ binary_out[1] & q_a[16])
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & !binary_out[2] &
  ↪ binary_out[1] & q_a[18])
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & binary_out[2] & !
  ↪ binary_out[1] & q_a[20])
| ( !binary_out[5] & binary_out[4] & !binary_out[3] & binary_out[2] & binary_out
  ↪ [1] & q_a[22])
| ( !binary_out[5] & binary_out[4] & binary_out[3] & !binary_out[2] & !
  ↪ binary_out[1] & q_a[24])
| ( !binary_out[5] & binary_out[4] & binary_out[3] & !binary_out[2] & binary_out
  ↪ [1] & q_a[26])
| ( !binary_out[5] & binary_out[4] & binary_out[3] & binary_out[2] & !binary_out
  ↪ [1] & q_a[28])
| ( !binary_out[5] & binary_out[4] & binary_out[3] & binary_out[2] & binary_out[
  ↪ 1] & q_a[30])
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & !binary_out[2] & !
  ↪ binary_out[1] & q_a[32])
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & !binary_out[2] &
  ↪ binary_out[1] & q_a[34])
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & binary_out[2] & !
  ↪ binary_out[1] & q_a[36])
| ( binary_out[5] & !binary_out[4] & !binary_out[3] & binary_out[2] & binary_out
  ↪ [1] & q_a[38])
| ( binary_out[5] & !binary_out[4] & binary_out[3] & !binary_out[2] & !
  ↪ binary_out[1] & q_a[40])
| ( binary_out[5] & !binary_out[4] & binary_out[3] & !binary_out[2] & binary_out
  ↪ [1] & q_a[42])
| ( binary_out[5] & !binary_out[4] & binary_out[3] & binary_out[2] & !binary_out
  ↪ [1] & q_a[44])
| ( binary_out[5] & !binary_out[4] & binary_out[3] & binary_out[2] & binary_out[
  ↪ 1] & q_a[46])
| ( binary_out[5] & binary_out[4] & !binary_out[3] & !binary_out[2] & !
  ↪ binary_out[1] & q_a[48])
| ( binary_out[5] & binary_out[4] & !binary_out[3] & !binary_out[2] & binary_out
  ↪ [1] & q_a[50])
| ( binary_out[5] & binary_out[4] & !binary_out[3] & binary_out[2] & !binary_out
[1] & q_a[52])
| ( binary_out[5] & binary_out[4] & !binary_out[3] & binary_out[2] & binary_out[1] & q_a[54])
| ( binary_out[5] & binary_out[4] & binary_out[3] & !binary_out[2] & !binary_out[1] & q_a[56])
| ( binary_out[5] & binary_out[4] & binary_out[3] & !binary_out[2] & binary_out[1] & q_a[58])
| ( binary_out[5] & binary_out[4] & binary_out[3] & binary_out[2] & !binary_out[1] & q_a[60])
| ( binary_out[5] & binary_out[4] & binary_out[3] & binary_out[2] & binary_out[1] & q_a[62]);




assign q_out = binary_out; // Time measurment in binary

assign out_thermo = q_a; // thermomter code for the Wallce Tree encoder

endmodule
```

Listing C.3: TDC Top Model

```
module MasterprjTDC #( // top level for the project which initialize the TDC and a Wallce tree encoder

 )(

 input logic ck,
 input logic arst,
 input logic start,
 input logic stop,

        // These are real values of delay time plus variations for the delay segments in the TDC generated in the TestBench
 input real start_chain_10ps[62:0],
 input real stop_chain_10ps[62:0],

 input real start_chain_20ps[30:0],
 input real stop_chain_20ps[30:0],

 input real start_chain_40ps[14:0],
 input real stop_chain_40ps[14:0],

 input real start_chain_80ps[6:0],
 input real stop_chain_80ps[6:0],

 input real start_chain_160ps[2:0],
 input real stop_chain_160ps[2:0],

 input real start_chain_320ps,
 input real stop_chain_320ps,

 output logic [5:0] delay_10ps,
 output logic [5:0] delay_10ps_wallce
 );

 logic [5:0] q_out;
 logic [62:0] out_thermo;
 logic [5:0] out_wallce;

 generate
   if (1) begin

     // --------------------------------
     // -- MasterprjTDCCore
     // --------------------------------

     MasterprjTDCCore #(

     ) u_Core(

        // Inputs
        .ck (ck),
        .arst (arst),
        .start (start),

        .start_chain_10ps (start_chain_10ps),
        .stop_chain_10ps (stop_chain_10ps),

        .start_chain_20ps (start_chain_20ps),
        .stop_chain_20ps (stop_chain_20ps),

        .start_chain_40ps (start_chain_40ps),
        .stop_chain_40ps (stop_chain_40ps),

        .start_chain_80ps (start_chain_80ps),
        .stop_chain_80ps (stop_chain_80ps),

        .start_chain_160ps (start_chain_160ps),
        .stop_chain_160ps (stop_chain_160ps),

        .start_chain_320ps (start_chain_320ps),
        .stop_chain_320ps (stop_chain_320ps),

        .stop (stop),

        // Outputs
        .out_thermo (out_thermo),
        .q_out (q_out)
     );

        // flip flop for holding the final time measurement reading from the proposed encoding circuit
     always_ff @(posedge ck or posedge arst) begin
        if(arst) begin
          delay_10ps <= 0;
        end else begin
           delay_10ps <= q_out ;
        end
     end

```

```
      WallaceEncoder #(  // 6 bits Wallce Tree encode initializing for
          ↪ comparison
      .NOB ( 6 ),
      .DK ( 62 ) // this is for the carry in bit for the output adders should be
          ↪  (2**NOB)-2
      )
      u_test(
      .in_thermo(out_thermo),
      .out_delay (out_wallce)
    );
        // flip flop for holding the final time measurement reading from the
            ↪ Wallce tree encoding circuit
      always_ff @(posedge ck or posedge arst) begin
        if(arst) begin
          delay_10ps_wallce <= 0;
        end else begin
           delay_10ps_wallce <= out_wallce;
        end
      end

    end

  endgenerate



endmodule
```

Listing C.4: TDC TestBench

```
  `timescale 1ps / 1ps
module test_MasterprjTDC (
 );
  localparam T_CK16M = 62.5ns;

  inTest_MasterprjTDC uin_MasterprjTDC(); // initializing a TDC DUT
  logic start;
  logic stop;
```

```

// ------------------------------
// -- Clock and Reset
// ------------------------------

initial begin
  uin_MasterprjTDC.arst = 0;
  uin_MasterprjTDC.ck16M = 0;
  uin_MasterprjTDC.start = 0;
  uin_MasterprjTDC.stop = 0;
  uin_MasterprjTDC.Delay_10ps = '0;
  uin_MasterprjTDC.out_wallce = '0;

fork
      // generating a 16MHz clock just for refernce and handling data
      //between DUT and TestBench
      //( although the TDC can operate up to 100MHz this was done for
          ↪ simplicity)
forever begin
    #(T_CK16M/2);
    uin_MasterprjTDC.ck16M = !uin_MasterprjTDC.ck16M;
end


join_any

end

initial
begin
  $timeformat(-6, 3, "us", 0);
end


// ------------------------------
// -- DUT and assignments
// ------------------------------
```

```

MasterprjTDC #( // connecting the inputs and outputs of the DUT
  ) u_MasterprjTDC (

  .ck ( uin_MasterprjTDC.ck16M ),
  .arst ( uin_MasterprjTDC.arst ),

  .start_chain_10ps (uin_MasterprjTDC.start_chain_10ps),
  .stop_chain_10ps (uin_MasterprjTDC.stop_chain_10ps),

  .start_chain_20ps (uin_MasterprjTDC.start_chain_20ps),
  .stop_chain_20ps (uin_MasterprjTDC.stop_chain_20ps),

  .start_chain_40ps (uin_MasterprjTDC.start_chain_40ps),
  .stop_chain_40ps (uin_MasterprjTDC.stop_chain_40ps),

  .start_chain_80ps (uin_MasterprjTDC.start_chain_80ps),
  .stop_chain_80ps (uin_MasterprjTDC.stop_chain_80ps),

  .start_chain_160ps(uin_MasterprjTDC.start_chain_160ps),
  .stop_chain_160ps (uin_MasterprjTDC.stop_chain_160ps),

  .start_chain_320ps(uin_MasterprjTDC.start_chain_320ps),
  .stop_chain_320ps (uin_MasterprjTDC.stop_chain_320ps),

  .start ( uin_MasterprjTDC.start ),
  .stop  ( uin_MasterprjTDC.stop ),

  .delay_10ps ( uin_MasterprjTDC.Delay_10ps ),
  .delay_10ps_wallce( uin_MasterprjTDC.out_wallce)
 );

//-----------------------------------------------------
// Main Stimulus
//-----------------------------------------------------
initial begin
  string testName;
```

```
    testName = "";
    $display("---␣Starting␣simulation␣---");
    if($test$plusargs("TESTNAME")) begin
      assert ($value$plusargs("TESTNAME=%s", testName)) else begin
        $error("Wrong␣+TESTNAME");
        $finish;
      end
      case(testName)
        "Ideal_Test" :ta_Ideal_Test();
        "Symmetrical_Test" :ta_Symmetrical_Test();
        "Unsymmetrical_Test" :ta_Unsymmetrical_Test();
        default :begin
          $error("The␣test␣you␣selected␣(%s)␣does␣not␣exist!", testName);
          $finish;
        end
      endcase // testName
    end
    else begin
      // Run all tests
      $display("No␣TESTNAME␣set;␣running␣all␣tests.");
      ta_Ideal_Test();
      ta_Symmetrical_Test();
      ta_Unsymmetrical_Test();
    end
    fu_printEndStatus();
  end

  task ta_Reset(); // reset task
    uin_MasterprjTDC.arst = 1'b1;
    @(posedge uin_MasterprjTDC.ck16M);
    uin_MasterprjTDC.arst = 1'b0;
  endtask

  task ta_Init();
    // Intialize input signals
    uin_MasterprjTDC.start = '0;
    uin_MasterprjTDC.stop = '0;
    // Reset
    ta_Reset();
  endtask

// Ideal task assume no variations in the delay segments

  task ta_Ideal_Test();
    real start_10ps[62:0];
    real stop_10ps[62:0];

    real start_20ps[30:0];
    real stop_20ps[30:0];

    real start_40ps[14:0];
    real stop_40ps[14:0];

    real start_80ps[6:0];
    real stop_80ps[6:0];

    real start_160ps[2:0];
    real stop_160ps[2:0];

    real start_320ps;
    real stop_320ps;

    $display("\n--------------------------------------------");
    $display("%t␣-␣%m␣starting␣IDEAL␣TEST", $time);
    $display("--------------------------------------------");
    $display("");
    ta_Init();


// The "uin_MasterprjTDC.start_chain_10ps[i] =
// start_10ps[i]/uin_MasterprjTDC.stop_chain_10ps[i] = stop_10ps[i]"
// sends the generated delay with variations from the testbench to the DUT


```

```
/////////////////////////////////////////////////

  for (int i = 0; i < 63; i++) begin // 10ps Start delay chain
    start_10ps[i] = 135; // Ideal case
    uin_MasterprjTDC.start_chain_10ps[i] = start_10ps[i];
  end


  for (int i = 0; i < 63; i++) begin // 10ps Stop delay chain
    stop_10ps[i] = 125; // Ideal case
    uin_MasterprjTDC.stop_chain_10ps[i] = stop_10ps[i];
  end

///////////////////////////////////////////////////

  for (int i = 0; i < 31; i++) begin // 20ps Start delay chain
    start_20ps[i] = 145; // Ideal case
    uin_MasterprjTDC.start_chain_20ps[i] = start_20ps[i];
  end

  for (int i = 0; i < 31; i++) begin // 20ps Stop delay chain
    stop_20ps[i] = 125; // Ideal case
    uin_MasterprjTDC.stop_chain_20ps[i] = stop_20ps[i];
  end

/////////////////////////////////////////////////////

  for (int i = 0; i < 15; i++) begin // 40ps Start delay chain
    start_40ps[i] = 165; // Ideal case
    uin_MasterprjTDC.start_chain_40ps[i] = start_40ps[i];
  end

  for (int i = 0; i < 15; i++) begin // 40ps Stop delay chain
    stop_40ps[i] = 125; // Ideal case
    uin_MasterprjTDC.stop_chain_40ps[i] = stop_40ps[i];
  end

```

```
///////////////////////////////////////////////////////

  for (int i = 0; i < 7; i++) begin // 80ps Start delay chain
    start_80ps[i] = 205; // Ideal case
    uin_MasterprjTDC.start_chain_80ps[i] = start_80ps[i];
  end

  for (int i = 0; i < 7; i++) begin // 80ps Stop delay chain
    stop_80ps[i] = 125; // Ideal case
    uin_MasterprjTDC.stop_chain_80ps[i] = stop_80ps[i];
  end

////////////////////////////////////////////////////////

  for (int i = 0; i < 3; i++) begin // 160ps Start delay chain
    start_160ps[i] = 285; // Ideal case
    uin_MasterprjTDC.start_chain_160ps[i] = start_160ps[i];
  end

  for (int i = 0; i < 3; i++) begin // 160ps Stop delay chain
    stop_160ps[i] = 125; // Ideal case
    uin_MasterprjTDC.stop_chain_160ps[i] = stop_160ps[i];
  end

  /////////////////////////////////////////////////////

  start_320ps = 445; // Ideal case
  uin_MasterprjTDC.start_chain_320ps = start_320ps;
  stop_320ps = 125; // Ideal case
  uin_MasterprjTDC.stop_chain_320ps = stop_320ps;

  ////////////////////////////////////////////////////
  ////////////////////////////////////////////////////
  ////////////////////////////////////////////////////

    for (int i = 0; i < 640; i++) begin
      @(posedge uin_MasterprjTDC.ck16M);
      @(posedge uin_MasterprjTDC.ck16M);

// generating multipule time measurements by seting Start to one and
// waiting for i (ps) to Set stop to one, the time sequnce will be from 0 to 639
    ↪ ps
      fork
      begin
        uin_MasterprjTDC.start = 1'b1;
      end

      begin
        #(i)
        uin_MasterprjTDC.stop = 1'b1;
      end
    join_any

    @(posedge uin_MasterprjTDC.ck16M);
    @(posedge uin_MasterprjTDC.ck16M);

        //ploting the output data
    $display("Delay_ideal␣=␣%d",i);
    $display("Delay_actual␣=␣␣␣␣%d",uin_MasterprjTDC.Delay_10ps);
    $display("Delay_wallce␣=␣␣␣␣%d",uin_MasterprjTDC.out_wallce );

    ta_Init();

    end
    $display("\n---------------------------------------------------");
    $display("%t␣-␣%m␣complete\n", $time);
    $display("---------------------------------------------------\n");

    #600000
    $stop;
  endtask

// symmetrical task assume same variations in the delay segments
//(same random seeding for both delay line, start and stop)
```

```

task ta_Symmetrical_Test();

  real start_10ps[62:0];
  real stop_10ps[62:0];

  real start_20ps[30:0];
  real stop_20ps[30:0];

  real start_40ps[14:0];
  real stop_40ps[14:0];

  real start_80ps[6:0];
  real stop_80ps[6:0];

  real start_160ps[2:0];
  real stop_160ps[2:0];

  real start_320ps;
  real stop_320ps;

  int seeed_fixed[62:0];
  int seeed[62:0];

  int test;

  $display("\n---------------------------------------------------------");
  $display("%t␣-␣%m␣starting␣Symmetrical␣TEST", $time);
  $display("----------------------------------------------------------");
  $display("");

  ta_Init();
@(posedge uin_MasterprjTDC.ck16M);
@(posedge uin_MasterprjTDC.ck16M);

for (int i = 0; i < 63; i++) begin // create a Fixed seed numbers
  seeed[i] = $urandom(i*231687);
```

```
    seeed_fixed[i] = seeed[i];
  end

//////////////////////////////////////////////////////////

  for (int i = 0; i < 63; i++) begin // 10ps Start delay chain
    start_10ps[i] = $dist_normal( seeed[i],135030,5330);
    start_10ps[i] = start_10ps[i]/1000;
    uin_MasterprjTDC.start_chain_10ps[i] = start_10ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  for (int i = 0; i < 63; i++) begin // 10ps Stop delay chain
    stop_10ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_10ps[i] = stop_10ps[i]/1000;
    uin_MasterprjTDC.stop_chain_10ps[i] = stop_10ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

////////////////////////////////////////////////////////

  for (int i = 0; i < 31; i++) begin // 20ps Start delay chain
    start_20ps[i] = $dist_normal( seeed[i],145500,5800);
    start_20ps[i] = start_20ps[i]/1000;
    uin_MasterprjTDC.start_chain_20ps[i] = start_20ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  for (int i = 0; i < 31; i++) begin // 20ps Stop delay chain
    stop_20ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_20ps[i] = stop_20ps[i]/1000;
    uin_MasterprjTDC.stop_chain_20ps[i] = stop_20ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

////////////////////////////////////////////////////

  for (int i = 0; i < 15; i++) begin // 40ps Start delay chain
    start_40ps[i] = $dist_normal( seeed[i],165100,6790);
    start_40ps[i] = start_40ps[i]/1000;
    uin_MasterprjTDC.start_chain_40ps[i] = start_40ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  for (int i = 0; i < 15; i++) begin // 40ps Stop delay chain
    stop_40ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_40ps[i] = stop_40ps[i]/1000;
    uin_MasterprjTDC.stop_chain_40ps[i] = stop_40ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

////////////////////////////////////////////////////

  for (int i = 0; i < 7; i++) begin // 80ps Start delay chain
    start_80ps[i] = $dist_normal( seeed[i],205460,8790);
    start_80ps[i] = start_80ps[i]/1000;
    uin_MasterprjTDC.start_chain_80ps[i] = start_80ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  for (int i = 0; i < 7; i++) begin // 80ps Stop delay chain
    stop_80ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_80ps[i] = stop_80ps[i]/1000;
    uin_MasterprjTDC.stop_chain_80ps[i] = stop_80ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

//////////////////////////////////////////////////

  for (int i = 0; i < 3; i++) begin // 160ps Start delay chain
    start_160ps[i] = $dist_normal( seeed[i],284970,12780);
    start_160ps[i] = start_160ps[i]/1000;
    uin_MasterprjTDC.start_chain_160ps[i] = start_160ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  for (int i = 0; i < 3; i++) begin // 160ps Stop delay chain
    stop_160ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_160ps[i] = stop_160ps[i]/1000;
    uin_MasterprjTDC.stop_chain_160ps[i] = stop_160ps[i];
  end

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  ////////////////////////////////////////////////

  start_320ps = $dist_normal( seeed[0],446540,21040); // 320ps Start delay chain
  start_320ps = start_320ps/1000;
  uin_MasterprjTDC.start_chain_320ps = start_320ps;

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  stop_320ps = $dist_normal( seeed[0],125010,5510); // 320ps Stop delay chain
  stop_320ps = stop_320ps/1000;
  uin_MasterprjTDC.stop_chain_320ps = stop_320ps;

  for (int i = 0; i < 63; i++) begin
    seeed[i] = seeed_fixed[i];
  end

  ////////////////////////////////////////////////
  ////////////////////////////////////////////////
  //////////////////////////////////////////////////

   for (int i = 0; i < 640; i++) begin
      @(posedge uin_MasterprjTDC.ck16M);
      @(posedge uin_MasterprjTDC.ck16M);

// generating multipule time measurements by seting Start to one
// and waiting for i (ps) to Set stop to one, the time sequnce will be from 0 to
    ↪ 639 ps

      fork
      begin
        uin_MasterprjTDC.start = 1'b1;
      end
      begin
```

```
        #(i)
        uin_MasterprjTDC.stop = 1'b1;
      end
    join_any

    @(posedge uin_MasterprjTDC.ck16M);
    @(posedge uin_MasterprjTDC.ck16M);

        //ploting the output data
    $display("Delay_ideal␣=␣%d",i);
    $display("Delay_actual␣=␣␣␣␣%d",uin_MasterprjTDC.Delay_10ps);
    $display("Delay_wallce␣=␣␣␣␣%d",uin_MasterprjTDC.out_wallce );

    ta_Init();
    end

    $display("\n-----------------------------------------------------");
    $display("%t␣-␣%m␣complete\n", $time);
    $display("-------------------------------------------------------\n");
    #600000
    $stop;
  endtask

// unsymmetrical task assume different variations in the delay segments
//(different random seeding for both delay line, start and stop)

  task ta_Unsymmetrical_Test();

    real start_10ps[62:0];
    real stop_10ps[62:0];

    real start_20ps[30:0];
    real stop_20ps[30:0];

    real start_40ps[14:0];
    real stop_40ps[14:0];

    real start_80ps[6:0];
    real stop_80ps[6:0];

    real start_160ps[2:0];
    real stop_160ps[2:0];

    real start_320ps;
    real stop_320ps;

    int seeed[62:0];

    int test;

    $display("\n-----------------------------------------------------");
    $display("%t␣-␣%m␣starting␣Unsymmetrical␣TEST", $time);
    $display("------------------------------------------------------");
    $display("");

    ta_Init();

  @(posedge uin_MasterprjTDC.ck16M);
  @(posedge uin_MasterprjTDC.ck16M);

  for (int i = 0; i < 63; i++) begin // create a Fixed seed numbers
    seeed[i] = $urandom(i*457);
  end

///////////////////////////////////////////////////////

  for (int i = 0; i < 63; i++) begin // 10ps Start delay chain
    start_10ps[i] = $dist_normal( seeed[i],135030,5330);
    start_10ps[i] = start_10ps[i]/1000;
    uin_MasterprjTDC.start_chain_10ps[i] = start_10ps[i];
  end

  for (int i = 0; i < 63; i++) begin // 10ps Stop delay chain
    stop_10ps[i] = $dist_normal( seeed[i],125010,5510);
```

```
    stop_10ps[i] = stop_10ps[i]/1000;
    uin_MasterprjTDC.stop_chain_10ps[i] = stop_10ps[i];
  end

////////////////////////////////////////////////////


  for (int i = 0; i < 31; i++) begin // 20ps Start delay chain
    start_20ps[i] = $dist_normal( seeed[i],145500,5800);
    start_20ps[i] = start_20ps[i]/1000;
    uin_MasterprjTDC.start_chain_20ps[i] = start_20ps[i];
  end

  for (int i = 0; i < 31; i++) begin // 20ps Stop delay chain
    stop_20ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_20ps[i] = stop_20ps[i]/1000;
    uin_MasterprjTDC.stop_chain_20ps[i] = stop_20ps[i];
  end

////////////////////////////////////////////////////

  for (int i = 0; i < 15; i++) begin // 40ps Start delay chain
    start_40ps[i] = $dist_normal( seeed[i],165100,6790);
    start_40ps[i] = start_40ps[i]/1000;
    uin_MasterprjTDC.start_chain_40ps[i] = start_40ps[i];
  end

  for (int i = 0; i < 15; i++) begin // 40ps Stop delay chain
    stop_40ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_40ps[i] = stop_40ps[i]/1000;
    uin_MasterprjTDC.stop_chain_40ps[i] = stop_40ps[i];
  end

/////////////////////////////////////////////////////

  for (int i = 0; i < 7; i++) begin // 80ps Start delay chain
    start_80ps[i] = $dist_normal( seeed[i],205460,8790);
    start_80ps[i] = start_80ps[i]/1000;
    uin_MasterprjTDC.start_chain_80ps[i] = start_80ps[i];
  end

  for (int i = 0; i < 7; i++) begin // 80ps Stop delay chain
    stop_80ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_80ps[i] = stop_80ps[i]/1000;
    uin_MasterprjTDC.stop_chain_80ps[i] = stop_80ps[i];
  end

//////////////////////////////////////////////////

  for (int i = 0; i < 3; i++) begin // 160ps Start delay chain
    start_160ps[i] = $dist_normal( seeed[i],284970,12780);
    start_160ps[i] = start_160ps[i]/1000;
    uin_MasterprjTDC.start_chain_160ps[i] = start_160ps[i];
  end

  for (int i = 0; i < 3; i++) begin // 160ps Stop delay chain
    stop_160ps[i] = $dist_normal( seeed[i],125010,5510);
    stop_160ps[i] = stop_160ps[i]/1000;
    uin_MasterprjTDC.stop_chain_160ps[i] = stop_160ps[i];
  end

  ////////////////////////////////////////////////////////

  start_320ps = $dist_normal( seeed[0],446540,21040); // 320ps Start delay chain
  start_320ps = start_320ps/1000;
  uin_MasterprjTDC.start_chain_320ps = start_320ps;

  stop_320ps =$dist_normal( seeed[0],125010,5510); // 320ps Stop delay chain
  stop_320ps = stop_320ps/1000;
  uin_MasterprjTDC.stop_chain_320ps = stop_320ps;

  /////////////////////////////////////////////////////////
  /////////////////////////////////////////////////////////
  /////////////////////////////////////////////////////////

  for (int i = 0; i < 640; i++) begin

     @(posedge uin_MasterprjTDC.ck16M);
     @(posedge uin_MasterprjTDC.ck16M);

// generating multipule time measurements by seting Start to one
// and waiting for i (ps) to Set stop to one, the time sequnce will be from 0 to
   ↪  639 ps

     fork
     begin
       uin_MasterprjTDC.start = 1'b1;
     end
     begin
       #(i)
       uin_MasterprjTDC.stop = 1'b1;
     end
    join_any

    @(posedge uin_MasterprjTDC.ck16M);
    @(posedge uin_MasterprjTDC.ck16M);

       //ploting the output data
    $display("Delay_ideal␣=␣%d",i);
    $display("Delay_actual␣=␣␣␣␣%d",uin_MasterprjTDC.Delay_10ps);
    $display("Delay_wallce␣=␣␣␣␣%d",uin_MasterprjTDC.out_wallce );

    ta_Init();
    end

    $display("\n---------------------------------------------");
    $display("%t␣-␣%m␣complete\n", $time);
    $display("---------------------------------------------\n");

    #600000
    $stop;
  endtask

  function void fu_printEndStatus;
    $display("");
    $display("");
    $display($time, "␣ns:");
    $display("-----------------------------------------------");
    $display("------------------------------------------------");
    $display("");
    $display("");
    $display("");
    $display("");
    $display("--------------------------------------------");
    $display("-------------------------------------------------");
    $display("");
    $display("");
  endfunction

endmodule
```

Listing C.5: Wallace Tree Encoder

```
module WallaceEncoder #( // A standard parameterizable Wallce tree encoder using a standard fulladders and 7to3 Wallce tree encoder
                        //number of bits must be bigger than 3
  parameter NOB,
  parameter DK // remember this is for the carry in bit for the output adders the first one should be (2**NOB)-2
)(

  input logic [(2**NOB)-2:0] in_thermo,
  output logic [NOB-1:0] out_delay
  );

  localparam NEW_NOB = NOB-1;

  logic [(NOB-1):0] a = '0;
  logic [(NOB-1):0] b = '0;
  logic [(NEW_NOB-2):0] cout = '0;

  if (NOB > 3) begin

  FullAdder #()
  u_fulladder_0(
  .a (a[0]),
  .b (b[0]),
  .ci (in_thermo[DK]),
  .s (out_delay[0]),
  .co (cout[0])
  );

  for (genvar i=1 ; i<(NEW_NOB-1) ; i++) begin // using recursive function to
      ↪ call the same code to generate an upper and lower branches for the
      ↪ encoder
    FullAdder #()
    u_fulladder(
    .a (a[i]),
    .b (b[i]),
    .ci (cout[i-1]),
    .s (out_delay[i]),
    .co (cout[i])
    );
  end

  FullAdder #()
  u_fulladder_last(
  .a (a[(NEW_NOB-1)]),
  .b (b[(NEW_NOB-1)]),
  .ci (cout[(NEW_NOB-2)]),
  .s (out_delay[(NEW_NOB-1)]),
  .co (out_delay[(NEW_NOB)])
  );

  if (NEW_NOB > 3) begin
    WallaceEncoder #(
      .NOB(NEW_NOB),
      .DK ((DK-2)/2)
      )
    u_UpperTreeEncoder( // upper tree for the most significant bits in the
        ↪ thermomter code
      .in_thermo (in_thermo[DK-1:(2**NEW_NOB)-1]),
      .out_delay (a)
      );
    WallaceEncoder #(
      .NOB(NEW_NOB),
      .DK ((DK-2)/2)
      )
    u_LowerTreeEncoder( // lower tree for the least significant bits in the
        ↪ thermomter code
      .in_thermo (in_thermo[(2**NEW_NOB)-2:0]),
      .out_delay (b)
      );

    end else if (NEW_NOB == 3) begin // generate a stander 7to3 bits Wallce tree
        ↪ encoder

    WallaceEncoder7to3 #()
    u_UpperBlock(
      .arst (0),
      .enable (1),
      .in (in_thermo[DK-1:(2**NEW_NOB)-1]),
      .out (a)
      );

    WallaceEncoder7to3 #()
    u_LowerBlock(
      .arst (0),
      .enable (1),
      .in (in_thermo[(2**NEW_NOB)-2:0]),
      .out (b)
      );
    end
```

```
end

endmodule
```

Listing C.6: 7to3 bits Wallace Tree Encoder

```
module WallaceEncoder7to3 #( // Simple 7to3 bits Wallce tree encoder using 4
    ↪ standard full adders
                             // which is used to implement a parameterizable
                                 ↪ Wallce tree encoder
)(
  input logic arst,
  input logic enable,
  input logic [6:0] in,
  output logic [2:0] out
  );

  logic [1:0] sum;
  logic [3:0] carryout;

      FullAdder #()
      fulladder_1(
      .a (in[0]),
      .b (in[1]),
      .ci (in[2]),
      .s (sum[0]),
      .co (carryout[0])
      );

      FullAdder #()
      fulladder_2(
      .a (in[3]),
      .b (in[4]),
      .ci (in[5]),
      .s (sum[1]),
      .co (carryout[1])
      );

    FullAdder #()
    fulladder_3(
    .a (sum[0]),
    .b (sum[1]),
    .ci (in[6]),
    .s (out[0]),
    .co (carryout[3])
    );

    FullAdder #()
    fulladder_4(
    .a (carryout[0]),
    .b (carryout[1]),
    .ci (carryout[3]),
    .s (out[1]),
    .co (out[2])
    );

endmodule
```