The Efficient Design of Time-to-Digital Converters

By

SCOTT TANCOCK

Department of Electrical and Electronic Engineering
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Engineering.

DECEMBER 2021

Word count: 50786

Abstract

Time-to-Digital Converters (TDCs) perform the conversion (sampling and quantisation) of time periods into digital (binary) numbers and are vital for Time-of-Flight (ToF) systems such as RADAR, LiDAR, SONAR and Ultrasonics. They are also vital in frequency-locking applications such as All-Digital Phase-Locked Loops (ADPLLs) and Delay-Locked Loops (DLLs), quantum applications such as Quantum Key Distribution (QKD), Single-Photon LiDAR / Single-Photon Time-of-Flight (SPToF) and coincidence counting.

The literature on TDCs is fragmented, with research clustering around specific applications and little cross-pollination of ideas between them. Therefore, I reviewed all these areas for novel TDC architectures and found several new designs including algorithmic, successive approximation and wave union TDCs. Most designs were based on Application Specific Integrated Circuits (ASICs) due to the hardware flexibility, with the rest on Field Programmable Gate Arrays (FPGAs), all of which have been verified.

One of the most promising hardware architectures for TDCs are FPGAs as they are of a lower cost than ASICs. I have presented the first TDC implementation on the FPGA's DSP blocks and resolved the extreme non-linearity in the DSP blocks with multisampling techniques to produce an effective delay line with performance comparable to carry chains (13.60 ps single-shot precision). DSP delay lines avoid the use of general-purpose fabric, allowing larger quantities of channels or more applications to be integrated on to a single device.

The long bubbles caused by applying the wave union multisampling technique were unable to be corrected by existing bubble correctors. Therefore, a new hardware bubble corrector, which operates at line rate with zero dead time (130 MHz, 144 bits/cycle on my TDCs), was designed and tested.

Acknowledgements

I could not have completed this PhD project without the help of many individuals and organisations, and so I would like to thank them for their support. If any of these people were missing, this project would not have succeeded as it did.

Firstly, to the Engineering and Physical Sciences Research Council (EPSRC) and the University of Bristol for providing the funding, resources, and stipend to complete this project. It's hard to perform good research without suitable equipment, spaces, and daily necessities, so this project could not have been completed without them.

Second, to my supervisor, Professor Naim Dahnoun. He has been instrumental in pushing this PhD forward, improving the quality of my written work and communication, discussing problem points and issues with my designs, and helping me focus on the bigger picture. Without him, this PhD would never have reached completion and my papers would likely not have been accepted for publication.

Third, to my co-supervisor, Professor John Rarity. He has been very helpful in providing equipment, funds, opportunities to network with other researchers and perspective on my topic and its applications. It is because of him that my research has been able to create an impact on other engineering and scientific fields rather than being purely confined to the realms of the theoretical.

Fourth, to my interim assessor Dr Krishna Coimbatore Balram for his guidance on progressing through the PhD. His input during our yearly reviews was indispensable for keeping the PhD focused and on-track, having a plan for progression and submission and understanding the PhD assessment process.

Fifth, to Dr. Jose Nunez-Yanez for his assistance related to FPGA implementation, routing, and toolchains. The architectures of FPGAs and tools to configure them can often be outright cryptic, and his deep knowledge on the matter really helped me move on with my research rather than scratching my head at the tools.

Sixth, to Dr. Dinesh Pamunuwa for his assistance related to ASIC design and manufacture. A large quantity of TDC literature is inherently intertwined with ASIC design and manufacture, a lot of which would have been very difficult to understand if not for Dinesh's expertise on VLSI design. He was a great resource for understanding the more complex designs and physical effects encountered in TDC papers.

Seventh, to my co-researcher Ekin Arabul for the time spent discussing TDC designs and implementation aspects. TDC papers, word limited as they are, often do not explore the tiny details and common errors in implementation, so having another researcher whom I could discuss these matters with was invaluable. I hope he was able to benefit as much from my presence as I did from his.

Eighth, to Rich Walker and the staff in the engineering workshop who helped me bring physical aspects of my designs to reality, including PCBs, protective covers, and cabling. These matters can seem minor in the heat of the moment but can often be the difference between a reliable experimental setup and one that only works when the planets are aligned.

Ninth, to the professional services and technical services staff in the University who helped with administrative and organisational tasks such as booking flights and conferences, organising desk allocations and equipment procurement. Again, these can seem like minor matters individually, but without them no research could be done.

Tenth, to the various researchers I met, talked with, and listened to at conferences, as well as the reviewers of the papers I have submitted for publication. Your input and perspective inspired me to question my own work more deeply, to be more rigorous in my assertions, and to examine the problem from angles I would never have considered. I could not have become a competent researcher without the guidance of the many people I interacted with in the publication process and while presenting at conferences.

Finally, I would like to thank the developers of the various open-source tools I use daily to turn my ideas into reality. From large projects such as the Linux Operating System, Git, LaTeX, Golang and the GNU Project, to small ones such as Sublime Text, Inkscape and Krita, these tools are provided for free by developers with spare time and generosity without any remuneration, so I would like to thank them for making these excellent tools available to the community.

In addition to those who helped me in a professional capacity, I would like to thank those who helped me in a personal capacity.

I would like to thank my parents for their emotional support, providing accommodation and general tolerance of my unconventional work schedule. Despite the stresses of the PhD and the many things in their own lives, they have taken the time to help me in more ways than I could ever imagine.

Also, I would like to thank my brothers for their emotional support and assistance with proofreading and critical opinions. They have been incredibly helpful in ensuring that the text I've written and diagrams I've drawn are of a suitable quality, easy to understand and error-free.

Finally, my friends inside and outside the University, including Jamie Marsden, Alex Stanoev, Elliott Worsey, Roget Kou, Charles Khoury, Abdu Elturki, Alex Gregory and Ed Nutting. As well as their emotional support and willingness to let me bounce ideas off them, they have helped me to take a step back from my work and unwind when I've needed to, helping me avoid burnout for the better part of five years.

Without these people, this project would not have been possible, and so they have my heartfelt gratitude for everything they have done for me over the past four and a half years. I hope that one day I can return the favour.

AUTHOR'S DECLARATION

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .... DATE: ....

Table of Contents

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Research Motive

Time-to-Digital Converters (TDCs) are relatively obscure compared to their analog counterparts, Analog-to-Digital Converters (ADCs), due to the relative lack of time-domain systems to which they can be applied. However, this has changed in recent years. Light Detection And Ranging (LiDAR) systems have seen a surge in popularity due to the advent of autonomous ("self-driving") cars, improvements in pulsed lasers and Single-Photon Avalanche Diodes (SPADs) have allowed Quantum Key Distribution (QKD) to become a reality, Phase-Locked Loops (PLLs) are being replaced by All-Digital PLLs (ADPLLs) and many designers are looking to Time-Mode Signal Processing (TMSP) to solve problems that are not tractable with analog systems due to their poor area scaling. As these applications see more widespread adoption, so too will TDCs, and thus TDC designs need to be available to satisfy the performance and cost requirements of these applications.

The current literature on TDCs can best be described as fragmented. Communities have developed around specific applications such as Time-of-Flight (ToF), frequency-locking (such as ADPLLs) and quantum, with very little interaction between these communities. Some researchers are approaching TDCs from a more theoretical perspective, and interacting with many of these communities, but the penetration of ideas is not very high. Therefore, one of the main goals of this PhD project is to aggregate designs from across the literature and communities, compare and summarise the available architectures, and provide recommendations according to performance requirements.

TDCs are often costly components due to the specific hardware requirements for implementing them. As we will see in later sections, the hardware that can be used for TDC operation is quite limited. If the hardware is expensive, then the cost of integrating a TDC into a system will also be high. Therefore, the other main goal of this PhD project is to develop new TDC architectures which can be used to target unique cost-performance-complexity trade-offs and enable TDCs to be applied to new areas.

1.2 Aims and Goals

The key aims and goals of this PhD project are:

- To identify applications to which TDCs can be applied.
- To identify and define metrics by which TDCs can be compared.
- To review TDC architectures present in the literature, compare them in terms of defined metrics, and make recommendations for the identified applications.
- To develop new TDC architectures with alternative characteristics to those currently available in the literature.
- To identify methods to combine TDC architectures available in the literature to achieve improved characteristics.

1.3 List of Contributions

Table 1.1: List of contributions.

Publication	My Contribution	Advancement of the State-of-the-Art
S. Tancock and N. Dahnoun, “Digital Signal Processing Systems: Choosing a Processor,” in 2020 International Conference Engineering and Telecommunication (En&T), Nov 2020, Plenary Talk.	Details on CPU, GPU, FPGA and microcontroller architectures.	Summarises all the different architectures for custom algorithms and makes suggestions based on performance, cost and power.
S. Tancock, E. Arabul, and N. Dahnoun, “A Review of New Time-to-Digital Conversion Techniques,” IEEE Transactions on Instrumentation and Measurement, vol. 68, no. 10, pp. 3406–3417, 2019.	Review of all current literature, selection of techniques to include, comparison and summarisation of techniques.	Brought current review literature up to date with the inclusion of Successive Approximation, Algorithmic, Wave Union, SERDES and DSP Delay Line TDCs. Also summarises available calibration and linearisation techniques.
S. Tancock, J. Rarity, and N. Dahnoun, “Developments in Time-to-Digital Converters during 2020,” in 5th International Nordic-Mediterranean Workshop on Time-to-Digital Converters and Applications NoMe–TDC 2021, 2021.	Discovery, selection, comparison, summarisation and evaluation of papers to be included.	Summarises the 14 new TDC architecture papers, 4 new TDC application papers, and 4 review and analysis papers published in 2020.

Continued on next page

Table 1.1: List of contributions. (Continued)

S. Tancock, E. Arabul, N. Dahnoun, and S. Mehmood, “Can DSP48A1 adders be used for high-resolution delay generation?” in 2018 7th Mediterranean Conference on Embedded Computing (MECO), IEEE. Institute of Electrical and Electronics Engineers (IEEE), Aug 2018, pp. 1–6.	Investigation of the internal delay structure of the DSP48A1 blocks, collating and digesting results.	Provided an initial investigation into the feasibility of DSP blocks as delay generators.
S. Tancock and N. Dahnoun, “A 5.25 ps-resolution TDC on FPGA using DSP blocks,” in Proceedings of the Digital Image & Signal Processing’19 conference, October 2019.	Creation of the TDC core and read-out logic, collating and digesting data, speculation on the internal structure of the DSP block.	Produced a functional TDC using the FPGA’s DSP blocks instead of general purpose logic, reducing logic utilisation and providing a high-resolution TDC.

Continued on next page

Table 1.1: List of contributions. (Continued)

S. Tancock, J. Rarity, and N. Dahnoun, “Temperature Characterisation of the DSP Delay Line,” in 5th International Nordic-Mediterranean Workshop on Time-to-Digital Converters and Applications NoMe–TDC 2021, 2021.	TDC design, experimental set-up and temperature characterisation.	Significantly improved the characterisation of the DSP blocks as delay generators, providing metrics which were missing from previous papers as well as temperature sensitivity characterisation. Lent further evidence to our speculations on the internal structure of the DSP48E1 post-adder.
S. Tancock, J. Rarity, and N. Dahnoun, “The Wave-Union Method on DSP Blocks: Improving FPGA-based TDC resolutions by 3x with a 1.5x area increase,” IEEE Transactions on Instrumentation and Measurement (Accepted), 2021.	TDC design and data collation and digestion	Significantly reduced the DSP Delay Line’s area requirements using the Wave Union technique with an FSR injector.

Continued on next page

Table 1.1: List of contributions. (Continued)

S. Tancock, J. Rarity, and N. Dahnoun, "A Long-Range Hardware Bubble Corrector Technique for Short-Pulse-Width and Multiple-Registration Encoders," in 5th International Nordic-Mediterranean Workshop on Time-to-Digital Converters and Applications NoMe–TDC 2021, 2021.	Algorithm and hardware design, proof of correctnesss.	Created a new bubble corrector in hardware that operates with zero dead time and is capable of correcting multiple large bubbles which are infeasible with traditional bubble correction techniques.
S. Tancock, J. Rarity, and N. Dahnoun, "Improving TDC Resolution with the Multi-Chain Vernier Method," IEEE Access (submitted), 2021.	TDC design and simulation.	Proof of concept showing that multi-chain techniques may be possible on Vernier Delay Lines.
E. Arabul, S. Paesani, S. Tancock, J. Rarity, and N. Dahnoun, "A Precise High Count-Rate FPGA Based Multi-Channel Coincidence Counting System for Quantum Photonics Applications," IEEE Photonics Journal, vol. 12, no. 2, pp. 1–14, 2020.	Delay line optimisations, dual-clock multisampling, triggering logic design and design debugging	Demonstrated a 320 Mcps, 8.9 ps RMS, 8-channel coincidence counter which is much higher performance than competing 8-channel designs and operates beyond the saturation rate and significantly below the timing noise floor of the single-photon avalanche diodes.

1.3.1 Comparison between the Literature and My Contributions

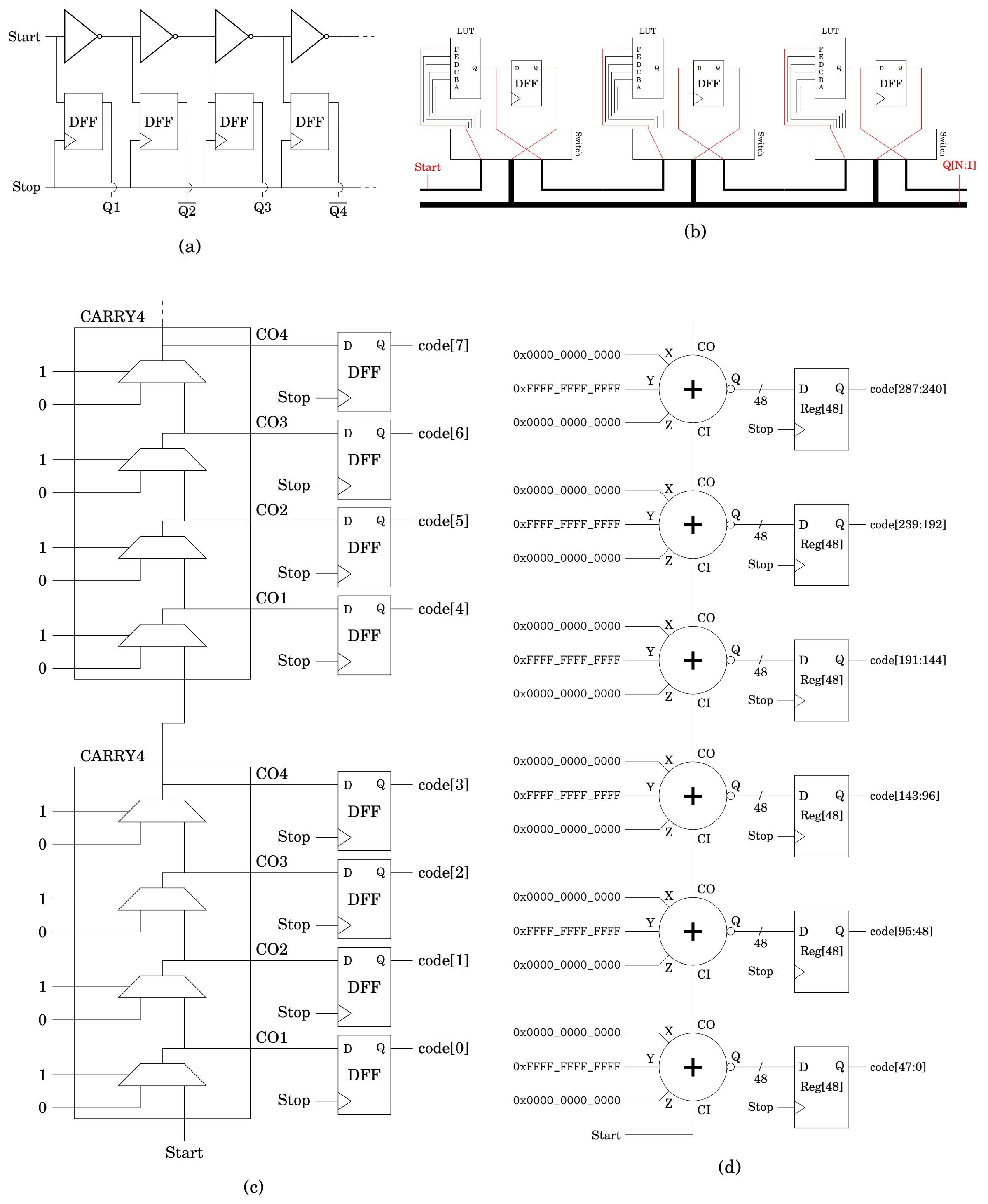
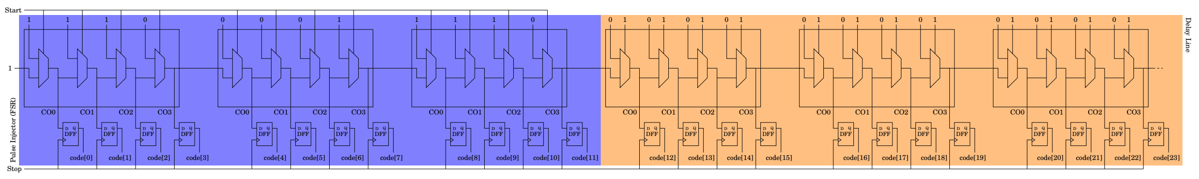
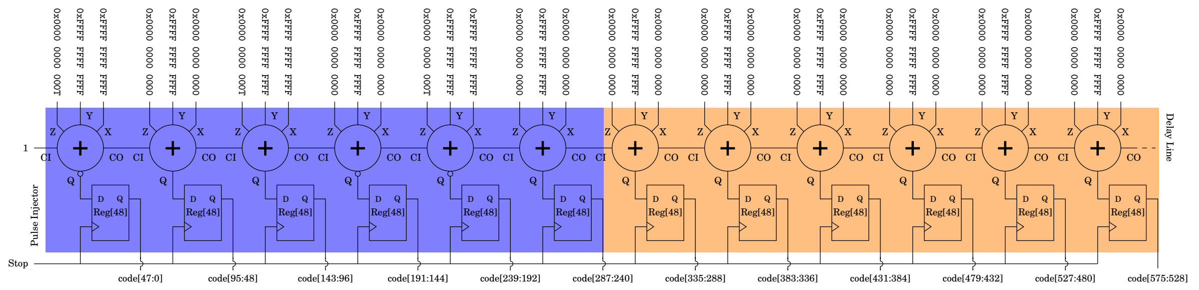


Figure 1.1: (a) A basic delay line on ASICs or Discrete Logic; (b) a basic LUT delay line on an FPGA; (c) state-of-the-art: a carry chain based delay line; (d) my contribution: a DSP block based delay line.



(a)



(b)

Figure 1.2: (a) Current state-of-the-art: the wave union method on carry chains; (b) my contribution: the wave union method on DSP blocks (T = Start).

Table 1.2: Comparison of DSP and carry chain delay lines.

Technique	Device	Resource	Resolution (average)	Utilisation (10 ns clock)
Delay Line	Spartan-6 FPGA	6-input look-up table (LUT6)	20 ps	128 CLBs
Delay Line	Artix-7 FPGA	6-input look-up table (LUT6)	15 ps	170 CLBs
DSP Delay Line	Spartan-6 FPGA	DSP Block (DSP48A1)	16.7 ps	13 DSPs
DSP Delay Line	Artix-7 FPGA	DSP Block (DSP48E1)	9.8 ps	22 DSPs
4x DSP Delay Line	Artix-7 FPGA	DSP Block (DSP48E1)	5.25 ps	74 DSPs
8x DSP Delay Line	Artix-7 FPGA	DSP Block (DSP48E1)	3.70 ps	146 DSPs

1.4 Structure of this Thesis

In the following sections, the applications of TDCs and metrics by which we evaluate them will be explained. Chapter 2 is the next chapter and describes the different hardware available for a possible implementation of a TDC. It will describe the benefits and drawbacks of various technologies before diving into more detail on Field Programmable Gate Arrays, which have been the main focus of this PhD project. After that, Chapter 3 will present a detailed review of TDC architectures in the available literature, showing the performance-cost trade-offs that are already available to designers. Related to this chapter are the first two major contributions of this PhD project, a review paper published in IEEE Transactions on Instrumentation and Measurement [8] and another published in the Nordic Mediterranean Time-to-Digital Converter Workshop (NoMeTDC) [9]. Then, Chapter 4 will describe the third major contribution of this PhD project, a new TDC architecture implemented on an FPGA's DSP blocks [10–13]. After that, Chapter 6 will describe the bubble correction logic designed for use in high-performance TDCs on DSP blocks [14]. This logic solves a major issue with the DSP block TDCs, namely the long bubbles that occur in the thermometer codes. Next, Chapter A will describe the final major contribution of this PhD project: the Multi-Chain Vernier TDC architecture [15]. Finally, Appendix B will detail other contributions I have made to the wider field of Electronic Engineering.

1.5 Applications

Time-to-digital converters (TDCs) play a vital role in almost all computational systems in existence. From their appearance in Phase-Locked Loops (PLLs), where they measure the difference between the loop and the reference clock to avoid clock drift, to Time of Flight (ToF) applications where the time between an emission and reception is measured to discover information about an object from which the signal was reflected or the environment through which the signal passed. In addition, there are also quantum versions of these applications, where the signal is a single quantum, and the PLL or time-of-flight measurement must perform well despite some quanta being lost in-flight. They also make an appearance in medical imaging, as some systems such as Positron Emission Tomography (PET) and Fluorescence Lifetime Imaging (FLIM) use the ToF or absorption time of tissues or substances to form an internal image of a complex structure such as a human body.

Table 1.3: Comparison of performance requirements for TDC applications.

Application	SSP	Dead Time	DNL (LSB)	INL (LSB)	Range	Channels	Power
LiDAR (automotive, single-point, forward-facing)	< 100 ps	< 10 ns	< 1	< 10	> 1 us[1]	1	< 10 W
LiDAR (automotive, single-point, all-around)	< 100 ps	< 1 ns	< 1	< 10	> 500 ns	1	< 50 W
LiDAR (automotive, multi-point, all-around)	< 100 ps	< 10 ns	< 1	< 10	> 500 ns	≥ 8	< 50 W
Coincidence Counting (Positron Emission Tomography)	< 10 ps	< 10 ns	< 5	< 20	> 10 ns	≥ 2	–[2]
Coincidence Counting (Quantum Photonics)	< 10 ps	< 10 ns	< 5	< 20	> 100 ns	≥ 4	< 10 W[3]
Spectrometry (Time-of-Flight Mass Spectrometry)	< 10 ps	–[2]	< 5	< 20	> 10 ms	1	–[2]
Spectrometry (Fluorescence Lifetime Imaging)	< 50 ps	< 1 ns	< 5	< 20	> 100 ns	1	–[2]
Frequency Locking (All-Digital Phase-Locked Loops)	< 300 fs	–[2]	–[2]	< 5	> 1 ns[4]	1	< 1 mW
Frequency Locking (Quantum Key Distribution)	< 50 ps	< 10 ns	< 5	< 20	> 1 us	> 2	< 10 mW

[1] > 150 m maximum distance.
[2] Not a major concern.
[3] Due to thermal noise.
[4] Dependent on clock period.

1.5.1 Coincidence Counting

Time-to-digital converters can be also utilised as time taggers in time correlation systems such as coincidence counters. These systems play an essential role in quantum physics experiments for gating the events of interest from the background noise and measuring the gamma ray correlation in PET systems. Coincidence counters are correlator tools which are tailored to measure the occurrences of simultaneous signal events over multiple channels. This is done by checking whether the events are happening within the same time window called the coincidence window.

In a PET scanner setup, a positron emitting radiotracer substance is introduced into the subject's body and the subject is surrounded by detectors which observe gamma rays. As the positron-emitting substance decays inside the patient's body, positrons meet with electrons, which results in annihilation of both the positron and electron. An annihilation of a positron and electron pair generates two gamma rays that travel in opposite directions towards two photon detectors placed in the surroundings. Counting the coincidences caused by the two gamma rays emitted allows analysis of the radiotracer's distribution in the body, which is then used for image formation. TDCs can be utilised to digitise gamma ray pairs' times of flight in such a set-up. Examples of such a scheme can be found in [16–20].

1.5.2 Spectrometry

Another area where TDCs are commonly used is spectrometry. Spectrometry can be defined as distinguishing mixed substances based on an interaction between light and their matter. Common spectrometry examples where TDCs can be used are Time of Flight Mass Spectrometry (TOFMS) and fluorescence spectrometry. In TOFMS, the times of flight of ions are used to measure the ions' mass to charge ratio. This process starts with ionising the atoms and molecules to be measured causing the required number electrons to be knocked off to form a positive ion. Then, the ions are accelerated to the same kinetic energy and projected for a known distance. Since heavier and lighter ions have different velocities due to their different masses, their time of flight will vary, and this reveals information about their charge to mass ratio [21]. ToF tomography is an example of mass spectrometry.

Fluorescence spectroscopy, which is commonly used in chemistry, biomedicine and medicine to analyse organic compounds [22], is used to determine the fluorescence content of the substance by measuring the decay time after the substance has been excited by a light beam. TDCs are employed as a part of the Time-Correlated Single Photon Counting (TCSPC) tools used to measure the decay time [23].

1.5.3 Rangefinding

Rangefinding ToF systems such as LiDARs are another common application of TDCs. In a typical LiDAR, the START signal corresponds to the time when the laser transmitter starts to illuminate the target with photons and the detection of reflected photons from the target by the receiver is denoted as the STOP signal. The differences between these values are used to determine the photons' time-of-flight between transmission and detection, and the photons' time-of-flight can be used to measure the distance. A TDC is employed to quantise and digitise the events of START and STOP signals in such applications.

For a photon or pulse of light travelling through free space, the speed of the photon is c. The distance to the object in question is d and the object is at an angle (θ,ϕ) (vertical, horizontal) from the transmitter. The time of emission is T_{st}, and the time of reflected pulse/photon reception is T_{sp}. Therefore, the distance d will be as in (1.1), noting that the pulse must complete a round trip ($2d$) between the start and stop. Then, the cartesian coordinates of the object (x,y,z) can be calculated as in (1.2) relative to the transmitter, with the x and z axes being perpendicular to the transmitter (plane of projection) and the y axis being parallel to the transmitter (into the plane). A visualisation of the cartesian (x,y,z) and spherical (d,θ,ϕ) coordinate systems with respect to a ToF sensor can be seen in Fig.1.3.

$$d = \frac{c(T_{sp} - T_{st})}{2} \tag{1.1}$$

$$\begin{aligned} x &= d\ cos\phi\ sin\theta \\ y &= d\ sin\phi\ sin\theta \\ z &= d\ cos\theta \end{aligned} \tag{1.2}$$

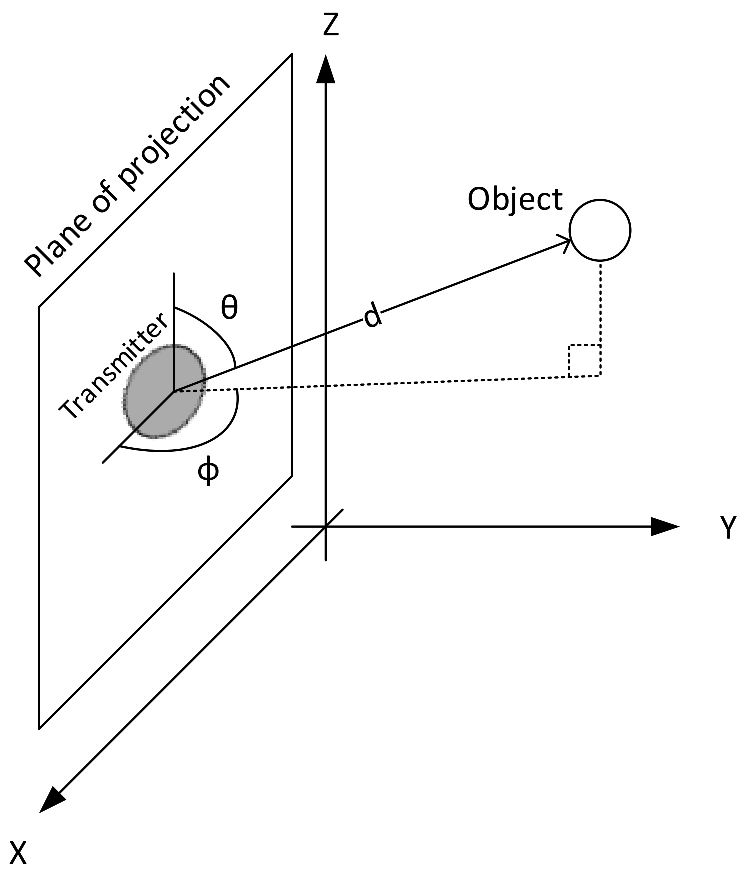


Figure 1.3: Visualisation of a Time-of-Flight (ToF) system, showing the cartesian as well as spherical coordinate systems.

As the time resolution of single-photon detectors is on the order of 100 ps [24], the desired resolution and precision of a TDC is approximately 10 ps. On the other hand, readily available Avalanche PhotoDiodes (APDs) can easily reach < 10 ps rise time error [25], so a TDC resolution and precision of < 1 ps is desirable to obtain the highest accuracies. The Differential Non-Linearity (DNL) must also be low enough to avoid significant mismeasurements if the largest code is hit by coincidence. For older PET detectors, the Full-Width Half-Maximum (FWHM) is on the order of 10 ps [20], so a detector resolution of 1 ns, achievable with a counter or multi-phase clock, is acceptable. However, for newer PET systems utilising Silicon Photomultipliers (SiPMs) and aiming to directly measure positron emissions without the aid of tomographic inversion, a resolution of 10 ps is desirable to obtain sub-millimeter accuracy [20].

1.5.4 Frequency Locking

Traditionally, frequency locking has been performed using phase-locked loops (PLLs). These are analog systems composed of a ring oscillator (RO), phase detector (PD), loop filter (LF) and charge pump (CP) [26, 27]. In normal operation, the ring oscillator will oscillate at a frequency F_{osc} somewhat similar to the input frequency F_{in}. The rising edge of the output of the RO will be compared to the rising edge of the input signal by the phase detector. If the edge of the RO is before the edge of the input, then the PD determines that the RO is running too fast ($F_{osc} > F_{in}$) and thus will instruct the CP to reduce the frequency. However, if the edge of the RO is after the edge of the input, then the PD determines that the RO is running too slow ($F_{osc} < F_{in}$) and thus will instruct the CP to increase the frequency. This mode of operation will also cause the phase of the oscillator Φ_{osc} to become aligned with the phase of the input signal Φ_{in}, since a phase mismatch would be seen as a frequency mismatch.

The charge pump controls the current available to the RO via current-starving transistors, thereby adjusting its period and frequency. Specifically, the charge pump increases and decreases the voltage at the gate of the current-starving transistors, which decreases or increases the source-drain resistance of the current-starving transistors respectively. If the resistance increases (the voltage at the gate decreases), less current will be available to the RO for the purpose of charging and discharging the internal node capacitance at each stage of the oscillator, and so the frequency will decrease. Conversely, if the resistance decreases (voltage at the gate increases), there will be more current available to the RO for charging and discharging node capacitances, and therefore the frequency will increase. By adjusting the voltage available at the gate, the charge pump can modify the RO frequency F_{osc}.

When a basic PLL reaches a steady state ($F_{osc} \approx F_{in}$ and $\Phi_{osc} \approx \Phi_{in}$), there is a tendency for F_{osc} to oscillate about F_{in} due to overshooting the adjustment to be made to reach the correct frequency, or due to minor perturbations in the phase. Therefore, most PLLs implement a low-pass filter as their LF to dampen changes in the frequency. This allows F_{osc} to become much closer to F_{in}, reduces the frequency at which F_{osc} varies and reduces the sensitivity of the PLL to noise.

However, the sensitive analog components present in PLLs do not scale efficiently to smaller process nodes [28]. As the process node gets smaller, PVT variations become more prominent relative to the smallest possible element. These variations reduce the performance and reliability of the PLL, and so much larger components with lower PVT variations are needed. The result of this is that the PLL is becoming a more significant aspect in chip area and power consumption, thus it needs to be optimised.

All-digital phase-locked loops (ADPLLs) operate similarly to PLLs but replace the analog components with digital ones wherever possible. The advantages of this are the improved power and area scaling to smaller process nodes as well as the addition of digital processing techniques that improve the performance of the PLL. However, this comes at the cost of design complexity. With an ADPLL, the PD, which can be relatively simple, is replaced with a high-precision low-range TDC, the CP is replaced by a digital-to-analog converter (DAC) and the LF is replaced by a digital filter. The RO and current-starving transistors often stay the same, although it is possible to replace the current-starving transistors with a digitally controlled capacitance to adjust the frequency of the RO. As the components are now digital, they can be scaled down to minimum size without compromising performance.

Key to this implementation is the TDC. The TDC has the purpose of determining the phase relationship between the RO output and the input frequency. Therefore, the TDC must be bidirectional (allowing negative delays, where the stop occurs before the start), high-precision (to accurately determine the phase relationship) and low-cost (minimal power and area as the ADPLL does no useful work for the user). As the clock period is often short, the range of the TDC can be short, and as the LF reduces the locking speed to improve noise immunity, the sample rate does not have to be high (TDC dead time can be relatively long compared to other applications).

1.5.5 Health and Monitoring

Monitoring system and environment parameters and health is important to ensure the longevity of a system and avoid malfunction or misuse. Occasionally, a TDC will be used in a monitoring application, usually with the aim of converting a pulse width to binary. For example, in [29], Chen et. al. employed a latching TDC for monitoring PLL health. The TDC should always give approximately the same results when triggered with the output of a healthy PLL as the on time and off time should remain relatively constant. Should the maximum or minimum period exceed normal operating ranges, this indicates that the PLL may need replacing due to degradation of the delay components or conditioning circuitry. The latching encoder expects some discriminators to always be triggered, while others are never triggered, and should this be violated, allows an alarm to be triggered to signal PLL degradation.

Chen et. al. also modified the pulse-generation element in a PS TDC to allow for temperature measurement as well as time interval measurement [30]. When in temperature mode, a fixed-width pulse is provided to the delay line, and the output code observed. As the temperature of the chip fluctuates, the reduction in pulse width per element will vary, resulting in higher or lower output codes for a known-width pulse. This was characterised experimentally and then used to calibrate the temperature readout. The temperature resolution was less than 0.1 K/LSB (Kelvin per Least Significant Bit) and the resolution of the TDC was 45 ps.

In [2], Ma et. al. applied TDCs to the hardware security domain. By exploiting VT variations on an FPGA, they were able to use the results from a TDC characterising the system clock and a neural network to detect the insertion of Trojan hardware on the FPGA fabric and thus avoid leaking information to the Trojan. They demonstrated that this approach was robust even under process and environment variations, which would normally be expected to upset such a scheme.

As a final example, Rostami et. al. demonstrated a Cyclic Successive Approximation (CSA) TDC for use in a Capacitance to Digital Converter (CDC) [31]. The proposed CSA TDC used a switched capacitor bank to adjust the delay of the SA stage. More importantly, though, is the fact that the scheme has memory. The scheme is able to improve its conversion time and therefore sample rate with slowly varying signals by starting estimation at the previous value of the signal, then adjusting in either direction (the TDC core is bipolar) to reach the new sample. This is an interesting concept and may be useful to include in future time-of-flight or other oversampling applications.

The common feature between all these TDC systems is that they have a knowledge of what is an ‘acceptable’ or ‘standard’ output and then report on differences between the observed sample and the expected value. This is quite different from other applications which output values without modification and may be a good target for application-aware optimisations.

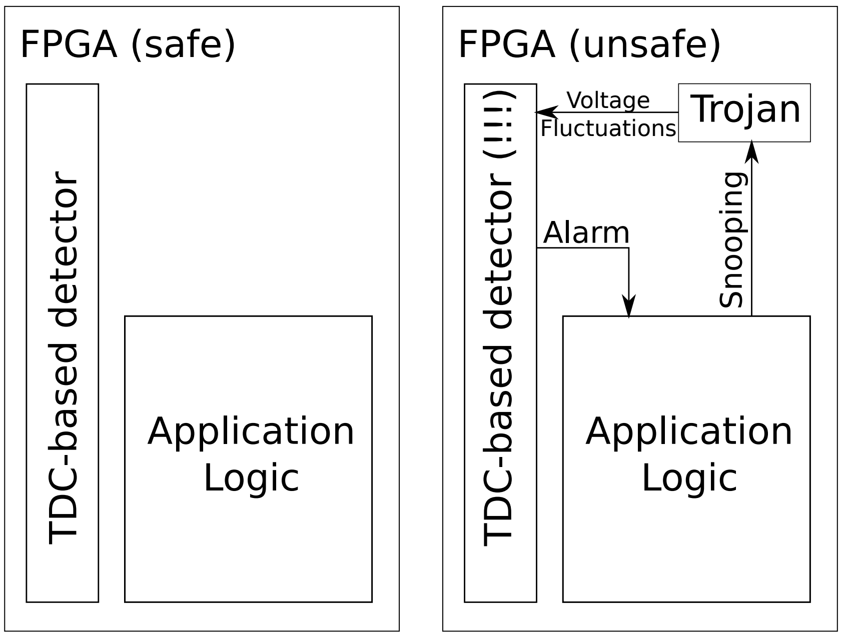


Figure 1.4: Application of TDCs to hardware security by Ma et. al. [2]. When a trojan is added to the FPGA device, the TDC-based detector observed changes in voltage and temperature and sounded the alarm.

1.5.6 Commercial Availability

Many TDC products are available commercially for different purposes. An example of a low cost two-channel LiDAR TDC is the Texas Instruments TDC7201 chip which provides 55 ps resolution and 67 ns dead time [32]. For applications such as coincidence correlation, a TDC such as Swabian Instrument's Time Tagger, which provides up to 144 channels with 4 ps RMS resolution and 2.1 ns dead time, can be used [33]. PicoQuant's Picoharp and Hydraharp series, which can provide down to 1 ps resolution in up to eigth channels of operation [34], and Becker Hickl's SPC series, which can provide 1.1 ps resolution with up to 512 channels in a multi-device system [35], are popular TDC products for Time-Correlated Single Photon Counting (TCSPC). However, they both suffer from long dead times of 80 ns and 100 ns respectively. In addition, IdQuantique's ID900 Time Controller is another example of a commercial time tagging box which provides 20 ps resolution with up to 64-channel operation and only 10 ns dead time [36].

1.6 Performance and Cost

In this section, we will explore the metrics by which we measure the performance of a TDC design.

1.6.1 Least Significant Bit (LSB)

When converting an analog quantity (time) into a digital quantity (binary code), the analog value must be quantised. Quantisation is the process of taking a continuous value and assigning it to one of many discrete values, often known as 'bins' or 'least significant bits' (LSBs). Each LSB will map to a range of input values (many different values fall into the same bin), with the average of these values considered the best estimate for the value of the bin (minimises the distance between the input value and the value of the bin).

The size of the LSBs (which may not be uniform) is a major contributing factor to almost all TDC metrics, except when so small as to become negligible. As it is almost entirely dependent on the physical manufacturing process, some metrics are quoted as multiples of the LSB to make their description agnostic to the manufacturing process.

1.6.2 Range

The range is an important yet often overlooked aspect of a TDC. The range of a TDC is the difference between the maximum measurable value and the minimum measurable value. If the maximum range is insufficient for an application, a TDC may suffer saturation conditions (reporting the maximum value for times above the maximum value), missed events (a pair of start and stop not reported due to the over-range condition), time-aliasing (a larger time being reported as a smaller time) or mis-attribution (a stop signal being attributed to the wrong start signal). This is most likely to happen in applications without a strictly bounded upper range (such as time-of-flight, where the maximal distance may not be known) or where the TDC range is intentionally under-specified (such as all-digital phase-locked loops, where the TDC is expected to saturate for large phase differences). Alternatively, if the minimum range is insufficient for an application, the TDC may suffer from missed events or miss-attribution. This is most likely to happen in short-range time-of-flight (where the channel delays are mismatched and the short range causes the stop to trigger before the start) and double-ended applications such as positron emission tomography (the pair of particles may hit the channels in any order).

For many TDCs, most notably those that measure the start and stop signals relative to an external reference, the minimum measurable value is zero, defined by the fact that the stop signal must occur after the start signal, and so any stop signals that arrive before the start signal are assumed to be related to a previous start signal. However, in other TDCs, most notably those that measure the stop signal relative to the start signal, there is a minimum measurable value defined by input delays, minimum pulse widths and setup times from the start to stop signals. Some TDCs allow double-ended operation, where the start and stop signals can occur in any order (thus, the minimum is the negation of the maximum). However, these double-ended TDCs often have limited ranges or cannot process multi-hit events due to the difficulty of determining which start signal a particular stop signal should be attributed to (example, start signals happening at 0 ns, 10 ns, 20 ns etc. and a stop signal at 6 ns; it is difficult to determine whether the stop signal is 6 ns after the first start or 4 ns before the second start).

An increase in range can often result in a decrease in accuracy, precision and resolution. For systems that measure the stop signal relative to the start signal, an increase in range may result in a proportional decrease in resolution as each possible quantisation is enlarged to cover the new range. Decreases in resolution will also cause precision to drop. In addition, as the range increases, the inaccuracy in the quantisation will often scale linearly (proportional difference) but sometimes non-linearly (as the operation of the TDC may change its characteristics due to process, voltage and temperature variation). Alternatively, for systems that measure the start and stop relative to an external reference, the accuracy of the external reference plays an important role in influencing the accuracy and precision as the range increases. When the range is small enough that the same external reference signal can be used for quantisation, the external reference has no effect. However, as the range increases, different reference signals will be required to perform the quantisation, and so the accuracy and precision of the difference between these reference signals will significantly affect the output result.

In this thesis, the range, R, of a TDC will be considered as the sum of the sizes of all the LSBs, as in (1.3), where τ_i is the size of the LSB with index i, N is the total number of LSBs, T_{max} is the delay after the final bin transitions and T_{min} is the delay before the first bin transitions. The number of bits at the output, B, will be the ceiling of the Effective Number Of Bits, $ENOB$ as per (1.4), which in turn is the base-2 log of the effective bin width τ_{eq} as in (1.6). The equivalent bin width is defined according to σ, the Single-Shot Precision (SSP), as per (1.5) [37].

$$R = \sum_{i=0}^{N-1} \tau_i = T_{max} - T_{min} \tag{1.3}$$

$$B = \lceil ENOB \rceil \tag{1.4}$$

$$\tau_{eq} = \sigma\sqrt{12} \tag{1.5}$$

$$ENOB = log_2\left(\frac{R}{\tau_{eq}}\right) \tag{1.6}$$

1.6.3 Accuracy

The accuracy is a commonly misinterpreted parameter of a TDC. It is commonly interpreted as the average *distance* (a positive value) between the measurements and the true value. However, the correct interpretation is the average *difference* (positive or negative). For example, with two estimates T_0 and T_1 of a true value T, where $T_0 = T + 1.01$ and $T_1 = T - 0.99$, the average of T_0 and T_1 is $\frac{T+1.01+T-0.99}{2} = T + 0.01$. The *distance* interpretation would describe an average error of $\frac{abs(1.01)+abs(-0.99)}{2} = 1.00$, whereas the *difference* interpretation would describe an average error of $\frac{1.01-0.99}{2} = 0.01$, which is clearly more correct.

The main factors contributing to inaccuracy in a TDC are channel offsets and miscalibration. If the delay from the input signal to the TDC core varies on a per-channel basis, the quantised delay will be offset by the difference in input delay between the start and stop channels. This channel offset can generally be considered constant on each device but may be affected by voltage and temperature fluctuations. Miscalibration is when the nominal value of a quantisation does not match the delay required to cause it. This is again caused by process, voltage and temperature fluctuations causing the delay of each LSB to fluctuate relative to their nominal value. While these errors can be very significant if not compensated for, compensation is a relatively easy task via calibration.

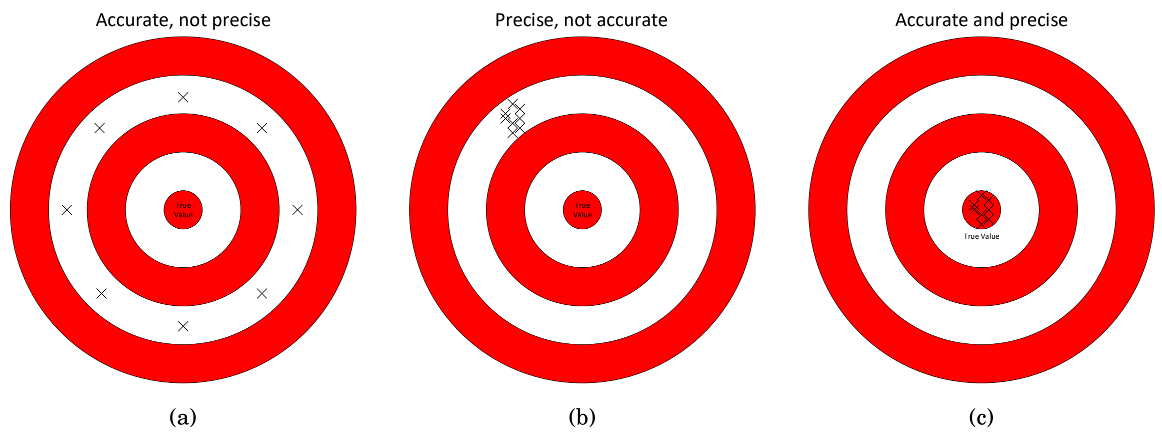


Figure 1.5: Centre of target: true value; (a) a process that is accurate but not precise; (b) a process that is precise but not accurate; (c) a process that is accurate and precise.

Calibrating for channel offset is a very simple task. One of the channels, normally channel zero, the start channel or the channel with the shortest delay, will be used as a frame of reference and be considered to have zero offset (the offset of a channel relative to itself must be zero). Then, a single signal is introduced to both the reference channel and another channel, and quantised. The average difference in the quantised value between the reference channel and channel in question will be the channel offset of the channel. As this method relies on accuracy within a channel, the channel offset calibration must be performed after calibrating the individual LSBs within a channel relative to its start.

Calibrating the LSBs within a channel can be performed at multiple different levels. The most basic is average delay calibration. The most accurate is code density calibration. Between these are various other methods that trade-off complexity, run-time and calibration accuracy. This is summarised in Figure 1.6.

Average delay calibration involves measuring the delay of the entire TDC and dividing by the number of LSBs to determine the size of a single LSB. For some TDCs, particularly multi-stage TDCs with a coarse quantiser and a fine quantiser, measuring the entire range of the TDC is intractable, and so a sub-section of the TDC (e.g., the fine quantiser, the length of a loop iteration in a looped TDC) is measured instead. This ensures that the average delay of an LSB is the same as the nominal delay of an LSB but does not account for individual variations in LSB size.

This method is often used to complement other methods, particularly to adjust for voltage and

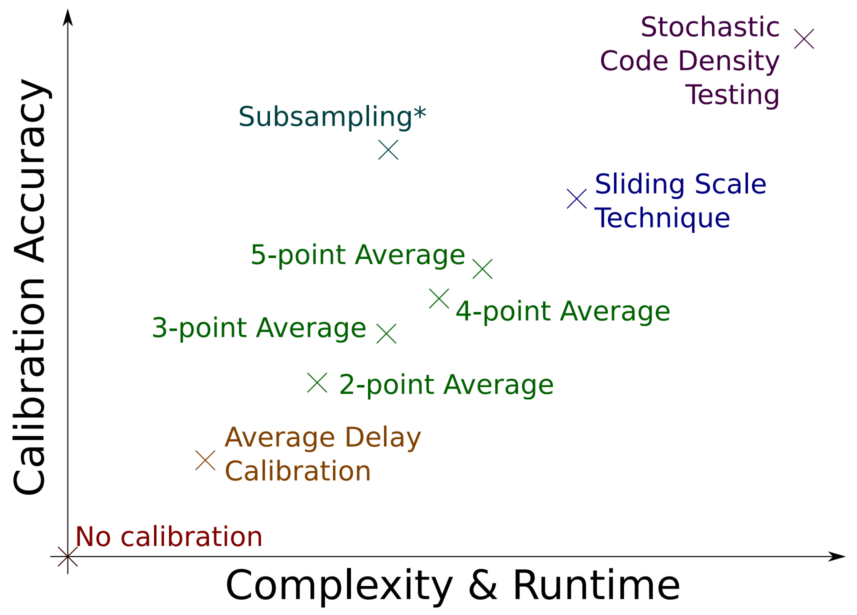


Figure 1.6: Comparison of calibration effort between methods. * Subsampling decreases resolution while improving calibration accuracy and linearity.

temperature variations at run-time (since it is assumed that voltage and temperature variations affect all LSBs proportionally). When performing the average delay calibration, two values are measured, T_{min} and T_{max}, which correspond to the delay immediately before the first bin transitions and the time immediately after the final bin transitions, respectively. The average delay of each element, $\overline{\tau}$ is calculated as in (1.7), and thus the nominal value of the LSB with index i, T_i, is defined as in (1.8). The $\frac{1}{2}$ component is required to ensure that the nominal value is at the centre of the bin.

$$\overline{\tau} = \frac{T_{max} - T_{min}}{N-1} = \frac{R}{N-1} \tag{1.7}$$

$$T_i = T_{min} + \overline{\tau}\left(i + \frac{1}{2}\right) \tag{1.8}$$

Code density calibration involves measuring the delay of each individual bin through a stochastic process. The TDC samples a uniform distribution multiple times, and as a result, quantity of measurements with a particular value (falling into a single LSB) will be proportional to the size of the corresponding LSB. A large LSB will result in numerous measurements with the corresponding value, while a small LSB will result in few measurements with the corresponding value. If the quantity of measurements (from here on referred to as 'hits') with the corresponding value is divided by the total hits, the size of an LSB can be represented as a proportion of the entire TDC. If the length of the TDC (or some subsection of it) is known, either via average delay calibration or via relation to an external reference of known period, then the size of an LSB can be directly represented as a delay.

Code density calibration necessarily requires a large number of hits to function as a stochastic process with any meaningful accuracy, and thus may take significant time to calibrate (meanwhile, the TDC may be non-operational). Code density calibration also improves integral non-linearity. The width of each LSB, τ_i, is defined as in (1.9), where h_i is the number of hits for LSB i, h_{total} is the total number of hits (as per (1.10)), and R is the range of the TDC. Then, the nominal value of LSB i, T_i is defined as the sum of all prior LSBs plus half the current LSB, as in (1.11).

$$\tau_i = \frac{R \times h_i}{h_{total}} \tag{1.9}$$

$$h_{total} = \sum_{i=0}^{N-1} h_i \tag{1.10}$$

$$T_i = T_{min} + \frac{\tau_i}{2} + \sum_{j=0}^{i-1} \tau_j \tag{1.11}$$

Another technique that may be considered is multi-point average delay calibration. Compared to single-point average delay calibration (which measures from the start to the end of the TDC's range), multi-point calibration measures multiple smaller subsections of the TDC. This significantly improves accuracy where drift is present within the TDC's range but may also reduce accuracy if the selection of subsections is unfortunate (this may also happen with single-point average delay calibration if there are large LSBs near the start or end of the TDC). When calibrating a subsection between LSBs i and j, the transition times of bins i and $j+1$, S_i and S_{j+1} are captured, and the average delay of all LSBs in the range, $\overline{\tau_{i,j}}$ is defined as in (1.12). Then, the nominal value of LSB k, T_k is defined as the delay of the start of the region (S_i), plus $k-i$ times the average delay within the region, plus half an average delay (to reach the centre of the bin), as in (1.13).

$$\overline{\tau_{i,j}} = \frac{S_{j+1} - S_i}{j+1-i} \tag{1.12}$$

$$T_k = S_i + \overline{\tau_{i,j}}(k + 1/2 - i) \tag{1.13}$$

The sliding scale technique operates similarly to multi-point average delay calibration in terms of measuring the delay of a section of the TDC, but instead of measuring a fixed section, the sliding scale technique measures a random section of the TDC similar to the code density test. This allows quick, relatively accurate calibration along the entire length of the TDC without significant negative effects when the measured TDC section is placed unfortunately (since the multiple random measurements even out any bad calibrations). However, the sliding scale technique, due to averaging across a large number of bins applies a smoothing effect to its output values and thus fails to properly recognise individual large LSBs in its normal mode of operation (although, a significantly advanced calibration engine with sufficient samples would be able to detect large bins by comparing adjacent quantisations). Implementation of the sliding-scale calibration may vary, but a common implementation would be an alpha filter, where the current estimate of each bin width $\overline{\tau_i}$ is modified slightly (by a factor of α) by the most recent measurement $\overline{\tau}$ to create an updated value $\overline{\tau_i}'$, as in (1.14). The nominal value of a bin is calculated as in code density calibration but may need to be normalised to ensure that the sum of the bin widths match the range.

$$\overline{\tau_i}' = \overline{\tau_i}(1-\alpha) + \overline{\tau}\alpha \tag{1.14}$$

Table 1.4: Comparison of calibration techniques.

Calibration Method	Calibration Accuracy	Calibration Time	Calibration Effort
Average Delay	Low	Low	Low
N-Point Average Delay	Medium	Low	Medium
Sliding Scale	Medium	Medium	Low
Code Density	High	Medium	High

1.6.4 Precision

Precision is arguably the most important metric for a TDC, given that inaccuracy can be removed via calibration and all the other performance metrics affect the precision. It also defines the usefulness of the results observed by the user. The precision of a TDC is defined as the spread of measurements about an average value (which may not be the true value). A TDC can be precise (have a small spread) without being accurate (close to the true value). Figure 1.5b shows an example of this, with the measurements being tightly clustered but far from the true value. If the offset between the cluster centroid and the true value is known and stable, then calibration can account for the offset, effectively converting Figure 1.5b into Figure 1.5c.

The spread of values about their centroid is normally measured by the Single-Shot Precision (SSP) or the Full-Width Half-Maximum (FWHM). The SSP is equal to the standard deviation of a single constant measurement, while the FWHM is the distance between the two points at which the probability of a measurement falls to half the maximum probability.

The main factors contributing to TDC imprecision are LSB resolution, miscalibration and reference signal jitter. As the size of an LSB increases (resolution decreases), the possible range of values which that LSB can represent will increase. As the best estimate for the value of an LSB is the centre of the range it covers, the greater the range of the LSB, the greater the probability of a measurement being further from the true value.

In addition, if the TDC is miscalibrated (or not calibrated at all), the nominal value of the LSB will not match the range of true values that cause it to occur, and so the difference between the average value and the individual measurements will increase significantly. If the difference between the true value and the measured value varies between LSBs, and multiple measurements cause hits in different LSBs, this will then impact precision (if the hits always occur in the same LSB, the result will be inaccurate but not imprecise).

For example, if a Time-to-Digital Converter (TDC) is free-running (measuring a trigger relative to a clock signal), it is unknown where the trigger may land in the TDC's range. When the TDC is miscalibrated, the difference between the true value and the quantised value varies across the range of the TDC. Therefore, multiple measurements will vary significantly in their distance from the true value, causing an imprecision. This is particularly noticeable in double-shot tests (start and stop signals are measured relative to the system clock), where the start and stop signal have a fixed time difference between them, but may occur at any point in the TDC's clock period. While the true value of their time difference is constant, variation in the difference between the measured values will be exhibited as imprecision.

Finally, for TDCs that measure relative to an external reference, variations in the reference signal will cause the delay between the reference and the start or stop to vary, increasing the spread of measured values.

Unlike accuracy, it is much harder to improve the precision of a TDC. A lack of calibration can be overcome by performing calibration, and this will significantly improve resolution where the TDC is highly non-linear. Reference signal jitter can often be improved by upgrading the external reference to one with a higher precision, such as an Oven-Controlled Crystal Oscillator (OCXO), and further calibrating this with an accurate time standard such as GPS or a local Rubidium time standard. The LSB resolution can only be improved by altering the architecture of the TDC or averaging multiple samples of the same input signal.

Determining the single-shot precision directly can be difficult as it requires a repeatable delay, which is difficult to ensure due to voltage and temperature variations. Therefore, the double-shot precision (DSP)[38] is often calculated as an intermediary, and then the SSP can be calculated as in (1.15). The DSP is the standard deviation of the time difference between two events T_{stop} and T_{start} as in (1.16). Achieving a repeatable delay between two events is trivial as both channels capturing the events can be provided the same signal, which will cause a repeatable delay equal to the channel offset.

$$SSP = \frac{DSP}{\sqrt{2}} \tag{1.15}$$

$$DSP = \sqrt{\frac{\sum(T_{stop} - T_{start})^2}{N-1} - N\left(\frac{\sum(T_{stop} - T_{start})}{N-1}\right)^2} \tag{1.16}$$

1.6.5 Resolution

Resolution is a very important metric for a TDC due to being the largest contributor to imprecision when greater than ~10 ps (below this value, other errors such as miscalibration, noise and jitter are more significant). The resolution is defined as the minimum possible difference between two measurements that can be reliably distinguished. Given that a measurement falls uniformly in each LSB, to reliably distinguish one measurement from another, the difference between the two must be at least the width of a bin to ensure that different LSBs are selected for the two measurements. Therefore, the resolution is nominally equal to the LSB size, unless other TDC components further limit the resolution. As the LSB size may vary across the TDC, the resolution can also vary. Therefore, the resolution must be defined statistically in relation to all the LSBs. Normally, the mean of the LSB sizes (as in (1.17)) is used as the resolution, however, as larger LSBs are more likely to experience a hit and affect the overall resolution or precision, there is an argument that the square mean (as in (1.18)) or cubic mean (as in (1.19)) are more appropriate. Alternatively, the resolution can be calculated as an equivalent width [39], that is, the width of a bin required to match the SSP provided by the TDC. The equivalent width can be calculated as in (1.20), either via summing the sizes of the bins or via computing the SSP (σ).

$$\overline{\tau} = \frac{\sum_{i=0}^{N-1} \tau_i}{N} \tag{1.17}$$

$$\overline{\tau} = \sqrt{\frac{\sum_{i=0}^{N-1} \tau_i^2}{N}} \tag{1.18}$$

$$\overline{\tau} = \sqrt[3]{\frac{\sum_{i=0}^{N-1} \tau_i^3}{N}} \tag{1.19}$$

$$\overline{\tau_{eq}} = \sqrt{\frac{\sum_{i=0}^{N-1} \tau_i^3}{NR}} = \sigma\sqrt{12} \tag{1.20}$$

Improvements in resolution can occur through three methods. First, an improvement in the underlying hardware may reduce the delay of each LSB, resulting in a smaller range of true values that can hit the LSB and thus improving the resolution. Next, a change in architecture can result in the LSBs becoming fundamentally smaller or being sub-divided by other LSBs. Finally, averaging can improve the resolution by making multiple measurements of the same value (with some random error) and then combining these to achieve a more precise (higher resolution) result. Averaging can be achieved either by taking multiple physical measurements of the desired phenomenon, or by replaying a single measurement multiple times with sufficient variation to allow averaging to occur.

It is important to note that averaging can only improve the resolution and precision if the measurement is not perfectly repeatable. A variation in the measurement normally reflects the relationship between the centre of an LSB and the true value: if the measurement occasionally increases by one LSB, the true value is most likely above the centre of the LSB. Alternatively, if the measurement occasionally decreases by one LSB, the true value is most likely below the centre of the LSB. If the measurement is evenly distributed between two LSBs, then the measurement is likely on the boundary between the two LSBs. The source of the variation can be quite varied; it could be differences in LSB sizes on different channels of the TDC, it could be different delays relative to the external reference (in which case, the difference must be known to adjust the various measurements to the same range), or the signal could enter the TDC at different positions in the quantiser (which would result in different LSBs being attributed to the same true value).

1.6.6 Linearity

Linearity is an important metric for a TDC, but like accuracy, it is easily corrected for. Linearity can be defined in two forms: the Differential Non-Linearity (DNL) is the maximum difference between the centres of two adjacent LSBs minus one, as in (1.21). This is a similar metric to the resolution and is often quoted as a multiple of the average LSB size (e.g., DNL = 1.7 LSB). The Integral Non-Linearity is the maximum difference between the midpoint of a bin (nominal value) and the ideal linear transfer function (as in the average delay calibration), as per (1.22). For a TDC where all LSBs are exactly the same size, the DNL is 0 LSB (the difference between the LSB centres is equal to the LSB size, therefore 1 LSB−1 LSB = 0 LSB) and the INL is 0 LSB (the midpoints exactly match the linear transfer function).

$$DNL = max(T_{i+1} - T_i) - 1 \tag{1.21}$$

$$\begin{aligned} INL_+ &= max\left(T_i - \left(T_{min} + \frac{R \times (i+0.5)}{N-1}\right)\right) \\ INL_- &= min\left(T_i - \left(T_{min} + \frac{R \times (i+0.5)}{N-1}\right)\right) \end{aligned} \tag{1.22}$$

For a TDC with uneven LSB sizes, the DNL and INL are generally worse. The DNL is at its maximum where the sum of two adjacent LSB sizes is largest, as this causes the difference between the centres to be highest. For the INL, the value is dependent on the presence of calibration. For an uncalibrated TDC, the maximum INL will likely be furthest from the point at which the nominal values and true values coincide. Depending on the TDC design, this could be at the start, the end, or at some point in the middle. The cumulative effect of inaccurate average bin sizes will dominate the INL, resulting in the point furthest away having the highest INL. For a TDC with average delay calibration, it is expected that the start and stop of the TDC are correctly calibrated (low INL at these points). Therefore, the point at which the INL is at its maximum will necessarily be near the middle, unless some architectural periodicity constrains it to another region. A good example of this is looped TDCs, where each loop will have the same characteristics, and so the INL passes through zero each time the loop repeats. Finally, with an accurate calibration such as code density calibration, the INL will be at its maximum at the boundary of the largest LSB. As the TDC is accurately calibrated, the nominal value matches the true centre of each LSB, so the maximum difference will occur at the boundary of an LSB where the true value is furthest from the centre (and thus furthest from the nominal value), and the boundary is furthest away from the centre in the largest LSB. A poor calibration will necessarily increase the INL.

Improvement of the linearity is difficult, but not as difficult as improving the resolution. For the DNL, improvements can be made by sub-dividing large LSBs. There are many ways to do this, with the most common being averaging multiple measurements, introducing multiple, branching paths through the TDC, or sub-dividing large LSBs with smaller ones in other channels. DNL may also be made less significant by excluding the large LSBs from the normal region of operation (this is most easily achievable when the large LSBs are near the ends) or by simply improving the base resolution or matching between components. INL is most significantly improved by calibration, and once that is complete, improvements to the DNL cause similar improvements to the INL (since both are based on the size of the largest bins).

1.6.7 Quantity of Channels

Unlike the previous metrics, which relate to the performance of a single measurement, the quantity of channels focuses on the number of measurements that can be made in parallel. The number of channels in a TDC defines the number of active inputs which can be simultaneously connected to the TDC. For some applications, such as Time-of-Flight (ToF) or All-Digital Phase-Locked Loop (ADPLL) applications, the number of channels is of minor importance, as there will often be only one start and one stop signal operating in parallel, and the accuracy, precision and repetition rate are generally more important. For other applications such as nuclear, quantum metrology and multi-hit ToF, having multiple channels to receive many start or stop signals is required, but these applications may not require high accuracy, precision or repetition rates.

Quantity of channels is highly related to area. If the area of a TDC is low, then the TDC can be duplicated to produce multiple channels. Similarly, if the area of a TDC is high, there may not be sufficient room for all the required channels. However, the quantity of channels may not be inversely proportional to the area. Introducing more channels to an existing TDC is often more efficient than duplicating the entire TDC, as the decode logic, serialisation logic, reference signal and even start channel can often be shared between channels. Also, some logic may be global and not require duplicating when operating multiple channels, such as the phases of the clock in a multi-phase clock system.

1.6.8 Repetition Rate

The repetition rate is highly related to the TDC architecture and is often the main driving force behind design choices. The repetition rate is defined as the maximum rate at which successive events can occur without any being missed. For many systems, the repetition rate is a small multiple of the system clock frequency, which is often strongly related to the external reference of that system. For others, the repetition rate will be defined by the maximum length of the TDC (for cases where the input signal travelled the entire length of the TDC) plus the time required to reset the device. Occasionally, the repetition rate is limited by the decode or serialisation functionality, but this is uncommon as providing high-bandwidth parallel logic is relatively simple compared to the TDC core.

Requirements for repetition rates are very specific to given applications. For ToF applications, the repetition rate may be related to the frequency of start pulses, the repetition rate of the receiver which feeds the stop channel, or the rate at which a surface needs to be scanned. For nuclear applications, the repetition rate will often be defined by the rate at which particles are emitted from the sample being characterised. For frequency locking applications, the repetition rate only needs to be slightly above the target frequency to quantise every edge. Finally, for quantum applications, the repetition rate only needs to be as high as the rate of the element that produces entangled particles.

1.6.9 Power

Power is a relatively uncommon consideration for TDCs as meeting performance, channel quantity and area targets are more important. However, when considering TDCs that must operate in adverse conditions or on finite power sources, power becomes an important metric. The power consumed by a TDC can be broken into two components: the static power, which is consumed while the TDC is powered on, and the dynamic power, which is consumed only while the TDC is operating on some data. The ratio between dynamic power and static power is dependent on the frequency of input events as well as the architecture. Some architectures have very little circuit activity while switched on, and so the static power becomes more significant. The same can happen when input events are uncommon: even if the cost of processing an event is high, if it rarely occurs, the overall cost will be low compared to the cost of keeping the TDC powered. Conversely, some architectures have large amounts of switching occurring even when there are no input events. This, or a high rate of input events, can cause the dynamic power to significantly exceed the static power.

The main techniques to reduce power include minimising switching (both while the TDC is active and while it is idle), reducing area (a smaller TDC has less logic for current to leak through) and turning off sections of the TDC while it is not in use (clock gating). For a system with genuinely random input events, clock gating is difficult. However, most systems are not truly random, often producing a hint that an input event will occur before it does. For example, in a ToF system, the activation of the start signal is often periodic, so the TDC can be powered down if it recognises that there will not be another start signal for a long time. Also, the stop signal often quickly follows the start signal, so once the stop has occurred, the stop channel can go to sleep until the next start signal.

Reducing switching in a TDC is most often achieved by careful design, avoiding wasted logic and avoiding processing while the input events are not active. For example, two designs of delay line TDCs Figure 1.9 have the same architecture but very different dynamic power requirements as one puts the clock signal through the delay line and captures it with the start and stop, while the other puts the start and stop through the delay line and captures them with the clock. When the TDC is inactive, the first design still causes the delay line to oscillate due to feeding the clock into the delay line, while the other does not oscillate as there is no start or stop. This results in the first design requiring more dynamic power than the other. Similarly, looped TDCs tend to have much lower static powers when idle than their non-looped counterparts, as they require significantly less components and so leak significantly less current. However, the counter attached to a looped TDC introduces more switching, and so they take a penalty in dynamic power compared to non-looped TDCs.

Fig. 1.10 demonstrates the three main forms of power draw. The first is dynamic power. When a logic gate (in this case, a CMOS inverter) output switches from a zero to a one, the intrinsic capacitance at each node in the gate will need to be charged from the power rail for the output to reach the required voltage. The capacitance stores $E = \frac{CV^2}{2}$ joules of energy, while the energy drawn from the power rail is $E = QV = CV^2$. Therefore, while charging, half the power is dissipated as heat through the PMOS transistor. When the logic gate output switches from a one to a zero, each internal node discharges to ground and the energy stored in the intrinsic capacitance is dissipated through the NMOS transistor.

Next is short-circuit power. When a logic gate switches, the PMOS transistors turn on as long as the voltage at the gate is less than $V_{dd} - V_{th}$, where V_{dd} is the source voltage and V_{th} is the threshold voltage, normally ~0.7 V. The NMOS transistors turn on if the voltage at the gate is more than V_{th}. If $V_{dd} > 2V_{th}$, there will be a period T_{sc} where both transistors are turned on and a low-resistance path will exist between the two power rails. During this period, a large amount of current will flow ($I_{sc} = \frac{V_{dd}}{R_P + R_N}$, R_P is the PMOS resistance, R_N is the NMOS resistance) and a high power draw is experienced ($P_{sc} = I_{sc}V_{dd}$, $E_{sc} = F_{sw}P_{sc}T_{sc}$, E_{sc} is the total energy drawn due to short-circuit conditions and F_{sw} is the switching frequency). However, this period is a fraction of the switching time and so is very rarely active.

Finally, static power is caused when current leaks through a transistor that is 'off'. A transistor which is off does not have an infinite resistance, but rather a relatively high resistance, often ~10000x the resistance when the transistor is on. However, a small quantity of current is able to make its way through the transistor from one power rail to the over, resulting in a small but constant power draw, $P_{leak} = VI_{leak} = \frac{V^2}{R_{off}+R_{on}} \approx \frac{V^2}{R_{off}}$.

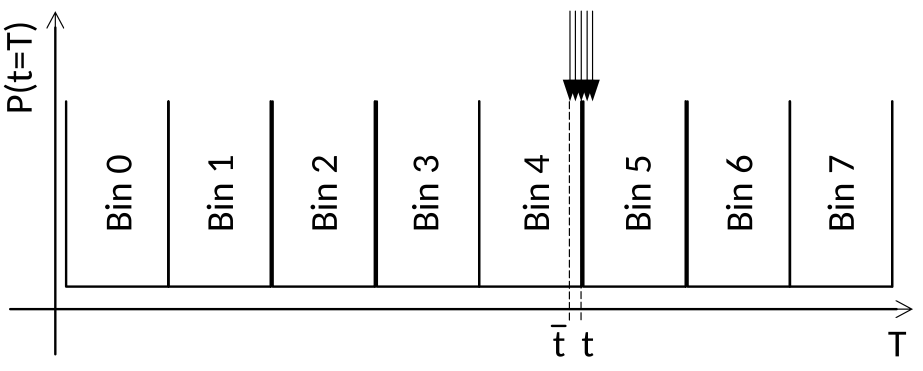
1.6.10 Area

Area constraints for a TDC are very common to allow for duplication to provide multiple channels, reduce cost by utilising smaller integrated circuits to implement the TDC, or to allow integration into a much larger system without requiring a separate integrated circuit. If area was not considered as an important metric, performance limitations would become trivial to overcome as the designer could simply add more TDC channels until the correct performance targets were met.

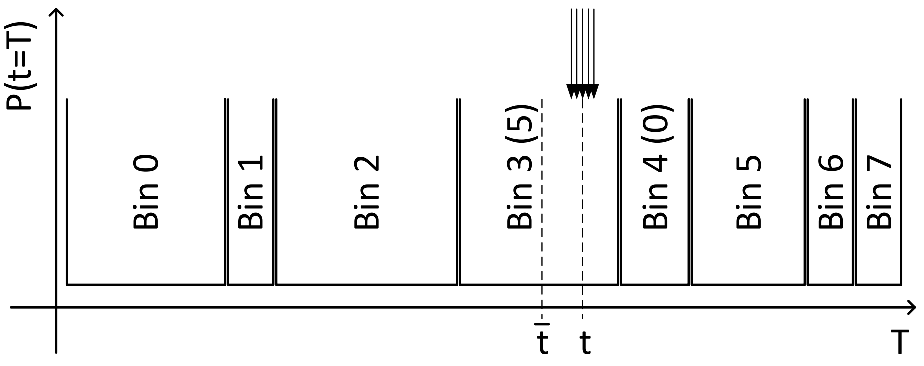
Area is predominantly affected by the architecture of the TDC and the number of channels present. Architectures which require more logic to produce a single delay, which require accurate analog components or have little common logic between channels cause significantly larger areas. Similarly, increasing the number of channels tends to significantly increase the required area, except in architectures where the cost of adding more channels is trivial. However, implementing looped TDC architectures allows for a significant reduction in area as the length of the TDC is folded back on itself, and the section of the TDC which corresponds to the true value is stored in a counter.

1.6.11 Time-to-Market and Technology

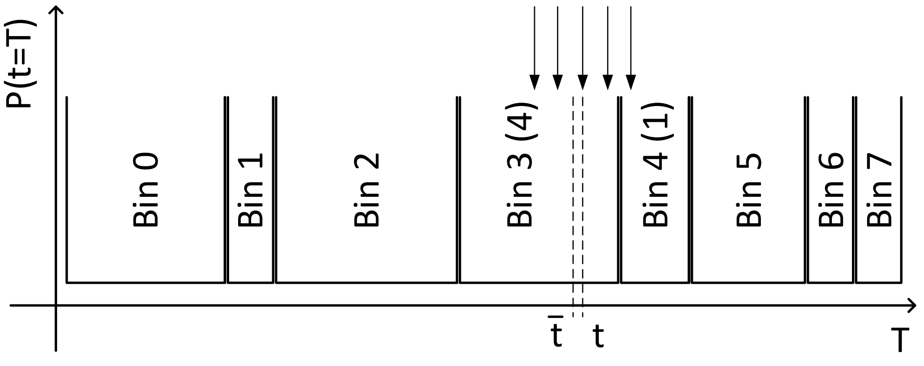
As most TDC research is either not commercially viable at the moment or involved in projects which may take many years to complete, time-to-market is not a significant concern to most TDC designers, but it deserves mentioning as it may become very important in the future. TDC designs with complex architecture that require large durations spent testing and debugging result in a much higher time-to-market, which may cause a user to miss the target window for usage of the TDC. Similarly, TDCs designed on Application-Specific Integrated Circuits (ASICs) generally require long times to prototype and manufacture, with production runs sometimes taking several months to complete, compared to designs on Field-Programmable Gate Arrays (FPGAs) which can be bought off-the-shelf and designed and deployed in a much smaller timescale. Architectures that are more generic and have faster time-to-markets generally cost more and have lower performance (as is the case for FPGAs), but provide benefits in terms of updatability, lower initial costs and lower development costs.



(a)



(b)



(c)

Figure 1.7: (a) A TDC with perfectly distributed bins. (b) A TDC with a poor DNL. The DNL will be at its maximum between bins two and three (both above-average size, bin ≡ LSB). (c) A demonstration of how averaging and sufficient measurement uncertainty can improve accuracy. The estimated value $\bar{t}$ is closer to the true value t as the distribution between bins three and four allows us to infer the value of t to a resolution higher than the bin size.

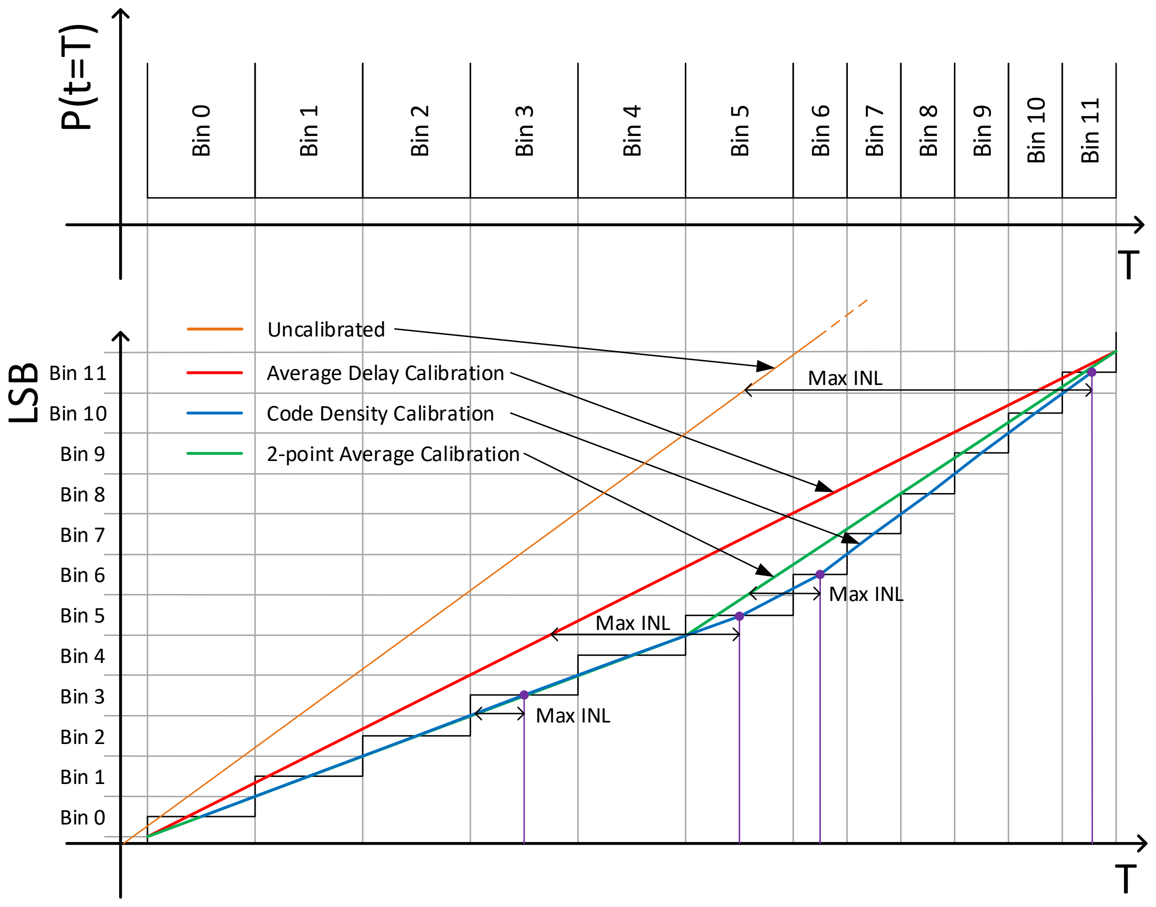


Figure 1.8: An example TDC with poor native INL, along with various methods to correct the INL. The maximum INL for each method is shown.

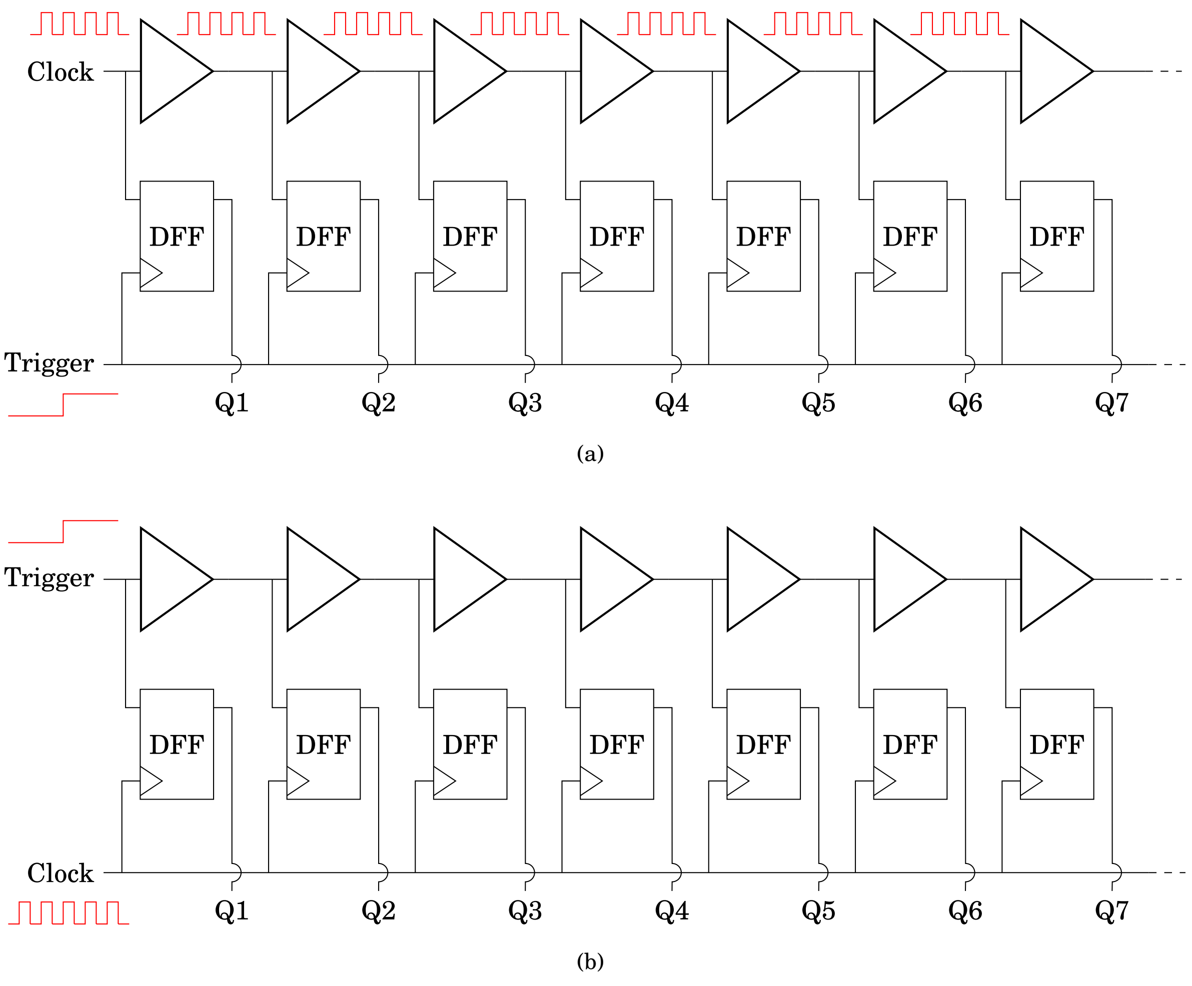


Figure 1.9: Comparison of power usage with different inputs to a delay-line TDC. (a) Clock as input to the delay line, trigger as input to flip-flops. (b) Trigger as input to the delay line, clock as input to the flip-flops.

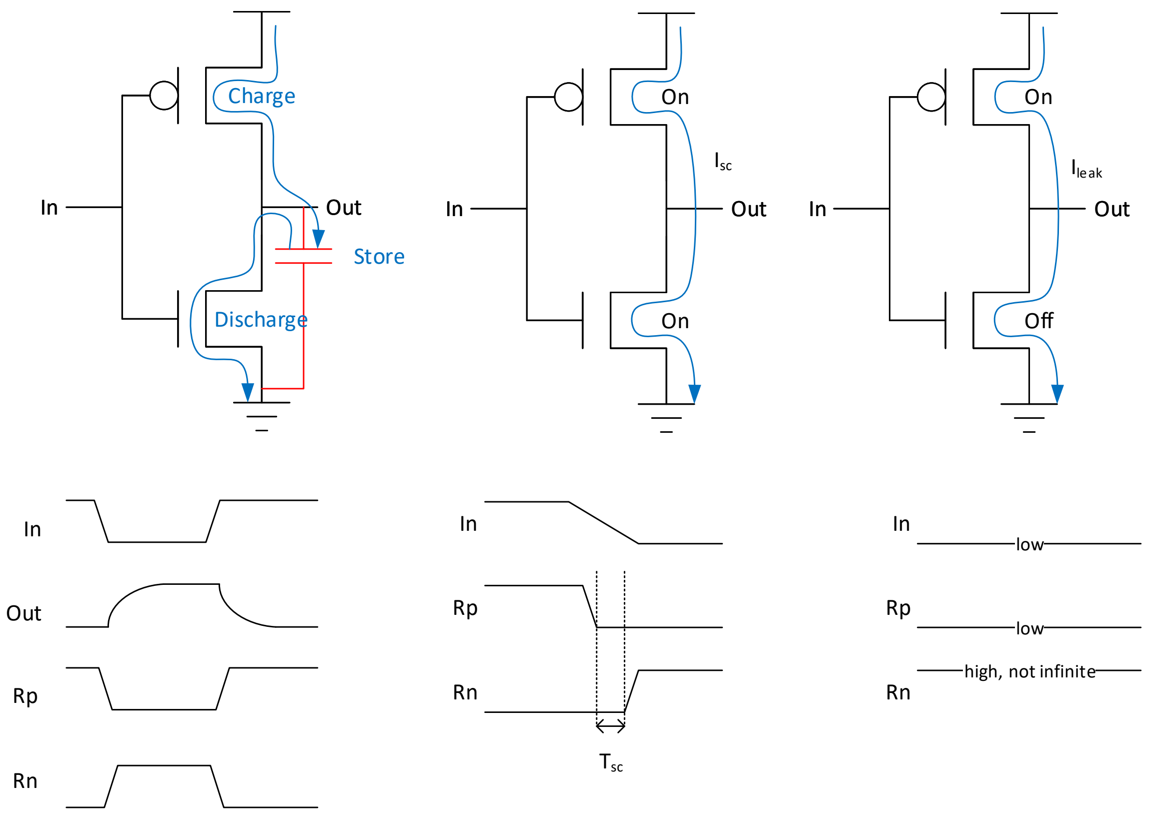


Figure 1.10: Demonstration of power consumption with a CMOS inverter (common delay element) as a sample circuit. Left: dynamic power; middle: short-circuit power; right: static power.

Chapter 2

Hardware

2.1 Abstract

Time to Digital Converters have been implemented on a large variety of technologies, from analog circuits [40] to discrete digital logic [41], from microprocessors [42] to Application-Specific Integrated Circuits (ASICs)[40], from Emitter-Coupled Logic (ECL) [43] to Complementary Metal-Oxide-Semiconductor (CMOS) [40], and everything in-between.

To understand the design, architecture, and implementation of TDCs, we must first understand the underlying technologies on which they are built, as this informs us both of what is possible, as well as what is efficient or high-performance. Therefore, this chapter is dedicated to describing the different technologies available for us to build TDCs on. Some platforms will be quickly discarded as they are unsuitable for TDC implementation.

In Section 2.2, we start off with Central Processing Units (CPUs), which are easy to program but relatively inefficient at their tasks. We then proceed through architectures which are successively lower-level and more difficult to design for, through Graphics Processing Units (GPUs) and Digital Signal Processors (DSPs), to Micro-Controller Units (MCUs). We then move on to simple analogue systems, and build up in complexity from there, through discrete logic and Field-Programmable Gate Arrays (FPGAs), to Application-Specific Integrated Circuits (ASICs).

We describe the characteristics of all the aforementioned platforms, their advantages and disadvantages, both directly related to the design space as well as touching on matters related to the development process. We will quickly see that programmable processors are a poor fit for TDC, and so it is the second half of the mentioned technologies (analog systems through to ASICs) which is more appropriate for the TDC task.

We will also see that the use of discrete analog and digital systems are incapable of the scales at which modern TDCs operate, relegating them to low-cost and historical systems. Modern designs are almost entirely constricted to the space of FPGAs and ASICs, the latter of which may be purely digital or mixed-signal (analog and digital).

In the following section (2.3), we will move on to exploring FPGAs in more detail, as the technology which was utilised for this PhD project. We will describe the building blocks present in an FPGA, how they operate, how they are utilised, and how they are composed to form a complex digital system. Many examples of Hardware Definition Language (HDL) code will be given along with its synthesised equivalents to demonstrate how code is translated to the FPGA's building blocks.

The pros and cons of all platforms are summarised in Table 2.1.

In Chapter 3, the various published TDC implementations on the technologies described in this chapter will be reviewed and compared.

Table 2.1: Table summarising the pros and cons of different hardware platforms.

Hardware	Pros	Cons	Suitability for TDCs
CPU	• Programmable to perform many tasks. • High single-thread performance. • Virtual memory allows for easy multi-threading. • Can run many in parallel.	• Response times are variable. • Virtual memory may cause accesses to take a long time. • Time to execute instructions varies. • No instruction to generate a small time delay.	None
GPU	• Highly parallel to optimise instruction and data throughput. • Simple cores optimise work done per Watt. • Programmable.	• All cores within a cluster must perform the same instructions, with only the data changing. • Code is difficult to write and optimise. • Control-style code is slow. • Single-threaded performance is low. • Data needs to be frequently exchanged between the GPU and a host CPU. • Slow, variable response times and high latency. • Need to submit data to process in batches. • No ability to generate small delays.	None

Continued on next page

Table 2.1: Table summarising the pros and cons of different hardware platforms. (Continued)

DSP	• Single-Instruction Multiple Data (SIMD) instructions allow multiple data items to be processed simultaneously. • Optimised performance per Watt for common mathematical instructions (multiplication and addition) and serial data input. • Consistent, predictable response times. • Direct Memory Access (DMA) engines to move data while CPU is performing useful work. • Direct access to peripherals to interface with outside world.	• Performance of control-style code is poor. • Writing code takes significant effort to achieve high performance. • Complex memory hierarchy needs to be considered when designing code. • Smallest delay is limited by system clock frequency, down to 1 ns.	Low
MCU	• Simple instruction set allows for small, low power, low complexity core. • Consistent and predictable response times. • Direct access to peripherals to interface with outside world.	• Low performance. • Limited available memory makes fitting code in available RAM difficult. • Smallest delay is limited by system clock frequency, down to 2 ns.	Low

Continued on next page

Table 2.1: Table summarising the pros and cons of different hardware platforms. (Continued)

Analog Systems	• Can work directly with time domain signals or convert to voltage domain. • *Theoretically* infinite resolution. • Remainders from conversion can be amplified and returned to the encoder. • Off-the-shelf availability.	• Resolution is *practically* limited by noise floor and component linearity. • All components are temperature- and voltage-dependent, so performance varies significantly over time. • Components are difficult to match precisely, and impossible to hit a specific value, causing inaccuracies. • Printed Circuit Board (PCB) needs to be designed to maintain signal integrity. • Cannot be changed after design is submitted.	Medium
Discrete Logic	• Cheap and simple for small circuits. • Off-the-shelf availability. • Can work directly with time-domain signals. • Can be integrated easily with analog systems.	• PCB needs to be designed to maintain signal integrity. • Voltage noise causes timing jitter due to limited slew rates. • Resolution is limited by bandwidth, delay and mismatch of signal traces and devices. • Cannot be changed after design is submitted.	Medium

Continued on next page

Table 2.1: Table summarising the pros and cons of different hardware platforms. (Continued)

FPGAs	• Can synthesise any digital system using array of look-up tables and storage elements. • Re-programmable to eliminate errors or change system design. • Close proximity of elements reduces signal degradation and impact of noise.	• Cannot perform any analog processes. • Synthesis of a digital system only matches functionality, delay may vary between the described system and synthesised system. • Limited bandwidth and higher delay compared to purpose-built system due to the addition of configurable routing and configurable logic. • In most cases, smallest delay limited by the delay of a logic element, down to 10 ps. • Difficult to program.	High
ASICs	• Can implement any electronic system. • Smallest delay can be based on differences between elements or analog values. • Custom routing allows for optimal performance and minimal delays.	• Up-front cost and time to market are high. • Minimum order quantities in tens of thousands. • Very difficult to design, often requiring a team multiple months (as opposed to a single person for other platforms). • Cannot be changed after the design is submitted. • A single mistake is very costly.	High

2.2 Hardware Types

2.2.1 CPUs

Central Processing Units (CPUs) are large, high-power processors that command the operation of a more complex system. They are characterised by their complexity, performance, access to large quantities of memory (which is often a separate device), and general-purpose interfaces designed to connect to peripherals (otherwise known as controllers) which communicate with the outside world.

They are generally non-deterministic, meaning the delay between an event happening and the system responding is variable. This makes them relatively inappropriate for real-time applications, and therefore not very useful as TDCs. While their high frequency suggests they may be able to implement a high-performance counter based TDC, the long and variable delay between events occurring and the CPU responding massively degrades the precision to the point where it is not an effective TDC, especially considering the cost of the CPU (as clock speeds increase, so does both the cost and the non-determinism).

2.2.2 GPUs

Graphics Processing Units (GPUs) are specialised processors designed for the purpose of processing graphics data quickly and efficiently. They are characterised by massive parallelism which allows large quantities of data to be processed simultaneously, high memory bandwidths and interface speeds to provide data to the massively parallel core, and lower clock speeds compared to CPUs (but higher than most other hardware in this section).

While the parallelism may lead the amateur to believe that these devices are a good choice for TDCs (as each core might be able to perform its own characterisation, and these characterisations then combined), this is not the case. Any signal input to the device is very quickly synchronised to the system clock a long time before reaching the highly parallel GPU cores, and so each core will observe the same signal with no variations between each core. This removes the measurement diversity required to make multiple measurements valuable, and therefore reduces the performance to the same as that of a single-core device, which is significantly lower than a

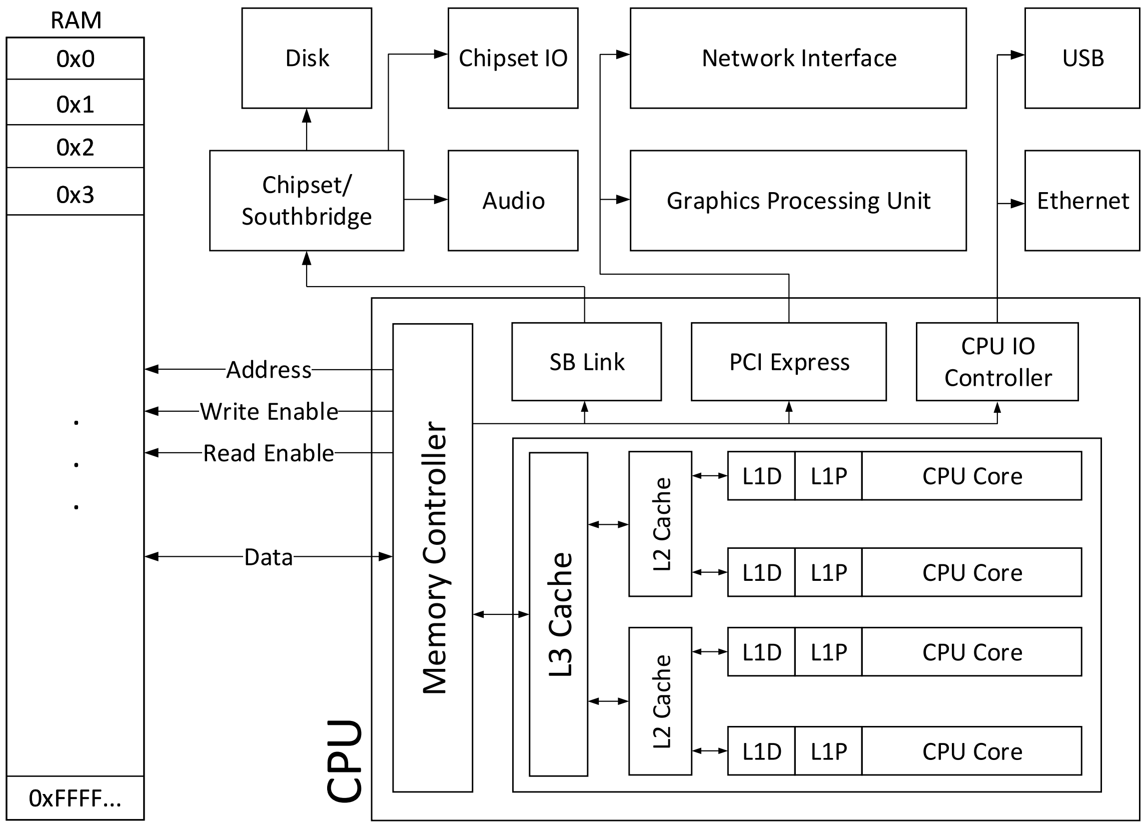


Figure 2.1: Common example of a CPU in a system. RAM = Random-Access Memory. L1P = Level 1 Program Cache. L1D = Level 1 Data Cache. SB = South Bridge. IO = Input/Output. PCI = Peripheral Component Interconnect. USB = Universal Serial Bus.

CPU despite GPUs costing significantly more than CPUs.

2.2.3 DSPs

Digital Signal Processors (DSPs) are specialised processors, similar to GPUs, designed to perform a single task very efficiently. In the case of DSPs, the task they specialise in is Digital Signal Processing (also DSP), which can generally be expressed as a repetition of multiplications and additions, operating on some data retrieved from the environment, and producing some data to be sent to the environment. For this purpose, the design of a DSP is focused around two main goals: high performance multiplication, addition and looping in the core, as well as efficient data reception and transmission from outside the processor. To achieve this, many of the hardware designed to optimise control code in a CPU is discarded in favour of fast multiplication and

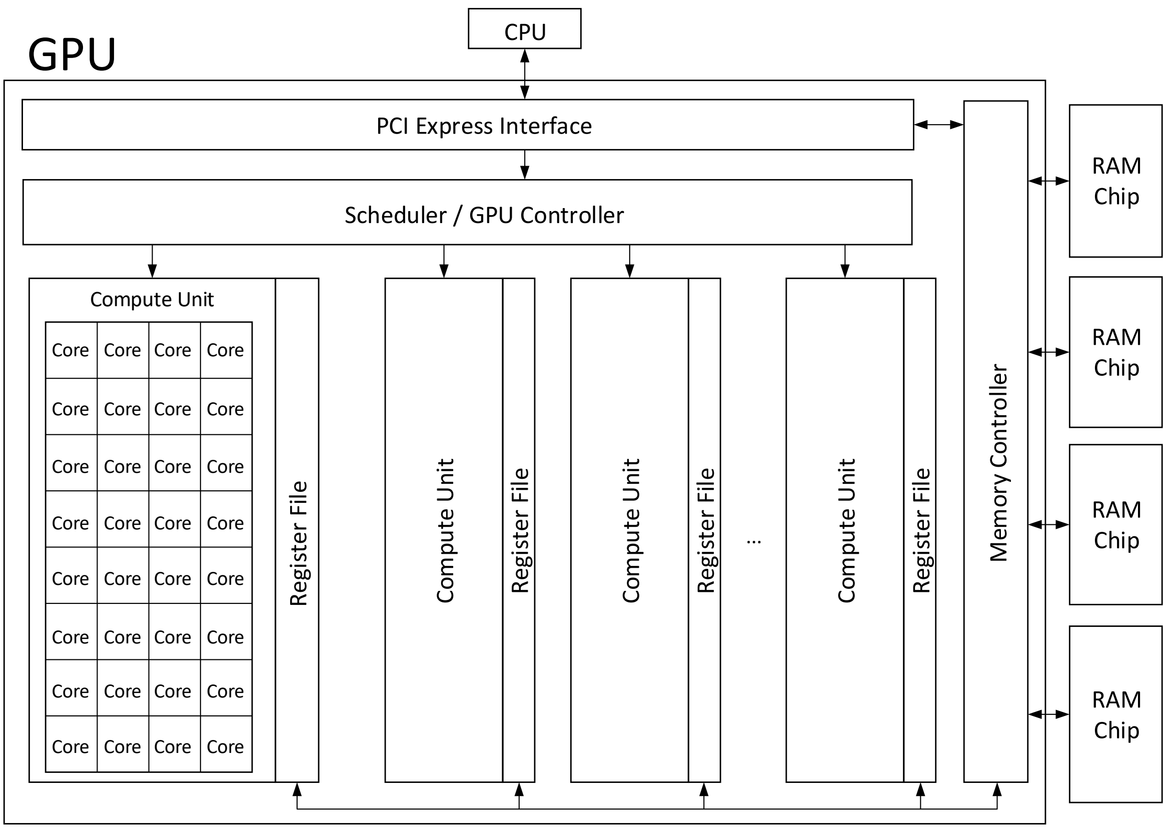


Figure 2.2: Internal layout of a GPU.

addition operating on many elements at once (Single Instruction Multiple Data, SIMD). Focus is also put on optimising the transfer of data in and out of the CPU, with peripherals known as Direct Memory Access (DMA) engines used to transfer data in the background, as well as complex peripherals designed for the purpose of communicating with common Analog-to-Digital Converters (ADCs), Digital-to-Analog Converters (DACs) and other data sources and sinks.

Similarly to MCUs, DSPs are very predictable and therefore excellent for real-time applications. As a result, they can also be used as rudimentary counter based TDCs. Unlike MCUs, DSPs have much higher clock rates (hundreds of megahertz), meaning that, when used in this mode, they perform significantly better. However, this comes at a significant cost, and so is not advisable (it is much more efficient to implement a counter in discrete logic). Another option with a DSP is to use the signal processing capabilities along with significant quantities of data to infer high-resolution timing. With a high-speed ADC or sufficiently low slew rate, it is possible to capture the rising edge of a signal as it is transitioning, rather than after the transition is complete. If the slew rate is sufficiently slow, it may be possible to capture the transition multiple times during its rise. The progress through the rising edge of the signal (and therefore the voltage observed by the ADC) will be proportional to the delay between the start of the signal's rising edge and the next clock pulse. Therefore, it is possible to interpolate the exact position of the start of the rising edge (or any other point on the rising edge) using this quantisation, to a much higher precision than would be provided by a simple counter. The drawback of this scheme is the reliance on analog components and a slow signal slew rate, which makes it susceptible to crosstalk and noise, similar to an analog system.

2.2.4 MCUs

Microcontrollers (MCUs) are very simple processors designed to directly control hardware. They take input directly from sensors and determine how to correctly drive corresponding actuators. Typical characteristics of microcontrollers include low clock speeds (tens of megahertz), small quantities of Random-Access Memory (RAM, tens of kilobytes) and Read-Only Memory (ROM, hundreds of kilobytes), and direct communication with pins, rather than communicating via a peripheral as a CPU would. Microcontrollers, due to the lack of hardware complexity, tend to be very predictable and much easier to tightly control, making them ideal for real-time applications, where clock speed and memory size allow.

Microcontrollers are common candidates for very low performance TDC designs. Due to their predictability and low-level control, it is possible to achieve reliable timing results with appropriate code. However, due to their low clock speed and therefore slow response to external stimuli, they are not capable of implementing high-performance TDCs. One common technique, used in low-performance ultrasonic Time-of-Flight (ToF) applications, is to capture the value of a counter when the ultrasonic signal is emitted (the 'start' signal) and then capture the value of the counter again when the ultrasonic signal reflects back into the transceiver (the 'stop' signal). The counter counts up once per clock tick, and therefore the resolution of the TDC is equal to the system clock period. As this may be in the region of tens of megahertz (16 MHz is a common clock frequency), the resolution would be in the range of tens of nanoseconds (66.67 ns

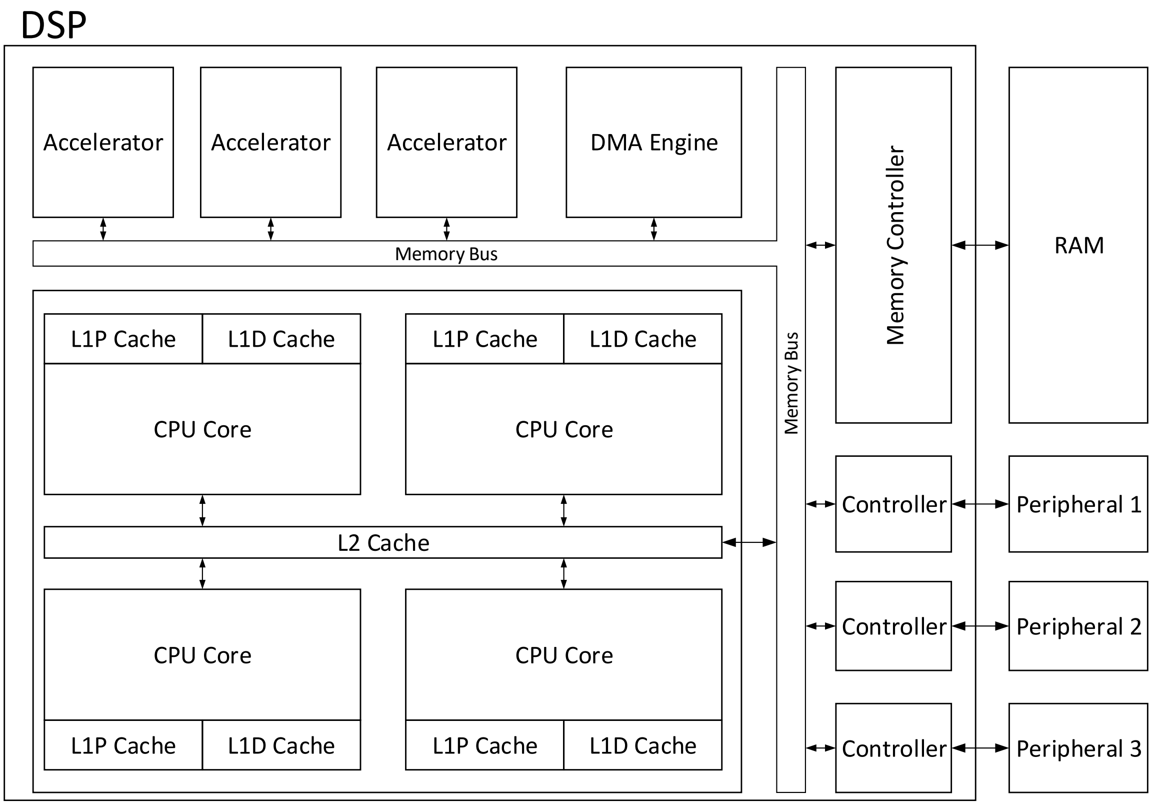


Figure 2.3: Example internal layout of a DSP.

for 16 MHz). Care must be taken when considering how and when the counter is captured: if there is uncertainty in the delay between toggling a pin on the CPU core and the physical pin toggling, or if there is uncertainty in the time taken for the CPU to react to a pin toggling (e.g., entering an interrupt service routine), this will affect the precision of the conversion. Therefore, code paths may need to be carefully designed to minimise or eliminate uncertainty and timing variation to achieve acceptable precisions, depending on the application.

2.2.5 Analog Systems

Analog systems are an exception compared to the other hardware types in this list, as they rely on the voltage of a signal to contain information rather than just the presence or lack thereof. Due to this, analog systems often require significantly fewer components than a digital system of equivalent performance. Therefore, they were ideal choices in the early days of TDC designs as each component required a separate Integrated Circuit (IC) which came with an associated cost and physical area on a circuit board, so keeping component count low was important.

However, for all their gains in reduced component counts, they introduced issues with signal integrity and noise. Unlike purely digital systems which suffer from near immunity to thermal and voltage noise, analog systems are very sensitive to noise and this limits their performance significantly. Voltage noise, thermal noise and leakage all distort the perceived voltage at the input to an analog component, reducing the precision, linearity, and accuracy of the conversion to a digital number. To correct for this, it is essential to introduce expensive components with low noise floors, complex power delivery systems with multi-phase power and lots of smoothing capacitors with low Equivalent Series Resistance (ESR) for high-frequency operation and employ careful circuit design to minimise signal crosstalk and thermal fluctuations. All of this increases system cost and design complexity, so as digital logic has become cheaper per-component (particularly with multiple components being implemented on a single IC), analog methods have fallen out of fashion.

Analog systems have seen a miniature revival with Analog Mixed Signal (AMS) and hybrid digital-analog designs. In these designs, all the analog circuitry is implemented on a single IC along with the digital circuitry. These designs reduce the Bill of Materials (BoM) and thus regain some of the losses seen when implementing high-performance analog systems with discrete components. In addition, the hybrid digital-analog systems do not rely entirely on analog conversion, but instead use a mix of digital and analog to achieve a higher performance with fewer components. Often, with these hybrid systems, due to the low resolution of the analog parts or the inherent matching achieved by having the components close together on the same integrated circuit, a lot of the issues with component variation and noise are no longer important. However, due to being near other components and digital logic, crosstalk becomes a much more significant issue and designs must be carefully planned to minimise crosstalk.

2.2.6 Discrete Logic

Discrete logic is the generic term for connecting multiple single-function Integrated Circuits (ICs) together to accomplish a more complex goal. For example, one might combine an IC that implements an adder with an IC that implements a multi-bit flip-flop (i.e., a register, alternatively two integrated circuits that implement multi-bit latches) and an IC that generates a clock signal to implement an accumulator circuit. With one of the inputs on the adder tied to the value of one, the accumulator becomes a counter implemented in discrete logic, which can then be combined with more flip-flops to create a TDC. While each IC is inherently simple in its purpose, cascading and looping these components can accomplish complex tasks, and this is how CPUs, GPUs and MCUs were built in their early days.

Discrete logic, while simple to implement, has many drawbacks, and so isn't seen often in modern systems. The design takes large amounts of space due to the packaging on each IC (including both the enclosure and the pins), the space required to route signals between ICs and the power distribution to each IC. It also has performance issues to the relatively low bandwidth and high delay between pins of individual ICs compared to within a single device. Cost is also a major drawback, as each IC has significant costs associated with it due to chip packaging and shipment. As a result, outside of some niche applications, discrete logic is a relic of the past since it is much more efficient to integrate all the required functionality onto a single chip.

2.2.7 FPGAs

Field-Programmable Gate Arrays (FPGAs) are the mid-point between programmable processors and custom integrated circuits. Unlike custom integrated circuits, the logic and wires present on an FPGA are fixed. However, unlike a programmable processor, the purpose of and interconnection between the logic is not fixed, but instead configurable. Using a layer of SRAM cells which store configuration bits for the FPGA, the routing is configured to arbitrarily connect logic components together, and the logic is configured to implement arbitrary binary functions. In addition, an FPGA will contain state-holding elements such as latches or flip-flops to store data from one cycle to the next, as well as dedicated clock routing resources that provide balanced delay paths to all elements on the device, reducing clock skew issues. Finally, the FPGA will include IO components to take signals in to and out of the chip, as well as 'IP' blocks which implement common functions efficiently.

FPGAs are a common choice for TDC designs. Using the configurable routing and program-ming the look-up tables, custom logic can be designed to perform TDC tasks. Common examples of this include using the carry chains as delay elements and the D-type flip-flops as arbitrators to capture the state of the delay line. It is common for FPGA designs to quantise with respect to a system clock since this can be distributed to the flip-flops via the clock tree and clock grid.

2.2.8 ASICs

Application-Specific Integrated Circuit (ASIC) is the general term for fabricating a custom IC to implement a specific application. Unlike the devices we have seen until now, they are generally not configurable or programmable outside the intentions of the designer (e.g., clock speed, number of active inputs, calibration etc.). The benefit of using an ASIC is the flexibility at design-time: any system that is physically possible in the available area can be implemented on an ASIC and the low per-unit cost at high volumes. However, the drawbacks are the large upfront cost, design complexity, long time-to-market, and lack of modifiability after fabrication.

ASICs are very popular platforms for TDC designs due to the lack of restrictions compared to FPGAs. Analog Mixed Signal (AMS) and hybrid analog-digital designs are only possible on ASICs, as are some of the more complicated architectures such as Vernier TDCs, Vernier Rings, Successive Approximation TDCs, Gated Ring Oscillators, Pulse-Shrinking TDCs and more. Also, ASIC performance is generally a factor of 10 higher than FPGA performance at the same process node due to the lower load per-cell, more optimised routing, more optimised logic elements and tighter control over sources of uncertainty such as clock skew. However, in the deep-sub-nanometre region that modern process nodes occupy, the prohibitively large development and manufacturing cost of the latest process nodes results in ASIC designs rarely being on process node parity with FPGA designs. The gap in process node parity between ASIC and FPGA designs has been swiftly growing as FPGAs can benefit from economies of scale, while ASICs are restrained by the cost of small production runs. This has resulted in the performance gap between FPGA designs and ASIC designs narrowing significantly even though the architecture gap is still present.

2.3 FPGAs in Detail

2.3.1 FPGA Overview

The goal of an FPGA architecture is to provide a flexible foundation on which a wide range of digital systems can be built. To accomplish this, it must determine an optimal trade-off between the flexibility of the architecture and the optimality for common designs. The more flexible the base architecture, the more types of digital system can be built. However, as elements become more flexible, they also become less optimal for common designs, resulting in greater area utilisation, slower clock speeds and higher costs to implement a digital system. Therefore, the architecture is optimised to perform well for common tasks and provide basic support for uncommon tasks.

The impact of non-optimal logic elements on area, clock speed and cost often compounds itself. The lower clock speeds can be solved through heavier pipelining, cascaded and parallel logic and cycle stealing, but this increases the required area. As the required area increases, the latency of routing signals around the design increases, thereby reducing clock speed again. Also, as the area increases, the cost of the FPGA increases, both in terms of running cost (energy consumption) and initial cost (purchase of a sufficiently large FPGA). This is the reason why Chapter 4 will investigate DSP blocks, as they are more heavily optimised and therefore can reduce area, increase clock speeds and reduce costs.

The architecture of the FPGA can be roughly divided into three aspects: the configurable logic which implements individual binary functions, the configurable routing which connects multiple binary functions together to produce a digital system, and the configurable peripherals which allow the digital system to communicate with the environment. All of these are configured via a layer of SRAM cells which enable and disable connections in the routing, compose the truth table of the logic and control voltages at the pins.

The configurable logic can similarly be broken down into three main components: the Look-Up Tables (LUTs) which generate arbitrary binary functions on parallel inputs, the carry logic which optimises the passing of signals between adjacent bits, and the storage elements which store the results for use on the next logic cycle. The carry logic is not necessary in a logic block, but due to the prevalence of binary functions that are defined relative to adjacent bits, operation without the carry block would result in significantly worse performance.

2.3.2 FPGA Building Blocks

2.3.2.1 LUT

The implementation of a LUT is often designed as a tree of multiplexers (MUXes) attached to a small SRAM memory and the input signals. The input signals attach to the select inputs of the MUX tree, and the SRAM memory is attached to the data inputs. When the input to the LUT changes, the MUX tree selects a different bit from the SRAM to forward to the output of the LUT. By changing the data stored in the SRAM, the logic function of the LUT can be changed by outputting a different bit for a particular combination of inputs.

For example, a LUT2 has four different possible combinations for its inputs, meaning the SRAM must contain four bits (one bit per possible input combination). There will be two layers in the MUX tree, with the first layer consisting of two MUXes with their select input attached to the first input bit (I0) and their data input bits attached to the four SRAM outputs. The second layer will contain a single MUX, with the select input connected to the second input bit (I1) and the data inputs connected to the outputs of the first layer of MUXes. When I0 changes, each of the two MUXes in the first layer will switch which of the two SRAM bits they are forwarding to the second layer, and the second layer will continue to pass one of these two bits to the output. When I1 changes, the MUX in the second layer will switch which first-layer MUX it is forwarding the output from. Due to this architecture, it is often the case that the last input bit (I1 in a LUT2, I5 in a LUT6, etc.) will have the shortest critical path, making it ideal for use on the critical path of the circuit, while other input bits have longer delays, making them ideal for non-critical paths.

2.3.2.2 Carry Chain

Many operations, such as addition and numerical comparison, result in dependencies between adjacent bits. In order to implement these very common operations quickly, the carry chain provides a fast logic path (main carry path) between adjacent bits, rather than using general-

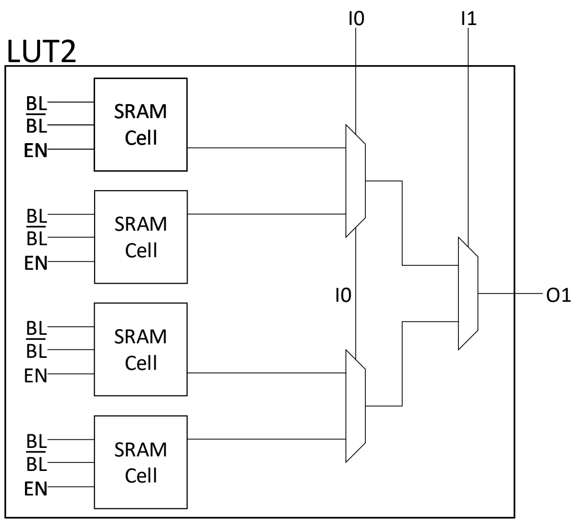


Figure 2.4: Implementation of a two-input LUT using multiplexers and SRAM cells. The SRAM cells store the truth table, and the multiplexers select the correct element from the truth table.

purpose routing to loop back to the input of the adjacent LUT. This fast path has been the target of most FPGA TDC designs in recent years.

The implementation of the carry chain is a relatively simple prospect. The main carry path consists of one two-input MUX per stage, which selects between a bit generated by the LUT (in the case of carry generation or deletion) and the carry from the previous stage (in the case of carry propagation). This minimalistic path allows the carry to propagate through the chain much faster than by using the switch to loop back to each successive bit, as the carry is always utilising a dedicated path. As the most common usage of a carry chain is for addition or subtraction, which also require an eXclusive OR (XOR) between the data and the carry bit to produce an output, there is also a built-in XOR gate to avoid routing to a LUT in another logic block to implement the XOR operation. This is all positioned between the LUT and the flip-flops, allowing a single logic block (or column of logic blocks where the addition has a large number of bits) to implement an addition and store the result in a register without using any general-purpose routing (switch, local or global interconnect).

2.3.2.3 Storage Element

The storage elements are an important component in an FPGA for the device to have memory, to interface with other systems using synchronous protocols, and to regulate the flow of data through the device. Without flip-flops, the designer would need to purposefully waste large chip areas to generate delays to keep data synchronised in the time domain and ensure that all data reaches its destination in the correct order. As most systems are synchronous systems, with developers aiming for high clock speeds and deeply pipelined logic or vast state machines, FPGAs include multiple storage elements in each logic block to provide sufficient granularity and quantities of storage.

Each storage element normally acts as a D-type Flip-Flop (DFF). As there is a custom connection to the clock grid to efficiently route the clock signal to the clock input of each DFF, the result presented by any LUT or carry chain can be quickly and efficiently captured by the nearby DFF. This trivialises the routing and some of the delay analysis for most systems. If a system is under-performing, the storage elements can be reconfigured to act as latches, which allows a technique known as 'cycle stealing' to be used to meet difficult timing constraints.

In addition to the individual DFFs present on the device, there are two other types of storage: distributed RAM and RAM blocks. RAM blocks are custom logic blocks designed to hold larger quantities of data than the DFFs in the logic blocks. As they are random-access memory, not all data can be written or read simultaneously, which is a disadvantage compared to the DFFs. However, they often hold around 32 kbits of memory, compared to DFFs which would store less than two hundred bits in the same area.

Distributed RAM refers to the use of the configuration memory for the LUTs as a RAM or shift register. Distributed RAM is much less dense than RAM blocks, with about approximately 5 kbits (20 slices, four LUTs per slice, 64 bits per LUT in Xilinx's case) in the same area as a 32 kbit RAM block. However, it has much wider data widths due to the shallow memory depth and is significantly more configurable than RAM blocks due to being implemented on the LUTs. It is also much faster to read than RAM blocks, as the look-up is asynchronous (although it can be registered through the DFFs if desired).

2.3.2.4 DSP Blocks

Due to the increasing complexity of designs targeting FPGAs, as well as the architectures themselves, it becomes more attractive to introduce more specialised, higher-performance logic to optimise for very common tasks. As multiplication and addition are the most common operations performed in a digital system, custom logic which directly implements these two tasks is beneficial in larger FPGAs. Therefore, most FPGA manufacturers have introduced Digital Signal Processor (DSP) blocks on their fabric.

A DSP block operates similarly to a scaled-down version of a DSP's arithmetic pipeline. The DSP block contains both a multiplier and an adder, with optional registers between each stage and configuration bits that can be set to modify the behaviour of the block. Unlike a DSP, it does not have the control logic to convert instructions into configuration bits or peripherals to interact with the environment - these matters are left to the FPGA engineer and general-purpose logic. The general-purpose logic is used to control the configuration of the DSP block, and then the DSP block processes data quicker and more efficiently than the general-purpose logic.

Considering DSP blocks from both major FPGA manufacturers (Intel, Xilinx), there are many common design decisions between the two. Both architectures include systolic registers, a pre-adder, a relatively large multiplier, a post-adder, a loopback path, and dedicated paths to chain DSP blocks together. These are included in both architectures as they are important operations for a DSP block. However, where Xilinx makes all its registers systolic for maximum flexibility, Intel makes only a few registers systolic to maximise clock frequency. This pattern seems to be reflected across the entire architecture, with Intel optimising more heavily for the most common designs, while Xilinx focuses more on flexibility and leans on its tools to efficiently utilise the fabric.

Most DSP blocks can be chained together for the purpose of operating on larger inputs or implementing filters with a pipeline methodology. DSP block inputs and outputs can pass control bits and partial results between DSP blocks via dedicated paths to perform larger operations.

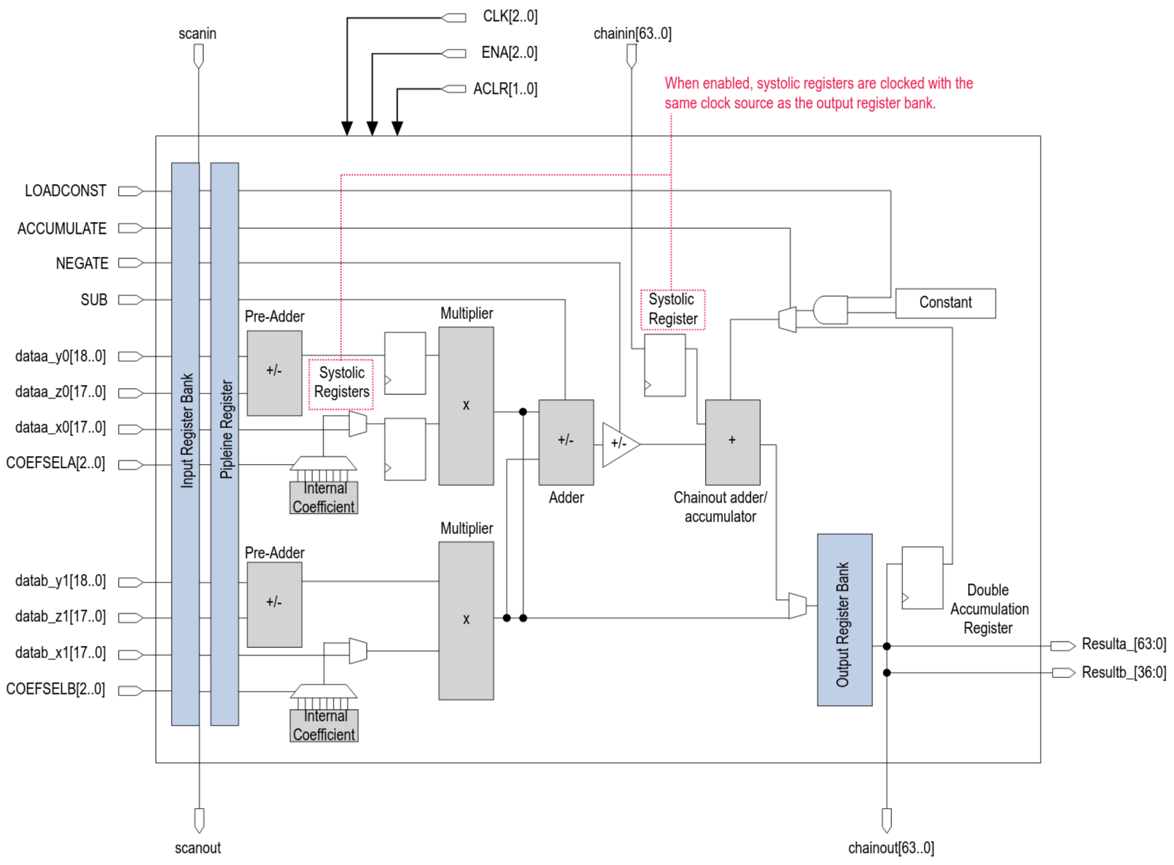


Figure 2.5: Architecture of Intel's variable-precision DSP blocks [3].

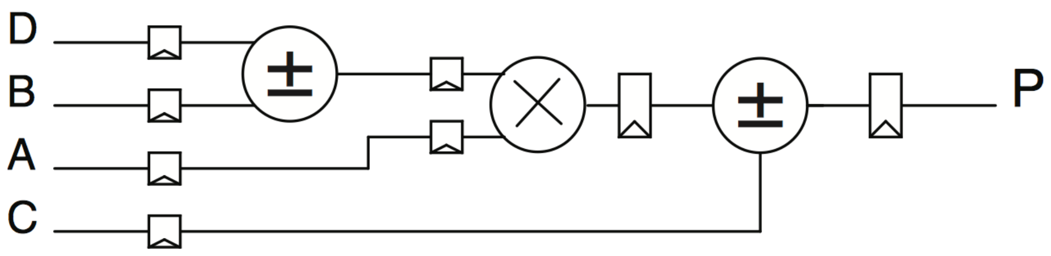


Figure 2.6: Simplified architecture of Xilinx's DSP48E1 blocks [4].

2.3.2.5 Input/Output Buffers

The input/output buffers (I/O blocks) control the connection between the FPGA fabric and external pins. When a buffer is set to output, an internal signal will be routed to the buffer's input, and the buffer will amplify this signal to the requested logic standard and drive the signal onto the pins.

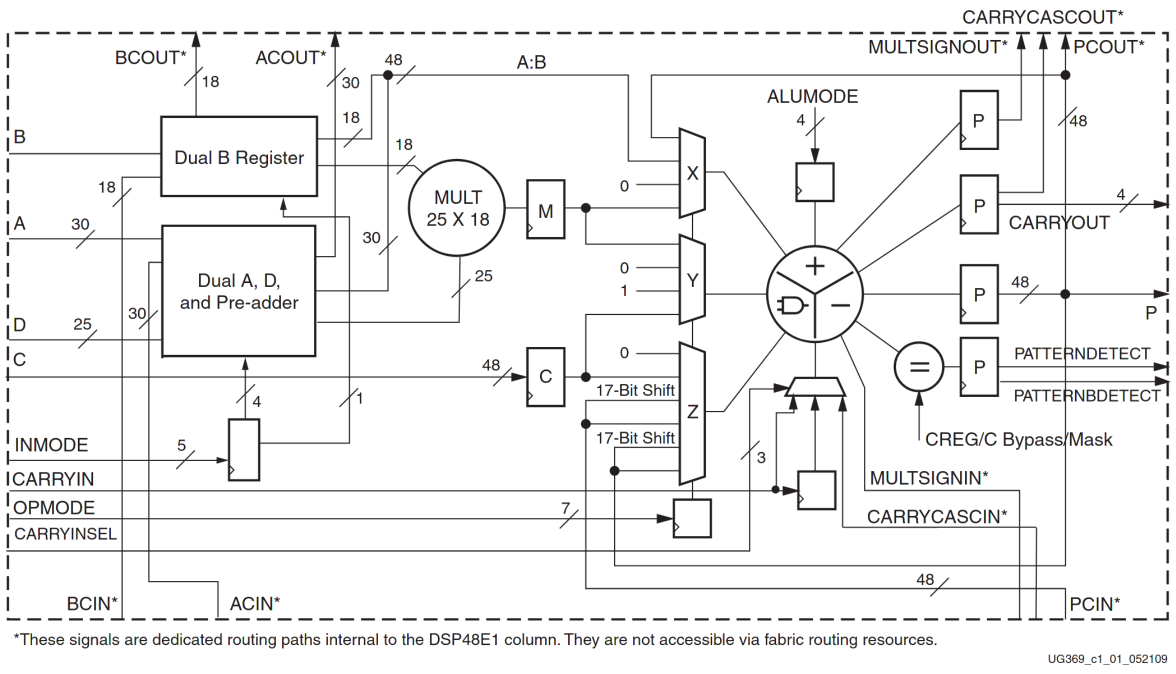


Figure 2.7: Detailed architecture of Xilinx's DSP48E1 blocks [4].

Alternatively, when the buffer is configured as an input, it will act as a comparator, comparing the voltage at the pin to a reference voltage and then driving a logical one or zero onto the FPGA fabric. In many cases, a pin must be required to act as both an input and output (for bi-directional data), in which case the buffer will act in a tri-state mode, with an extra input to toggle between input and output.

Input and output buffers must be suitably flexible to interface with most other common integrated circuits without external level-shifting circuitry. However, as the range of supported voltages and drive strengths increases, so too does the size of the I/O block as well as the supporting power conditioning circuitry required to achieve the correct voltage and current levels. Therefore, manufacturers limit the range of supported standards to those that are likely to be encountered by the FPGA. As a result, interfaces with higher speeds are often only supported at lower voltages and without tri-state support as most devices capable of high speeds are also low voltage devices, and tri-state operation is much slower. Comparatively, slow interfaces tend to be available in a wider variety of voltage standards and with tri-state support.

For very high-speed serial interfaces, a custom block called SERDES (SERialiser/DESerialiser) is used. When deserialising, this block samples a high-speed serial signal multiple times per clock cycle and then outputs these samples as a parallel chunk of data. Similarly, when serialising, the block takes a chunk of parallel data and outputs all bits sequentially in a single clock cycle. This allows the high-rate serial interface to be down-converted to a lower-rate parallel interface which can be processed by the FPGA fabric.

2.3.3 FPGA Architecture

2.3.3.1 Logic Blocks

The design of the logic blocks has seen a gradual increase in complexity as designs and chip sizes have increased. In the earliest FPGAs, a logic block might consist of one to four Look-Up Tables (LUTs) each with just two inputs (thus called LUT2), each connected to a latch. Either the direct output of the LUT or the latch would then be connected through the switch to the routing. As designs have become more complex, the logic block has increased in size. Modern FPGAs will often contain four to eight LUTs, each with six inputs (LUT6) and up to two outputs. These LUTs are then connected to between eight and sixteen Flip-Flops (FFs) which can also be configured as latches. In addition, there is often an optional carry chain placed between the LUTs and the FFs to accelerate addition and other multi-bit operations, as well as multiplexers to combine multiple LUTs into larger LUTs and XOR gates to accelerate addition and subtraction operations.

2.3.3.2 Routing Resources

Design of the routing becomes a very difficult task to manage as FPGA sizes increase. Originally, FPGAs were descended from CPLDs (Complex Programmable Logic Devices) and were fully connected, allowing any input to reach any output, and vice-versa. However, as the number of logic elements increases, the amount of routing in a fully connected scheme increases exponentially. Therefore, a more efficient method of routing is required. To solve the problem, the principle of locality is key: as the distance between two elements increases, the probability of needing to connect them decreases. It is very rare for a single signal to be distributed to the entire design (except the clock signal, for which we have dedicated routing). Therefore, it is not required to have sufficient routing to route all signals everywhere else simultaneously. Instead, routing is tiered, with more routing available to route signals over short distances, and less routing available to route signals over long distances. To maximise the locality of the design, it is important to place related logic elements close together, in a ratio that minimises the need to travel long distances. For this purpose, FPGA manufacturers decide on a set of logic that covers most applications and group it together in a logic block, then apply routing between logic blocks.

The routing tiers can generally be grouped into five categories: the dedicated paths, the local switch, the local interconnect, the global interconnect, and the clock tree. The local switch is the second tier of routing and is responsible for routing signals from outputs of a logic block to its inputs, as well as routing signals to and from other tiers of routing. For example, the switch may receive a signal from the local interconnect, as well as a signal from the output of the attached logic block, and route these two signals to two different inputs of the logic block, as well as routing the output signal to the global interconnect.

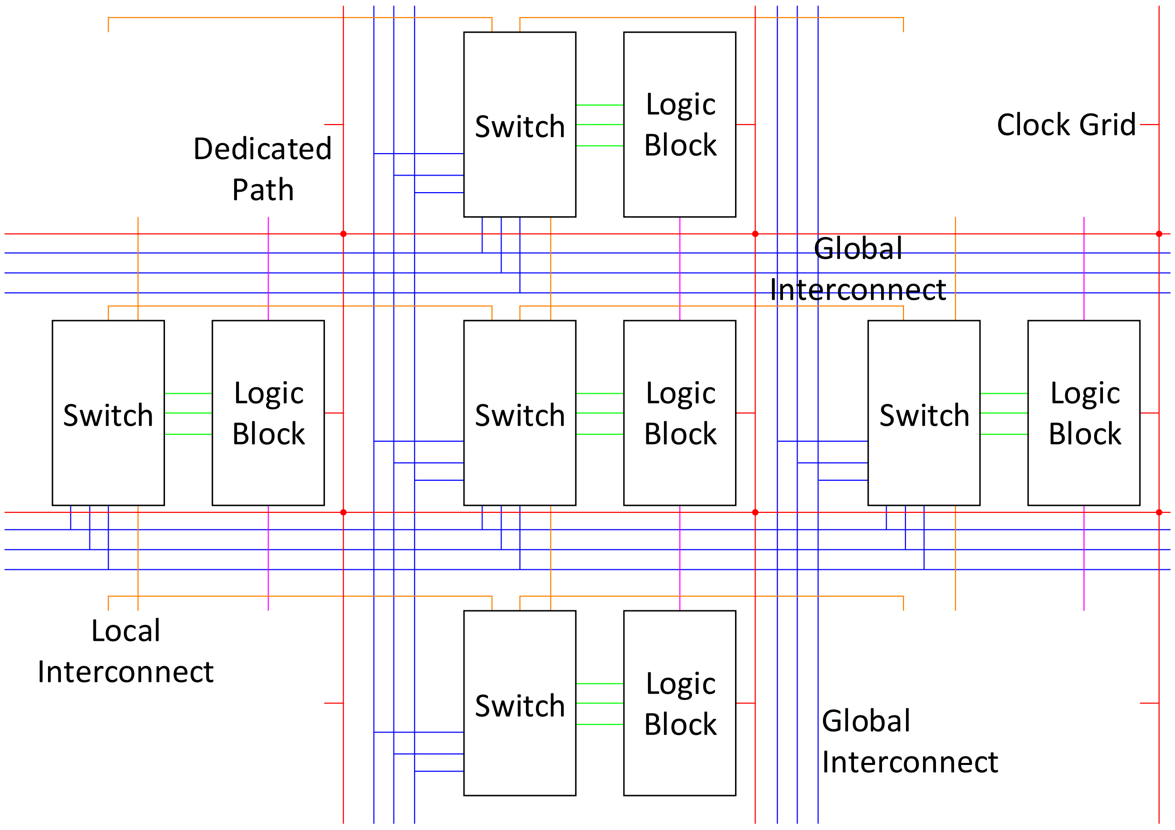


Figure 2.8: Routing structure of an FPGA, showing local switch (green), local interconnect (orange), global interconnect (blue), clock grid (red) and dedicated path (purple) connections.

The local interconnect is the third tier of routing. The local interconnect is a group of paths that connect adjacent or semi-adjacent logic blocks together. As these paths do not stretch the entire length of the device, and thus require less area, significantly more of them are available. These paths are most commonly used for cascaded or iterative logic, where a signal is passed from one logic element to the next with very little fanout (fanout: ratio between the quantity of destinations and strength of the source of a signal). Using the local interconnect and switches together, a signal can be passed over long distances, but this is very inefficient as the signal must pass through a large quantity of logic to reach its destination.

The global interconnect is the fourth tier of routing. The global interconnect usually consists of horizontal and vertical buses spanning the entire length or breadth of the chip, often with gates at regular intervals to regenerate the signal and allow the bus to be split into multiple smaller buses. If a signal needs to be transported over a long distance, or connected to multiple endpoints, it will be connected to one of these buses, and many of the buses may be connected together to deliver the signal over a wide area. For long-distance routing, these paths are more efficient than chaining local interconnect, but since they span the entire length of the device, they require a lot of area and thus are fewer in number. Also, as the connection from the source to each sink is a direct path on a Manhattan grid, the signal skew can be very large between different endpoints, so it is not ideal for timing-sensitive signals.

The clock tree is the fifth and final tier of routing. The clock tree is a balanced and heavily buffered path to all points on the device, minimising signal skew and maximising fanout at the expense of delay, flexibility, and quantity of paths. A modern FPGA may have only tens of clock trees, often ten or less per clock region, of which there are several per device. Clock routing may be split into global clock routing, which distributes a signal to multiple clock regions, regional clock routing, which distributes a signal to multiple clusters of logic, and local clock routing, which distributes a clock signal to individual logic elements. The global and regional clock routing is very carefully balanced, often utilising an H-tree to balance paths. Conversely, the local clock routing will often utilise a clock grid to reduce the delay and complexity of routing to each individual element, at the cost of a small quantity of clock skew.

The dedicated paths are the lowest level of routing, and comprise of a large number of very short, dedicated wires between logic elements. These paths are often within a logic block, for example from the output of a Look-Up Table (LUT) to the input of a flip-flop, but may also be between strictly adjacent logic blocks, for example the connection between the carry out of one carry chain and the carry in of another carry chain. They only have one or a few possible sources, and a single possible destination. These are the fastest paths in the device, but also the least flexible, and so their usage is heavily dependent on the target application.

Some FPGAs have started to utilise stacked integrated circuits to increase logic density per chip. This gives rise to a new tier of routing - Through-Silicon Vias (TSVs). This tier of routing very similar to global interconnect, however, global interconnect would best be described as 'planar interconnect' in this scheme as it is only able to route within a single plane of the device, and the TSVs are required to route between layers. As the accuracy of stacking ICs is not as high as the implanting of metal wires, TSVs require significantly more space than the planar interconnect and so even less are available, although not to the same extent as the clock trees. Similarly to the planar interconnect, they do not balance delay, and so skew between layers can become quite large, often exceeding the length of the clock cycle and requiring buffering.

2.3.3.3 Slice Organisation

Modern FPGAs also vary the logic blocks available to optimise for multiple applications. For example, some logic blocks may have more FFs, others may have more LUTs, and some will allow the configuration memory of the LUTs to be chained together to implement a shift register or distributed RAM. Manufacturers also replace a small subsection of the logic blocks with other custom logic. Most commonly among these are Digital Signal Processing (DSP) blocks which implement multiplication, accumulation, shifting, binary operations and pattern detection, RAM blocks which implement high-density SRAM, clock management blocks which generate and distribute high-frequency, low-noise, low-skew, and phase-related clock signals to the sequential elements on the device; and IO blocks which implement high bitrate communications (SERialiser-DESerialiser, SERDES).

Using Xilinx's Artix 7 FPGAs as an example, a modern logic block ('Configurable Logic Block' or 'CLB' in Xilinx terminology) will contain two 'slices', each of which contains four LUT6s; a four-element carry chain with dedicated paths from the previous logic block and to the next logic block; some extra custom gates such as two extra layers of multiplexers to combine the four LUT6s to two LUT7s or a single LUT8 and XOR gates to accelerate addition; and eight FFs which can also be reconfigured as latches. Xilinx connects each CLB (and thus its two slices) to a switch, which is then responsible for routing to other logic blocks. Dedicated carry paths operate between CLBs, so a carry chain in one CLB will connect to the corresponding carry chains in the vertically adjacent CLBs, with the carry input connected to the CLB below (south of) the current CLB, and the carry input connected to the CLB above (north of) the current CLB. This, along with the routing bias which contains more (and shorter) paths to connect the output of the current CLB to the input of the CLB to the right (east), results in a general preference bit parallelism to be arranged vertically (with the MSB being north of the LSB) and bit sequentialism to be arranged horizontally (with a bit to the right being the product of a bit to the left). In a sense, this is a self-fulfilling prophecy: by designing the routing as such, it encourages designs to follow this regular approach, and by having designs follow this regular approach, the routing can be optimised and reduced to just the small amount present. If a design were to intentionally go against these conventions, it would likely experience poor performance, routing congestion or even a failure to implement on the device due to a lack of available routing.

In comparison, Intel's modern logic blocks (Logic Array Blocks, LABs) contain two groups of five Adaptive Logic Modules (ALMs), with the LAB control placed between the two groups. Carry signals propagate south through ALMs in a LAB, with an optional dedicated path to the next LAB. The ALM contains four LUT4 blocks, two carry chain elements (resulting in twenty bits of carry chain per LAB), four DFFs (not configurable as latches, so no cycle stealing) and some extra multiplexers to combine the LUT4 blocks into a LUT6 with two outputs. Each ALM has eight inputs, with various degrees of sharing between LUTs, and two outputs, each of which can take the value of any of the four LUT4 elements (allowing for a true LUT6 as well as a complementary LUT6 using the same truth table and two bits to differentiate between the outputs). Similarly to Xilinx's LUT elements, the LUTs in Intel's ALMs can be used as distributed RAM, with a 32x2 (32 deep, 2 wide) memory per ALM, resulting in 32x20 per LAB (compared to 64x8 per CLB). Intel also has RAM blocks on their fabric, with 20 kbit per block and a data width of up to 40 bits.

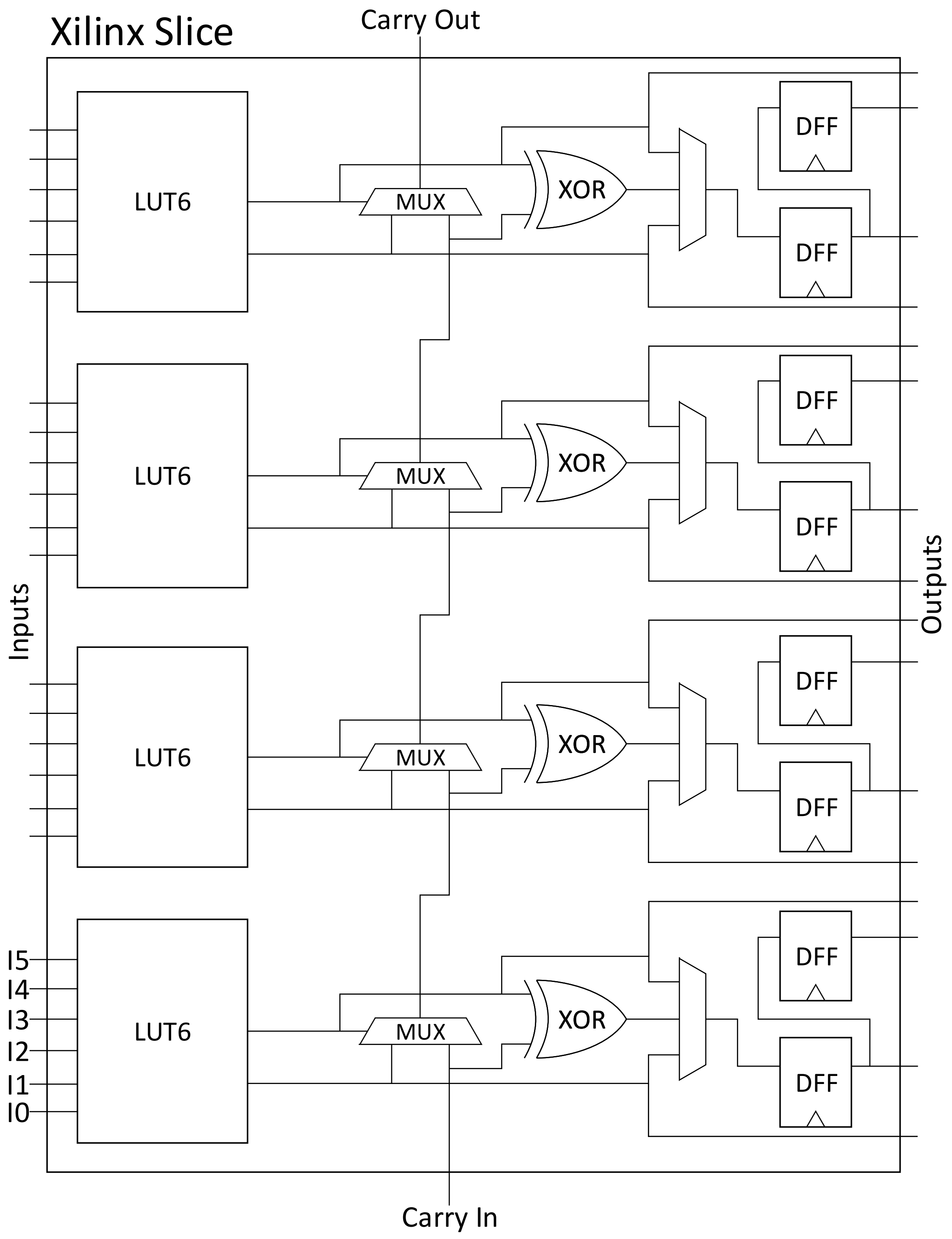


Figure 2.9: Architecture of a Xilinx ‘slice’ (two slices per CLB/logic block).

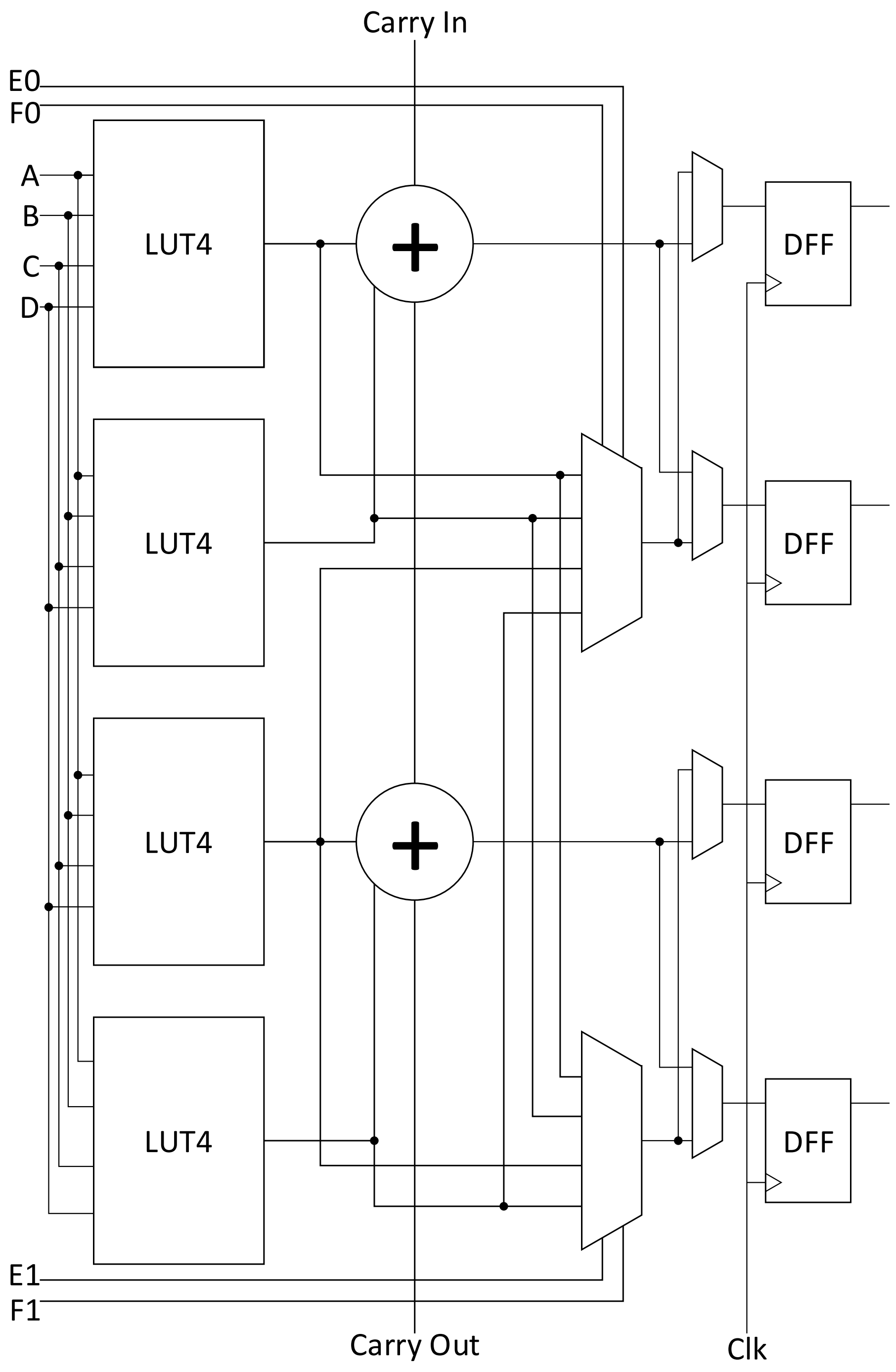


Figure 2.10: Architecture of an Intel 'ALM' (10 ALMs per LAB/logic block). Input port names truncated (A = dataa, B = datab, E0 = datae0 etc.).

2.4 Conclusion

In this section, we have described the various processor types available to use in a system. We have evaluated all of these with respect to TDC implementations and found that only FPGAs, and ASICs are suitable for high performance TDC designs. Analog systems and discrete logic were suitable for some older systems, but have significant drawbacks in terms of power, repetition rate or resolution, making them unattractive choices for modern TDCs, and CPUs, GPUs, MCUs and DSPs are only suitable for low-performance TDC applications such as Ultrasonics and SONAR.

Between FPGAs and ASICs, ASICs have been shown to be much more flexible due to the wider variety of logic types available, the capability to operate analog components and dynamic logic and more. However, the cost of ASICs is much higher than FPGAs. ASICs have a much higher tooling cost (static manufacturing cost), longer time-to-market, more complex development process and are less readily available than FPGAs. Also, reproduction of results obtained by other researchers is difficult without having access to the same fabrication facility. Comparatively, FPGAs are available off-the-shelf, can be developed for and tested relatively rapidly (although not as rapid as CPUs and MCUs), have a relatively short time-to-market and have a much lower tooling cost. Therefore, we examined FPGAs in more detail.

We described the main components of an FPGA in depth, with specific attention paid to the purpose of the components, the design trade-offs which lead to the current architecture and the applicability of each element to TDC designs. In particular, we observed the different levels of routing available to forward signals between logic elements and the effect that the routing has on signals. We also examined the logic elements in depth, describing the purpose of each component and its applicability to TDCs. Finally, we looked at some of the more specialist blocks such as DSP block and input/output block, observing their functionality and the role they might play in a TDC system.

While we have concluded ASICs and FPGAs to be the most appropriate for modern TDC implementations, a system that requires a TDC will often be required to perform other tasks such as device control, calibration, data read-out, signal fusion and signal processing. For this purpose, many different pieces of hardware (e.g., CPU, FPGA and DSP) will often be integrated onto a single integrated circuit (IC). Sometimes, this may even result in a change in hardware choice for the TDC: an ASIC may be optimal for a given task, but an FPGA may be chosen instead due to integrating a CPU or MCU on the same IC, which can then be used for control without introducing a second IC and interfacing the two. Similarly, many ASIC based TDCs use FPGAs to assist with data readout and processing to reduce development time and circuit complexity.

Chapter 3

Review

3.1 Abstract

Time-to-digital converters are used for a variety of purposes to collect data from the environment in which a system operates, and so the study of how to implement them to required standards and efficiently is an important topic. This chapter looks at the various methods of time-to-digital conversion available in the literature and compares them on the aspects of count rate, accuracy, precision, number of channels and logic utilisation. A truncated version of this review, focusing only on techniques that had not appeared in review literature, was published in IEEE Transactions on Instrumentation and Measurement as “A Review of New Time-to-Digital Conversion Techniques” [8]. A more recent review, focusing only on techniques published in 2020, was presented at NoMeTDC 2020 [9].

3.2 Introduction

Time-to-digital converters (TDCs) play a vital role in almost all computational systems in existence. From their appearance in Phase-Locked Loops (PLLs), where they measure the difference between the loop and the reference clock to avoid clock drift, to time-of-flight applications where the time between an emission and a reception is measured to discover information about an object from which a signal is reflected, or the environment through which the signal passes. In addition, there are also quantum versions of these applications, where the signal is a single photon, and the PLL or time-of-flight measurement must perform well despite some photons being lost in-flight. They also make an appearance in medical imaging, as some systems such as positron emission tomography (PET) and magnetic resonance imaging (MRI) use the time of flight or absorption time of tissues or substances to form an internal image a complex structure such as a human body.

To perform these applications efficiently, it is important to choose the correct TDC design to fit the application. Rarely does a system choose a TDC design from the outset, but rather choose a performance target and hence decide on the required TDC. This requirement will vary not only according to said performance requirement, but also according to the underlying technology on which the TDC is built. If the technology platform is advanced, then it may be possible to use a less accurate, low logic utilisation TDC, whereas on a more basic platform a more expensive TDC may be required to hit performance targets. Similarly, in some cases the logic utilisation of the TDC required to hit performance targets would be so large that it requires the designer move to a more advanced technology platform to reduce TDC complexity.

Therefore, this section aims to provide an overview of the TDC architectures available, along with their benefits and drawbacks, to aid the designer in choosing an appropriate TDC to satisfy their design requirements. It will look at TDCs implemented both on field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs), which differ vastly in terms of the resources and flexibility available, both in production and in the field and hence benefit more or less from various techniques.

The rest of this section will be structured as such: in section 2, various TDC architectures will be explored, with a subsection dedicated to each TDC design. Section 3 will then compare the results achieved from the various designs in the literature, while considering the difference in technology platform. This will lead on to section 4, where the various advantages will be compared, with section 5 concluding with recommendations on which architectures provide the best trade-offs. [40, 44–51]

3.3 TDC Architectures

Table 3.1: Summary of TDC Architectures.

Architecture	Precision (SSP)	Range	Dead Time	Area	Power
Counter	500 ps	> 1 s	1 ns	Very Low	Low
Delay Line	20 ps	10 ns	5 ns	Medium	Medium
Looped Delay Line	20 ps	> 1 s	5 ns	Low	Medium
Vernier Delay Line	1 ps	1 ns	100 ns	Medium	Medium
Vernier Ring	1 ps	> 1 s	10 ns	Low	Medium
Pulse-Shrinking	5 ps	5 ns	100 ns	Medium	Medium
Looped Pulse-Shrinking	5 ps	> 1 s	10 ns	Very Low	Medium
Local Passive Interpolation	3 ps	5 ns	2 ns	High	High
Stochastic	100 fs	30 ps	50 ps	Medium	Medium
Successive Approximation	1 ps	1 ms	10 ns	Low	Low
Algorithmic	200 ps	> 1 s	1 μs	Low	Low
Gated Ring Oscillator	20 ps	> 1 s	100 ps	Low	Medium
Stochastic Phase Interpolation	1 ps	2.5 ns	2.5 ns	High	High
Wave Union	< 1 ps	10 ns	10 ns	Medium	Medium
TAC-ADC	10 ps	1 μs	1 μs	Very High	Very High
SERDES	100 ps	> 1 s	200 ps	Low	Medium
Time Amplification	1 ps	1 μs	1 ms	Very High	Very High

3.3.1 Counter TDC

Perhaps the most obvious, and most basic, of TDCs, is the counter TDC. The counter TDC consists of a simple counter circuit with the signal to count upwards is attached to the system clock. In this scheme, the value of the counter is captured on the rising edge of the trigger (which may be synchronised to the system clock), and the difference between two triggers is the difference between their counter values multiplied by the clock frequency.

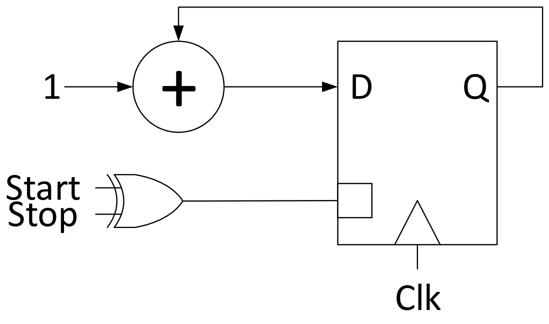
This scheme is very efficient in terms of logic utilisation. The counter's size increases linearly with the number of bits (using a ripple counter). Also, additional channels are cheap to implement, requiring only a single register to store the value of the counter each time.

However, the counter TDC falls behind all other TDCs in terms of accuracy. The clock frequency rises exponentially with regards to the number of bits, which not only imposes a hard limit caused by the speed at which the logic will operate, but also a soft limit caused by the power consumption, which increases linearly with clock speed until the limit of the logic speed, and then cubic with the clock speed as the voltage must be increased to satisfy timing requirements. This is expressed in (3.1), where T is the time period to be measured, T_{clk} is the clock period, N is the max count of the TDC, B is the number of bits in the TDC, A_{FF} is the area of a flip-flop (storage element) and $A_{I:O}$ is the average area of a function with I inputs and O outputs (for example, $A_{3:2}$ is the average area of a function with 3 inputs and 2 outputs).

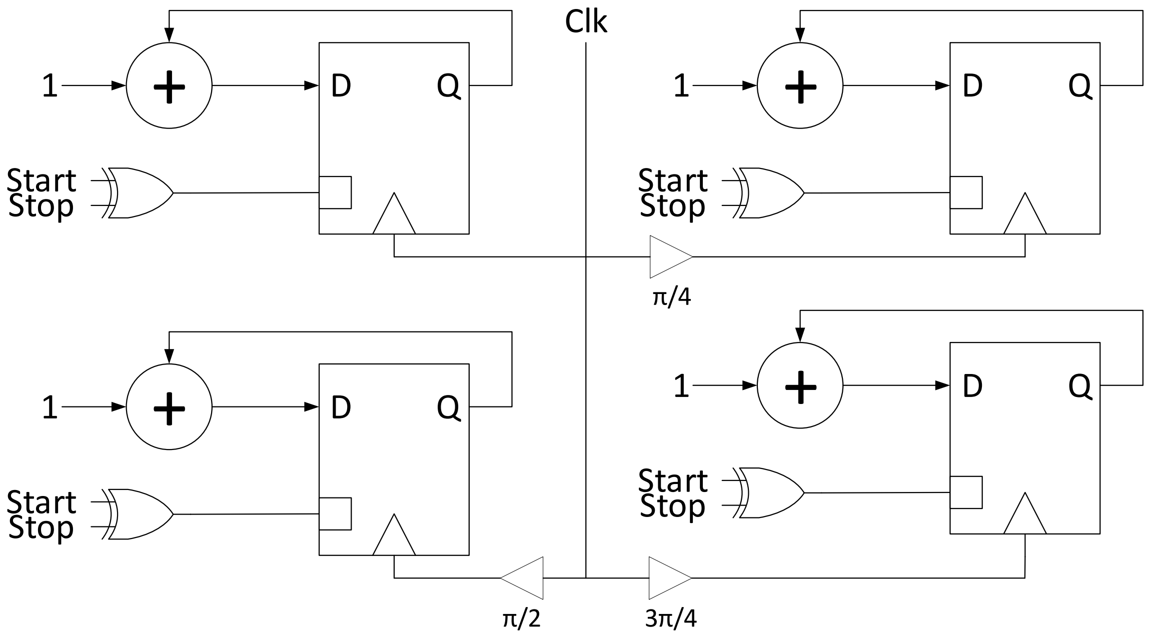
However, the counter TDC is used in most TDC applications. In some cases, it is because the counter TDC is sufficient for the purpose, such as ultrasonic range-finding where an accuracy of a few centimetres is sufficient. Alternatively, the counter TDC may form part of a multi-stage system, where the counter is the top level, producing most the range, and within each clock cycle there is a more accurate time measurement performed by another TDC. The dead time of the counter TDC is one clock cycle. The linearity is entirely controlled by clock drift and jitter, and can easily be controlled down to ~10 ps.

Also, there is a derivation of the counter TDC, which will hence be referred to as a Controlled Ring Oscillator (CRO) which allows some significant improvement without losing the benefits of the counter TDC. By creating multiple clocks with a known phase difference, the accuracy can be increased efficiently for low quantities of clocks. However, as this method is also exponential in the number of bits, beyond 2-4 bits' improvement (4-16 phase-shifted clocks) it also becomes ineffective. It's logic area utilisation is expressed in (3.2), where T_{min} and A_{min} are the delay and area of the minimal delay element in the clock divider / phase generator and N_{ph} is the number of phases in the TDC.

The counter TDC is applicable to all methods of implementation. ASICs pull ahead in terms of accuracy, as they can be seen to reach tens of GHz clock speeds, resulting in resolutions in the range of tens or hundreds of picoseconds. FPGAs experience much lower accuracy, generally only



(a)



(b)

Figure 3.1: (a) A counter TDC; (b) A multi-phase clock

reaching a gigahertz, but also significantly benefit from the reduced area, as most FPGAs will be designed to efficiently implement counters and adders. Also, they are the only form of TDC that can be implemented by software on a general-purpose processor, as a general-purpose processor is capable of both counting and storing the value of a counter.

It is the implementations in general-purpose processors for low-performance applications that make up most of the sole usage of this type of TDC. As these implementations are generally devoid of any scientific progress, they are generally unpublished and hence very few papers, if any, can be found on the sole use of counters as TDCs. On the other hand, almost all TDCs that will be seen from here on make use of a counter in some way: as a coarse stage in a multi-stage system, in each phase of an MPC (which may then be the coarse part of a multi-stage system), or as the low resolution, high range measurement device in a dual-slope TAC-TDC.

$$\begin{aligned} N &= \frac{T}{T_{clk}} \\ B &= log_2(N) \\ A_{CTR} &= B * (A_{FF} + A_{3:2}) \end{aligned} \tag{3.1}$$

$$\begin{aligned} N_{ph} &= \frac{T_{clk}}{T_{min}} \\ A_{MPC} &= B * N_{ph} * (A_{FF} + A_{3:2}) + N_{ph} * A_{min} \end{aligned} \tag{3.2}$$

3.3.2 Delay Line TDC

The delay line TDC is perhaps the most used after the counter TDC. This is a very simple design where the ‘start’ signal propagates through a series of logic elements, and the ‘stop’ signal captures the state of the delay line at a point in time. As each logic element has an associated propagation delay, the time between the start and stop signals is represented by the highest position of a ‘1’ in the delay line.

The delay line TDC suffers from two issues. The accuracy is limited by the smallest repeatable logic delay in the system, with the only way to increase it being to increase the voltage, up to a limit, and the number of delay elements needed (hence the logic utilisation) increases exponentially with the number of bits of range needed.

As a result, most designs couple the delay line TDC with a counter TDC, with the number of delay line elements defined by the optimal crossover in resource requirements between increasing the number of delay line elements and decreasing the clock period of the counter. The utilisation of a delay line TDC is as in (3.3) and (3.4), where T_{min} is the time of the minimal delay element, T is the total delay to be measured (normally 1 clock period) and N is the number of delay elements. A_{PE} is the area of the priority encoder, A_{min} is the area of the minimal delay element and A_{arb} is the area of an arbiter.

The dead time of the delay line TDC is entirely dependent on the decoding logic, with some designs being capable of capturing multiple trigger signals per system clock cycle. The linearity is mainly dependent on variation in the delay elements, both on a local scale (affects DNL) and on a global scale (affects INL).

There is a derivation of the delay line TDC known as the hierarchical TDC. This TDC obtains an improvement to the logic utilisation by using many separate delay lines, each one triggering the next and being half the length of the previous. This allows the hierarchical TDC to directly output the binary value rather than a priority code (which a delay line outputs), which requires more memory cells and a priority encoder to convert to binary. The logic utilisation of the hierarchical TDC is as in (3.5). There is also a version, known henceforth as a differential DL (DDL), which operates in a differential mode, with the difference between the positive and negative signals determining between 0 and 1. This method provides better signal integrity, reduces the effects of NMOS/PMOS mismatch, and mostly avoids metastability, but requires significantly more area than a standard DL, up to 4x in many cases.

Delay line TDCs are implementable both on ASICs and on FPGAs, as is the case for its various derivations. ASICs, as always, obtain better performance than FPGAs, however, the difference is much less than in other methods, as the primitives available on common FPGAs are close to their ASIC counterparts.

When implemented on ASICs, delay lines can feed their final bin into a phase detector (PD), which then controls a charge pump (CP) which feeds the gates of some current-starving transistors, which then current-starve the delay elements in the delay line. By providing a stable clock at the other input of the PD and the input of the delay line, the delay of the elements can

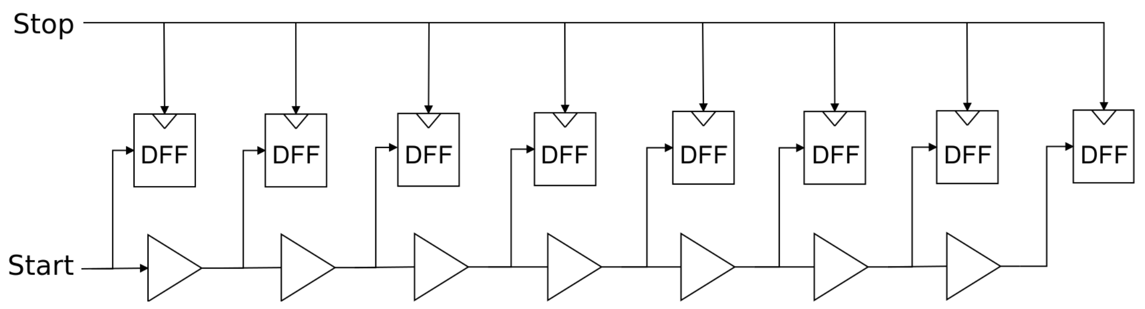


Figure 3.2: A delay line. DFF = D-type Flip-Flop.

be automatically adjusted to avoid voltage and temperature variation, and the average delay of an element will be known to be T_{clk}/N. These Voltage-Controlled Ring Oscillators (VCROs) then have their delay line state captured when a start or stop signal enters the system, providing the delay line functionality while remaining stable over variations in voltage and temperature, something not possible on FPGAs. Also, as the clock is being sent through the delay line rather than the start or stop signal, the same delay line can be used for multiple channels if the delay of the paths to each channel are balanced. This is not possible in an FPGA as the routing is fixed and generally unbalanced, however, in an ASIC, it is possible to balance the delay of the paths to each channel (an H tree, generally used in clock distribution, is a good example of a suitable structure) and hence create multiple channels with very little extra logic utilisation (only 1 more register required for each channel).

$$N = \frac{T}{T_{min}} \tag{3.3}$$

$$\begin{aligned} A_{DL\ TDC} &= A_{DL} + A_{PE} \\ &= N * (A_{min} + A_{arb}) + N * A_{3:2} \end{aligned} \tag{3.4}$$

$$\begin{aligned} A_{HDL\ TDC} &= A_{DL} * \left(\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{N}\right) \\ &= N * (A_{min} + A_{arb} * \sum_{i=1}^{log_2(N)} \left(\frac{1}{2^i}\right) \\ &\approx N * (A_{min} + A_{arb}) \end{aligned} \tag{3.5}$$

Due to their mediocre performance (as will be seen in comparison to other methods in the following sections), it is generally rare to see examples of literature with simple delay lines on

ASIC technologies, with most ASIC designs tending to opt for the slightly more complex, but better performing, VCROs when aiming for multiple channels or small footprint, and other higher resolution methods when aiming for performance. On the other hand, FPGA architectures often have elements ideal for implementing a delay line but very little available to implement other methods, especially methods that require balanced paths or analog control, such as the VCRO.

For example, [52–56] all present TDCs implemented on FPGAs, and all use the basic form of delay line to implement their fine time measurement. As a result, they are each limited by the smallest logic element available in their architecture, with [52] exhibiting the highest resolution at 17ps due to its use of dedicated carry logic in a Xilinx Virtex-5 FPGA. [55] claims to have a higher resolution of 6ps, but ultimately confuses resolution with precision, in fact resulting in an implementation with a 6 ps Single-Shot Precision (SSP) and resolution of $\frac{25ps}{\sqrt{2}} = 17.67ps$ as per the standard error of the mean (two samples caused by the 'minimal' wave union). All of these papers try to make use of specialised elements to obtain the smallest possible delay, but as these elements will vary by architecture, the end results vary significantly, with [54] obtaining the lowest resolution of 400ps on an Altera ACEX 1K.

On the other hand, [57–62] all opted for VCROs, due to their ASIC implementation allowing for the more advanced architecture and analog control. [60] could have made great use of this by distributing the result of the VCRO to four channels, with each channel consisting of only a register to store the state, but instead creates a VCRO for each channel. [59], on the other hand, uses a combination of the two methods, with a global counter and VCRO used for coarse and medium timing, while a delay line was used on each channel for fine timing. [63] uses a delay line despite being an ASIC implementation, however, as the focus of the paper is not on TDCs but rather on frequency synthesis for Bluetooth, this decision was probably made to save area and development time rather than to improve the performance of the TDC, which was shown to be suitable for the application even without temperature and voltage invariance.

3.3.3 Vernier Delay Line

To obtain higher accuracy than can be achieved with a standard delay line, the Vernier delay line was proposed. The Vernier delay operates using a 'fast' and 'slow' delay line, where the fast delay line is the same speed as a standard delay line, and the slow delay line is slightly slower. The start signal is sent through the slow delay line which feeds to the D inputs of flip-flops, and the stop signal sent through the fast delay line, which controls the clock inputs. The stop signal 'catches up' with the start signal at a rate equal to the difference between the two logic element delays. Flip-flops where the start signal arrived first will be '1', whereas elements where the stop signal arrived first will be '0', thereby generating a priority code.

As can be seen, this scheme does not rely on fast elements, but rather elements with a small but reliable difference between them. This allows a relatively slow technology to obtain relatively high accuracy, at the cost of double the logic elements, half of the logic elements no longer being minimum-sized, and a reduced range for the same number of elements. The area utilisation is expressed in (3.6), with T_1 and T_2 being the delays and A_1 and A_2 being the areas of the two types of delay elements. $T_1 < T_2$.

$$\begin{aligned} N &= \frac{T}{T_2 - T_1} \\ A_{VDL} &= N * (A_1 + A_2 + A_{arb}) \end{aligned} \tag{3.6}$$

$$\begin{aligned} L &= \frac{N * (T_2 - T_1)}{T_{cnt}} \\ A_{VR} &= \frac{A_{VDL}}{L} = \frac{T_{cnt} * (A_1 + A_2 + A_{arb})}{T_2 - T_1} \end{aligned} \tag{3.7}$$

The dead time of the Vernier delay line is dependent on the delay of the slow elements and the number of elements. Specifically, it is $N \times T_2$. As the range of the VDL is $N \times (T_2 - T_1)$, the dead time exceeds the range by a factor of $\frac{T_2}{T_2 - T_1}$. The linearity of the Vernier TDC is dependent

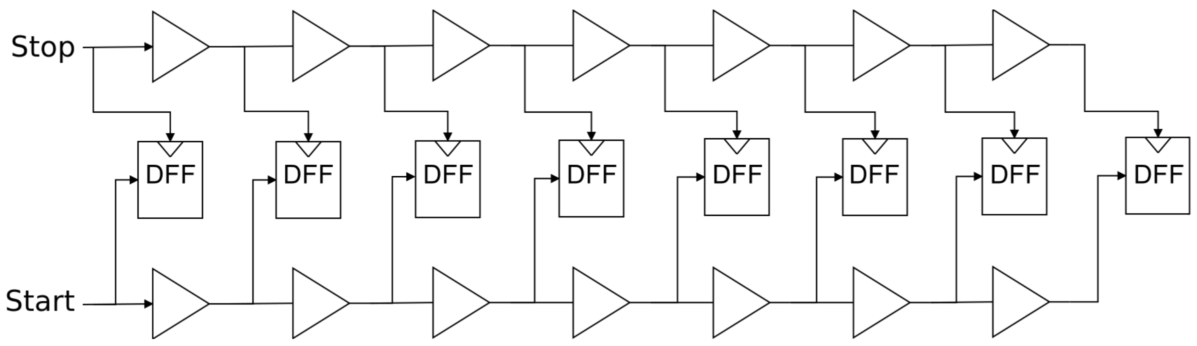


Figure 3.3: A Vernier delay line

on the matching of adjacent delay elements and the routing, which can generally be controlled quite well, particularly with non-minimum-width components.

A closely related concept to this is the Vernier ring. The Vernier ring involves a Vernier delay line where the output is the inversion of the input looped back on itself, and the D flip-flops replaced with counters. The counters count the number of times the start signal passes them before the stop signal, and hence the position in an equivalent conventional Vernier delay line can be calculated. Using this method, the logic utilisation is determined by the speed at which the counters can operate and the time difference between a fast and slow element. The time difference multiplied by the number of elements in the ring must meet the timing constraints (max frequency) of the counters. The area, as expressed in (3.7), is divided by the loop factor L, which is defined based on the maximum clock period of the counters, T_{cnt}.

Looped TDC architectures such as the Vernier Ring tend to have improved linearity as the loop length can be easily calibrated, bringing the calibrated INL to zero every $\frac{N}{L}$ elements. The dead time is also improved with the Verier ring as the signal only needs to propagate through a sufficient number of loops to convert the input, plus one more before the propagation can be halted, compared to the full range in the case of VDLs.

The vast majority ([64–75]) of Vernier implementations occur on ASIC platforms, as the generation of similar delays with a small, reliable difference is much easier in custom design due to the ability to individually size transistors. However, some FPGA architectures have been exploited to employ the Vernier technique, either making use of two differently-clocked oscillators ([76]) or differences in routing delay ([77]) to provide the time difference. While the ASIC technologies have been seen to provide much higher resolutions (880fs in [68]) than delay lines on equivalent process nodes, the FPGA implementations have fallen out of favour as the routing delay difference between the two paths is often much higher than an individual element's delay in modern FPGAs, resulting in standard delay lines emerging the superior implementation (the highest achieved resolution was 27ps in [76] as opposed to the 17ps seen in [52]). Also worthy of note is [78], which uses the vernier method on an FPGA to create a Digital-to-Time Converter (DTC), the inverse of a TDC, using the PLLs on Altera and Xilinx FPGAs, achieving a 1.58ps resolution on Altera Stratix III FPGAs. If this could be inverted to create a TDC, then this methodology would make a significant leap forward in FPGA TDCs, however, the PLLs cannot be easily started and stopped, which would be required to synchronise them to the input signals. It is noted that [76] is worthy of a revisit as FPGA technology has significantly advanced since this paper, and hence a higher resolution may be able to be achieved.

As for specific implementation details worthy of note, [64, 68] make use of PLL components such as PDs and CPs to ensure the stability of the fast and slow delay lines with respect to temperature and voltage variations, as well as to adjust them to obtain the finest-possible delay difference between them. However, this requires two clocks with a very small frequency difference, which is not easy to obtain. [65, 66, 75] all make use of multi-dimensional schemes to reduce the conversion time and area, with [65, 66] triggering a large number of vernier oscillators using the outputs of a standard delay line, while [75] performs comparisons between multiple miss-aligned elements in the vernier line to quickly achieve long delays without long vernier lines or dead times. [70] is probably the most interesting as it shows a method of using a single looped vernier delay line as the second level of a system, triggered by the residue (in comparison to the other methods mentioned, which connect an individual vernier oscillator or delay line to each tap of the higher level).

3.3.4 Pulse-shrinking TDC

Pulse shrinking TDCs operate by converting the start and stop signals into a single pulse-width encoded signal, which is then passed through pulse-shrinking elements until it becomes too small to be measured. The number of elements that it appears at, then, is proportional to the width of the pulse and hence the time difference between the start and stop signals. As this method does not rely on delay elements, it can achieve time resolutions smaller than that of the smallest delay element. However, the time taken to convert the pulse is often quite long as the size mismatches between the elements (required for the pulse shrinking) result in relatively long delays between each shrinking element. The area utilisation is expressed in (3.8), where A_{PSE} is the area of a pulse-shrinking element, comprised of a minimum-sized inverter (A_{inv}) and an unbalanced inverter (A_{ub}). The number of elements is inversely proportional to the shrinking time T_{shr}, which is the difference between the widths of the signal before and after encountering the pulse-shrinking element.

Pulse-shrinking TDCs have a long dead time similar to VDLs as the delay of a pulse-shrinking element is much greater than the quantity the pulse is shrunk by. This results in high accuracy but long dead times, equal to $N \times T_s$, where T_s is the delay of the pulse-shrinking element. Linearity is dependent on the precision of T_{shr}, the difference between the pulse width before and after the element. This can be made more accurate by using non-minimum-width components to reduce variation.

There is also a slight variation on this design where the pulse is passed through the same pulse-shrinking element multiple times over, and a counter is used to determine the number of times the pulse passes before becoming immeasurable. This alteration significantly decreases area utilisation, especially in the case of TDCs with a large number of bits (high range), but will often require a reduction in the count speed as the counter will not operate at the same rate as the pulse shrinker so extra delay elements are added. To solve this, some designs use a fixed number of pulse shrinkers in a loop (similar to a vernier ring), each connected to a counter (or just the final one connected to a counter), to achieve the maximum count rate while still retaining the excellent area scaling exhibited by the simple looped version.

$$\begin{gathered} N = \frac{T}{T_{shr}} = \frac{T}{W_a - W_b} \\ A_{PS} = N * A_{PSE} + A_{PE} = N * (A_{inv} + A_{ub}) + N * A_{3:2} \end{gathered} \tag{3.8}$$

In current literature, [79] is one of the few examples of a non-cyclic system, although it does make an attempt at reducing the area requirements using row and column decoders to reduce the detection logic. However, most systems make use of a cyclic system, with [80–82] using a

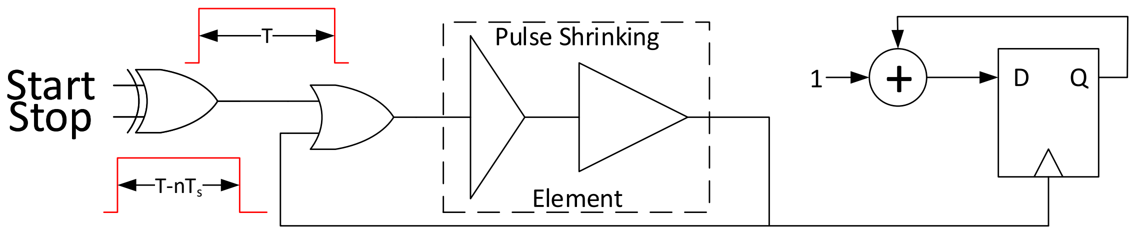


Figure 3.4: A pulse shrinking TDC

single pulse-shrinking element per stage, which obtains the highest linearity (only one element) but results in lower count rates due to the addition of a normal delay line to compensate for the maximum rate of the counter, whereas [83–86] use multiple pulse shrinking elements to obtain higher count rates at the expense of some linearity. The highest resolution was 6ps, as reported by [84], but it can be seen that this fails to exceed the resolutions achieved by other sub-bin methods such as the VDL. As a result of the aforementioned drawbacks, this method is rarely used.

3.3.5 Local Passive Interpolation

The Local Passive Interpolation (LPI) TDC replaces the delay elements with differential delay elements, with a feedback path consisting of equal-valued resistors leading back to the same-polarity input. The resistors act as a voltage divider between the input and output, and as the output will be retarded with respect to the input, the nodes between the resistors will cross the threshold voltage at different times, therefore decimating the time below that of a single delay element. The output of this scheme is a differential signal which could be captured by a sense amplifier flip-flop or a standard S-R latch, depending on the metastability requirements of the system.

In the ideal case this scheme requires 2^N resistors to provide a N-bit resolution increase, and in the absence of noise, this would be the case, as the differential mode of operation compensates for component mismatching due to local process variations. However, due to the thermal noise, power supply noise and induced noise from the high frequency switching of digital components, this performance is severely degraded. Also, the logic requirements of this scheme require many times the area of other schemes due to the cost of creating a differential delay line, as well as ohmic resistors. The area utilisation is expressed in (3.9), and is composed of some new primitives, including the area of a differential delay element A_{dmin}, the area of a differential arbiter, A_{darb}, and the area of an ohmic resistor, A_{res}. N here represents the number of differential delay elements, whereas N_{res} represents the number of resistors per differential delay element.

The DNL of the LPI TDC is mainly based on the matching between the analog components. When implemented with resistors, which require a large area, the mismatch is generally low and so the DNL is small. When implemented with diodes, the mismatch is much higher, resulting in more variation and higher DNL. The INL is dependent on mismatch between the differential amplifiers, which is generally low as the components need to be non-minimum-size to drive the analog sections of the LPI TDC.

$$\begin{aligned}
N &= \frac{T}{T_d min} \\
T_{min} &= \frac{T_d min}{N_{res}} \\
A_{LPI} &= N * A_{LPI\ Elem} + A_{PE} \\
&= N * (T_{dmin} + N_{res} * (2 * A_{res} + A_{darb})) \\
&\quad + N * N_{res} * A_{3:2}
\end{aligned} \tag{3.9}$$

LPI TDCs are a relatively recent development in the field of TDCs, and as such, most papers are authored by Henzler et. al. [87–89]. These systems show a LPI TDC with 4x interpolation achieving a 4.7ps resolution. Kim et. al. [90] also implemented Henzler's design in 180nm (compared with 90nm) and achieved 7.84ps resolution, with the multiplexers in Henzler's design replaced with tri-state inverters to reduce the time delay between coarse stages. This increased the resolution from 14.3ps, which was achieved using Henzler's original design.

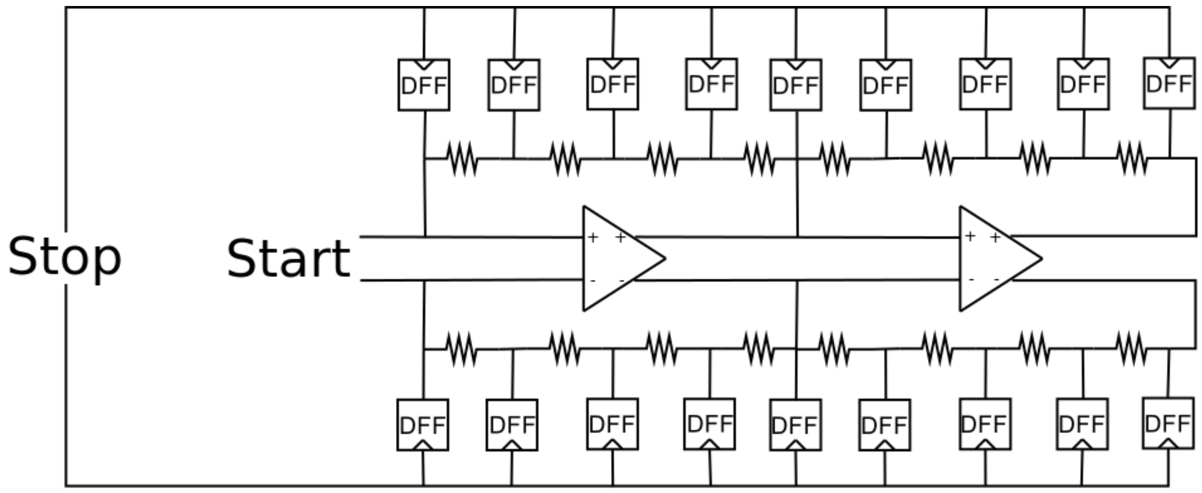


Figure 3.5: An LPI TDC

3.3.6 Stochastic TDC

When high accuracy is required without regard for range, a stochastic TDC (STDC) is often employed, such as in PLLs. These devices already have a good approximation of the time at which a signal will occur, and only require a small range to identify how far out they are. For this purpose, the stochastic TDC is a good option. The stochastic TDC utilises the metastable nature of flip-flops when their inputs change near to their clock edge. The further a signal is retarded relative to the clock edge, the less likely it is to appear at the output. These statistics can then be used by employing many flip-flops to estimate the chance of an individual flip-flop showing some value, and then using this probability to estimate the time difference between the clock edge and the signal.

While the stochastic TDC can provide extremely accurate timing, it does so at a cost. A single stochastic TDC is incapable of providing timing estimates outside of the range in which it exhibits metastability, and so many individual stochastic TDCs offset by delays would have to be used to be used to provide a larger range. Also, the number of logic elements rises exponentially with the number of bits of accuracy. Finally, the jitter in the timing measurements is higher than most other TDCs, being the entire width of the metastability region at most, meaning it may not always be accurate. As with any unweighted statistical process, the accuracy increases with the square root of the number of samples (standard error of the mean) and the range is bounded by the sample minimum and sample maximum. Depending on the process and type of arbiter used, this will result in a maximum range in the order of tens of picoseconds, and a theoretically infinite resolution (although this is somewhat limited by the increasing area utilisation), as expressed in (3.10

The dead time of a stochastic TDC is dependent on the settling time of the flip-flops. This is generally on the order of tens or hundreds of picoseconds, depending on process node, resulting in TDCs that can operate at very high speeds. Therefore, decoding the output (population counting and calibration) tends to be the bottleneck. The linearity of a stochastic TDC is very high in the center of its available range (due to a large quantity of flip-flops dividing the range finely), but reduces near the edge of its range (due to relatively fewer flip-flops having extreme blackout times and thus a more coarsely divided range).

$$\begin{aligned} T_{min} &= \frac{T_{smax} - T_{smin}}{N} \\ A_{STDC} &= N^2 * A_{arb} \end{aligned} \tag{3.10}$$

In the current literature, there are two main designs of stochastic TDC. The first is the style used in All-Digital Phase Locked Loops (ADPLLs), where a stochastic TDC is used in isolation for frequency/phase error tracking [91–98]. As the aim of these systems is to make two periodic signals align, the limited range seen in these TDCs is not an issue, as over successive measurements the stochastic TDC will always output the same value (either maximum or minimum) until the time difference of two signals is within the range of the TDC, at which point the TDC can then operate in its normal mode, and the two signals will not drift out of phase far enough to surpass the range of the TDC. In these systems, we see resolutions as high as 20fs in [95], with ranges in the region of $\pm 15ps$ in the aforementioned to $\pm 35ps$ in [92].

The other design seen in current literature is the use of a stochastic TDC as the fine stage of a multi-level TDC [99]. As the range of black-out times or metastability period is often in the range of 1-10 inverter delays, it is often possible to connect a stochastic TDC to each stage of a delay line to interpolate between the delay line's time bins. Systems using this method generally exhibit the high range of a delay line, especially when paired with a counter, as well as a much greater resolution than can be achieved with a delay line. However, they generally do not have the same resolution as their purely stochastic counterparts, as to increase the resolution more arbiters must be added to each stage of the delay line, making a high resolution prohibitively large on-die.

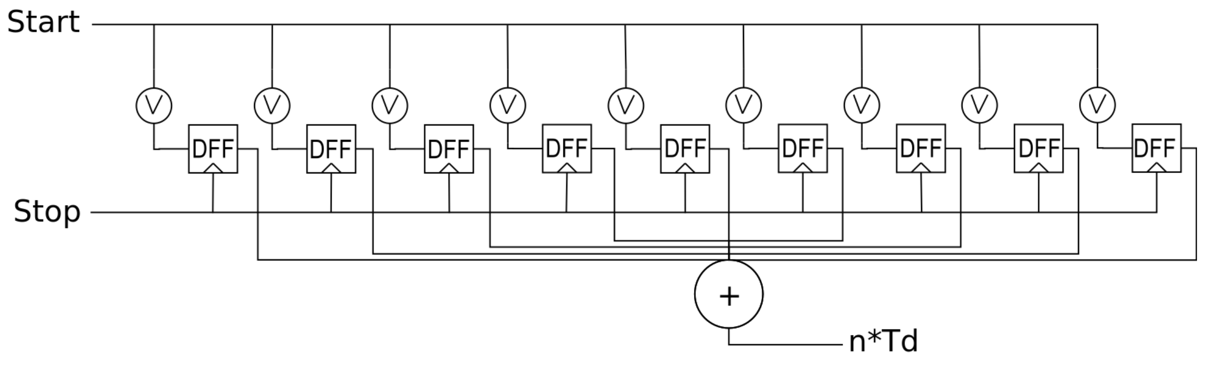


Figure 3.6: A stochastic TDC

It would be worth looking into a residue-selection system (which occasionally appears in a wide range of designs) to decouple the stochastic TDC from individual time bins so that one highly accurate STDC could service the entire delay line.

As a side note, a few designs have been described, such as [100], which claim to be "quasi-stochastic". These designs, however, are not operating on the traditional stochastic principles, but rather using bit counting (rather than a priority encoder) to obtain a more reliable edge in the delay line under metastability conditions. With very high resolutions, some arbiter designs (particularly D Flip-Flops) will have such long metastability times that it covers tens of bins, and due to the Gaussian nature of the blackout time, the highest bin number can vary wildly over many measurements. As a result, the often-used priority encoder is no longer suitable in these circumstances. Instead, these systems resort to bit counting (also known as population counting or ones-encoding), as this relies on the number of bits rather than the position of the highest bit. This is roughly equivalent to sorting the bits to be in-order then applying the priority encoder. As the number of bits in the metastability region is more reliable than the position of the highest bit, this increases the precision of the TDC.

3.3.7 Successive Approximation TDC

Successive Approximation TDCs (SA TDCs), named due to their derivation from successive approximation Analog-to-Digital Converters (ADCs), operate by delaying the start and stop signals using a variable delay line and comparing which signal exits first, then routing the signal to propagate through a shorter delay line.

Simple linear SA TDCs are relatively straightforward to implement, as each stage of the approximation is a fixed delay line with the propagation delay being half that of the previous stage. However, they subsequently suffer in terms of large area utilisation and poor matching over the course of the TDC due to local process variations. The delay can be stabilised by applying bias voltages derived from a delay-locked loop (DLL) that divides the previous stage's time by two, but this adds extra complexity, and therefore cost, to the circuit.

The dead time of a successive approximation TDC is slightly longer than its maximum range. At each stage, either the start or stop signal is delayed by a power-of-two fraction of the maximum range, plus some extra delay due to routing and comparison. The sum of all these fractions of the maximum range adds to give the maximum range itself, while the N copies of the extra delay add on to extend the dead time slightly further than the maximum range. The linearity is dependent on the accuracy of the delays. As large delays are easy to make accurate, the INL tends to be low, but inaccuracies in the generation of small delays result in a fairly high DNL.

Therefore, Cyclic SA TDCs (CSA TDCs) have been proposed where the signal is routed repeatedly through the same delay element - a Digital-to-Time Converter (DTC), which has its delay repeatedly halved as the start and stop signals converge. As the same delay elements are being used each time, the effect of local process variation is much smaller, and for the same range and precision, only half the delay elements are needed, as per Equations 3.11, 3.12 and 3.13. This means that the full-scale range of the CSA TDC (without the addition of another level of TDC) is twice the length of the largest generatable delay ($\sum_{i=0}^{\infty}\left(\frac{1}{2^i}\right) = 2$).

$$N = \frac{T}{T_{min}} \tag{3.11}$$

$$\begin{aligned} A_{SA} &= A_{min}\left(\frac{N}{2} + \frac{N}{4} + \ldots + \frac{N}{N} + \frac{N}{N}\right) \\ &\quad + B(2A_{mux} + A_{arb}) \\ &= A_{min}\left(\frac{N}{N = 2^B} + \sum_{i=1}^{B}\left(\frac{N}{2^i}\right)\right) \\ &\quad + B(2A_{mux} + A_{arb}) \\ &= N * A_{min} + B(2A_{mux} + A_{arb}) \end{aligned} \tag{3.12}$$

$$\begin{aligned} A_{CSA} &= A_{DTC} + A_{arb} + A_{sel} \\ &= \left(\frac{N * A_{min}}{2} + (B-1)A_{mux}\right) \\ &\quad + A_{arb} + (2A_{mux} + 2A_{OR}) \end{aligned} \tag{3.13}$$

Specifically, Equation 3.11 describes the number of elements N in the DTC as a function of the full-scale range T and the minimum time resolution T_{min}. For simplicity, only a homogeneous DTC (only one type of delay element) is considered in these equations. Equation 3.12 describes the area of a linear SA-TDC in terms of the area of a minimum-sized delay element A_{min}, N as defined in Equation 3.11, the number of bits of resolution B, the area of a multiplexer A_{mux} and the area of an arbitrator (flip-flop, SR latch, current-sense amplifier etc.) A_{arb}. It can be seen that the result includes roughly the same number of elements as a simple delay line TDC, so the linear SA-TDC only benefits when non-homogeneous elements are used (e.g., by adjusting load through capacitors).

Equation 3.13 shows the area of a CSA-TDC in terms of the area of a DTC A_{DTC}, the area of an arbitrator A_{arb} and the area of a selector A_{sel}. The DTC only needs to cover half the full-scale range T of the TDC so only needs half the delay elements $\frac{N}{2}$, and then must use $B-1$ multiplexers to enable or disable sections of the DTC. The selector can be composed of two multiplexers and two OR gates.

Fig. 3.7 shows the operation of a CSA TDC. The “Arbiter and Selector” block in Fig. 3.7() chooses to forward either A’ and B or A’ and B’ to its outputs depending on whether A’ or B arrives first. If A’ arrives before B, this implies that the delay between A and B is longer than the period of DTC$_a$ (Digital to Time Converter ‘a’), therefore it forwards A’ and B to reduce this delay and then halves the delay for the next cycle. If B arrives before A’, then the delay between A and B does not exceed DTC$_a$, therefore it forwards A’ and B’, which have the same delay difference as A and B, so that the delay is maintained for the next round where the delay is once again halved. If A’ arrives before B, a ‘1’ is output on T_b when B arrives, otherwise a ‘0’ is output on T_b when B arrives. The A signal can be used to allow double-ended operation (stop before start) or to clear the state of the arbiter for the next round. The TDC here outputs the code “1011001”, which is a fractional number with an MSB of 1, resulting in $1+0.25+0.125+0.015625 = 1.390625$, which is the closest number below 1.4.

The hierarchical TDC [51] is observably similar to the SA TDC, and in fact can be considered a less efficient form of the SA TDC, as the next stage is triggered twice, whereas the SA TDC only triggers the next stage once. Also, as the hierarchical TDC does not choose between the delayed and non-delayed versions of the signals, some conditional bit flipping is needed on the outputs which is not needed in the SA TDC.

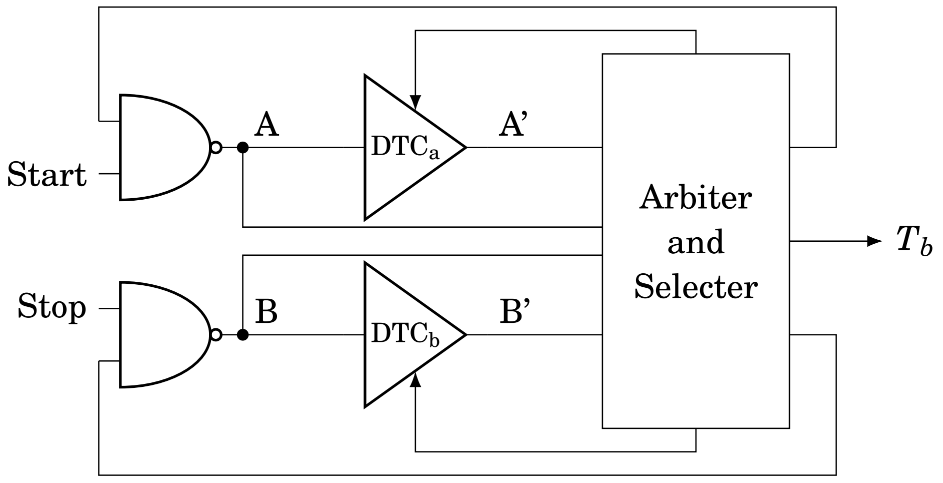
The asynchronous pipelined TDC demonstrated in [101] is also a form of successive approximation TDC. However, since the aim is to quantify pulse time (rising edge to falling edge), the residue is formed by finding the dead time or overlap between the signal and the delayed version of the same signal. If the signal overlaps it's delayed version, then the period of the signal is longer than the delay, otherwise it is shorter. The residue, which is the quantity by which the signal overlaps or misses the delayed version, is then quantised by an exponentially shorter delay. The authors of [101] achieved a resolution of 200 ps due to the dead zone of the residue generator, which was shown to be 189.7 ps.

Similarly to Local Passive Interpolation (LPI) TDCs (see Henzler [40]), SA TDCs are relatively new in the current literature, with more than half of the papers published by Mäntyniemi et al. [102–104]. These papers show a TDC operating at up to 610 fs resolution with a 5 ns range (or 1.21 ps resolution with a 328 us range) using a switched capacitor array (and a Voltage-Controlled Oscillator (VCO) for the 328 us range) for the Digital-to-Time Converter (DTC), which is explained in more detail in [105]. Building on this, Chung et al. [106] propose unrolling the SA loop to increase sample rates. Using 65 nm CMOS (compared with 350 nm in [104]), 80MS/s was achieved, compared to 5MS/s in [104]. However, due to the inferior switched-capacitor implementation, [106] only managed a 9.77 ps resolution. [107] presents a technique one might call a linear SA TDC, in that it linearly increases the delay until the two signals align. However, as it has no signal recovery, duplication, or residue, it requires multiple samples of the input signal before it can detect the correct time difference, which is not tolerable in many applications. Also, this system exhibits a 474 ps resolution, which is well behind even delay line implementations available in 2016 (17 ps in [107]), when this paper was written. [48, 108] also present a linear (unrolled) implementation of the SA-TDC, achieving 25 ps and 12.5 ps respectively on 180 nm.

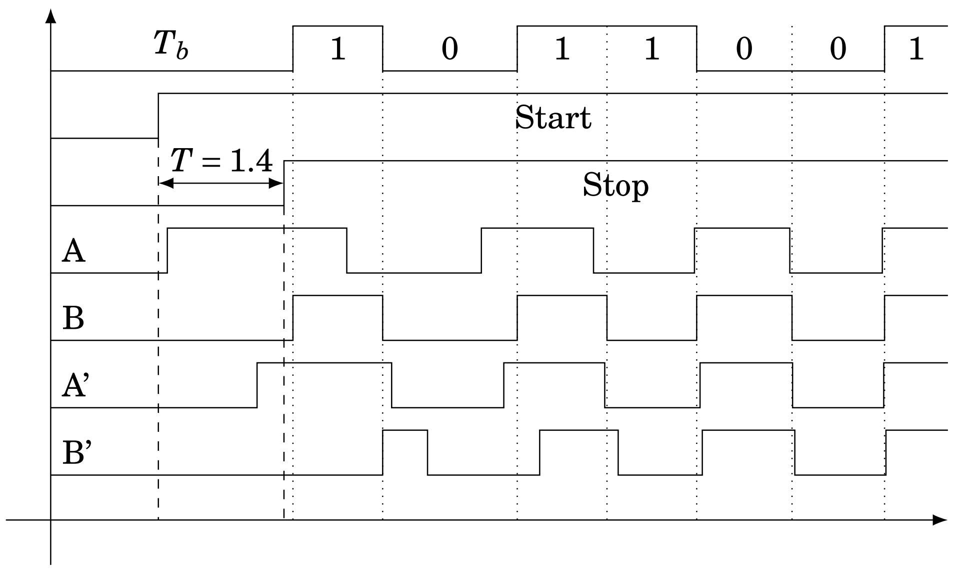
3.3.8 Algorithmic TDC

Algorithmic TDCs, first proposed by Keranen and Kostamovaara in [5] and used again in [109], are functionally similar to CSA TDCs (see Section 3.3.7). However, at each stage of the successive approximation, instead of reducing the delay exponentially, it amplifies the residue exponentially and re-quantises the amplified residue with the same delays.

Keranen's paper uses a scheme similar to a dual-slope TAC, but instead of increasing and decreasing the amplitude, it increases the phase of a ring oscillator at two different speeds. The

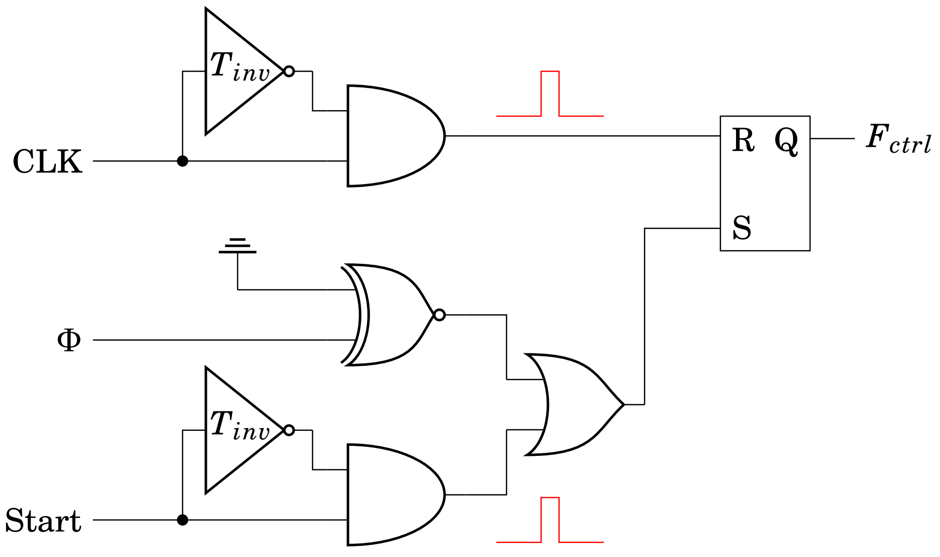


(a)

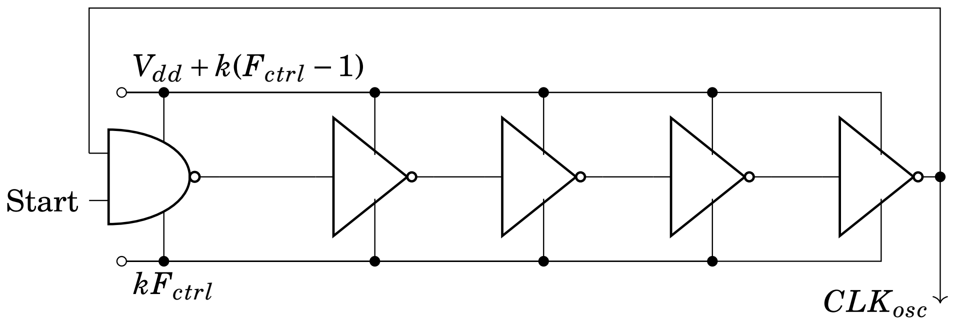


(b)

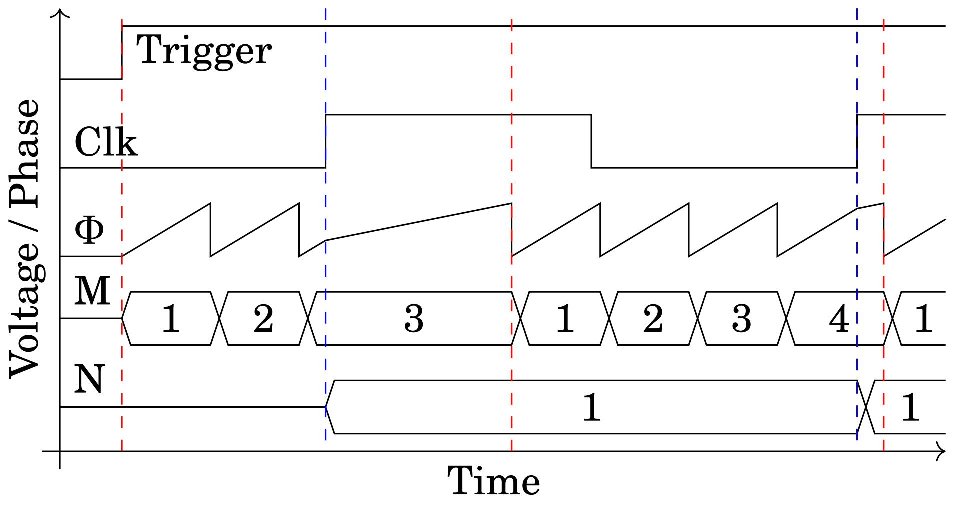
Figure 3.7: (a) Block diagram and (b) waveform of a CSA TDC.



(a)



(b)



(c)

Figure 3.8: Algorithmic TDC. (a) Frequency Control. (b) Ring Oscillator. (c) Wave trace for a sample conversion based on [5] (residue = 38% of a clock period). Outputs (M, N, M, N, . . .) are 3, 1, 4, 1, ...

number of 'fast' oscillations between the start and stop (system clock edge) are counted, and then the oscillator is switched to its 'slow' mode. The time taken for the oscillator to reach a full oscillation (phase is zero) will be dependent on the quantisation residue of the counter process, and will be amplified by a ratio of $\frac{F_{fast}}{F_{slow}}$, where F_{fast} is the frequency in 'fast' mode, and F_{slow} is the frequency in 'slow' mode.

Fig. 3.8(b) shows the wave trace for an algorithmic TDC. The ring oscillator starts oscillating at a high frequency (e.g., $F_{fast} = 6 \times F_{clk}$) when the trigger signal transitions from 0-1 (first dashed line, red) and oscillates until the next clock edge (second dashed line, blue). At this point the value of the M counter (which counts ring oscillator periods) is sampled to produce the first residue. Then, the ring oscillator is switched to a low frequency (e.g., $F_{slow} = 2 \times F_{clk}$) and the N counter (which counts whole clock periods) is started. This runs until the ring oscillator wraps around to $\Phi = 0$ at which point the N counter is sampled to produce the next residue and the counter is set back to the high frequency. At each stage, the algorithmic TDC is amplifying and quantising the residue from the previous stage through the change of the ring oscillator's frequency. If F_{slow} were to be slower than F_{clk}, it would be possible to reach $N > 1$ and increase the amplification at each stage, at the expense of longer conversion times.

The dead time of the algorithmic TDC is very high due to requiring several clock cycles to perform conversion. On stage of the conversion is done per each slow clock cycle, with multiple stages requried for a relatively high resolution. However, the linearity of the algorithmic TDC is relatively high, as it is dependent on oscillators, which can be accurate and precise to sub-picosecond levels if required.

In [109], a second oscillator is started in fast mode while the first is in slow mode to quantise the amplified residue, and then the first oscillator is used to quantify the second amplified residue (etc.). On the other hand, in [5], the system clock is used to quantify the amplified residue, and then a further residue is generated from the time between the oscillator reaching zero phase and the next system clock edge, which is quantified using the same method as the original pulse (using the oscillator in fast mode to quantify, then switching to slow mode to amplify).

3.3.9 Gated Ring Oscillator

Gated Ring Oscillator (GRO) TDCs are similar to the multi-phase clock TDCs mentioned in 3.3.1. However, they use a gated ring oscillator instead of multiple clock phases, which has the property of maintaining it's phase between measurements. As a result, a first-order noise-shaping effect is applied to the quantisation noise, resulting a lower average error. This results in better performance when multiple measurements are made with the TDC, but does not increase the accuracy of a single measurement, the so-called single-shot precision (SSP). If Nth order noise-shaping is required to obtain the required accuracy, then it has been suggested to use N relaxation oscillators with a digital filter to neutralise the error. The areas expressed in (3.14), along with an example showing how the accumulated errors eventually wrap around to 0 when they reach a whole clock phase.

$$
\begin{aligned}
A_{GRO} &= A_{MPC} \\
R_0 &= 0 \\
T_1 &= N_{11} * T_{clk} + N_{12} * T_{ph} + R_1 \\
\frac{\bar{T_1}}{T_{ph}} &= (N_{11} * T_{clk} + (N_{12}+1) * T_{ph} \\
&\quad - R_0) DIV\ T_{ph} \\
T_1 - \bar{T_1} &= (R_1 + R_0) MOD\ T_{ph} \\
T_2 &= N_{21} * T_{clk} + N_{22} * T_{ph} + R_2 \\
\frac{\bar{T_2}}{T_{ph}} &= (N_{21} * T_{clk} + (N_{22}+1) * T_{ph} \\
&\quad - (R_1 + R_0)) DIV\ T_{ph} \\
T_2 - \bar{T_2} &= (R_2 + R_1 + R_0) MOD\ T_{ph}
\end{aligned}
\tag{3.14}
$$

The maintained phase is caused by the use of an enable signal, which is high when a start has occurred but the stop has yet to occur. This signal is used to connect the GRO to the voltage rails, and hence when the signal is low, the nodes in the GRO maintain their voltage via the parasitic capacitance at each node. When the enable signal becomes high again, the GRO can hence start from where it left off. However, as with the LPI TDC, the usage of analog voltages in the TDC results in the performance degrading with increased induced noise in the device. This is significantly worse inside the GRO TDC, as the voltage is being stored in capacitors with no method to replenish themselves, which effectively integrates the noise over the period that the enable signal is low. This, along with leakage current through the capacitors, results in a minimum required count rate to continue working which becomes worse in noisier environments. The minimum required count rate will often be satisfied in single-particle systems due to dark counts in the detectors. However, in high-energy and regular time-of-flight applications, this could pose a significant problem for the system.

The dead time of a gated ring oscillator is effectively zero, as the TDC is immediately ready to continue oscillating. Therefore, the dead-time will be controlled by the performance of the decoding logic. In return for this, the gated ring oscillator has a minimum input frequency required to keep the internal nodes charged and avoid loosing its state (at which point erroneous measurements may occur). The DNL of a gated ring oscillator is dependent on the local variation between delay elements, while the INL is generally low due to the looped nature of the device.

Due to the other benefits provided by GROs, most literature is not focused on the absolute resolution of the GRO, but rather the integrated noise or noise shaping properties [110–112]. However, some systems try to push the boundaries of resolution with GRO-based systems. [113–115] make use of a multipath ring oscillator instead of a standard ring oscillator to reduce the time between stages by connecting outputs from multiple stages before to the current stage. However, they do this at the expense of power consumption due to the increased time the inverters spend in short circuit mode. Despite this, these systems are attractive as they achieve higher

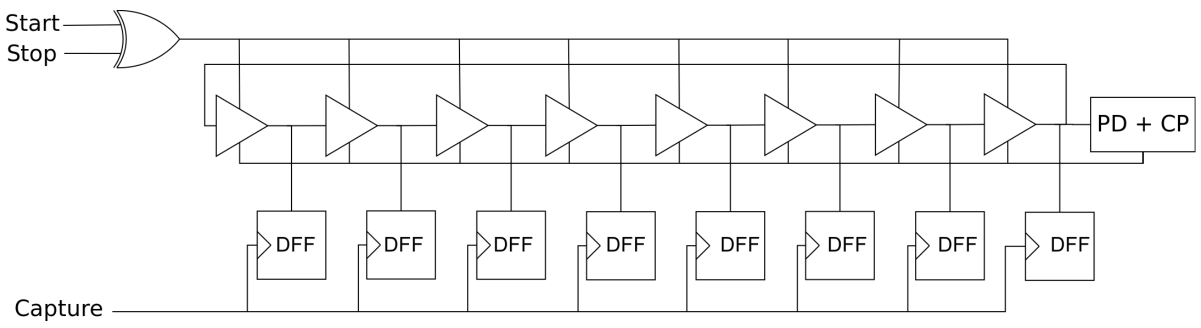


Figure 3.9: A gated ring oscillator.

resolutions than their simpler counterparts (6ps compared with the 50ps achieved in [110]), hence further improving the noise characteristics of the measurement.

[111] is rather unusual in that it uses a relaxation oscillator with a current source as opposed to a ring oscillator, and hence manages to obtain 6ps resolution without assistance from bypass schemes. [116, 117], on the other hand, opt to incorporate the vernier technique into their GROs, resulting in much higher resolutions than normally achievable with a GRO, as low as 5.8ps in [117].

3.3.10 Stochastic Phase-Interpolation TDC

The Stochastic Phase-Interpolation (SPI) TDC was first introduced by Kim et. al. [100] and then later analysed in Gammoh. et. al. [118]. In this scheme, multiple delayed versions of the clock signal are created via a delay line, then gated by the and between start and an inverted stop. These gated clock signals are then introduced to the clock inputs of D flip-flops with the D input tied to logical one. During the period where the start signal is high and the stop signal is low, clock edges arriving at taps in the delay line cause the corresponding D flip-flops to latch a logical one. Meanwhile, edges that arrive at a tap when start is low or stop is high are gated out, therefore leaving the D flip-flop at logical zero. The distribution between ones and zeros allows the TDC to determine the portion of the clock period for which start was high and stop was low, and therefore the fine time value. If the difference between start and stop was greater than a clock cycle, then all D flip-flops would output a logical one, therefore showing that the range of this TDC is no higher than a clock cycle.

This scheme has been described as “PVT-tolerant” as the quantisation does not rely on the delay of the individual delays of the buffers delaying the clock, but rather the distribution of ones and zeros. For the linearity of the TDC to be disturbed, it is not sufficient for a single area of the TDC to suffer from degraded performance. Instead, multiple areas of the TDC, spaced such that they match the period of the clock signal, must suffer from similarly degraded performance to adjust the overall distribution of the edges. For example, if all the elements of the delay line were to suffer from degraded performance due to a temperature increase, then the increase in element delay would result in each clock edge reaching fewer bins during the active time of the

TDC, but more clock edges would be present in the delay line due to the increased overall delay. The increase in clock edges would compensate for the reduction of bins reached by each clock edge, maintaining linearity and precision.

The dead time of the SPI TDC is dependent on the decoding and reset logic. Once the TDC has triggered, all the flip-flops must be reset to zero before the TDC can be triggered again. The logic must read out data before resetting the flip-flops. Therefore, the reset procedure often takes two clock cycles to complete. Linearity is high due to the stochastic nature of the TDC, relying on the distribution of ones and zeros rather than one particular delay.

The main drawback of this TDC is the required area. As multiple clock edges must be held within the delay line to perform quantisation, the length of the delay line must vastly exceed the clock period. For example, the original paper [100] uses 2^{14} delay elements to achieve a resolution of 10 bits, creating a redundancy of $2^{14}/2^{10} = 16$. Similarly, Gammoh's paper [118] uses 16x redundancy, but with a 10-bit quantisation and 6-bit output ($2^{10}/2^{6} = 16$).

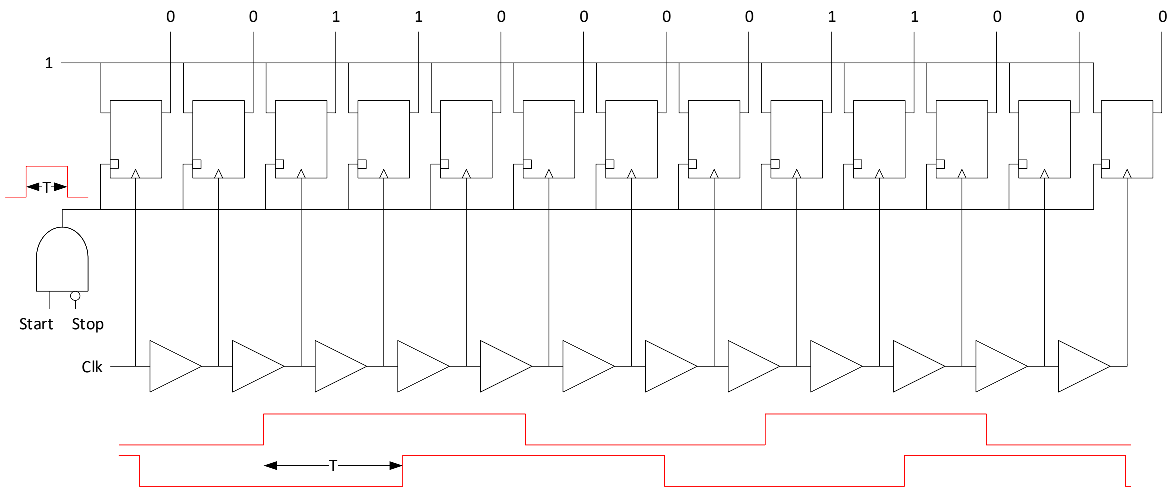


Figure 3.10: Architecture of a Stochastic Phase Interpolation TDC. The proportion of ones at the output is dependent on $\frac{T}{T_{clk}}$.

$$N = 2^B \tag{3.15}$$

$$M = 2^{B+k} \tag{3.16}$$

$$\begin{aligned} A_{SPI} &= A_{DL} + A_{PC} && (3.17)\\ &= M * (A_{min} + A_{arb}) + \frac{M}{2} * A_{2:2} + \frac{M}{4} * A_{4:3} && (3.18)\\ &\quad + \frac{M}{8} * A_{6:4} + \ldots + \frac{M}{M} * A_{(2*(B+k)):(B+k+1)} && (3.19)\\ &= M * (A_{min} + A_{arb}) + \sum_{i=1}^{B+k} \frac{M}{2^i} * A_{(2*i):(i+1)} && (3.20) \end{aligned}$$

3.3.11 Wave Union Launchers

In [1], Wu and Shi propose a method of improving precision past the gate delay: the wave union TDC. Rather than dispatching one edge per trigger and quantising this edge, the authors suggest dispatching multiple edges and quantising all of them by a method similar to the gated ring oscillator (GRO), but without the need for more than one input sample.

The authors suggest two methods for doing this. The first (type A) is to store a wavelet inside a delay line and release it on incidence of a trigger. When the stop signal occurs, the wavelet is held in place and quantised. Each edge in the wavelet is individually quantised and the edges are then combined to give a more accurate measurement of the original trigger position. This is referred to as an FSR (finite step response) wave union launcher.

Fig. 3.11 shows the design of an FSR wave union TDC. The first M bins are used to store the

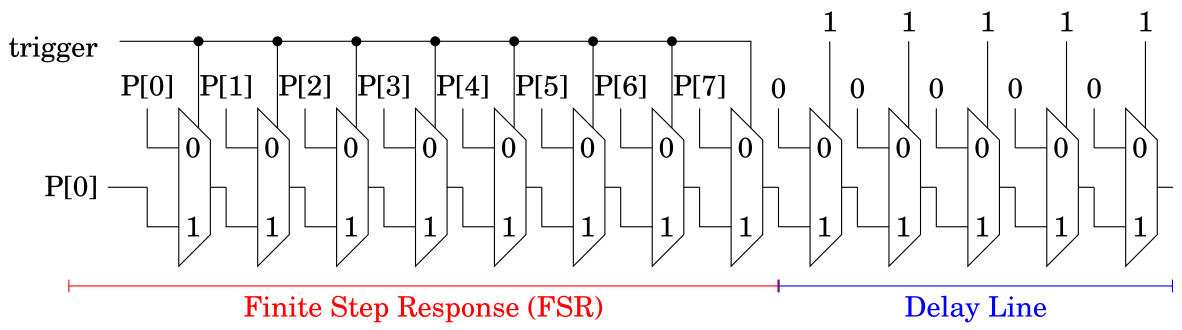


Figure 3.11: Initial section of a wave union launcher using multiplexers (the delay line continues further).

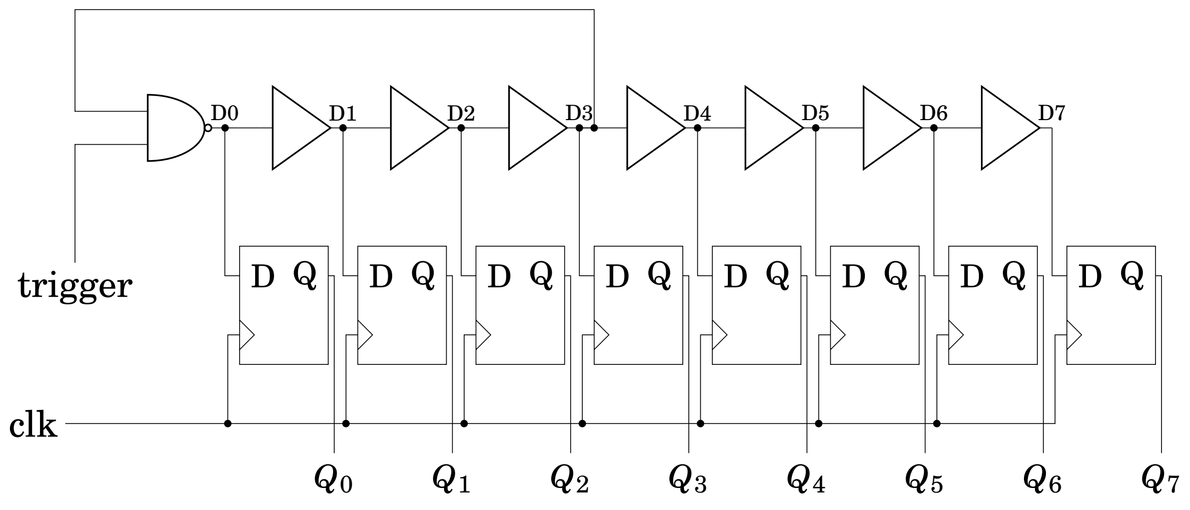
FSR pulse and for quantisation, with the remaining N bins being used solely for quantisation. In [1], M was 16 bins (with the distance between edges in the FSR being 13 bins), while N was 48 (i.e., the delay line was three times the length of the FSR storage).

The second method (type B) is to attach a startable ring oscillator to the front of the delay line. The trigger signal starts the ring oscillator, which then oscillates for several cycles before stopping. This is referred to as an ISR (infinite step response). The oscillations happen over multiple system clock (stop) cycles.

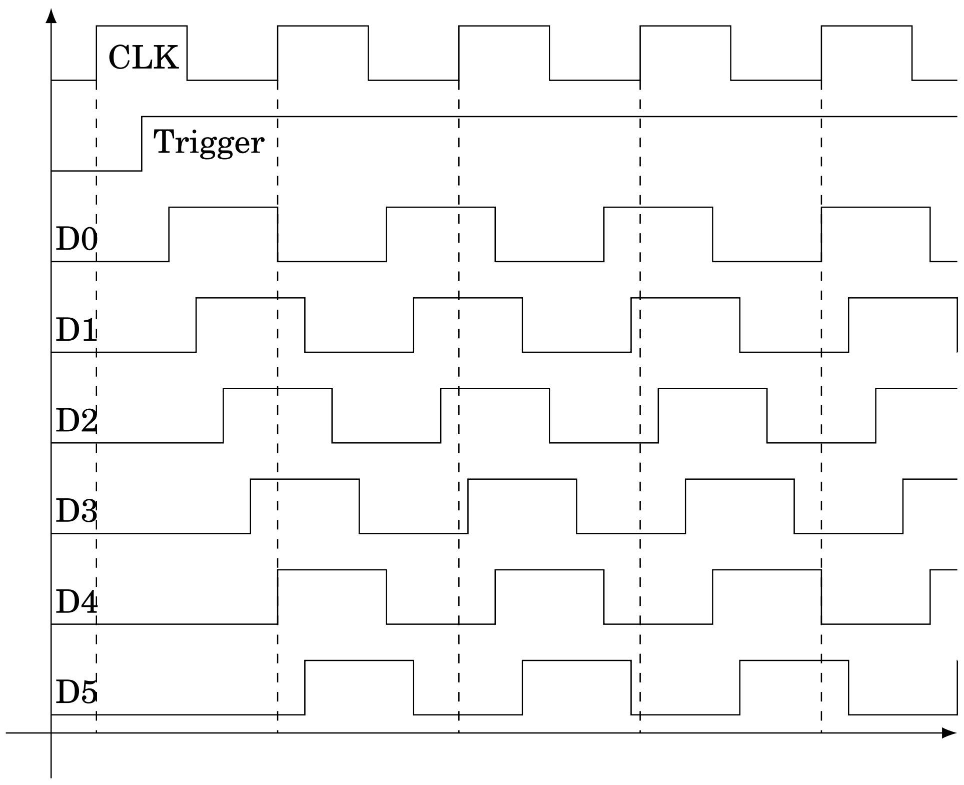
In the type B wave union TDC, shown in Figs. 3.12() and 3.12(a), the period of the oscillator must not be too similar to the period of the system clock so that the sampling process does not repeatedly hit the same large bin (as this would result in a large DNL). However, this means that there will be cases where ring oscillator edges will not be seen once. The authors of [1] identify three possible cases: U, V and W patterns, corresponding to jumps of zero, one and twwo ring oscillator periods. The jump type is determined from the output values of the priority encoder:

- For a value in the range $3N/4 \rightarrow N$ followed by a value in the range $N/4 \rightarrow N/2$, this implies both signals were the same ring oscillator edge (based on the operation of the priority encoder) and hence is a case of the U pattern.
- For a value in the range $N/4 \rightarrow N/2$ followed by a value in the range $3N/4 \rightarrow N$, this implies that a ring oscillator edge has been missed, and hence is a case of the W pattern. This will only happen if the ring oscillator is faster than the clock period (meaning an edge can pass between two samples).
- All other jumps are classified as V patterns, and are indicative of the standard operation of the TDC.

The first method can increase the accuracy quite significantly, from 165 ps per bin worst case and 60 ps per bin average case (in the original TDC), to 65 ps per bin worst case and 30 ps per bin best case. It does this without significantly increasing the dead time (2.5 ns to 5 ns) but the decoding complexity increases due to increasing the number of edges to be decoded per output (although this was performed on a computer in the original paper). The second method was measured through the RMS error of measuring a fixed time difference and resulted in an



(a)



(b)

Figure 3.12: Example of a type B (ISR) wave union launcher. (a) Gate-level implementation; the NAND gate and first three buffers act as the startable oscillator. (b) Waveform of a type B wave union launcher where $F_{CLK} > F_{OSC}$.

improvement from 40 ps to 10 ps for 16 measurements (in comparison to 25 ps RMS for the FSR method), albeit at an 18 times dead time increase (2.5 ns to 45 ns). This is summarised in

Table 3.2.

DNL is generally low in wave-union TDCs as smaller bins tend to sub-divide larger bins, increasing resolution and decreasing non-linearity. INL is generally quite high before calibration due to differences in bin density at different points in the delay line compounding, but is generally small after calibration due to the small bins and relatively similar bin sizes. Dead time is slightly higher than one clock cycle, as this is how long it takes for the edges to leave the delay line ready for the next trigger signal.

Subsequently, the authors of [55] made use of the wave union TDC and managed a 1.8 times improvement on the bins inside their Virtex 4 FPGA from 16 ps RMS to 9 ps RMS.

In [119], Hu et al. suggest a Stepped-Up Tree Encoder (SUTE) to efficiently encode the edges on a Virtex 4 FPGA in the presence of bubbles and the non-thermometer code presented by the type A wave union TDC. The encoder uses a pre-processing stage capable of removing single-bit bubbles (e.g., 0000 1011 1111) which encodes the position of the $0 \rightarrow 1$ edge in subgroups of four bits, plus a flag to determine if the transition occurs in that subgroup and a flag to determine if a transition happens on the border of subgroups.

The 4-wide grouping suppresses the single-bit bubbles and hence allows the resultant outputs to be sent to an array of standard priority encoders for encoding via some switching multiplexers which distribute the edges to the encoders. This ensures that multiple edges can be encoded in a single clock cycle, and the maximum number of edges is determined by the number of terms in the FSR (wavelet generator).

3.3.12 TAC-ADC

Other than the purely digital techniques explored so far, there are also some analog techniques that can be used to achieve the same thing, generally in the form of a time-to-analog converter

Method	Mean Bin	Worst Bin	RMS Error	Dead Time
Delay line	60 ps	165 ps	40 ps	2.5 ns
Type A (FSR)	30 ps	65 ps	25 ps	5 ns
Type B (ISR)	–	–	10 ps	45 ns

Table 3.2: Performance of the wave union TDCs in [1].

(TAC) paired with an analog-to-digital converter (ADC). This converts the time between the start and stop signals to an analog signal such as voltage (usually by charging a capacitor), then converts the analog signal to a digital one using an ADC.

Due to the extensive research done on ADCs, the accuracy is often very high. However, the cost of the discrete analog components needed to hold the analog signal between the stages is high, since the signal integrity decreases when miniaturised onto SOI technologies. Also, due to the time taken to raise and then sample the analog signal, the count rate of these devices is often much lower (dead time much higher) than all-digital methods. (3.21) expresses the resolution of the TAC-ADC system, where E_{NL} represents non-linearities in the time-to-voltage transfer characteristic of the capacitor and current source, such as due to the parasitic inductance of the system.

INL is generally dependent on the quality and limitations of the components used in the design, with poor-quality voltage sources and leaky / high ESR (Equivalent Series Resistance) capacitors leading to high INL, while high-quality current sources and capacitors produce low INLs. DNL is mainly dependent on the quality of the ADC, with flash ADCs often having poor DNL and successive approximation ADCs having much better DNL.

To mitigate this, the ADC architecture with the fastest conversion rate, a flash ADC, is almost always employed. However, the area utilisation of the flash ADC is exponential in the number of bits B, as shown in (3.22). Also, compared to other TDCs, most of the the equation has been multiplied by the technology scaling factor S to account for the fact it does not shrink with technology improvements as digital methods do (and hence gets comparatively worse as technology scales). In the equation, A_{DCAP} represents a capacitor for digital purposes (i.e., one that is allowed to scale with technology improvements), A_{CSRC} represents a current source for digital purposes, and A_{CMP} represents a comparator for digital purposes.

$$T_{min} = T_{clk} * \left(\frac{1}{2^B} + E_{NL}\right) \tag{3.21}$$

$$\begin{aligned} A_{TAC} &= A_{ADC} + A_{ACAP} + A_{ACSRC} \\ &= 2^B * (S * A_{CMP} + A_{FF} + A_{3:2}) \\ &\quad + S * (A_{DCAP} + A_{CSRC}) \end{aligned} \tag{3.22}$$

The TAC-ADC combination mainly appears in older literature, and regularly exceeds all other methods available at that time in area and precision. As digital logic scaled, the difference between TAC-ADC methods and digital methods lessened, until eventually digital elements overtook the resolution of TAC-ADCs, which rarely improve in precision. For example, [120] was written in 1988, well before other digital methods even reached sub-nanosecond timing, yet managed 10ps timing resolution. However, the resolution achieved was highly dependent on the quality of analog components used, with [121–124] achieving resolutions between 10ps and 312.5ps, as well as count rates between 100KHz and 100MHz. Also, it can be seen by comparing the count rate and speed for various implementations that the count rate generally increases as the resolution decreases, since higher resolution ADCs either need significantly more area than a low-resolution counterpart, or they need to apply another conversion technique with higher resolution, but slower operation.

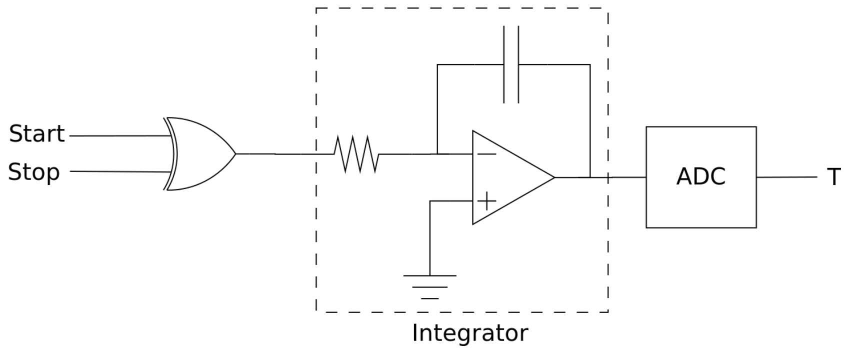


Figure 3.13: A TAC-ADC combination acting as a TDC. The integrator could be replaced with any other design.

3.3.13 SERDES TDC

When operating on an FPGA, serialiser deserialiser (SERDES) blocks can be used to create uniform delay elements to form high resolution fine time interpolation TDCs. SERDES are generally used in high-speed communication applications where the input/output (I/O) channels are limited. The transmitter's parallel input is serialised using a high-speed clock. At the receiver, the data is deserialised to the original parallel format. In other words, SERDES blocks have lower data rates at the input, they conduct the transmission at a faster clock frequency and have the lower data rate again at the output. As a result, the I/O required for the transmission is minimised and no data is lost during transmission due to the faster data rate. Modern FPGAs offer SERDES blocks which can provide 10 times clock multiplication and so 10 times faster serialisation. Since SERDES blocks are uniform chains of shift registers which are synchronised with a high-speed clock throughout the transmission, they provide high resolution fine time quantisation. As described in [125], a SERDES based 96-channel TDC was implemented on two Altera Stratix EP1S30F780C6 FPGAs which achieved a 1.2 ns resolution.

Due to the constantly-converting design of the SERDES TDC, the dead time is effectively zero (limited by the decoding logic and resolution). The INL is also low, being dependent on the quality of the input clock, which can be improved to sub-picosecond accuracy if needed. However, the DNL may be fairly high due to inaccuracies in generating the derivative (bit) clocks from the input clock.

3.3.14 Time Amplification

To increase the accuracy of the TDC, some have proposed using time amplifiers (TAs) to multiply the time domain signal, measure the multiplied signal, and then divide by the gain of the TA. This way, the accuracy is multiplied by the TA's gain, however, it also introduces a relatively large dead time where the detector is not able to detect a signal on the same channel due to the previous signal is in the process of being amplified or measured, resulting in a lower count rate and missed counts. Also, the range which can be obtained from this level of the TDC is divided by the TA's gain. These effects are represented in (3.23), where G_T is the gain of the TA, T_{unamp} is the resolution of the TDC when unamplified, and T_1 to T_4 are the start, stop, amplified start and

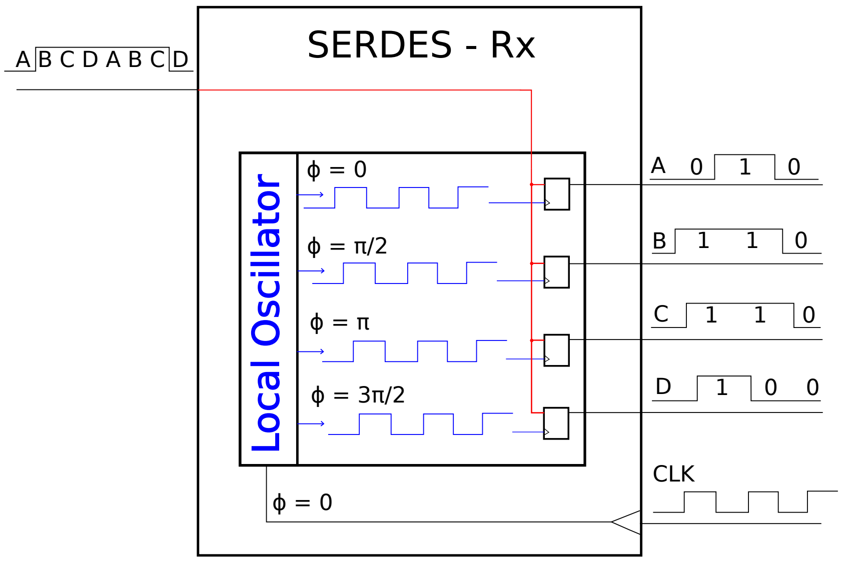


Figure 3.14: A SERDES TDC with 4 x interpolation. Φ is the phase of the clock signal from 0 to 2π.

amplified stop signals respectively. The range, $N * T_{min}$, is hence going to be smaller than the range of the unamplified TDC, $N * T_{unamp}$, while the resolution is 'higher' (lower valued).

TAs are generally designed in one of two ways: either they exploit metastability in a system to amplify the time difference, or they use an analog dual-slope design. The metastability method, which puts some logic into a metastable state and then quantises the time taken to settle, is limited in range to the metastability time of the components. This can be increased by lowering the speed of the components (by increasing the length of the transistors), but this increases area. Dual-slope systems convert the time difference into an analog voltage, and then convert this analog voltage back into time, but with a different gain factor ($G_{ATC} = k * GTAC$). The time of either the entire process or just the falling slope is quantised to obtain the time. Dual-slope methods suffer from the same drawbacks as TAC-ADCs: Bad scaling with process technology and slow conversion times.

The use of time amplification generally increases dead time, as the period to be converted is longer and the number of elements to convert it must also increase. DNL generally decreases due to the residue of large bins being broken down by the time amplifier and re-converted, while INL tends to suffer due to inaccuracies in the amplification process.

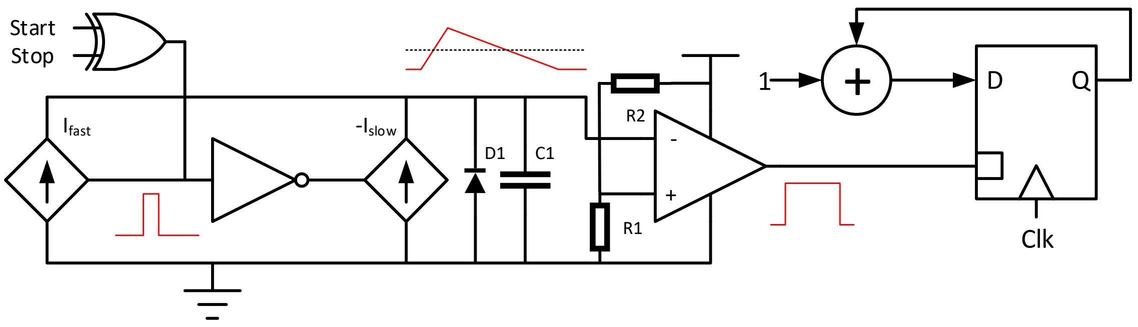


Figure 3.15: A dual-slope time amplifier.

$$
\begin{aligned}
T_2 - T_1 &<= N * T_{min} \\
T_4 - T_3 &<= N * T_{unamp} \\
T_4 - T_3 &= (T_2 - T_1) * G_T \\
T_{min} &= T_{unamp}/G_T \\
N * T_{min} &= N * T_{unamp}/G_T
\end{aligned}
\tag{3.23}
$$

Usage of metastable TAs is generally split into two areas: small time delays that need to be recorded extremely precisely, and as a component in a multi-level system. The systems that need to measure small time delays usually directly amplify the input signal, then measure with a suitably high precision TDC (such as a DL or VDL) [126]. As the range is bounded by the metastability time, it is possible to employ relatively inefficient methods (in terms of scaling in number of bits) at this stage to obtain higher accuracies. On the other hand, the systems that need to measure long time delays will generally produce a "residue" - the difference between the measured time value (for example, 14ps in a 1ps resolution system) and the actual time value (for example, 14.47ps). This residue (0.47ps in the example) is then amplified to a much higher value (for example, 18.8ps under a 40x amplification) and measured again, possibly using a more accurate TDC than the one that generated the residue (for example, a DL generates the residue while a VDL consumes the amplified residue) [127–129].

A few systems [130–132] take this a step further and repeatedly amplify the time residue, either applying this to a 1-bit TDC to compare the time difference to a known quantity, or measuring the time difference at each stage and generating a new residue. As is suggested by the number of references, the two-stage systems, often called pipeline TDCs, are by far the most common as the need to measure an extremely small maximum time difference is quite rare, except in ADPLL applications, which are often served by STDCs or other high-resolution, short-range TDCs.

As for dual-slope systems, these were most popular at the same time as TAC-ADCs, as they would often be connected to the enable signal of a simple counter as they charged or discharged. [133, 134] show modern realisations of a dual-slope TACs, with the one employing a dual-downwards slope with a capacitor, current source, and comparator, while the other employs a scheme where multiple parasitic (gate) capacitors are charged or discharged in a delay line, with the progression of a signal through the delay line being measured. Such a system may be considered similar to a gated ring oscillator in terms of its operation. However, it doesn't exhibit noise shaping properties, and hence it is not useful to classify it as one. The more traditional implementations, such as [135, 136], have fallen out of favour due to increased component mismatch making current sources, and hence linear transfer characteristics, harder to achieve as the process node shrinks.

In terms of resolution, the highest achieved was 630fs in [132], which also achieved a range of 1.3ns. This was most likely due to the repeated use of time amplifiers to amplify the residue at each stage, using an architecture similar to a linear SA-TDC. Despite the possibility of the pipeline TDCs achieving a much longer range that 1.3ns, none decided to do so, either because the range was unneeded or because the area efficiency (when a TA is connected to each intermediate output stage) is too bad to create more that approximately 10 bits of resolution.

3.4 Performance

Discussion of various architectures' merits and demerits is, unfortunately, insufficient to make a decision on which architecture is best for a particular use case. Hence, this section analyses a portion of TDCs available in current literature, showing the achieved performance in each case as well as the architecture and process technology used. Table 3.3 displays key performance metrics from many recent (last 20 years) papers on time-to-digital converters. The table is sorted by resolution, as this is arguably the most important metric for a TDC, but it also includes information on process technology, integral and differential non-linearity, single-shot precision, number of bits and range, number of channels, and the architecture used.

Not all information can be expressed in such a table, and so exceptions worthy of note are as follows: [74], [136] and [123] are the only papers in the table that do not use CMOS, using an unspecified FPGA, a $0.8\mu m$ BiCMOS process and an unspecified ECL process respectively. [59], [146] and [110] are unique in that they also integrate arrays of single-photon avalanche photodiode (SPAD) pixels on the same chip, with either a one-to-one or one-to-many matching between TDC and SPAD, thereby incorporating an entire depth-mapping system onto a single chip.

[79], [143] and [140] are also worthy of mention, as they use multi-dimensional schemes to reduce area requirements. The former arranges it's pulse-shrinking elements into a two-dimensional grid, with row and column decoders being used to ascertain the position at which the pulse was extinguished, and the latter two employ a scheme of splitting their delays into a sequential set (e.g., 1, 2, ..., 8) which operates on each column, and a sparse set (e.g., 1, 9, 17, ...) which operates on the rows, with the arbiters comparing the two delayed signals to each other, as opposed to a delayed signal against an undelayed signal. Finally, the usage of switched capacitor arrays in [104] and [102] and resistive dividers in [87] and [89] shows how analog components can be used to great effect while still exhibiting technology scaling improvements.

3.5 Architecture Comparison

It can be seen from Table 3.3 that main trade-off many designs make is between conversion time, resolution, and range. Stochastic and metastable time amplifier systems are both similar in that they sacrifice their range to increase their resolution and conversion rate. This makes them excellent for systems that have events happening very quickly (such as short-range time-of-flight (ToF)) or close to a known reference (such as frequency synthesis), but makes them bad for applications that require a large dynamic range, such as long-range ToF systems. However, due

Method	Technology*	Resolution (ps)	DNL (LSB)	INL (LSB)	SSP (LSB)	Channels	Ref
Algo TDC	350	0.61	±0.4	±4.5	±1.2	1	[109]
CSA-TDC + Counter	350	0.61	N/A	N/A	N/A	1	[104]
STDC	130	0.7	±1.4	±2.4	N/A	1	[92]
Branching CRO	65	0.85	±0.27	±2.94	1	1	[137]
TAC-ADC	N/A	1	N/A	±10	1	1	[138]
CSA-TDC + Counter	350	1.2	N/A	±6.67	3	1	[102]
DL + TA + DL	90	1.25	±0.8	±3	1	1	[127]
Algo TDC	350	2	N/A	±1.25	±0.15	1	[5]
Looped LPI-TDC	90	4.7	±0.6	±1.2	0.7	1	[87]
Looped LPI-TDC	90	4.7	±0.5	±1.0	N/A	1	[89]
Wave Union A	FPGA	6	N/A	N/A	1	48	[55]
GRO	130	6	N/A	N/A	N/A	1	[115]
VDL + DL + Counter	180	10	N/A	N/A	N/A	1	[70]
Wave Union B	FPGA	10	N/A	N/A	1	8	[1]
TAC-ADC + Counter	ECL	10	N/A	±2	1.5	1	[123]
TA + DL	180	11.25	N/A	N/A	1.33	1	[139]
Flash (2D) + CRO + Counter	350	12.2	N/A	±0.41	0.66	1	[140]
Vernier SA-TDC	180	12.5	±0.4	±0.4	N/A	1	[48]
Looped PS	800	20	±0.5	N/A	1	1	[82]
TA + SA-TDC	90	20	N/A	N/A	N/A	8	[141]
DDL + Counter	90	21	±0.7	±0.7	N/A	1	[142]
VDL + CRO + Counter	350	24	±0.55	±1.5	N/A	1	[72]
Flash (2D) + CRO + Counter	600	30	N/A	±1.33	32	1	[143]
VDL	700	30	N/A	±1.0	0.2	1	[64]
Hierarchical TDC	90	31.25	N/A	N/A	N/A	1	[144]
Dual-Slope TAC + Counter	800	32	N/A	±0.16	0.94	1	[136]
DL + Counter	130	40	N/A	N/A	N/A	1	[63]
CSA-TDC + CRO + Counter	350	42	N/A	N/A	N/A	1	[50]
GRO	130	45	N/A	N/A	N/A	1	[145]
CRO + Counter	130	50	±0.5	±2.4	N/A	1024	[110]
PS (2D Array)	800	50	N/A	±3.5	N/A	1	[79]
CRO + Counter	130	55	±0.3	±2.5	N/A	20480	[146]
Wave Union A	FPGA	60	+1.17	±1.08	0.42	1	[1]
CRO + Counter	180	61	±0.23	±0.3	1	24	[61]
VDL + DL + Counter	FPGA	75	N/A	N/A	0.53	1	[74]
CRO	65	80	N/A	N/A	N/A	1	[58]
DDL + CRO + Counter	350	97	±0.09	±1.89	N/A	32	[59]
TAC-ADC + Counter	500	312.5	±0.2	±0.3	0.32	1	[122]
CRO	800	530	±0.36	±0.36	N/A	1	[57]
SERDES, 2 chips x 48 channels	FPGA	1200	±0.167	N/A	±0.5	48x2	[125]

Table 3.3: Comparison of TDCs in the current literature

to their high count rate and resolution, they are also useful as the lowest level in a multi-level system, where they provide the least significant bits while leaving the more significant bits to other methods.

Vernier, pulse-shrinking, dual-slope time amplifier and, to a lesser extent, successive approximation systems instead decide to sacrifice conversion rates and area for better range and resolution. This is excellent for systems with a low repetition rate or where the repetition rate can be controlled (such as long-range ToF), but inadequate when a high repetition rate (such as short-range ToF or quantum key distribution) is required, as often the only option is to employ an interpolation or pipelining scheme which can massively hurt area efficiency.

Delay line and controllable and gated ring oscillator methods avoid low count rates and low range, but suffer in terms of resolution, meaning they are often a good choice either as a mid or upper level of a multi-level TDC, or in the case where the application does not require high precision, such as for low-rate frequency synthesis, long-distance low-resolution time-of-flight and quantum key distribution with low dead-count rates. The GRO method can also suffer from some signal integrity issues in low count rate systems but excels when measuring the same time period multiple times over due to its first order noise shaping.

TAC-ADCs and local passive interpolation methods perform very well in all three areas but suffer from signal integrity and area efficiency problems, as well as a lack of technology scaling in the TAC-ADC's case. If an LPI TDC were extended to its logical extreme with several resistors in its potential divider, it could probably achieve much higher resolutions than shown in Table 3.3, although the area utilisation would increase exponentially due to the number of resistors needed.

Flash TDCs are highly configurable, with high resolutions and ranges available at the expense of area and conversion time, making them excellent as a mid or low-level section of a system, although they don't achieve the same resolutions as stochastic, time amplification, vernier or local passive interpolation methods.

3.6 Conclusions and Recommendations

In conclusion, time to digital conversion has developed many approaches over the last two decades, each of which has its own unique characteristics. When measuring the same signal many times over, it is worth employing a gated ring oscillator for its first order (and higher order when multiple GROs are used together) noise shaping. If the conversion rate of the target system is of little concern, then, depending on area constraints, utilising looped vernier or pulse shrinking methods is advised, as the analog components in dual-slope systems do not scale well with technology.

Beyond this, the systems that have been shown to perform best are multi-level systems that exploit the benefits of multiple architectures while covering the weaknesses with the others. Most notably, SA-TDCs are delay-element agnostic, meaning they can incorporate a large variety of delay generation methods to assist in obtaining the correct range and resolution, and output the residue (the difference between the measured and actual value of the time difference) at each stage, which can then be passed directly to a higher resolution TDC. It is suggested that, should a small-enough residue be obtained at the final output of the SA-TDC, either a stochastic or metastable time amplification method be used to obtain optimal accuracy. Similarly, if a CSA-TDC were employed, a time amplifier could be switched in as the system approaches the lowest bits to amplify the time difference and hence reduce the requirements for small delays.

Methods that use traditionally analog components in a way that allows them to scale, such as local passive interpolation and switched capacitor arrays (as seen in the CSA-TDC papers) allow for very small time differences typically not seen in digital methods. However, these systems require careful choice of components and layout to minimise non-linearity and interference from other components, and often require large areas of the layout (although not as much as other analog methods). Hence, they are worth consideration if the expertise and design constraints allow for such an approach.

Chapter 4

DSP Blocks as Delay Generators

4.1 Abstract

In this section, a novel TDC architecture involving DSP blocks on FPGAs is described. There are three papers related to this section: [10], [11] and [12]. Therfore, this section is split into an introduction section and three investigation sections, each of which describes a different stage of our investigation into the DSP blocks.

Section 4.3 describes our initial investigation into the DSP blocks. We observed the properties of the Xilinx Spartan 6 DSP48A1 blocks when configured as a classic delay line, and ultimately concludes they are unsuitable for use in isolation due to the high differential non-linearity but may be suitable as a semi-fine stage of a multi-stage TDC or when combined in an equivalent coding line. Section 4.4 then moves to the Artix 7 DSP48E1 blocks that will be used for the rest of the section and shows how we were able to achieve 5.25 ps resolution with four parallel DSP delay lines and ones-encoders for decoding. Then, Section 4.5 characterises the DSP blocks in more depth and with respect to temperature, observing that some measures of performance improve with temperature due to the reduction in the size of the Ultra-Wide Bins (UWBs). Following this, Chapter 5 will demonstrate the introduction of the wave union technique to a DSP delay line to achieve comparable resolution to the parallel DSPs with a significant reduction in area. However, the very large bubbles along with multiple edges presented a decoding challenge that was solved with the bubble corrector described in Chapter 6.

Table 4.1: Comparison of DSP and carry chain delay lines.

Technique	Device	Resource	Resolution (average)	SSP	Utilisation (10 ns clock)
Delay Line	Spartan-6 FPGA	6-input look-up table (LUT6)	20 ps	25 ps	128 CLBs
Delay Line	Artix-7 FPGA	6-input look-up table (LUT6)	15 ps	13.16 ps	170 CLBs
DSP Delay Line	Spartan-6 FPGA	DSP Block (DSP48A1)	16.7 ps	–	13 DSPs
DSP Delay Line	Artix-7 FPGA	DSP Block (DSP48E1)	9.8 ps	52.56 ps	22 DSPs
4x DSP Delay Line	Artix-7 FPGA	DSP Block (DSP48E1)	5.25 ps	–	74 DSPs
8x DSP Delay Line	Artix-7 FPGA	DSP Block (DSP48E1)	3.70 ps	–	146 DSPs

4.2 Introduction

4.2.1 Initial Investigation

When looking specifically at FPGA-compatible TDCs, the most common method utilised appears to be delay lines, as there is almost always some form of delay element available on an FPGA. Sometimes, an FPGA will have a suitable architecture for implementing a vernier delay line [147], and sometimes an element can be found which is suitable for pulse shrinking, but this is dependent on the particular FPGA architecture. Delay lines can be generated in various ways, such as with the look-up tables (LUTs) configured as buffers, but the highest resolution found so far appears to be the carry chain [148] made available for addition operations.

As FPGAs are often required to perform addition and subtraction, most vendors implement carry logic with a fast interconnect between logic blocks [3, 149], as this is commonly the critical path, and so fast carry logic may drastically increase the speed of the design. The carry logic also generally propagates perpendicularly to the rest of the logic interconnects, as this allows parallel signals which are part of the same number to have similar routes through the FPGA, thereby increasing the probability of meeting timing requirements. These carry chains are also ideal for implementing fast delay lines which output a priority code based on the delay between the start and stop signals.

However, there is another source of fast carry propagation: DSP blocks. Both Intel (formerly Altera) and Xilinx have DSP blocks on their FPGA fabric to accelerate multiplication and addition operations even further than is possible on the general-purpose fabric [3, 6]. But despite the importance of generating high-resolution delays, there are seemingly no reports on the viability of the FPGA's built-in DSP block as a delay generator, despite the architecture suggesting that it is suitable. Therefore, in Section 4.3, we examine the capability of Xilinx's DSP48A1 blocks, present on their Spartan 6 series FPGAs, to generate small delays and hence act as a delay line.

If the implementation of delay lines on DSP blocks is successful, it is expected that we can achieve higher resolutions and precisions compared to carry chains (due to dedicated addition logic) while using less area (2 DSP blocks for every 10 slices in applicable tiles, which is 2.5x as dense) and freeing general-purpose logic for other purposes (other system tasks or more delay lines).

4.2.1.1 Structure

Section 4.3 will be structured as follows: first, there will be a section describing the changes made to the DSP48E1, how they affect the design, and how the system is configured to obtain delays from these elements. Then, there will be a section detailing our testing methodology. After that, there is a section on the results obtained from our tests. Next is a section discussing the results obtained. Finally, there will be a conclusion section and discussion of further work.

4.2.2 Solving the Non-Linearity

With the introduction of its 7-series FPGAs, Xilinx introduced a new iteration of their DSP48 block, the DSP48E1. In Section 4.4, we examine how the DSP48E1 has changed and what this means for the generation of high-resolution delays with this block. We also apply multichain techniques (Equivalent Coding Line, ECL) to four parallel DSP delay lines to create a higher-resolution TDC.

The DSP48E1 blocks are a significant upgrade compared to the DSP48A1 blocks seen in Section 4.3, implementing a pattern recogniser, deeper pipelining, higher clock speeds, SIMD and 2-input logic functions [4]. We once again utilise the post-adder in the DSP as this provides a fast carry through multiple elements and perform code density tests to determine the feasibility of using the new DSP blocks as delay generators.

4.2.2.1 Structure

Section 4.4 will be structured as follows: first, there will be a section describing the changes made to the DSP48E1, how they affect the design, and how the system is configured to obtain delays from these elements. Then, there will be a section detailing our testing methodology. After that, there is a section on the results obtained from our tests. Next is a section discussing the results obtained. Finally, there will be a conclusion section and discussion of further work.

4.2.3 Temperature Characterisation

Characterisation of the temperature dependence of a TDC is vital to the applicability of the device to real-world scenarios. As events are captured, converted, and output, the TDC core consumes energy and the power dissipation rises, providing a vector for temperature increases in the device. In addition, most TDCs will be part of a much larger system, and changes in load on other parts of the system may cause a rise in chip, board or ambient temperature which could negatively impact the TDC. This is even more important when events must be captured at picosecond timing resolution, as even a small change in temperature can result in erroneous measurements. Therefore, characterisation results that can be used to adjust measurements in post-processing or inform the need for recalibration of the system are vital.

In Section 4.5, we will be taking a closer look at the DSP delay line, focusing on the characterisation of the DSP delay line across a wide range of temperatures as well as in-depth analysis of the Differential Non-Linearity (DNL), Integral Non-Linearity (INL), Single-Shot Precision (SSP), resolution and channel offset.

4.2.3.1 Structure

Section 4.5 will be structured as follows: Section 4.5.1 will detail the design of the system, test setup and characterisation methodology. Section 4.5.2 will then present the characterisation results and highlight some trends in the data, specifically with regards to temperature. Section 4.5.3 will then present some discussion on the implication of the observed results and Section 4.5.4 will summarise our findings and present some possible avenues for future investigation.

4.2.4 Delay Lines

The basic operation of a delay line (regardless of the architecture and delay element used) is as follows: a start signal (Fig. 4.1: trigger) enters a cascade of delay elements. Each of these elements require a time delay (T) for a signal at the input to propagate to the output. By cascading these delay elements, we can form an array of signals with incrementally increasing delay. These elements are input to the data input (D) of a discriminator (displayed as Flip-Flops in Fig. 4.1). At some point after the start signal, a stop signal (Fig. 4.1: clock) causes all discriminators to latch. If the start signal arrived at the discriminator before the stop signal, then the discriminator will latch a '1'. If the stop signal arrived before the start signal, the discriminator will latch a '0'. At the data outputs (Q) of the discriminators, a thermometer code will therefore be formed, with each bit in the thermometer code signifying a time increase of T relative to the previous bit. This thermometer code is then converted to a binary value before being output.

The resolution of a delay line is dependent on the delay of the elements used in its construction. For example, in a modern FPGA, the CARRY4 elements used in most delay lines have an average delay of 15 ps (each CARRY4 has four taps and a total delay of 60 ps). Comparatively, a modern ASIC may have delay elements as small as tens of femtoseconds. However, when the delay of the elements is non-uniform (due to process variations and uneven loading), the performance and resolution of the TDC will be degraded compared to the average. Therefore, the equivalent width (4.1) [37] is a better measure of resolution, and non-linearity numbers such as the Differential Non-Linearity (DNL) and Integral Non-Linearity (INL) are often quoted.

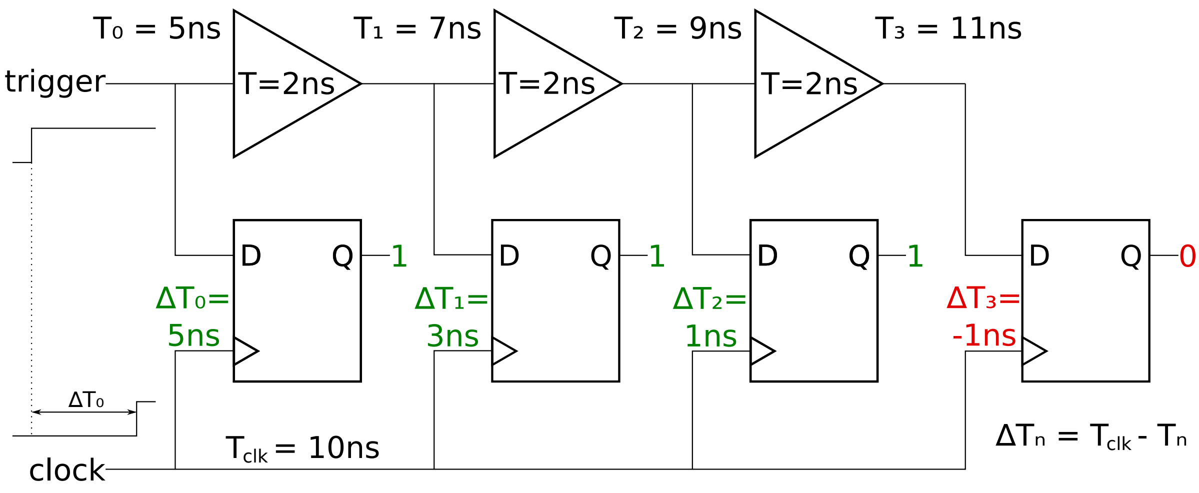


Figure 4.1: General operation of a delay line. Output code 1110 is a thermometer code reading $4ns < T_0 < 6ns$ (2 ns LSB). Green: positive ΔT_i (output '1'); Red: negative ΔT_i (output '0').

$$w_{eq} = \sqrt{\frac{1}{R}\sum_{i=0}^{N-1}\tau_i} \qquad (4.1)$$

4.3 Initial Investigation

4.3.1 DSP48A1 Architecture

Almost every DSP algorithm has MAC (Multiply Accumulate) operations. In the Spartan-3A architecture, the DSP48A slice is available for MAC operations and it has been extended into the DSP48A1 slice in the Spartan-6 series. In Spartan-6 FPGAs, DSP48A1 slices are organized as vertical columns along with some additional dedicated logic and routing [6]. One of the most important features is the ability to cascade a result from one DSP48A1 slice to the next without the use of general fabric routing. Each DSP48A1 slice contains an 18-bit input pre-adder followed by an 18 x 18 bit two's complement multiplier and a 48-bit sign-extended post-adder/subtracter/accumulator, a function that is widely used in digital signal processing. Fig. 4.2 shows a simplified block diagram of the DSP48A1.

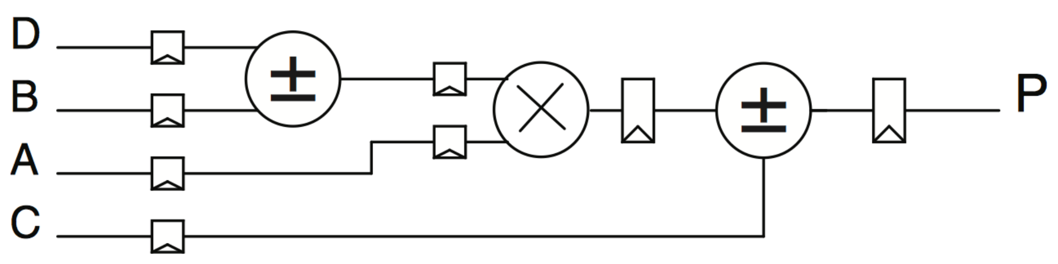


Figure 4.2: A simplified block diagram of the DSP48A1 [6]

All these arithmetic operations are hard-wired into the silicon at fabrication time. Every arithmetic function has pipeline registers on its input and output. These registers are very close to the corresponding function logic and can be bypassed, so arithmetic functions (Combinational Logic) can directly connect to FPGA fabric. The performance of DSP48A1 is explained in [150].

4.3.2 System Design

4.3.2.1 Generating a delay line inside a DSP48A1

The first task was to attempt to generate a delay line inside a single DSP48A1 block. A delay line requires a single input that changes from 0 to 1 (or vice-versa), and multiple outputs which transition successively. Preferably, it would also have unregistered inputs and outputs from adjacent blocks. For this purpose, we chose the post-adder in the DSP block. The benefit of the post-adder is the carry in and carry out connections, which can be configured to be unregistered. Also, it contains the shortest path to the output pins of the DSP48A1. Finally, the P register can be configured to provide the output discrimination required, thereby reducing the logic utilisation.

To configure the DSP48A1 block, all register bypass MUXes (as shown in Figure 4.3) were used to bypass their associated registers. The X input to the post-adder was set to use the D:A:B concatenated input port, which had all 48 bits set to 1, thereby causing the post-adder to carry any additional non-zero input through the internal carry chain. The Z input was set to use the C input where Z[0] is connected to the trigger and all other bits are connected to 0. The connection of C[0] to the trigger allows for the trigger to be input to the DSP carry chain.

When chaining multiple DSP blocks together, only the first DSP block has the Z input con-

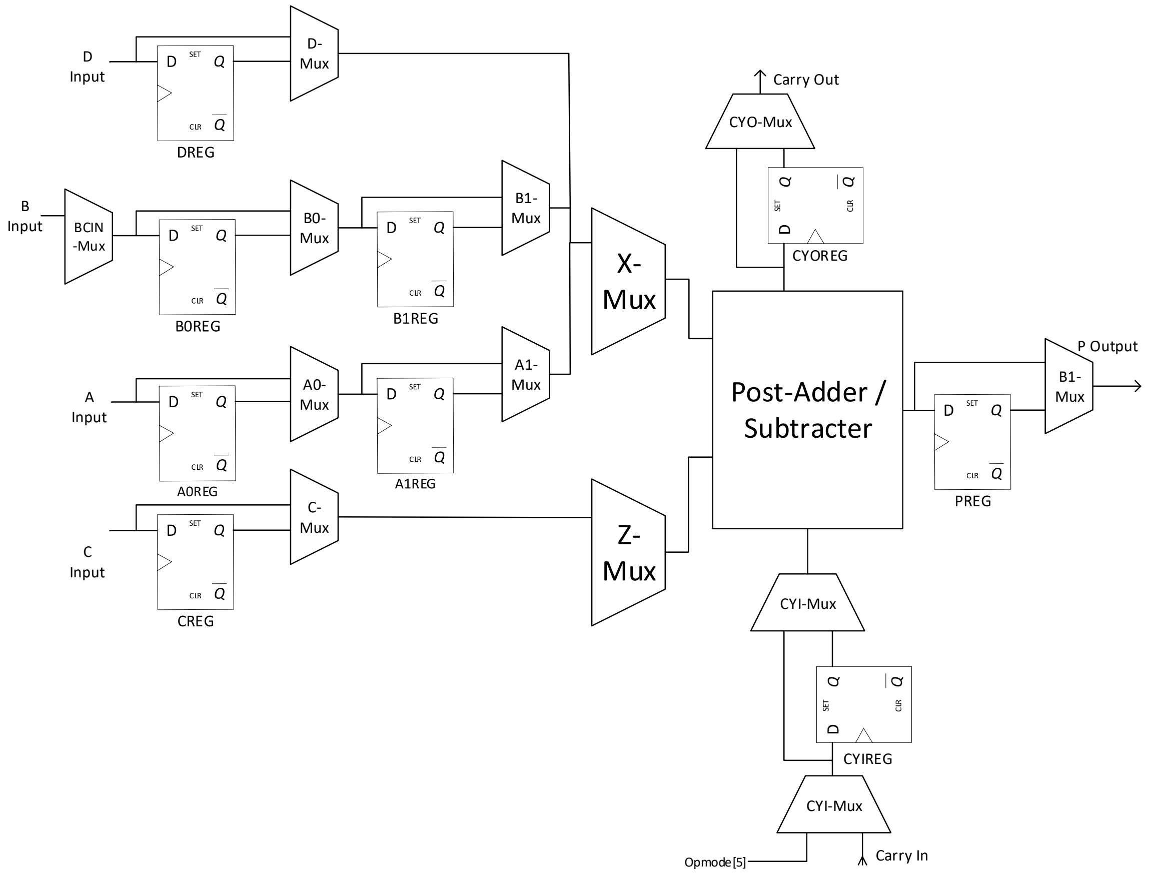


Figure 4.3: The internal architecture of the DSP48A1 block (utilised components only), based on [6].

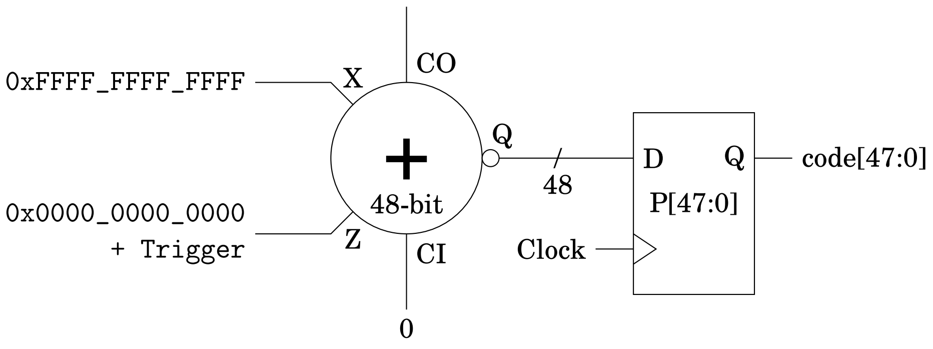


Figure 4.4: Configuration of a single DSP block as a delay generator.

nected to the trigger. All subsequent DSP blocks have their Z inputs connected to zero. However, whereas the carry input of the first DSP block is always zero, subsequent DSP blocks have their carry input connected to the carry output of the previous DSP block via the CARRYCASCOUT

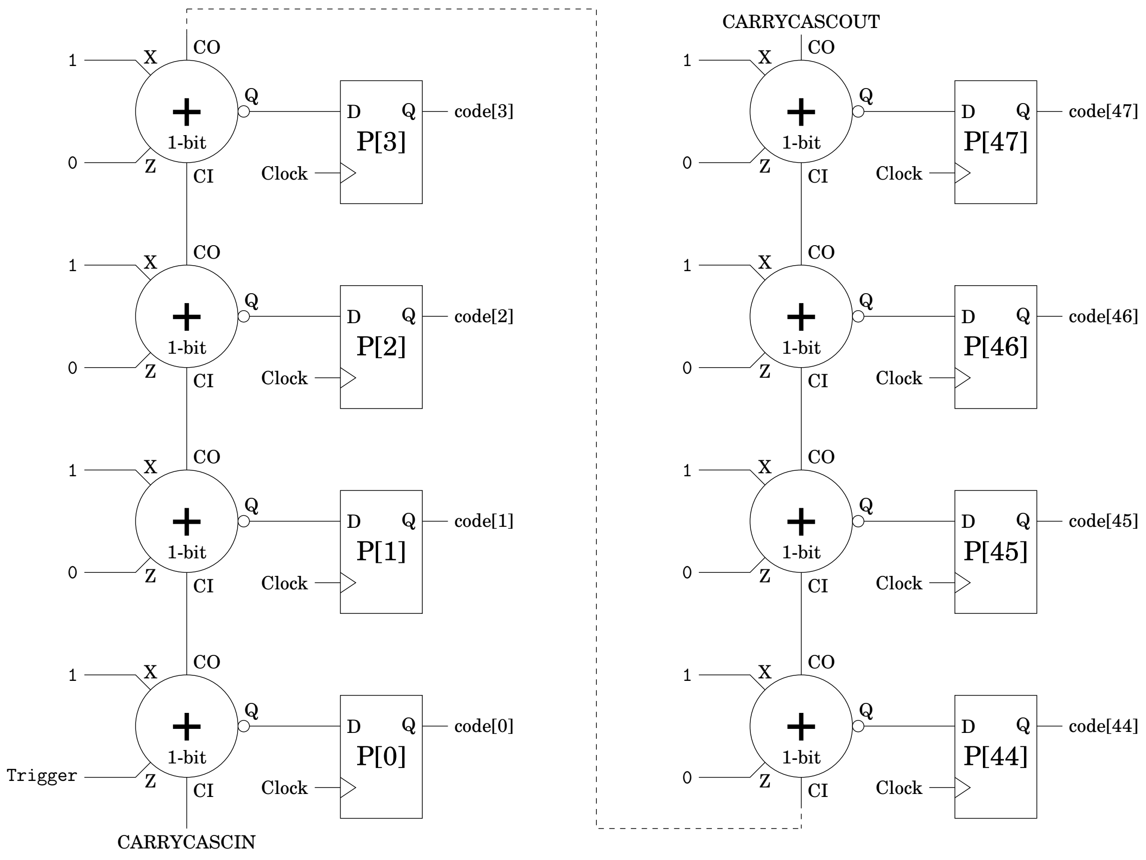


Figure 4.5: Theoretical internal operation of a DSP block as a delay generator, shown per-bit (carry look-ahead not shown). Bits 0-3 and 44-47 shown.

and CARRYCASCIN ports (CARRYCASCIN of block i is directly connected to CARRYCASCOUT of block $i-1$).

4.3.2.2 Placement on the FPGA Fabric

The carry in and carry out are dedicated connections between the DSPs, which are placed column-wise. By using the device-native blocks in the chosen hardware description language, the synthesis tool is forced to use adjacent DSP blocks with minimal inter-DSP latency. As a result, the delay between DSP48A1 blocks is minimised, resulting in optimal differential linearity in the delay line. A snippet of the instantiation of a DSP48A1 block can be seen below, full details are available in Xilinx UG615 [151]:

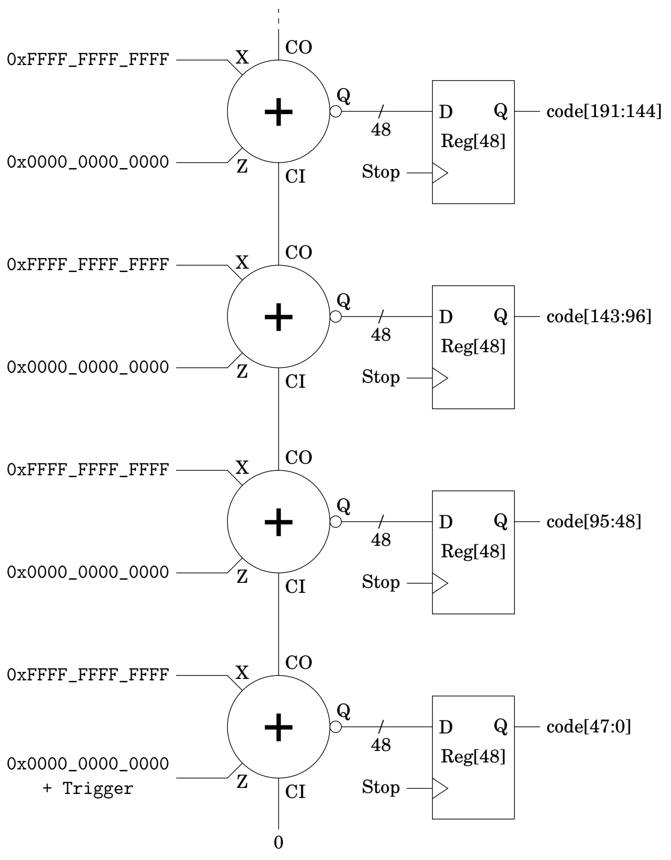


Figure 4.6: Configuration of a chain of DSP blocks for use as a delay line.

```
carry48: DSP48A1 generic map (

  -- Registers enabled or disabled with 0 or 1.
  A0REG => 0,
  A1REG => 0,
  ...
  CARRYINSEL => "OPMODE5", -- Carry in mux
  RSTTYPE => "ASYNC" -- Asynchronous reset signal.

) port map (

  -- Output signals
```

```
    CARRYOUT => carrys(1),
    P => code(47 downto 0),
    BCOUT => open,
    PCOUT => open,
    CARRYOUTF => open,
    M => open,

    -- Input signals
    CLK => clk,
    A => "00" & X"0000",
    B => "0" & X"0000" & trigger,
    C => X"FFFF_FFFF_FFFF",
    D => "00" & X"0000",
    CARRYIN => carrys(0),
    OPMODE => "00001111",
    PCIN => X"0000_0000_0000",

    -- Clock Enable signals
    CEA => '0',
    CEP => '1', -- Clock Enable for P is the only one high.
    ...

    -- Reset signals
    RSTA => '0',
    RSTB => '0',
    ...
);

```

4.3.2.3 Triggering Logic

To avoid reading out when there is no valid data in the delay line, some triggering logic was implemented. The trigger was passed through a synchroniser which normalised it to the clock edge, and then the current value of the trigger was compared to the value of the trigger on the previous clock cycle. If the trigger is high on the current clock cycle and was low on the previous clock cycle (rising edge triggered), then this implies a new trigger came in and so there is valid data in the delay line. This is then used to enable the priority encoder and read-out logic, which convert the priority code to binary and transmit the binary to a host computer respectively.

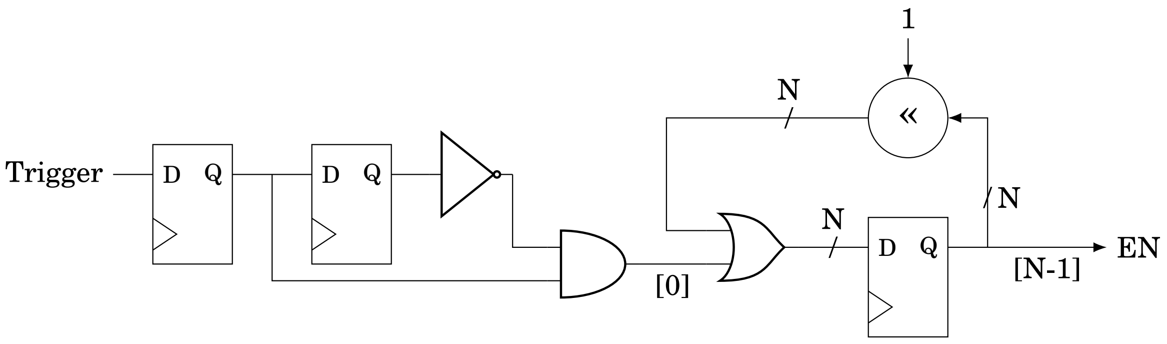


Figure 4.7: Schematic of triggering logic used in delay line.

To do this, the trigger signal is delayed by multiple clock cycles so that it stays in-line with the valid conversion as it propagates through the synchronisers and priority encoder, before finally becoming the enable signal for the read-out logic, which is described in the Section 4.3.2.4.

4.3.2.4 Read-Out Logic

Read-out was performed using the parallel interface provided by the Opal Kelly SDK [152]. This interface allows 16-bit quantities to be transferred to a host PC using the USB present on Opal Kelly FPGA carrier boards. As the tags are 10-bit quantities and the coarse count is not required for code density testing, the upper six bits of each word were set to 0, while the 10-bit bin number was stored in the lower bits of the word. A 10-bit quantity was required as 20 DSP blocks were implemented (960 bins) to ensure no tags exceeded the end of the delay line.

The transmitted data was collected by software on the host PC in a comma-separated variable format, then imported into GNU Octave for analysis. The analysis could have been performed on-chip using a histogrammer, but the USB readout was chosen to enhance debugging capabilities and allow more detailed analysis than is possible on-chip.

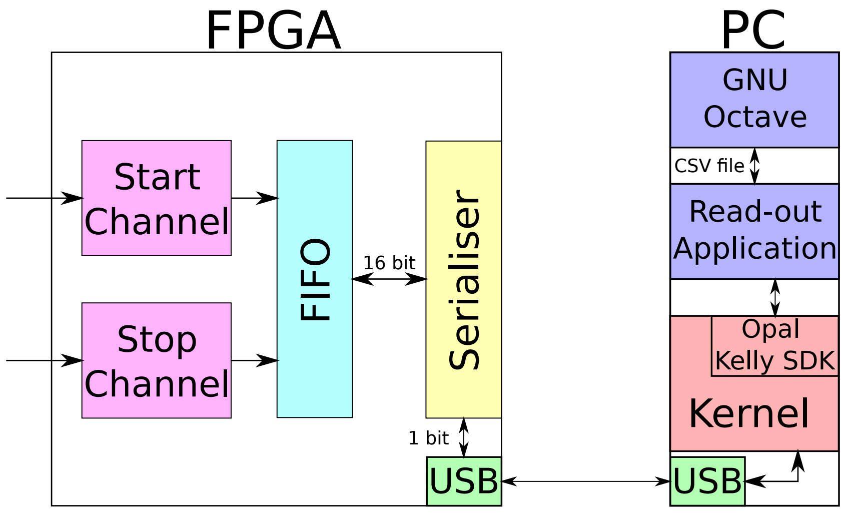


Figure 4.8: Readout logic for the TDC channels implemented with DSP48A1 blocks.

4.3.3 Test Methodology

High-resolution and PVT-stable delay lines and capture registers are required for TDC applications. As we use the DSP48A1's post-adder carry propagation logic (48 bits) and the P register as the capture register, we can expect the delay line's characteristics to be similar to that of a carry look-ahead adder (the architecture used in most adders). The carry lookahead logic is defined by (4.2), where C_i is ith carry output and it is produced by $G_i = A_i.B_i$ (carry generation) or $P_i = A_i \oplus B_i$ (carry propagation).

$$C_i = G_i + P_i.C_{i-1} \tag{4.2}$$

We model the DSP delay line using the generic model shown in Fig. 4.9. Each buffer here corresponds to the carry propagation logic between one bit of the adder and the previous bits of the adder. This allows us to use standard average delay and code density testing methods to determine the delay of each bit of the adder. In some cases the delay between adjacent bits may be nominally negative. This is due to carry bypass logic forwarding a bit by multiple stages, and so a bit i is being driven by bit $i-k$, where $k > 1$. This results in bit i transitioning before bit $i-1$, thus the relative delay is negative.

In the average delay tests, we determine the time taken for the start signal to reach the final time bin (end of the last 'buffer') as shown in Fig. 4.9. This implies an output code of 0xFFFF...FFFF and the MSB being '1'. In the code density tests, we determine the propagation delay of each 'buffer' by providing random signals in the time domain, uniformly distributed with respect to the stop signal, and creating a histogram of the priority encoder outputs. In the case that the stop signal is not equally routed (there are buffers between each bin on the stop line), the bin size will increase by the delay to the corresponding register and decrease by the delay to the previous register.

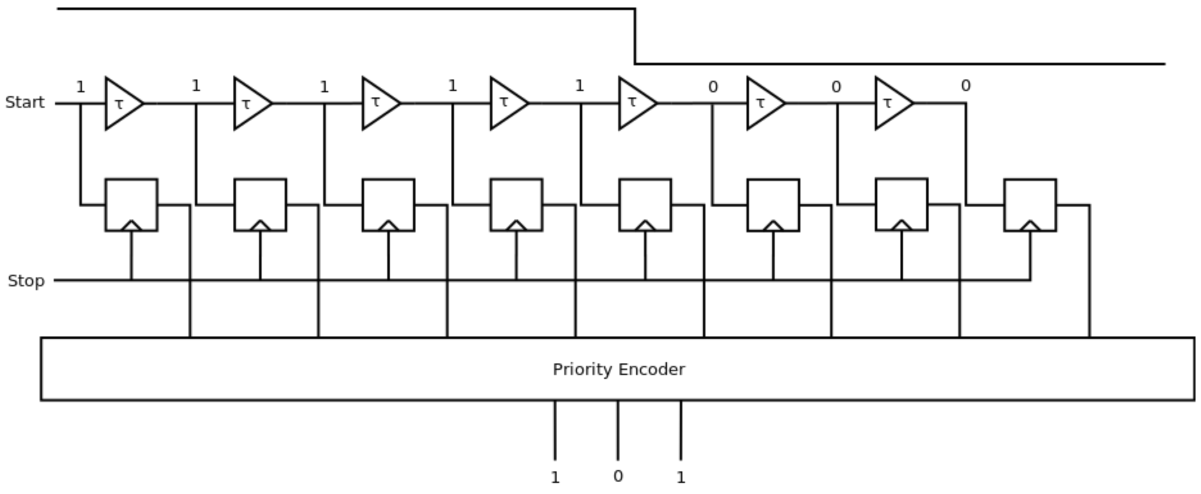


Figure 4.9: Generic model of a delay line.

4.3.3.1 Average Delay Testing

In the average delay test, the post-adder was set up to have all the bits on one of its inputs high, and all the bits on the other input low except the LSB, which was connected to the trigger signal. As we are adding one to the maximum representable number, all the bits will roll around to 0 through the carry propagation logic. To determine the average delay, we calculate the time difference between the LSB and MSB transitioning.

To implement this, we first connected two probes from a 1GHz oscilloscope to adjacent FPGA outputs (same IOB). These outputs were both connected to the LSB of a DSP block with the

D:A:B input (through the X-mux) being 0xFFFFFFFFFFFF to enable the carry chain and the C input (through the Z-mux) being 0 except the LSB, which was the trigger generated by an on-chip oscillator. The probe delays were then adjusted to tear (align) the signals before re-routing one output to the MSB of the DSP block, which resulted in an almost identical path while including the full delay of the DSP. The nets were inspected in the FPGA editor tool to ensure that the routing of the two signals was matched (close together) and to allow us to re-route or re-constrain the signals if not.

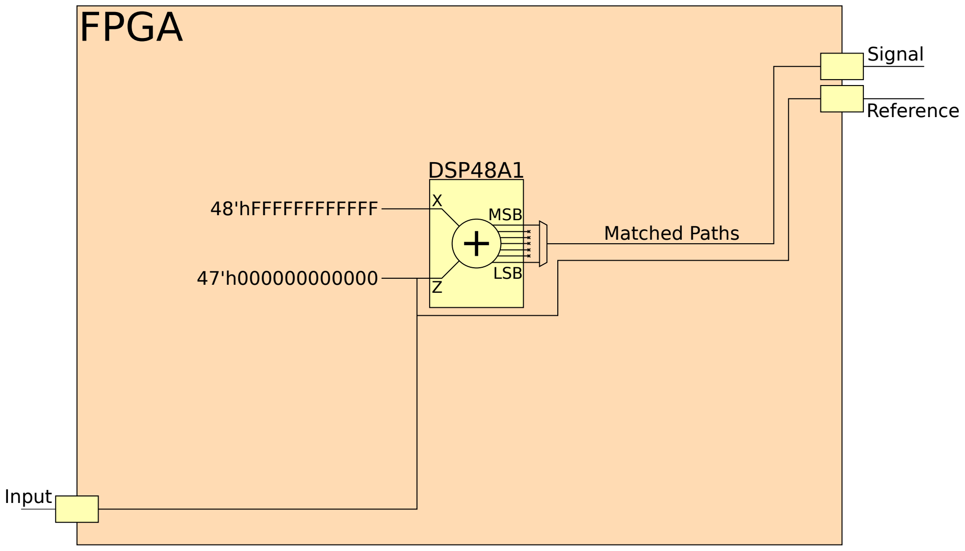


Figure 4.10: Method to test the average delay of a single DSP48A1 block. 48'hNNNNN: a 48-bit bus with hexadecimal value NNNNN.

4.3.3.2 Code Density Testing

In addition to the average delay testing, code density testing is an important component in determining the viability of a delay generation method. Even if the average delay is exceptionally low, a TDC is unusable unless the non-linearity characteristics are decent. A good parallel to this is the linearity of an ADC. Even if an ADC can be found to have 20 bits of resolution (well in excess of most modern ADC designs), this is useless if one bin covers half the range of the ADC while the other 1048575 cover the other half. In such a case, the effective resolution is at most two bits. Such an ADC is useful, however, as putting another ADC in parallel but offset in the voltage domain could allow the two ADCs to efficiently cover the entire voltage range. On the other hand, this increases the area requirements. The same can be said for a bad TDC; if the code density is decent, corrections can be made to cover for its weakness using another TDC in parallel. We will be using three types of code density test to examine the DSP blocks: a linear code density test, a cyclic code density test and a subdivided code density test.

Linear Code Density Testing Code density testing is one of many techniques to measure the linearity of the TDC. The code density test provides random, uniformly distributed signals to the TDC and stores the conversion results. When provided with a uniformly random signal (in the time domain), the number of times a particular code will be output by the TDC will be proportional to the size of the time bin corresponding to that code. A larger time bin (longer delay until the next storage element) will accumulate more 'hits' (number of times the corresponding bin code is output) whereas a smaller time bin will accumulate less hits. Once a suitable number of hits has been accumulated to obtain an accurate measure of the width of each bin, the size of time bin i, τ_i, will be calculated as in (4.3).

$$\tau_i = \frac{H_i \times T_{clk}}{H_{total}} \tag{4.3}$$

$$T_i = \sum_{j=0}^{i} \frac{H_j \times T_{clk}}{H_{total}} \tag{4.4}$$

In (4.3), H_i is the number of hits for bin i, H_{total} is the total number of hits for all bins, and T_{clk} is the period of the system clock (which acts as our stop signal, and so the period is the maximum possible time difference). Similarly, the cumulative form of this equation, T_i, can be used to determine the delay up to a certain bin (and is hence used for calibration). The uniformly random pulses were provided to the TDC by an external pulse generator outputting a square wave at 10 MHz with a 10% duty cycle. Data was read out via USB (University of Bristol) or serial connection (Quaid-i-Azam University) after encoding and serialisation.

Cyclic Code Density Testing In this test, the post-adder P register is enabled, and two independent oscillators are used to produce a random phase drift. The first oscillator is similar to the average delay test and is connected in the same way to generate the first carry. A 1kHz trigger signal with a different oscillator is connected to the P pipeline register to capture the carry propagation randomly. In the capture register, invalid codes (0x000000000000 and 0xFFFFFFFFFFFF) are removed, and the rest of the codes are transferred to a PC via a serial interface.

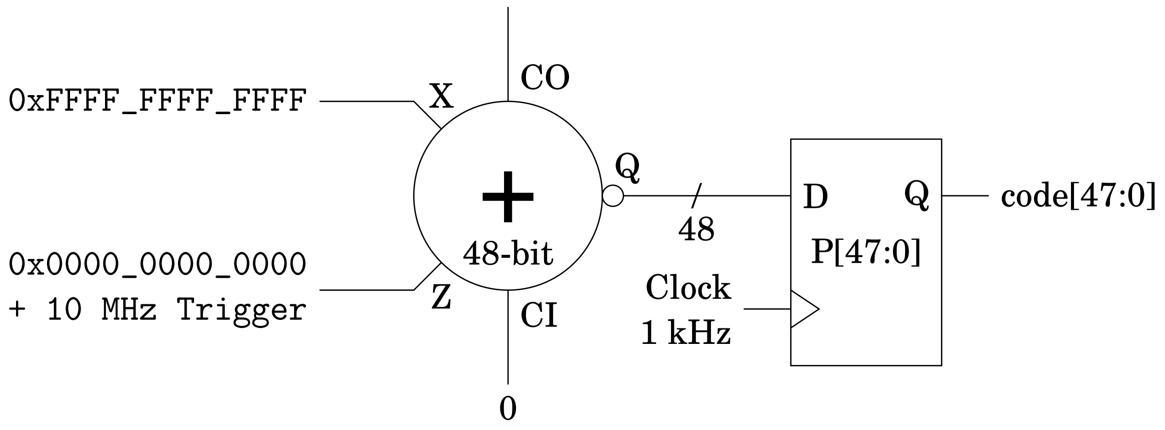


Figure 4.11: Diagram of the cyclic code density test.

Subdivided Code Density Testing To eliminate the possibility of carry look-ahead logic eliminating some bins by forwarding the carry across 24 bins at once, we implemented a method which simultaneously tests both the lower and upper 24 bits without continuity between them. In this test, a 10MHz trigger signal was input into a DSP block at positions 0 and 24 through the C input and Z-mux, while the D:A:B input and X-mux was set to 0x7FFFFF7FFFFF so that the carries would stop at positions 23 and 47 respectively. The number of cases of each output (23 and 47) being high was recorded on a host PC via a serial connection.

4.3.3.3 Validation

To ensure the results we achieved were accurate, we cross-validated our results across two labs, designs and pieces of hardware. The first design used linear code density testing on a Spartan-6 LX150 FPGA packaged by Opal Kelly [153] on a custom signal breakout board at the University of Bristol. The other device, a Spartan-6, used a cyclic code density test at Quaid-i-Azam University. If our results are to be believed, we expect that the individual bin sizes will not be the same, but the patterns in bin sizes should match. For example, if there is a recurring pattern every four bins on one device, we would also expect to see a similar recurring pattern on the other device.

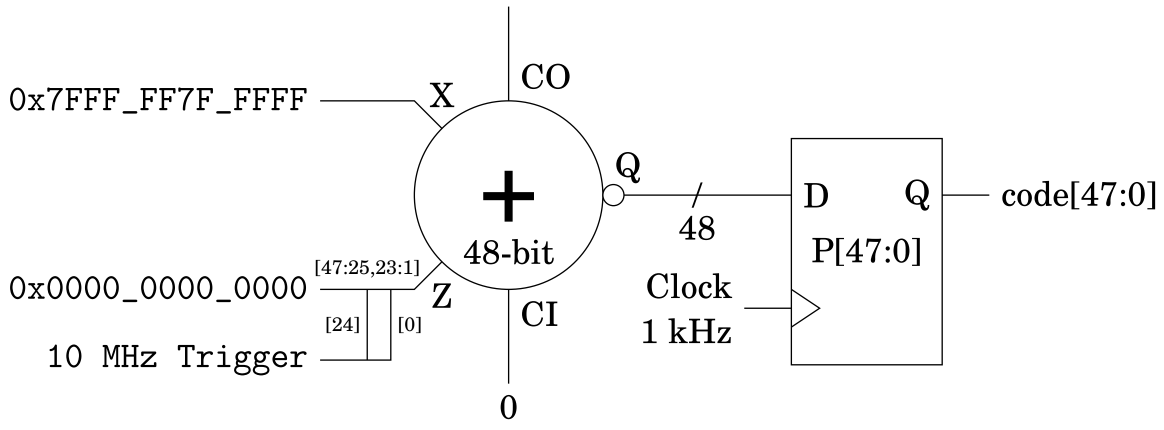


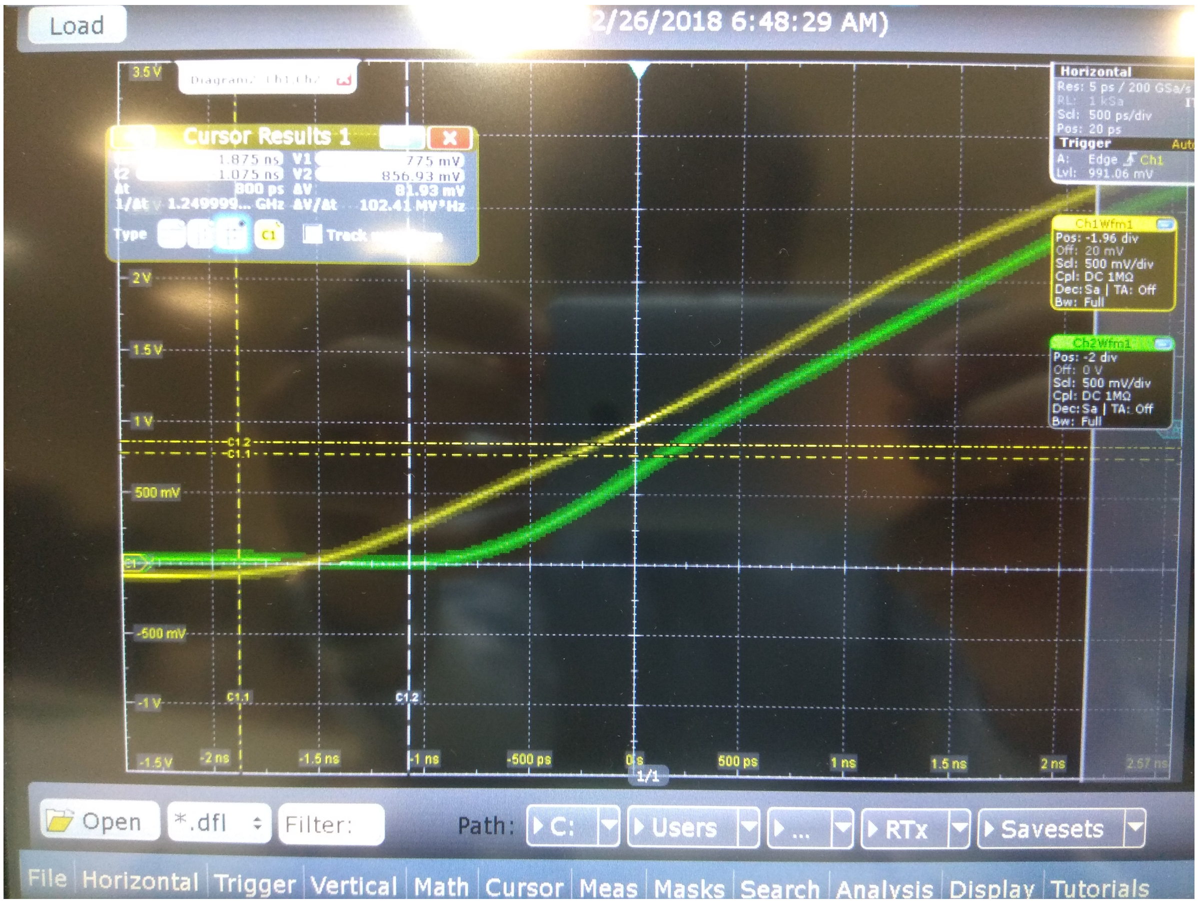
Figure 4.12: Diagram of the subdivided code density test. The missing bits on the X input disable some of the carry chain.

If this does not occur, then this suggests that the external components in the design (test input signal, read-out logic, system clock etc.) are interfering with the measurement and hence the design can be fixed to remove the effect of these external influences.

4.3.4 Results

4.3.4.1 Average Delay Testing

Under testing, it was found that the delay between the two output signals was 800 ps (Fig. 4.13), representing the entirety of the DSP delay. As there are 48 bins, this evaluates to a mean bin width of 16.7 ps, which is significantly better than the 21 ps present in Spartan 6 carry chains. A similar test was performed, but measuring the time to output bit 23 rather than bit 47. This gave a relative delay of 400 ps, showing that the total delays in the first and second halves of the DSP are equal.


2/26/2018 6:48:29 AM)

Cursor Results 1
1.875 ns V1 775 mV
t2 1.075 ns V2 856.93 mV
Δt 800 ps ΔV 81.93 mV
1/Δt 1.249999... GHz ΔV/Δt 102.41 MV*Hz

Horizontal
Res: 5 ps / 200 GSa/s
RL: 1 kSa
Scl: 500 ps/div
Pos: 20 ps
Trigger Auto
A: Edge Ch1
Lvl: 991.06 mV

Ch1Wfm1
Pos: -1.96 div
Off: 20 mV
Scl: 500 mV/div
Cpl: DC 1MΩ
Dec: Sa | TA: Off
Bw: Full

Ch2Wfm1
Pos: -2 div
Off: 0 V
Scl: 500 mV/div
Cpl: DC 1MΩ
Dec: Sa | TA: Off
Bw: Full

Figure 4.13: Time difference between the two output signals on a 1GHz oscilloscope.

4.3.5 Code Density Testing

Histograms from the code density tests conducted at the University of Bristol and Quaid-i-Azam University were collected, with the bin widths for a single DSP block shown in Fig. 4.14 and the calibration graph (cumulative bin widths based on stochastic code density testing) for a large (960 bin = 20 DSP) delay line shown in Fig. 4.16. The diagrams have been normalised to the number of hits and the clock period to obtain the delay times of each bin. As the delay is on the y axis, a large vertical jump (large bin width) is undesirable (low resolution), whereas a small, non-zero vertical jump is desirable (high resolution). No vertical ascension between bins implies a lost code, which can be ignored.

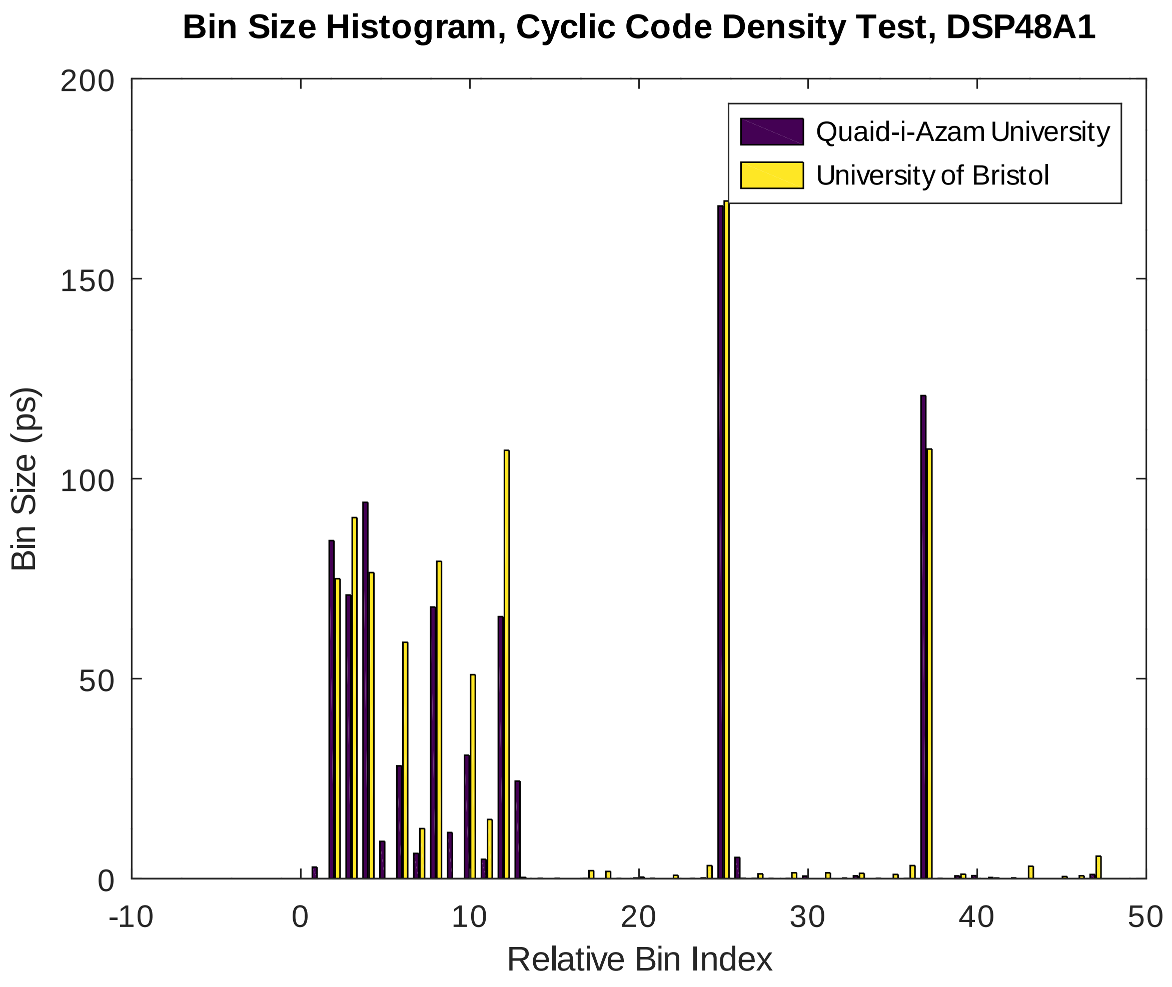


Figure 4.14: A histogram of a 384k-hit cyclic code density test, converted to bin sizes. Results from the Univeristy of Bristol and Quaid-i-Azam University are shown in parallel.

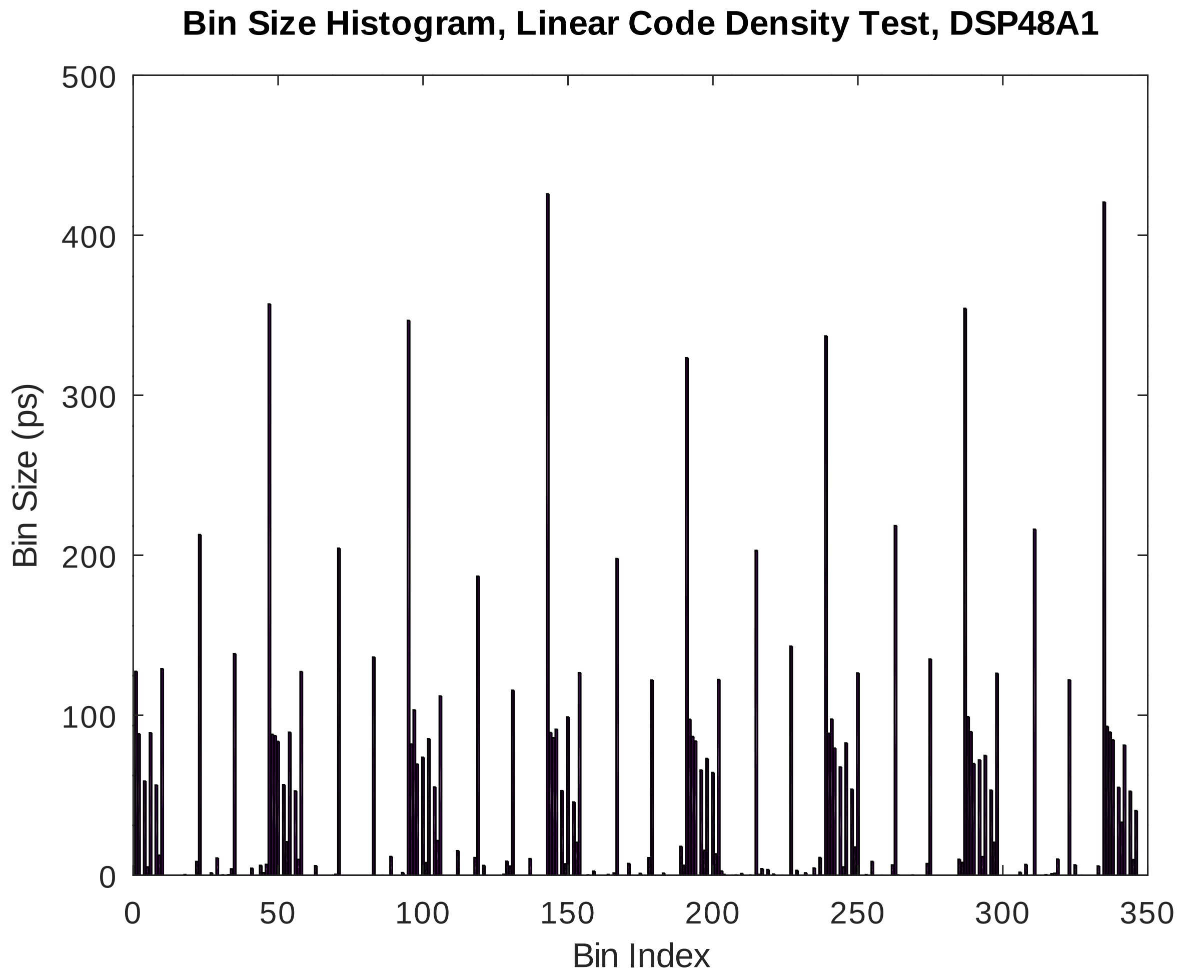


Figure 4.15: A histogram of the 384k-hit linear code density test, converted to bin sizes.

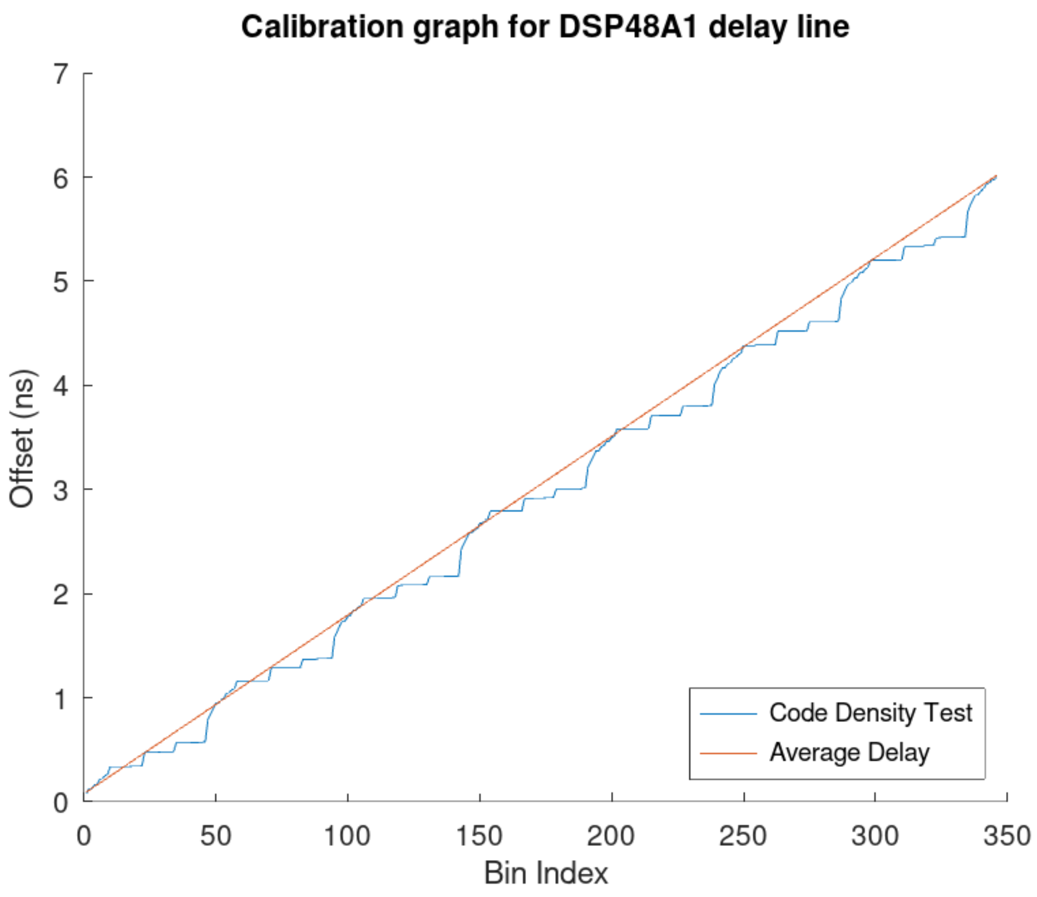


Figure 4.16: The calibration chart obtained from a histogram of a 384k-hit linear code density test.

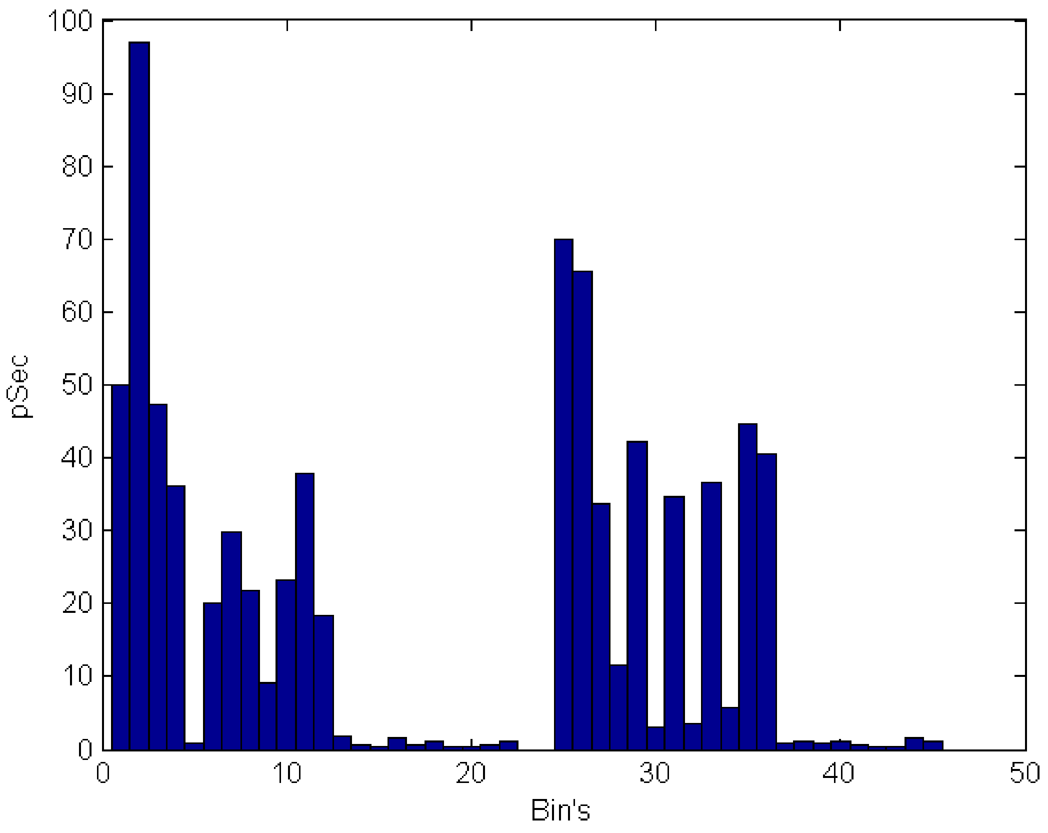


Figure 4.17: The histogram of the subdivided test.

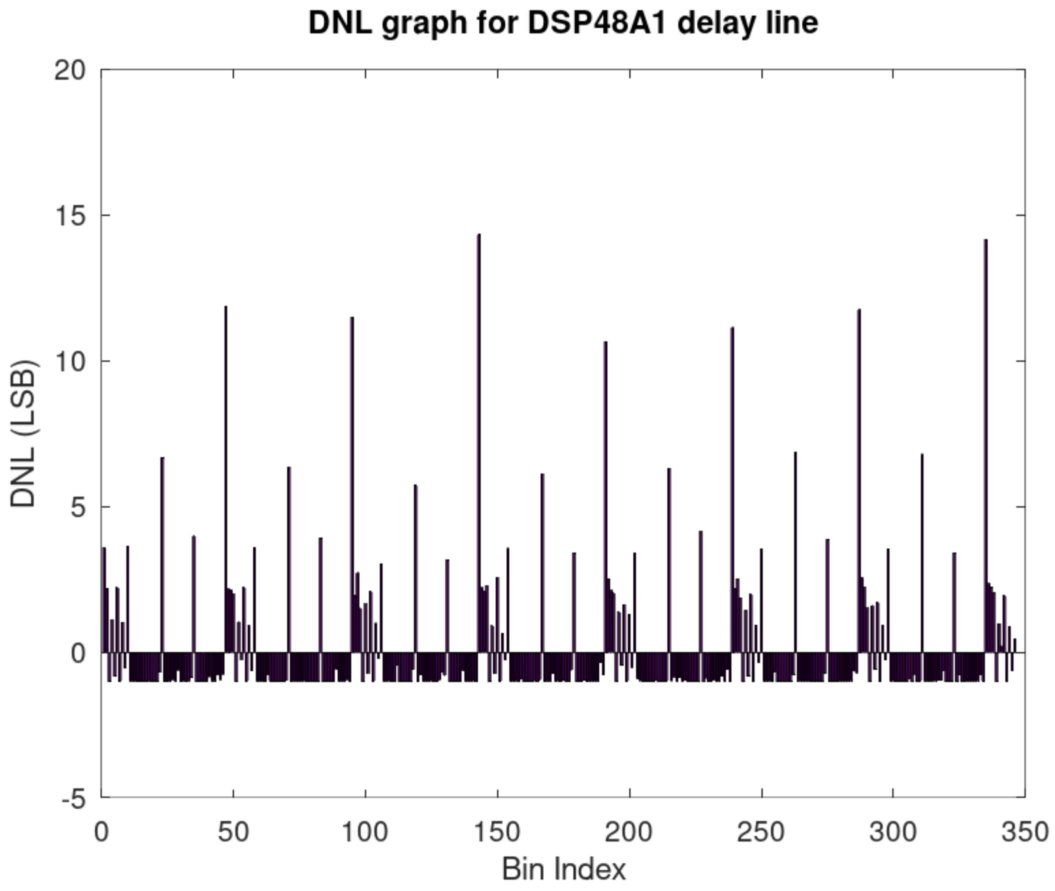


Figure 4.18: DNL graph for a DSP48A1 delay line.

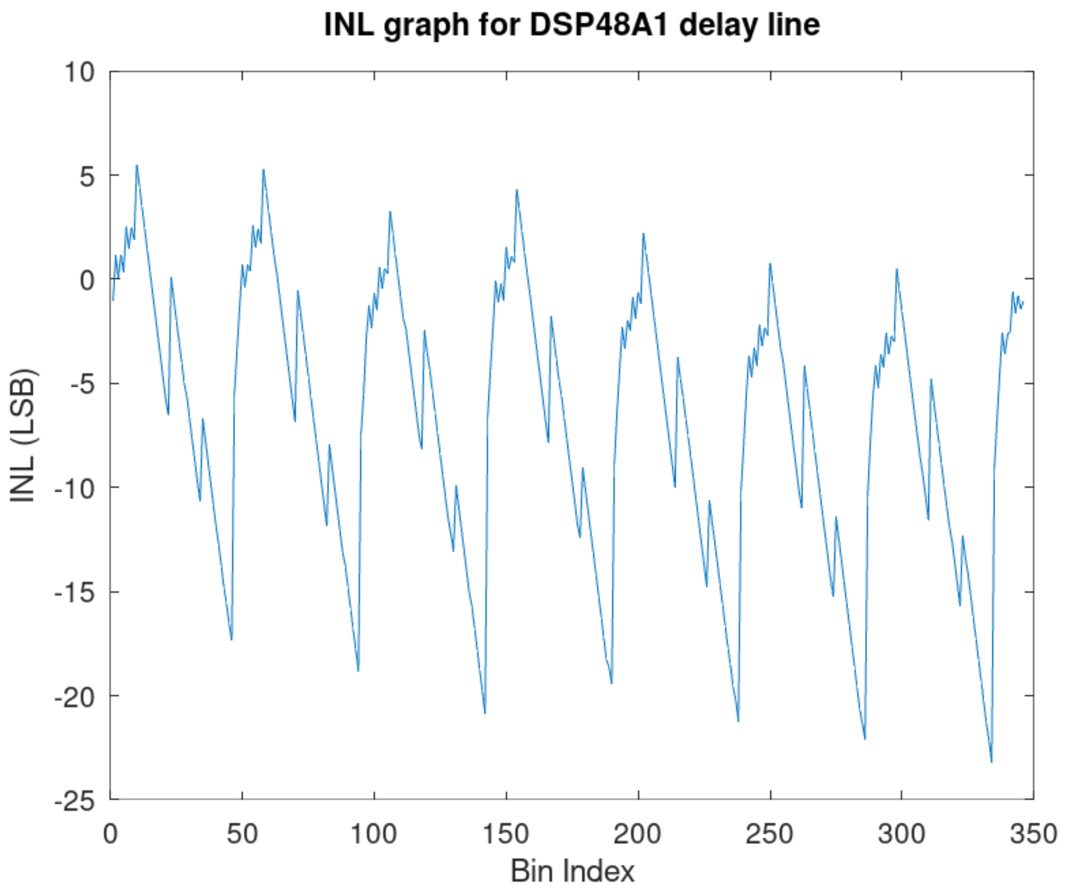


Figure 4.19: INL graph for a DSP48A1 delay line.

4.3.6 Discussion

4.3.6.1 Average Delay Testing

The average delay testing showed that the signals that emerged from the output pins of the FPGA had a difference of 800 ps between them. This suggests that the average delay of the bins is 16.7 ps. This is a 20.5% improvement on the CARRY4 chain, presuming that all bins are in use (non-zero effective width). However, as bins that have a zero effective width do not contribute to the resolution of the system (they are missing bins), the more missing bins we discover in the code density test, the lower the effective resolution.

Parameter	**CARRY4**	**DSP48A1**
Total Delay	80 ps	800 ps
Quantity of Taps	4	48
Average Tap Delay	20 ps	16.7 ps
DNL	2 LSB	15 LSB
INL	22 LSB	23 LSB

Table 4.2: Parameters of CARRY4 and DSP48A1 blocks.

4.3.6.2 Code Density Testing

As can be seen from the code density tests in Figs. 4.14 and 4.16, a single DSP48A1 does not create effective delay lines. While the average bin size per DSP is small (16.7 ps from the average delay testing), the bin sizes are very widely skewed, with most bins exhibiting an effective width of 0 ps (or so small that they could not be measured). Meanwhile, the minority of bins exhibited large delays in the range of 40-170 ps and the few bins at the boundary of a DSP block exhibited delays between 200 and 350 ps. As the missing bins (zero effective width) do not contribute to the resolution of the system, the actual resolution is 34.8 ps with a standard deviation of 51.3 ps.

This is a significantly worse resolution (larger time difference) than has been achieved by the carry chains ($(\mu,\sigma) = (34ps, 51ps)$ compared with $(21ps, 25ps)$). As a result, we can conclude that these devices are unsuitable for generating high-resolution delays (i.e., acting as the final stage in a multi-stage TDC) in isolation. However, as they do generate a significant delay, they may be useful as a semi-fine stage in a multi-stage TDC, thereby reducing the required length of the fine delay line (and hence reducing the overall logic utilisation).

Alternatively, multiple DSP blocks could be utilised in a parallel staggered fashion, allowing delay lines to sub-divide each other in an equivalent coding line. As the resolution increases with the square root of the number of delay lines, 2.62 (i.e., 3) DSP48A1 delay lines would be needed to obtain the same resolution as a CARRY4 chain. The rest of this section will be dedicated to examining the cause of the observed delays.

First, we observe the large bins at multiples of 48 in the linear test. A comparatively large bin in a structure usually denotes some form of discontinuity in the logic structure, and in this case the discontinuity is the crossing between DSP48A1 blocks. Each DSP48A1 block provides 48 time bins, and then a carry out which is directly connected to the next DSP block. While this direct connection is fast compared to most FPGA routes, it is still significantly slower than the routes inside the DSP block as the signal must travel through three configuration multiplexers between the post-adders of each DSP block.

Second, we notice that the bins which are a multiple of 96 (even multiples of 48) are larger than the odd multiples of 48. Unlike the odd multiples, the even multiples of 48 not only need to propagate between two DSP48A1 blocks, but also across a clocking discontinuity. One in every four DSP interconnects is on a clock management tile (CMT) boundary, while another is bridging the CMT's dedicated clock routing area. These jumps cause the multiples of 96 to have larger bins than the odd multiples of 48.

Beyond this, we notice that bins which are an odd multiple of 24 are also relatively large. The available documentation from Xilinx gives no indication as to the cause of this, and so it can only be presumed that there is some discontinuity in the internal addition logic present in the DSP48A1. An example of this would be carry look-ahead logic, which in a 48-bit adder would forward the carry signal to multiples of 24.

Under the split test, we see that the presence of the bins after bin 24 becomes much more pronounced. This adds credibility to the suggestion that the missing bins are caused by carry look-ahead logic is bypassing bins to increase adding speed, but at the expense of delay line differential non-linearity.

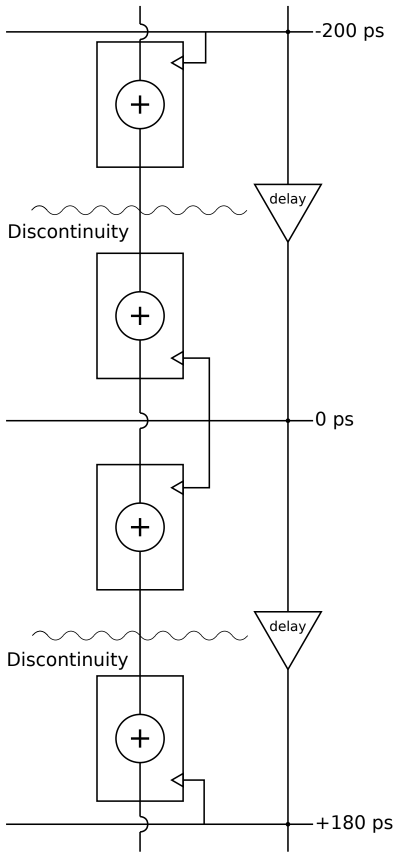


Figure 4.20: A diagram to show how clock routing causes larger delays every 96 bins.

4.3.7 Conclusion

In conclusion, we have found that the post-adder in a DSP48A1 block can be used to generate delays with an average length of 34.8 ps and a standard deviation of 51.3 ps. This is a lower accuracy than the competing CARRY4 chain solutions, but this could be remedied by using multiple staggered DSP48A1 blocks in parallel. The larger delay generated by the DSP48A1 blocks was found to be suitable as a semi-fine delay generation as would be required in a multi-stage TDC. We discovered this using average delay calculation and code density testing. We also found that the INL is comparable to a carry chain implementation ~23 LSB, while the DNL is significantly worse ~15 LSB compared to ~2 LSB.

4.4 Solving the non-linearity

4.4.1 System Design

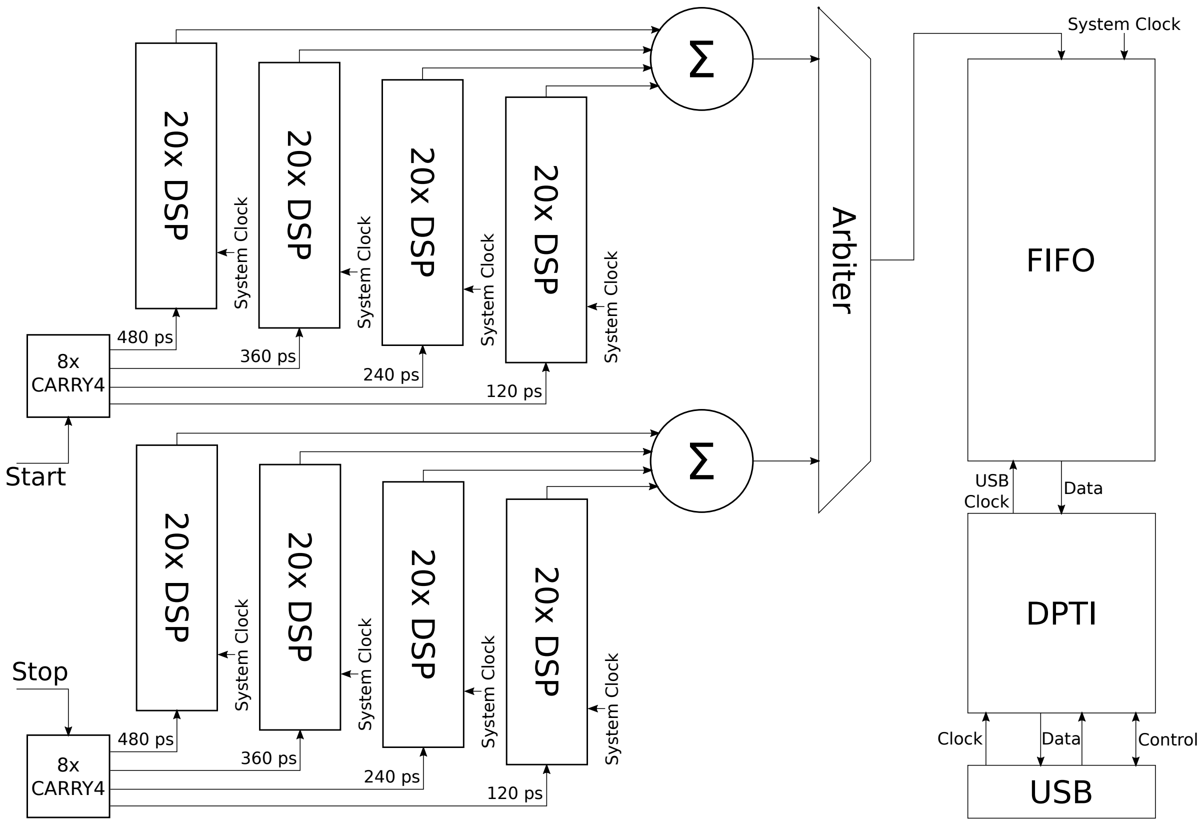


Figure 4.21: Block diagram showing the architecture of the parallel system.

4.4.1.1 Singular DSP

As our aim was to turn the 48-bit post-adder in the DSP48E1 into a delay line, we needed to examine its operation. The post-adder has three major inputs, the 48-bit X, Y and Z multiplexers (MUXes), as well as 1 minor input (the carry input), one major output (the P output), 1 minor output (the carry out) and two control inputs, the operation mode and arithmetic logic unit (ALU) mode. In addition, there are several configuration registers to enable or bypass various pipeline registers in the DSP48E1.

The operation of the DSP's post-adder (Fig. 4.22) is to take the outputs of the X, Y and Z multiplexers (which are fed by external inputs, constants, loopback paths or other logic components such as the multiplier) and add them together with the carry in (CIN) to produce a 49-bit output, with the upper bit being the carry out (COUT) and the lower 48 bits being the P output of the DSP block. Both the Z input and the P/COUT output can be inverted to enable addition and subtraction while still retaining carry functionality.

Unlike the DSP48A1, the Y mux has an option to use a hard-coded 48 binary ones as the input, so we decided to use this rather than manually inputting the same pattern. Similarly, for the trigger, the DSP48E1 adds to the DSP48A1 by including a dedicated carry input from the general-purpose FPGA fabric, so we were able to use this for the least-significant DSP's carry input and have all DSPs with the X and Z inputting all zeros, again through a hard-wired input to the mux.

The X, Y and Z multiplexers are configured using the OPMODE configuration input. In the basic propagating configuration, the Y multiplexer uses bits two and three of OPMODE and is always configured to forward the value 0xFFFF FFFF FFFF to the post-adder. The X multiplexer uses bits zero and one of OPMODE and is configured to introduce zero to the post-adder. The Z multiplexer is set to use zero for a standard delay line. The carry in is connected to the previous DSP block in most cases, but to the trigger signal (start / stop) in the case of the first DSP.

Once the trigger signal is introduced to the delay line, the edge(s) propagate through the internals of the DSP, causing bits to swap from zero to one or one to zero. These bits are sent to the P output, where they are then captured by flip-flops on the FPGA's general-purpose fabric at the positive edge of the clock.

As with the DSP48A1, all pipeline registers were disabled except for the P output register. In the DSP48A1, the carry cascade output was connected to a different register to the P output, so the P output register could be used as a discriminator while the carry register was bypassed. In the DSP48E1, the carry output and P output both go through the same register (see Figure 4.22) so the register must be disabled to allow the carry to propagate to the next DSP asynchronously. Therefore, external fabric flip-flops were required to register the output.

In some other designs [154] (published after [11]), the carry signal propagates through the pre-adder cascade (ACIN/ACOUT or BCIN/BCOUT) which allows for the DSP's P register to be used. However, we are using the post-adder cascade (CARRYCASCIN/CARRYCASCOUT) in our

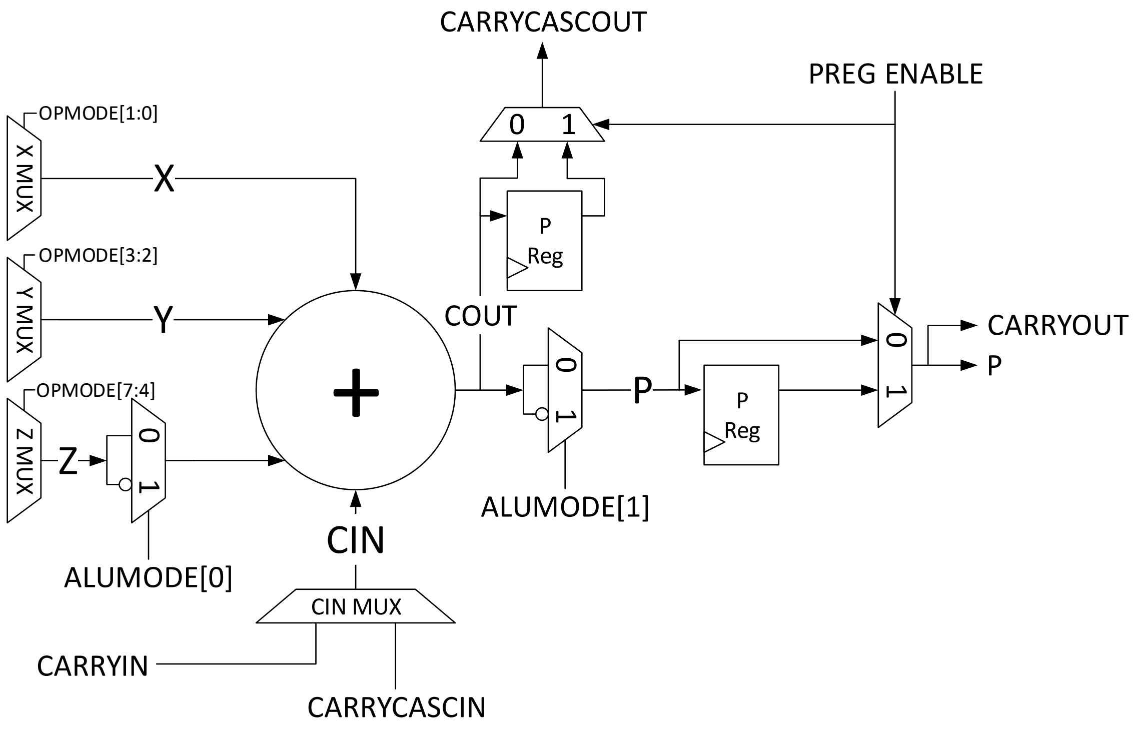


Figure 4.22: DSP add/sub operation, showing the purpose of the ALUMODE[1:0] and OPMODE[7:0] inputs, based on [4].

design, which excludes the use of the P register in order to operate an asynchronous carry path (enabling the P register would interrupt carry propagation with a register synchronised to the system clock).

4.4.1.2 Multiple DSP and TDC Design

Once we had determined the set-up for a single DSP, we then sequenced 20 of them to be sure we would cover the whole clock period, resulting in a maximum of 960 bins. These bins were registered by two levels of flip-flops on nearby FPGA fabric (to eliminate metastability), and then inverted and consumed by a priority encoder, which determines the most significant position of a one (a zero before inversion) to see how far the delay propagated. These most significant positions were then output to a PC via a parallel interface (Digilent's DPTI) for histogramming.

The DSP blocks in the design were configured with a 'head' block which accepted an external carry input and multiple 'tail' blocks which accepted a carry input via the dedicated routing from the previous DSP block. The trigger signal was introduced to the first DSP block as the external carry signal, while all DSPs were configured to calculate ¬(0xFFFF_FFFF_FFFF + Carry). In this configuration, the internal adder starts with all the sum bits equal to one, resulting in an output code of all zeros and an output carry of zero. When the input carry toggles from zero to one, the internal adder starts propagating the new carry bit up the carry chain, resulting in all the internal carries transitioning to one and all the sum bits transitioning to zero, thereby resulting in a successive toggle of output bits from zero to one. Once the carry bit reaches the Most Significant Bit (MSB) of the adder (which may be before it reaches other intermediate bits), the carry leaves the DSP and enters the next DSP via dedicated routing. This is demonstrated in Figure 4.23.

For our delay line of 960 bins (corresponding to 20 DSP blocks of 48 bins each), we need a 10-bit code to fully encode the quantity of ones. This was combined with an eight-bit channel identification code, a two-bit clock identification code (for synchronising multiple DSP delay lines when the trigger is near to a clock edge) and a 44-bit coarse counter to produce a 64-bit output code. This 64-bit output code (produced in the TDC clock domain at 120 MHz, with up to two channels triggering at once, and up to one trigger per channel every four clock cycles) was then passed through several serialisation stages to arbitrate between the two channels, serialise the data stream into bytes, buffer it and switch to a 60 MHz USB clock domain. Once it entered the 60 MHz USB clock domain, it was sent to an attached PC via a USB 2.0 interface (Digilent's DPTI) at $\sim 480 Mb/s$. The data was captured by the PC and post-processed for calibration and characterisation.

4.4.1.3 Ones-Encoder

Due to the disappointing results of the histogram (multiple missing bins, see Section 5), we determined that we needed a way to recover these missing bins. The missing bins occur due to the flip-flop on bin i+1 passing its blackout time before the flip-flop in bin i, meaning the priority encoder does not pick up the transition of bin i since bin $i+1$ is already high.

In effect, the priority encoder is discarding the majority of the bins present due to them not transitioning in-order. Therefore, we switched to an order-invariant encoder (the ones-encoder)

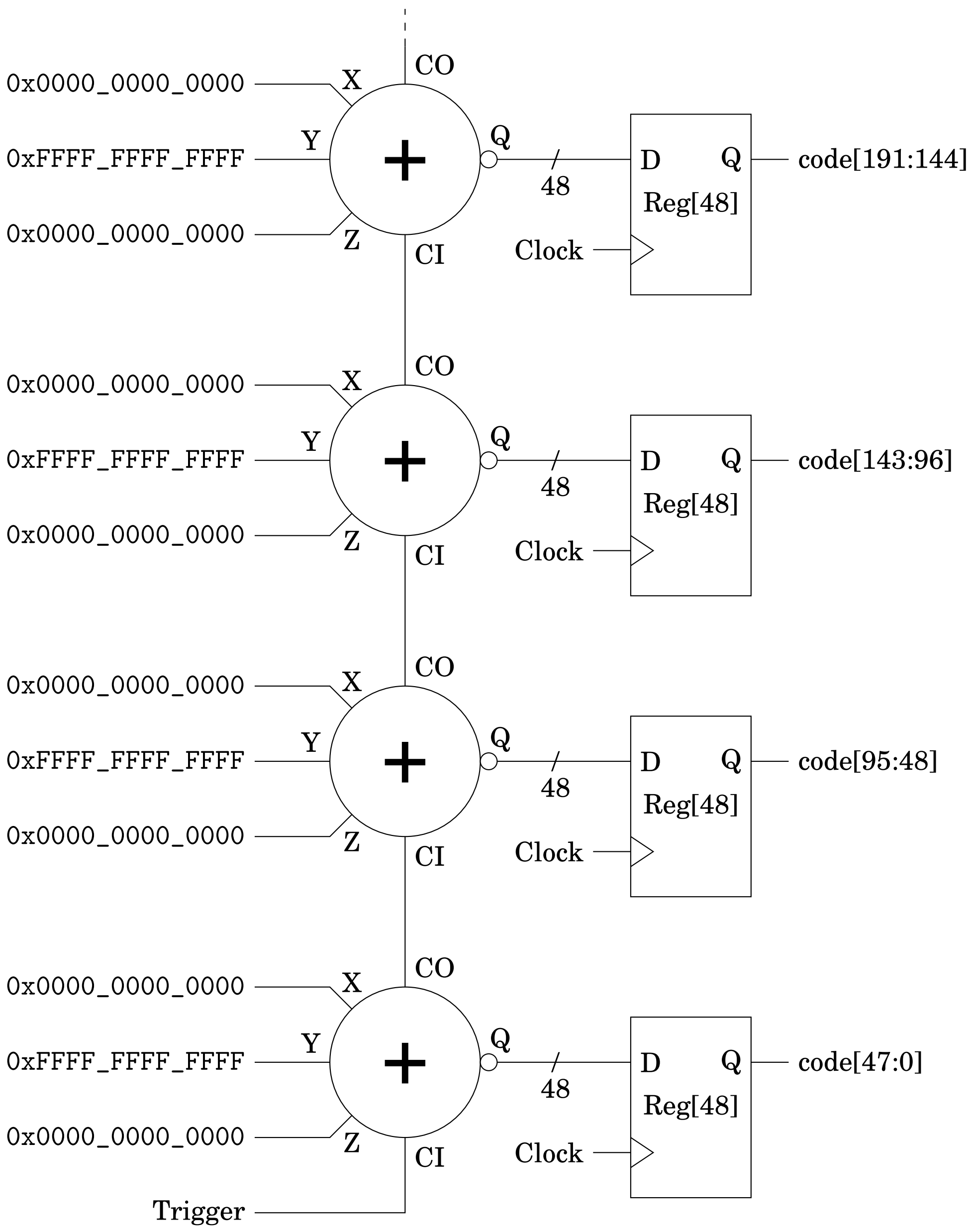


Figure 4.23: Configuration of multiple DSP blocks as delay generators.

to recover these bins. However, the ones-encoder does not retain edge position information, and so is not capable of operating when the on-time of the trigger is shorter than the clock period (a ones-encoder cannot distinguish between `1100 0000` and `0000 0110`, both have a value of '2').

As the number of bins that are high is monotonically increasing, we can instead use a ones-encoder for un-ordered bin detection. With a ones-encoder, even if bin i transitions after bin $i+1$, the two transitions will still give two different outputs, separated by a single LSB. With this, we

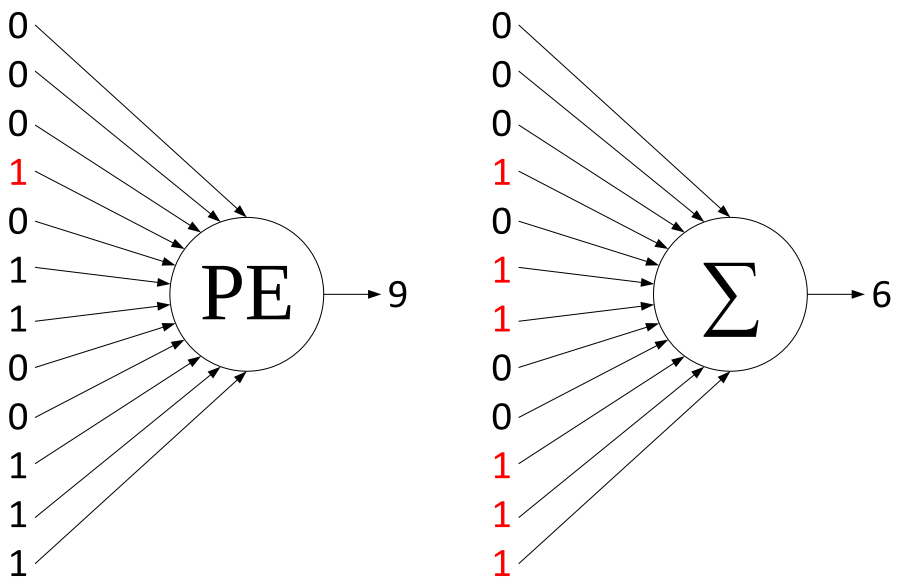


Figure 4.24: (left) A priority encoder which encodes the highest position of a ‘1’ bit and ignores other bits; (right) A ones-encoder which encodes the quantity of ones.

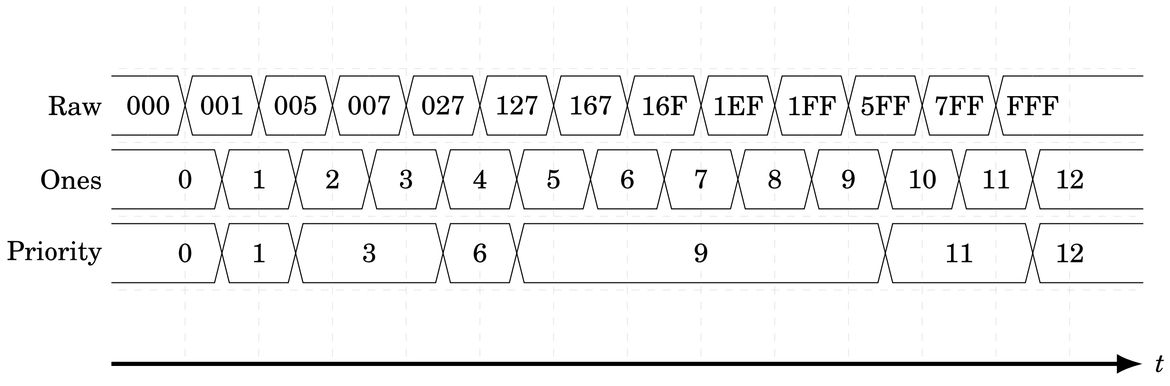


Figure 4.25: A waveform showing how a priority encoder misses some codes while a ones-encoder retains all codes.

were able to recover the missing bins, although these bins were extremely small.

4.4.1.4 Parallel Delay Lines

Although the bins were recovered, there was still a significant difference in size between the last bin in a DSP and the rest. As the sum of the rest of the bins was more than 1/3 of the size of the large bin (170 ps vs 310 ps), it was possible to sub-divide the large bins by creating multiple delay lines with fractions of a DSP as an offset, and then summing the results of these delay lines. As each delay line is monotonically increasing, stepwise, as the time between the trigger and clock increases, the sum of the delay lines must also monotonically increase, with each delay line sub-dividing the others' large bins.

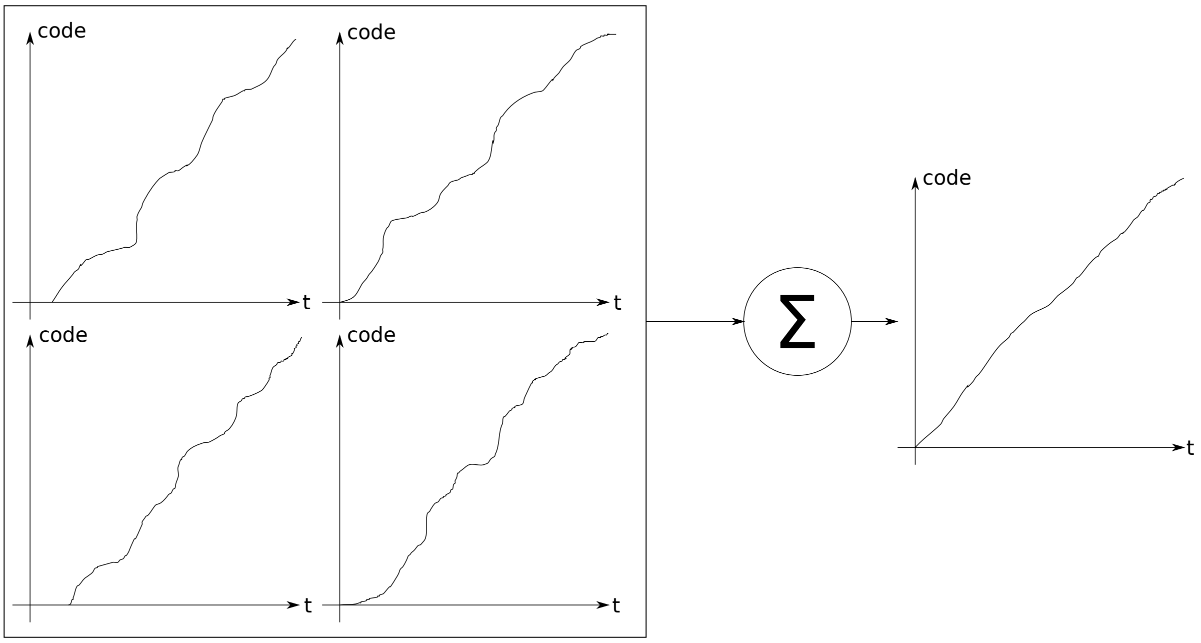


Figure 4.26: When multiple delay lines which monotonically increase are combined (outputs summed), the result also monotonically increases.

Specifically, each delay line has a low-resolution region (the last bin in each DSP) and a high-resolution region (the intermediate bins) per DSP. The aim is to have at least one of the parallel delay lines in its high-resolution region at all times. Therefore, the start of each delay line is successively offset by a fraction of the delay line shorter than the high-resolution region. This ensures that one DSP chain will enter its high resolution region before the previous chain leaves its high-resolution region.

The offsets, being much smaller than a DSP on its own, were provided by CARRY4 blocks. As

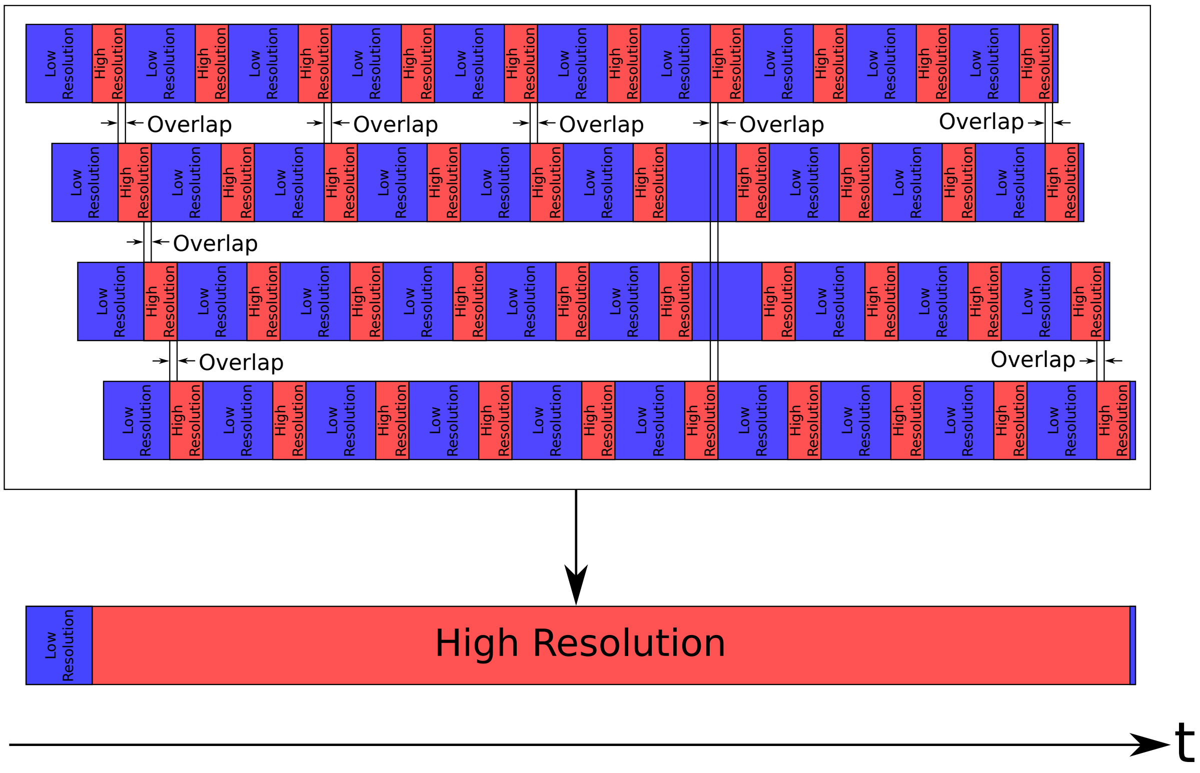


Figure 4.27: A block diagram showing how the high resolution regions of the DSP blocks overlap in the time domain to create an equivalent coding line with high resolution along its length (with the exception of the initial region, which can be removed by the triggering logic).

each CARRY4 block is, on average, 65 ps and the delay of a DSP block is 550 ps, two CARRY4 blocks are sufficient to provide the first delay, with subsequent delays successively adding pairs of CARRY4 blocks to generate successive offsets.

4.4.2 Methodology

For the DSP48E1, we used a linear code density test to determine the delay of each component. The results of the code density test were then histogrammed per-DSP and per-bin to determine the overall delay of each DSP block in the chain as well as the bin distribution within a DSP. We used a 1 MHz external pulse generator to generate the rising edges (start signals) asynchronously to the system clock (stop signal) to obtain an even distribution of delays with respect to the system clock, which had a frequency of 120 MHz. Read-out was performed live for 1,041,043 tags and histogramming performed off-line.

The input signal was a single 1 MHz, 3.3 V (0 V - 3.3 V) square wave produced by a BK Precision 4054B 30 MHz function generator. To ensure a stable delay between the start and stop channels, measuring only the variability caused by the TDC core, ther signal was split on-chip to feed the two delay lines. As the clock of the function generator is asynchronous to the TDC core (supplied by a 120 MHz Phase-Locked Loop), this 100 kHz input is effectively random over a period sufficient to ensure the clocks drift in and out of phase. These tags were collected in the normal mode of operation, with all tags used for calibration and pairs of tags with equivalent coarse counts used for testing (to remove any clock jitter from the measurement). A diagram of this can be seen in Figure 4.36 and a picture of this in Figure 4.29.

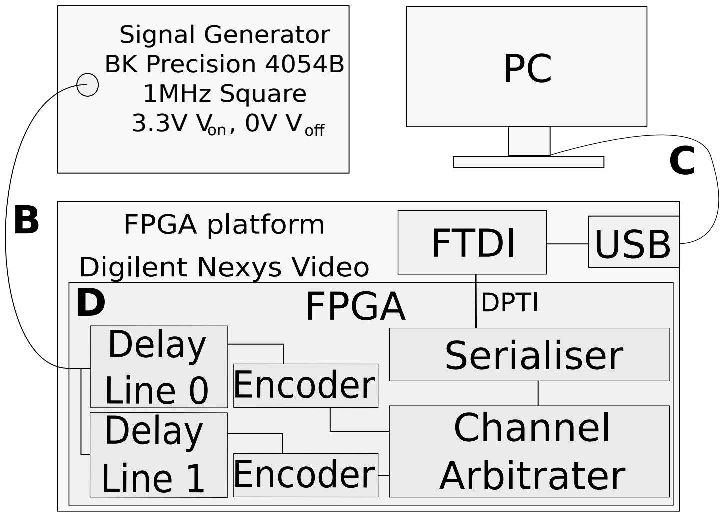
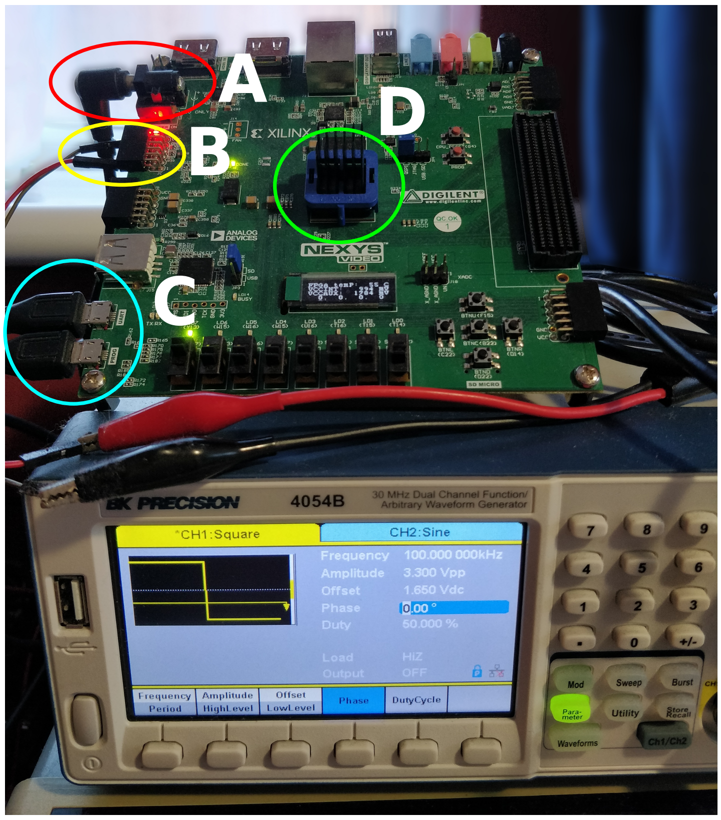


Figure 4.28: Block diagram of test setup.

4.4.3 Results

4.4.3.1 Per-DSP

Fig. 4.30 shows the time bins on a per-DSP basis. When discarding outliers, we observe a 553 ps mean delay with 74 ps standard deviation. DSP 0 is an outlier that describes the path mismatch between the trigger / coarse counter and the fine counter. DSPs with indexes greater than 14 are completely outside the clock period and only occur due to metastability in the triggering logic, while DSP 14 is only partially (25%) within the clock period and so can be considered an outlier.



Frequency 100.000 000kHz
Amplitude 3.300 Vpp
Offset 1.650 Vdc
Phase 0.00 °
Duty 50.000 %
Load HiZ
Output OFF

Figure 4.29: Block diagram of test setup.

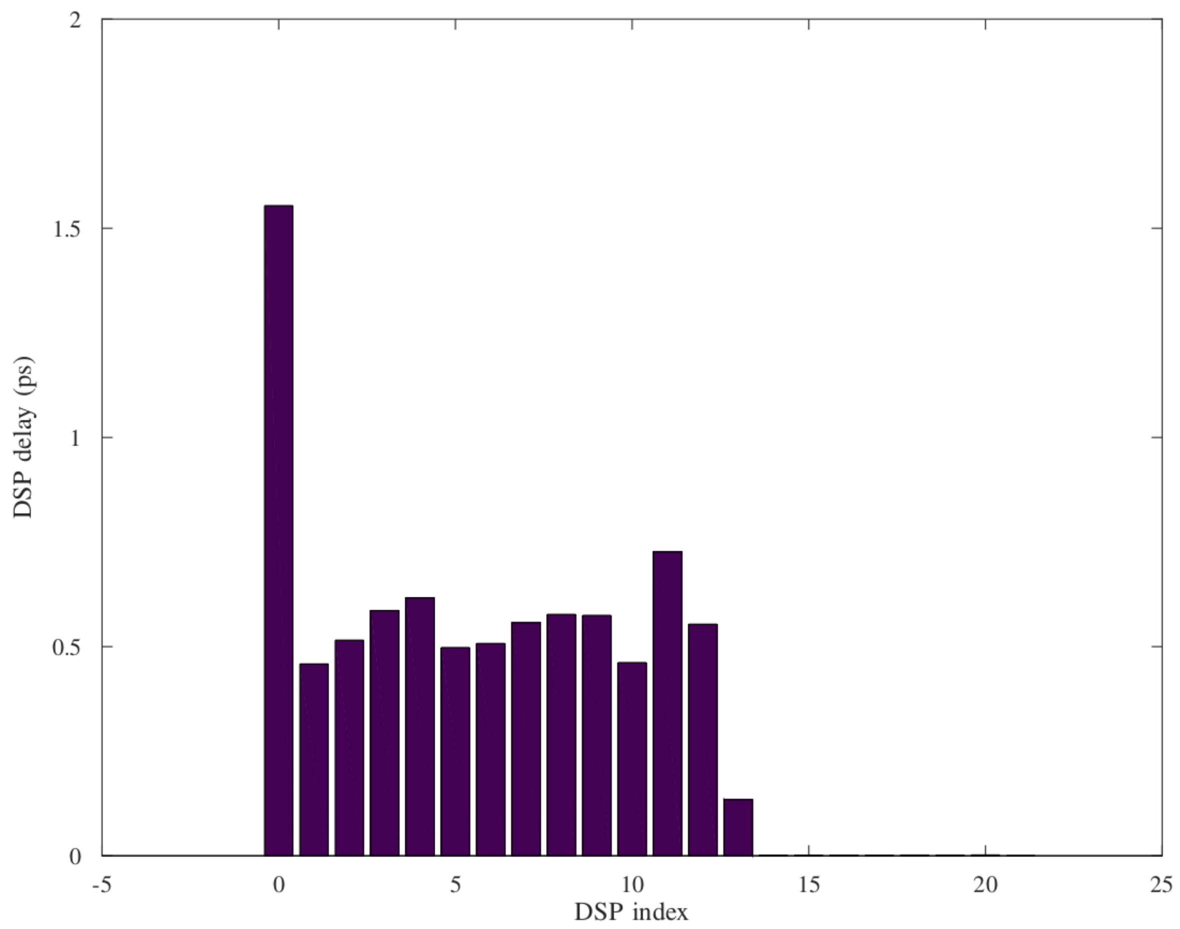


Figure 4.30: Delays of the DSP blocks.

4.4.3.2 Per-bin

Fig. 4.31 shows the time bins on a per-bin basis, averaged across all non-outlier DSPs. We see that most of the delay is concentrated in the final bin (bin 47), while the rest is scattered across the bins approximately evenly. However, due to the high final bin, the mean is 11.5 ps and the standard deviation 45.1 ps (would be 5.17 ps and 9.72 ps respectively if not for the final bin).

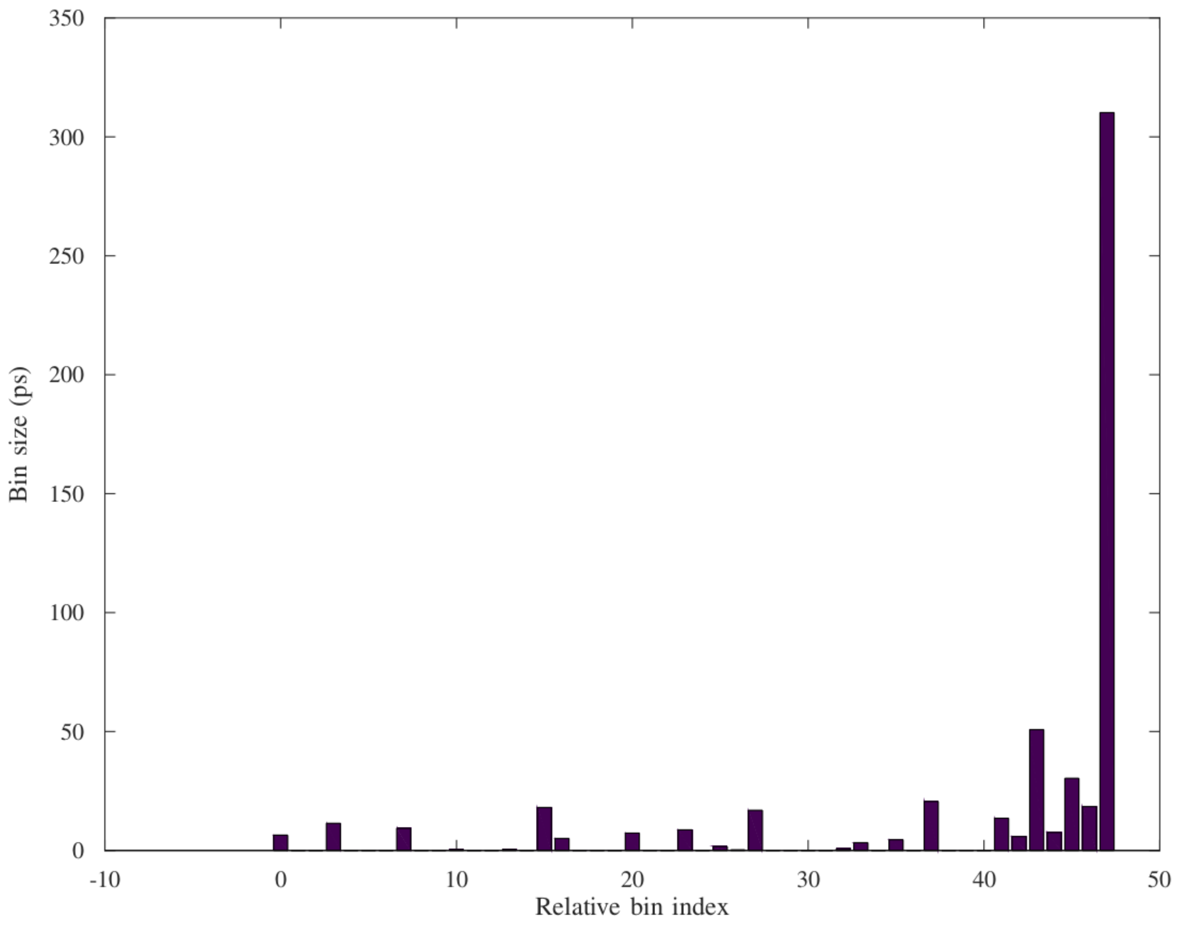


Figure 4.31: Average delay within a DSP block.

4.4.3.3 Overall

Fig. 4.32 shows the time bins across the DSP blocks. All the large peaks occur in the last bin (47) of a DSP, demonstrating the repeatability of the results across multiple DSPs.

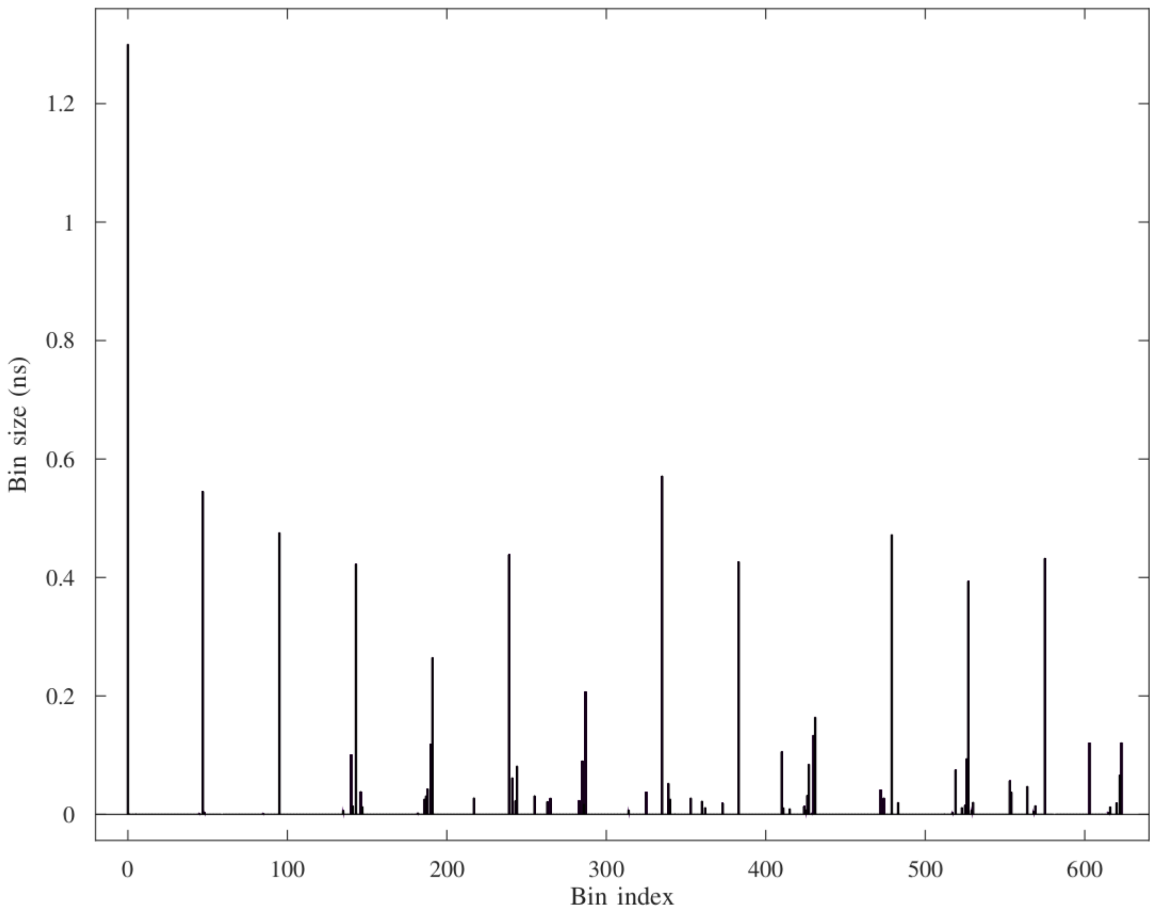


Figure 4.32: Linear code density test.

4.4.3.4 With Ones-Encoder

Fig. 4.33 shows the whole-system histogram when the ones-encoder is used with a single delay line. Unlike the results shown in Fig. 4.31, it shows that each bin other than the large one at the end has some hits in it, corresponding to approximately 5.21 ps per bin. The large final bin still exists.

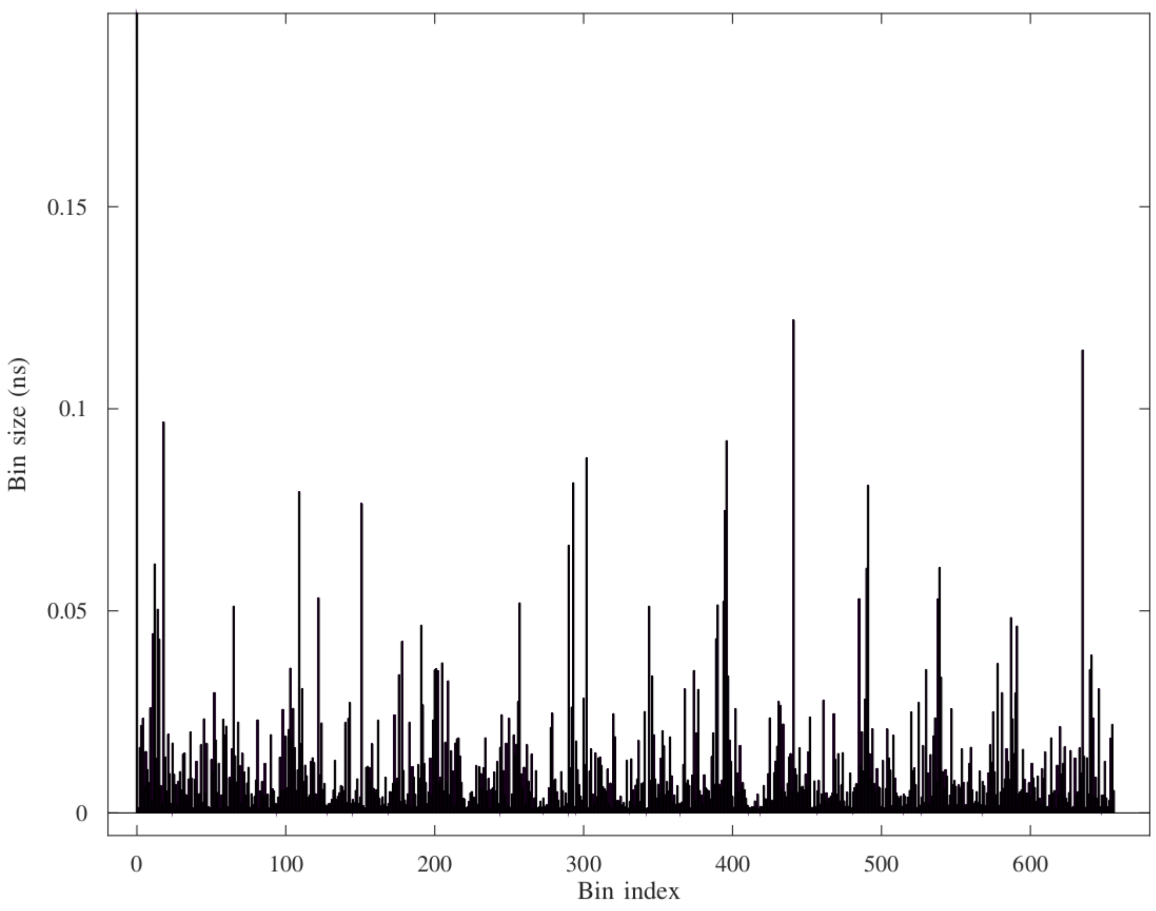


Figure 4.33: Linear code density test with ones-encoder. Bin 0 is 447ps.

4.4.3.5 Parallel Summed Delay Lines

Fig. 4.34 shows the whole-system histogram when the ones-encoder outputs of four offset delay lines are summed together. Worthy of note in this diagram is that the maximum bin number is quadruple that of a single delay line (as we are summing four delay lines) and that the large bins present in the previous tests no longer exist. This shows that offsetting the delay lines to overlap the high-resolution regions with the low resolution regions and then creating an equivalent coding line was effective in improving the overall resolution to match that of the high-resolution regions, which was 5.21 ps.

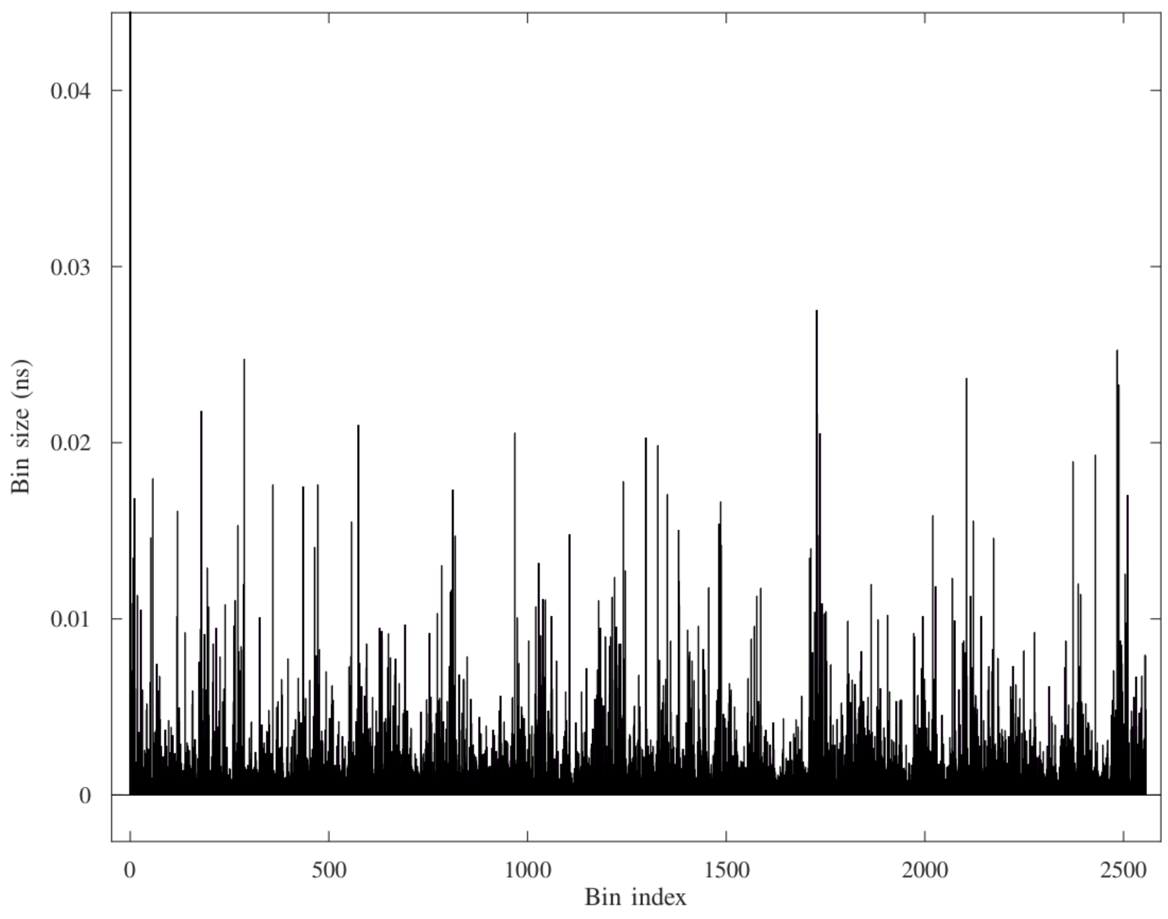


Figure 4.34: Code density test with four offset delay lines. Bins 0 and 1 are 710ps and 106ps respectively.

4.4.4 Discussion

Similarly to the results seen on the DSP48A1 blocks, the DSP48E1 blocks suffer from both missing bins and large non-linearities when used with a priority encoder. As we can be sure that the outputs are monotonically increasing, we can instead use a ones-encoder (see Section 4.4.1.3), and this removes the issue of missing bins. This creates a delay line with a large delay of (on average) 308 ps, with the other 245 ps rest of the delay split between 47 bins for an average of 5.21 ps per bin.

To sub-divide these large bins, an equivalent coding line [37] can be created by summing together the outputs of multiple DSP delay lines. As the large bin is many times larger than the average bin or the standard deviation, we cannot rely on natural drift due to process variation, and so must insert delays to offset the DSP delay lines relative to each other. This is done by inserting varying numbers of CARRY4 chains before the delay lines to form the correct delay duration.

Once four DSP delay lines have been successively offset by a quarter of the duration of a delay line each, their outputs are summed to form an equivalent coding line (ECL) of 2557 bins length. This ECL has a max bin size of 27.04 ps and a cubic mean bin size of 5.25 ps, with the first two bins discarded due to system offsets (which can be compensated for). 5.25 ps is an acceptable resolution for the quantity of logic (9% of the system's DSP blocks) required to implement it and so this can be considered a successful implementation. Further testing revealed that the system's maximum bin size can be dropped to 22.35 ps with a cubic mean of 3.70 ps using 18% of the system's DSP blocks.

While there is no documentation on the internals of the DSP block, the order and pattern present in the order of bins transitioning suggests a three-level carry-lookahead adder is present. Fig. 4.35 shows the presumed internals of the DSP48E1 post-adder, with modified full-adders (FA) that generate the sum ($s_i | i = [0..47]$), propagate ($p_i | i = [0..47]$) and generate ($g_i | i = [0..47]$) bits from the two (or three) data inputs ($x_i, y_i, z_i | i = [0..47]$) and carry in ($c_i | i = [0..47]$). The first-level carry propagators (second column) control carry propagation between groups of four full-adders, the second-level carry propagators (third column) control blocks of four first-level carry-propagators (and thus 16 full-adders), and finally a third-level carry propagator (fourth column) controls three second-level carry propagators and generates the output carry. Due to the hierarchical nature of the design, the delay from the carry input to some blocks in the middle may be significantly longer than the delay from the carry input to the carry output (or other blocks later in the chain). This in turn produces the out-of-order transitioning observed in the output of a DSP block.

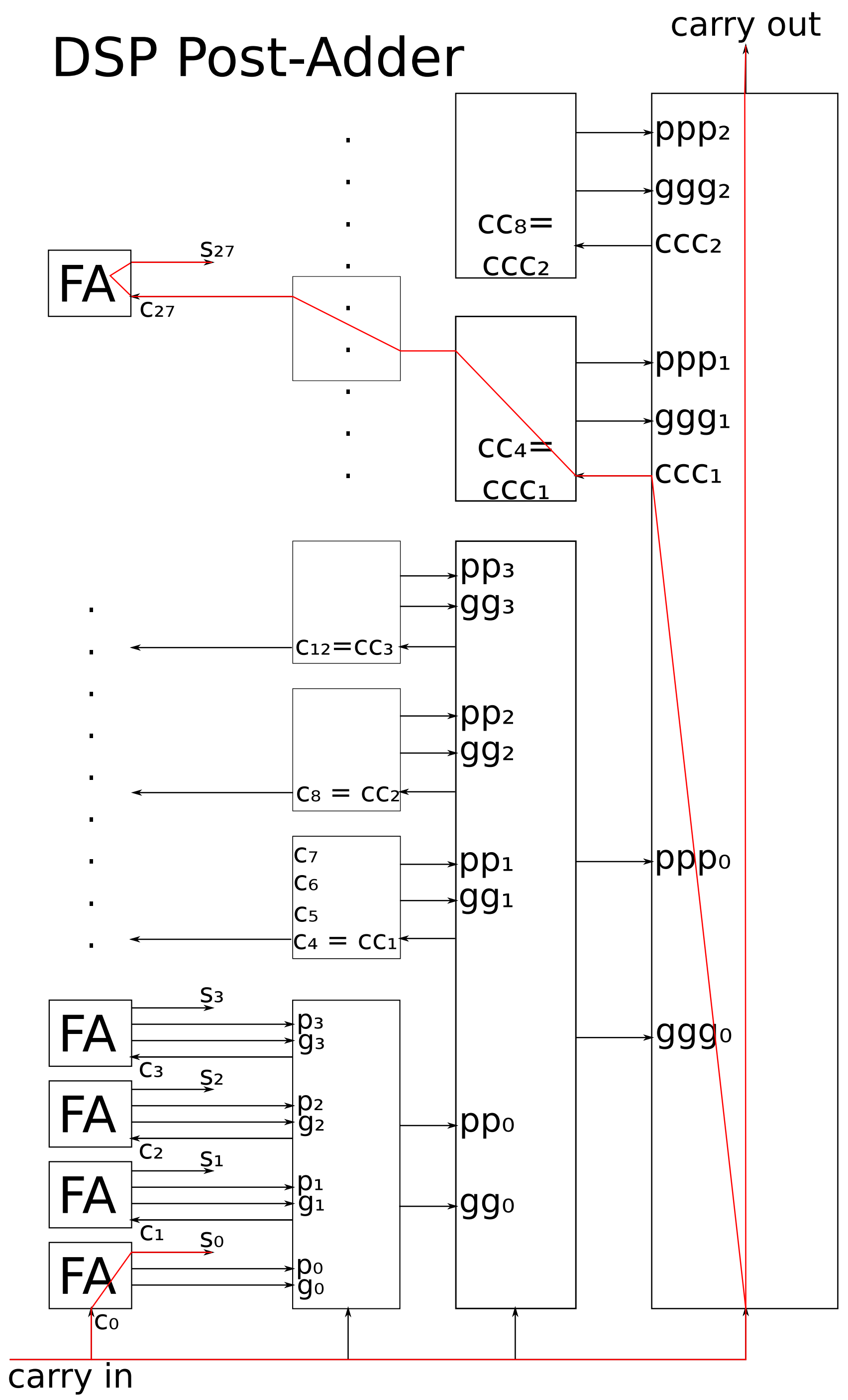


Figure 4.35: Internal logic of a DSP48E1 post-adder, demonstrating the out-of-order timings. FA: full adder; s_i: sum output i; c_i: carry signal i; g_i carry-generate signal i; p_i: carry-propagate signal i.

Carry Look-Ahead Operation

The operation of the carry lookahead (CLA) structure is defined by the propagation and generation signals. The propagate (p) signal is set high when a bit of the adder would propagate a carry bit from input to output (the sum of the input bits is one, so a carry in would cause a carry out); the generate (g) signal is set high when a bit of the adder generates a carry bit (the sum of the input bits is greater than one so the carry bit is always active), and neither signal is high when a bit of the adder would delete a carry (the sum of the input bits is zero, so a carry is absorbed and the carry out is always zero).

The CLA structure takes the propagate signals, generate signals, and carry in as its inputs, then generates intermediate carry signals, a propagate signal and a generate signal as its outputs. The intermediate carry signal at position i is calculated as (4.5), where S is the look-ahead factor (with a value of four in this example). This equation is often unrolled to reduce the length of the critical path (e.g., as in (4.6)). The output propagate signal pp_j is defined as the logical AND of all the propagate bits (4.7), while the generate is defined as the presence of a generate in the inputs which is propagated to the output (4.8).

$$c_i = \begin{cases} g_{i-1} \vee (p_{i-1} \wedge c_{i-1}), & i \bmod S \neq 0 \\ c_i = c_{in}, & i \bmod S = 0 \end{cases} \tag{4.5}$$

$$\begin{aligned} c_3 = {} & g_2 \vee (g_1 \wedge p_2) \\ & \vee (g_0 \wedge p_1 \wedge p_2) \\ & \vee (c_{in} \wedge p_0 \wedge p_1 \wedge p_2) \end{aligned} \tag{4.6}$$

$$pp_j = \bigwedge_{k=0}^{S-1} p_{S\times j+k} \tag{4.7}$$

$$gg_i = \bigvee_{k=0}^{S-1} \left(g_{S\times j+k} \wedge \bigwedge_{m=k+1}^{S-1} p_{S\times j+m} \right) \tag{4.8}$$

$$cc_2 = gg_1 \vee (gg_0 \wedge pp_1) \vee (c_{in} \wedge pp_0 \wedge pp_1) \tag{4.9}$$

Higher levels of the CLA structure operate the same as the lower levels. For example, the cc_1 signal is calculated as in (4.9).

4.4.5 Conclusion

In conclusion, the DSP blocks present in the Artix-7 Series FPGAs are highly non-linear (more than half the delay is contained within a single bin) and suffer from out-of-order bins (which show up as missing bins with a priority encoder). By introducing a ones-encoder to convert the output code, the out-of-order bins were rectified, and by offsetting four parallel delay lines (9% DSP resources), the large bins were sub-divided through the equivalent coding line method. This produced a 5.25 ps bin resolution with 27.04 ps max bin size and doubling the number of parallel delay lines to eight (18% DSP resources) produced 3.70 ps resolution with a 22.35 ps maximum bin size.

4.5 Temperature Characterisation

4.5.1 System Design and Methodology

As in the previous sections, a DSP delay line was configured on a Xilinx Artix 7 200t FPGA (Digilent Nexys Video development board) to propagate a signal through the post-adder operating without the P register enabled. This allows the carry signal to propagate between DSP blocks asynchronously, with the output of the adder being sampled by attached registers on the positive edge of the system clock. The number of bins which have transitioned from a zero to a one will indicate how far the carry bit propagated through the post-adder chain in the DSP blocks, which can then be related to the time between the trigger signal and the edge of the system clock.

By determining the time between the edges and the system clock, as well as counting system clock periods, we can determine the difference in time between two trigger signals. Conversely, if we know the time between two trigger signals transitioning, then we can determine the variability of the measurement and therefore the precision of the device. Given some pairs of measurements from two channels (channel one and channel two, channel numbers are arbitrary) which relate to a known time difference between two channels, the channel offset $\overline{T_{2-1}}$ and channel-to-channel variance $Var(T_{2-1})$ can be calculated. The SSP is defined as the standard deviation of measurements on a single channel, which can be approximated as $\sqrt{Var(T_{2-1})/2} = \sigma_{2-1}/\sqrt{2}$.

The input signal was a single 100 kHz, 3.3 V (0 V - 3.3 V) square wave produced by a

BK Precision 4054B 30 MHz function generator. The clock of the function generator is again asynchronous to the TDC core (supplied by a 100 MHz MEMS oscillator). 261865 tags were collected, ~130932 from each channel. A diagram of this can be seen in Fig. 4.36.

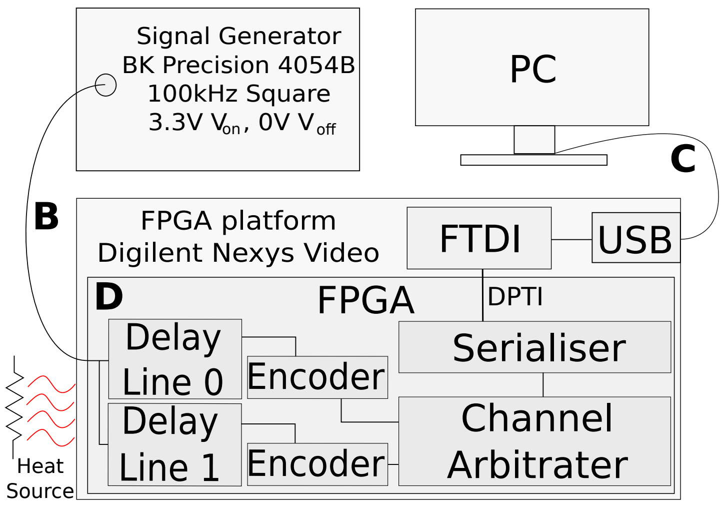


Figure 4.36: Block diagram of test setup.

Temperature characterisation was performed by correlating the measured temperature on-chip with the calibration results once the chip had reached equilibrium. The equilibrium point was adjusted by influencing the ambient temperature and supply of air via a variety of resistive heating elements and axial fans. The on-chip temperature diode, available via the XADC system monitor [155], was used to determine chip temperature. Once the chip had reached an equilibrium temperature while data was being collected (since data collection increased chip power and thus influenced the equilibrium point), the data to be used for calibration and characterisation was then collected and stored to the PC. Once the data was stored, the source of ambient heat/air was then adjusted to a new value and the system left to reach equilibrium again.

Temperature monitoring was performed by implementing a Xilinx Microblaze soft-core on the FPGA fabric (away from the TDC core to avoid interference with the delay line) and connecting this to the XADC system monitor and a UART transceiver via an AXI4 bus. Xilinx's Vitis SDK was used to write code for the Microblaze core and load binaries onto the core, and PuTTY was used to monitor the serial output for the temperature result. The code used was a minor modification to the XADC polled printf example (removing any unneeded components, looping the code, and introducing a 0.5 s delay between samples) provided by Xilinx.

With the various ambient heat and air sources, the measured temperature equilibriums were:

1. 32.7° C (source A, no heating).
2. 40.8° C (ambient / no source).
3. 46.3° C (source A, temperature 1).
4. 60.2° C (source A, temperature 2).
5. 81.8° C (source B, temperature 1).
6. 101° C (source B, temperature 2).

It is noted that the uppermost temperature is beyond the rated temperature of this device (0° C to 85° C for commercial-grade devices). This is performed intentionally in the knowledge that very few manufacturers use different designs for their high-temperature-rated chips, and instead simply bin existing silicon into groups that are either capable or not capable of operating within specification at those temperatures. Some manufacturers do not perform any binning at all and simply charge more for chips with higher ratings. Therefore, it is unlikely that the chip will completely stop working at this temperature, as shown in our results, although the performance is not guaranteed to be within design specifications as per Xilinx's datasheets.

4.5.2 Results

4.5.2.1 Resolution, SSP & Offset

Over the 68.3° C temperature range, the SSP varied by 8.23 ps. Specifically, at 32.7° C, the SSP was 60.66868 ps, while at 101° C, the SSP was 52.43871 ps. This is a gradient of -0.1205 ps/° C, i.e., the precision increases (SSP drops) by 120.5 fs for every 1° C increase in temperature. The full table of results and graph can be seen in Table 4.3 and Fig. 4.37 respectively.

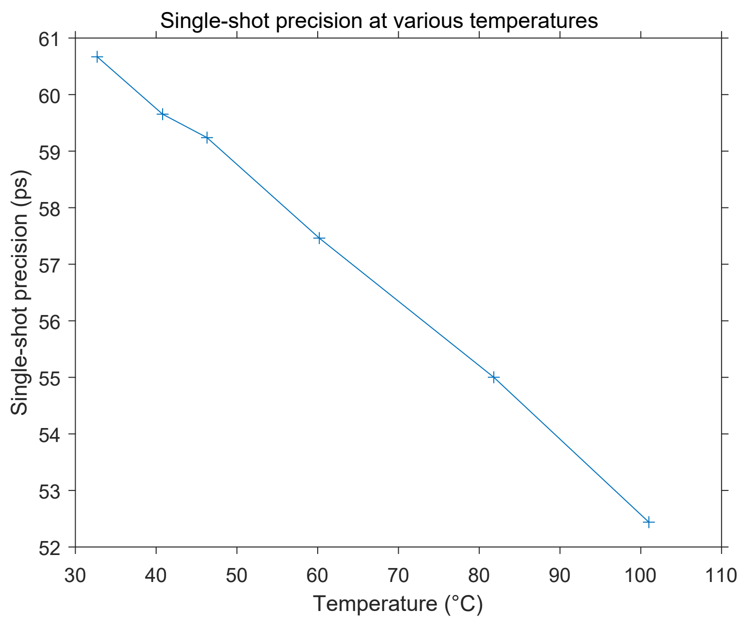


Figure 4.37: SSP plotted v.s. temperature.

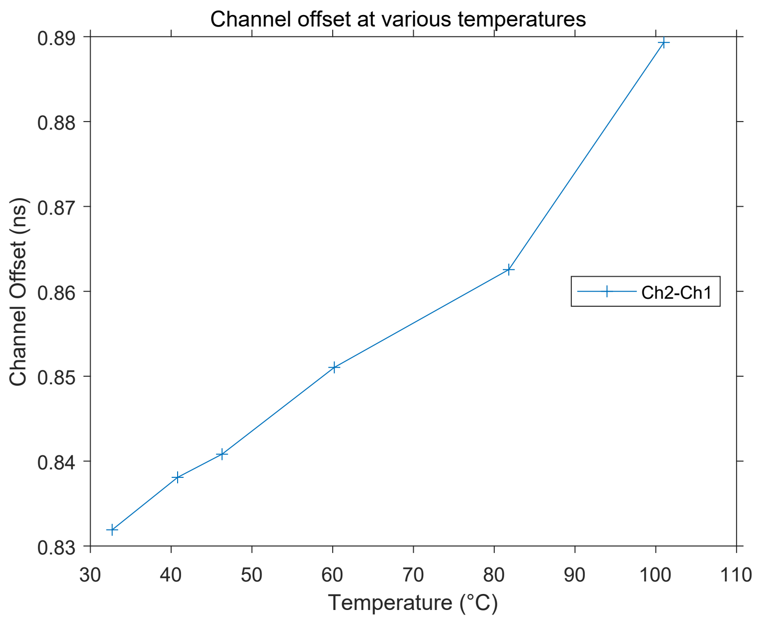


Figure 4.38: Channel offset plotted v.s. temperature.

Conversely, the channel offset increased as the temperature increased. At 32.7° C, the channel offset was calculated to be 831.909 ps, while at 101° C the offset was 889.320 ps, a difference of 57.411 ps and a gradient of 0.8406 ps/° C, i.e., the channel offset increases by 840.6 fs for every

1° C increase in temperature. Again, the full table of results and graph can be seen in Table 4.3 and Fig. 4.38 respectively.

The resolution is calculated as the cubic mean of the bin widths (4.10), where τ_i is the width of bin i and N is the total number of bins, to help with comparison to our other papers, as well as the equivalent bin width described in [39]. The resolution at various temperatures is shown in Table 4.3. Histograms of the bin sizes have also been calculated. The histograms for 32.7° C and 101° C (limited to the range 0 ps to 50 ps for readability) can be seen in Fig. 4.39 and Fig. 4.40 respectively, while Fig. 4.41 shows the variation in cubic mean resolution with temperature. Note that the histograms omit the 19 ultrawide bins (bin 0 of each DSP) to show the relevant bins on a reasonable scale. The average bin size on a per-DSP basis can be seen in Fig. 4.42 and Fig. 4.43. Bin zero relative to the start of each DSP block can be seen to constitute the majority of the delay, at 250 ps to 300 ps, while the majority of the bins (4th to 43rd) have a delay below 10 ps. The gradient of the resolution is -3.8877 ps over 68.3° C, or -56.9 fs/° C.

$$Res = \left(\frac{\sum(\tau_i^3)}{N}\right)^{\frac{1}{3}} \tag{4.10}$$

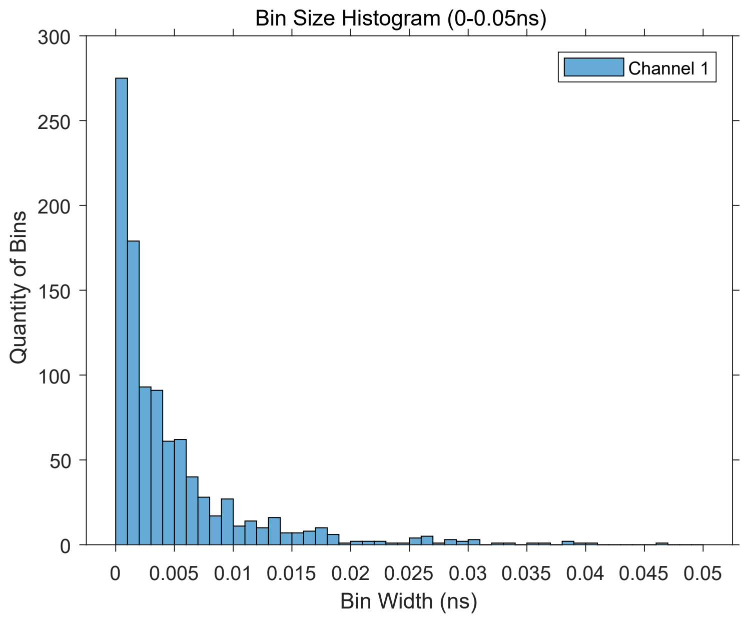


Figure 4.39: Bin width histogram for channel 1, 32.7° C.

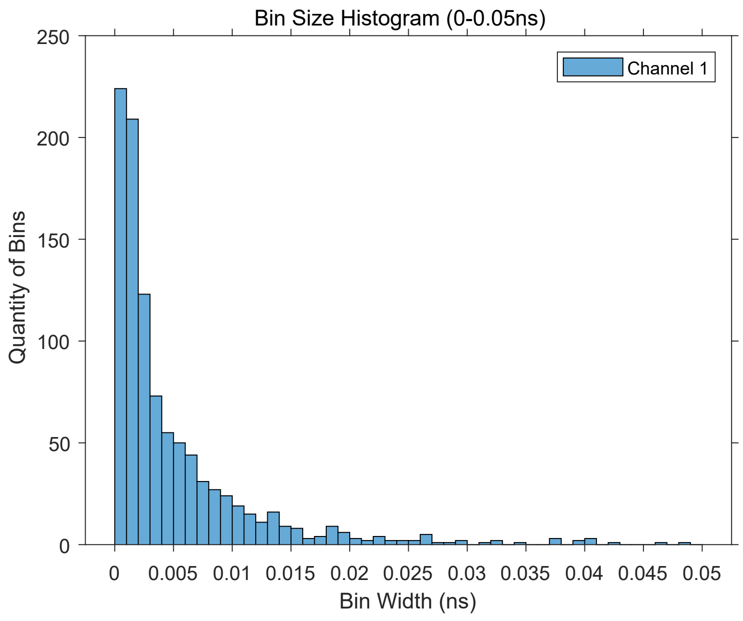


Figure 4.40: Bin width histogram for channel 1, 101° C.

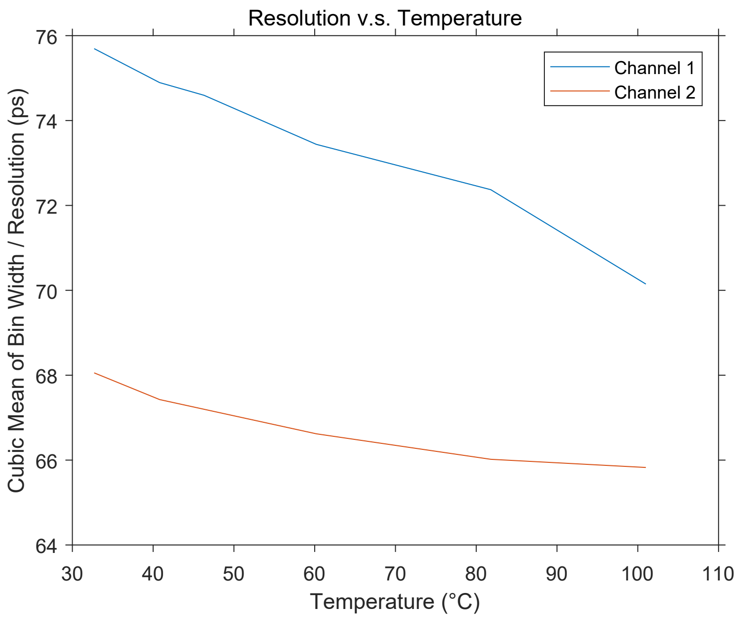


Figure 4.41: Variation in resolution (cubic mean bin width) with temperature.

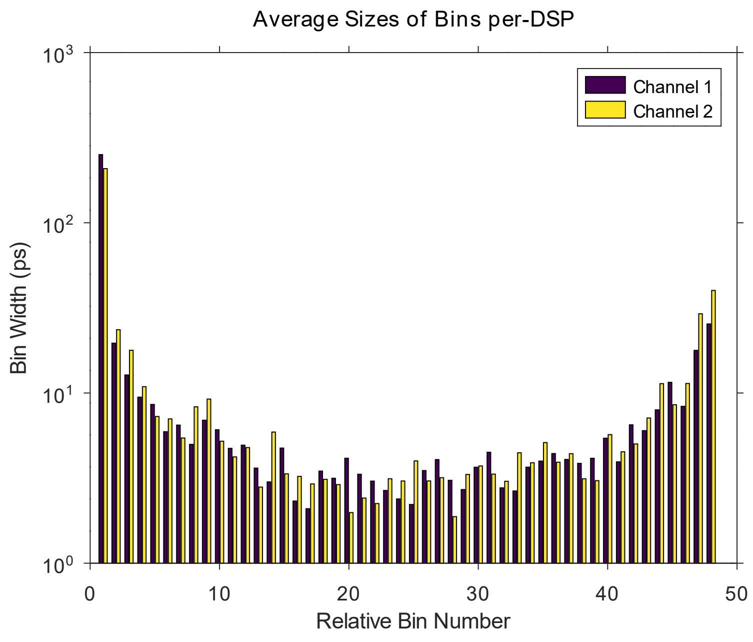


Figure 4.42: Per-DSP bin size, 32.7° C.

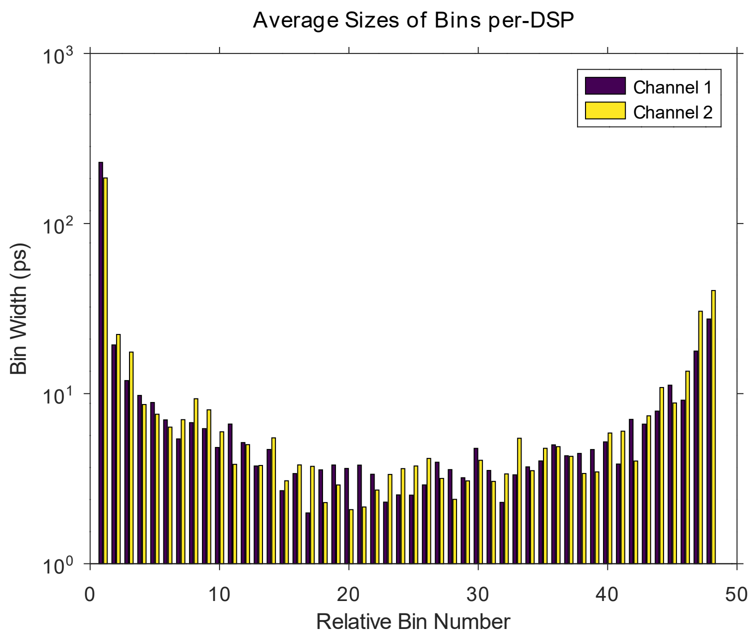


Figure 4.43: Per-DSP bin size, 101° C.

Temperature (°C)	Cubic Mean Resolution (ps)	Equivalent Width (ps) [39]	Offset (ps)	SSP (ps)	DNL (LSB)	INL (LSB)	Notes
32.7	71.875	184.2	831.909	60.669	-1 / +3.99	-5.64 / +4.05	Source A, No heating
40.8	71.160	181.5	838.089	59.654	-1 / +4.03	-5.73 / +3.44	Ambient / No source
46.3	70.897	180.5	840.816	59.239	-1 / +4.01	-5.75 / +3.13	Source A, Temperature 1
60.2	70.030	177.1	851.040	57.462	-1 / +3.98	-5.82 / +2.20	Source A, Temperature 2
81.8	69.196	174.0	862.560	55.001	-1 / +3.92	-6.04 / +0.904	Source B, Temperature 1
101	67.987	169.4	889.320	52.439	-1 / +3.87	-6.28 / +0	Source B, Temperature 2

Table 4.3: Characterisation results at various temperatures.

4.5.2.2 DNL & INL

The DNL and INL were calculated relative to the Least-Significant Bit (LSB) of the DSP delay line, which is defined as the cubic mean of the bin widths (i.e., the resolution). For the DNL, the size of each bin is divided by the LSB size to give the step size in LSB, and the ideal step size of 1 LSB is subtracted to give the DNL. For a zero-width bin, this would give a DNL of -1, and for a bin twice the normal size, this would give a DNL of +1.

Due to the distribution of bin sizes (and the presence of the 'ultrawide' bins between DSP blocks), the majority of the bins are more than an order of magnitude smaller than the resolution of the TDC ($-1 < DNL < 0.9$), with one in 48 bins (19 in total) being very large (approximately 4x the resolution). As a result, most bins sit at the positive and negative limits of the DNL, with very few in the middle.

The INL is similarly unfortunate. Due to the ultrawide bins which occupy ~55% of the time a signal spends inside a DSP (the smaller bins account for just ~45% of the total time spent inside the DSP), the INL varies by a significant amount per each DSP cycle.

As the temperature rises, we see each bin get smaller relative to the clock period, resulting in the INL dropping down to more negative values. The slack in the delay line is being picked up by bin 912, which is an 'ultrawide' bin (bin 0 of DSP block 20). At 32.7° C, bin 912 picks up a slack of just 16.5 ps before reaching the clock period of 10 ns, whereas at 101° C, bin 912 picks up a slack of 297.9 ps to reach the end of the clock period. The INL graphs for 32.7° C, 60.2° C and 101° C are Fig. 4.44, Fig. 4.45 and Fig. 4.46 respectively.

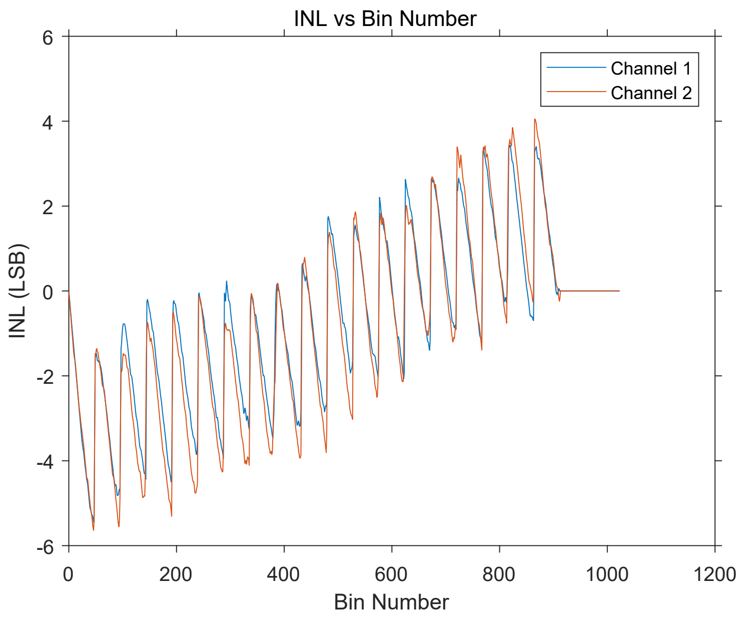


Figure 4.44: INL graph at 32.7° C.

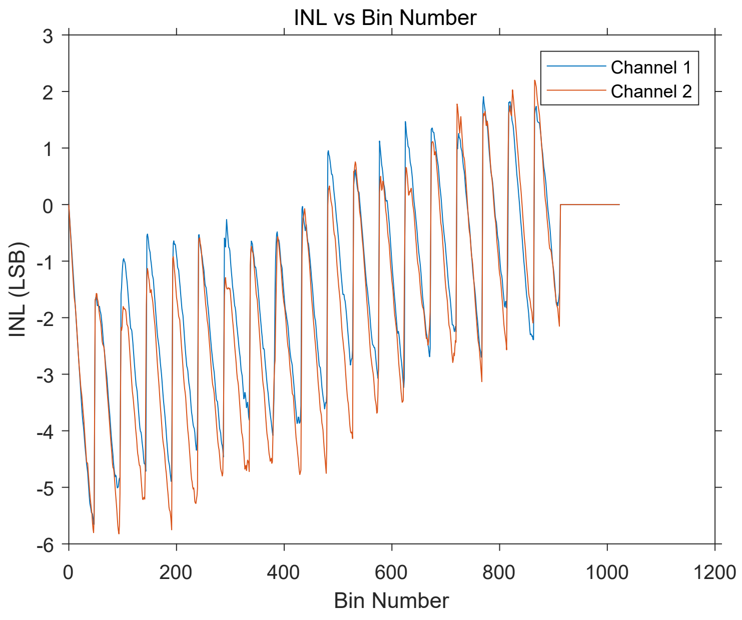


Figure 4.45: INL graph at 60.2° C.

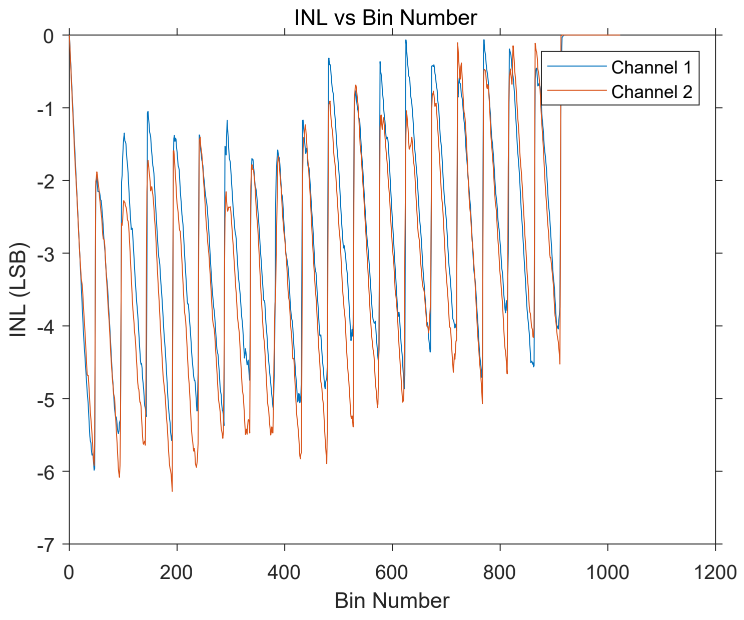


Figure 4.46: INL graph at 101° C.

4.5.3 Discussion

Consistently across all graphs, an improvement in bin size has been demonstrated, which is surprising given that transistors, MOSFETs and metal routing tend to slow down as the temperature increases above ~200 K. There is also an increase in the leakage at higher temperatures, so maybe this is allowing a partial charging of internal nodes which results in a faster transition when the trigger signal arrives.

Alternatively, given that sum nodes in the adder are specifically being discharged (and then inverted to give the thermometer code) while the carry nodes are being charged, perhaps the temperature dependence of the P-type MOSFETs is more significant than the N-type MOSFETs, resulting in a stronger N-type compared to the P-type and therefore increasing the discharge speed compared to the charge speed. This could be tested in future work by testing for the falling edge of the trigger (charging of sum, discharging of carry) and observing the temperature correlation in that scenario.

It was also notable how the bin sizes become more even at higher temperatures. Compared to Fig. 4.39 (32.7° C), Fig. 4.40 (101° C) shows less 'ultrasmall' (< 1 ps) bins (225 compared to 280) and more small bins (210 compared to 180 at 1-2 ps). It can also be observed that there are still significant quantities of bins up to 25 ps, whereas the quantity of bins drops off by 20 ps at 32.7° C. Despite this, we see an improvement in resolution, SSP and INL at higher temperatures due to the much shorter 'ultrawide' bins.

Considering the speculation we made on the architecture of the post-adder in [10], a possible suggestion for the increase in resolution and SSP (numerical decrease) could be that the sizes of the small bins are increasing at a faster rate than the sizes of the ultrawide bins, due to the small bins being dominated by transistor delay and the large bins being dominated by wire propagation delay.

If we assume that our speculations about the presence of a carry-propagation system are correct, then the carry is bypassing most of the post-adder's carry chain and spending a long time between DSPs. While the carry is travelling to the next DSP, most of the internal carry bits flip, causing time bins to form which do not impact the delay of the ultrawide bin.

As part of the ultrawide bin's delay is being masked off by the small bins, an increase in the size of the small bins relative to the ultrawide bins will cause a greater portion of the ultrawide bin to be masked off, resulting in better DNL and INL performance as well as better resolution and precision (due to less uncertainty from the ultrawide bin, which is the dominating factor).

The increase in channel offset as the temperature rises is as expected. The routing delay will increase as the temperature increases, and therefore a longer route will increase more than a shorter route. As a result, a positive offset will become more positive (magnitude increase) and a negative offset will become more negative (magnitude increase). As the offset between channels one and two is positive (channel two outputs higher average values than channel one), a positive coefficient was both expected and observed.

4.5.4 Conclusion

In conclusion, we have characterised the tapped delay lines implemented in DSP blocks, which were presented in our previous work, with respect to temperature. We measured the resolution, channel offset, single-shot precision, differential non-linearity, and integral non-linearity at multiple different temperatures and observed the effects of temperature on these metrics.

We observed an increase in the channel offset of 840.6 fs/° C, an improvement in the SSP of -120.5 fs/° C (smaller values are better), a resolution improvement of -56.9 fs/° C and more moderate DNL and INL ranges (DNL range: from 4.99 to 4.87 and INL range: from 9.69 to 6.28 as the temperature increased from 32.7° C to 101° C).

We suggested some possible causes for the improvements in SSP, resolution and INL despite expectations of these metrics worsening. In particular, the speculations made in the previous section provide a simple and robust explanation for the observed temperature effects, and thus our confidence in their validity is improved.

4.6 Conclusion

In this section, DSP blocks have been extensively investigated and tested as possible targets for high resolution delay generation in TDC delay lines. First, we looked at the post-adders of DSP48A1 blocks on Xilinx Spartan 6 devices, and observed that, while the average delay specifications were promising, there were large non-linearities in the output code which made a single DSP delay line ineffective in isolation.

Then, we looked at the DSP48E1 adders on Xilinx Artix 7 devices and observed similar non-linearities to the DSP48A1 blocks. However, with a ones-encoder to capture the out-of-order bins and Equivalent Coding Line (ECL) techniques combining four parallel staggered delay lines, we were able to achieve 5.25 ps resolution on the DSP blocks using just 10% of the FPGA fabric (slices).

Finally, we fully characterised the DSP delay line with respect to resolution, Single-Shot Precision (SSP), Differential Non-Linearity (DNL) and Integral Non-Linearity (INL). We also determined the sensitivity of this characterisation with respect to temperature. Surprisingly, some of these characteristics improved with an increase in temperature due to a reduction in the size of the Ultra-Wide Bins (UWBs), which appears to be an interaction between the highly non-linear carry architecture of the DSP post-adder and the temperature scaling of wires and logic in semiconductors.

In the next chapter, we will explore the application of wave union techniques to the DSP delay lines developed here.

Chapter 5

Multisampling

5.1 Introduction

Until recently, the highest single-quantiser (excluding averaging or equivalent coding lines [37, 156]) resolutions achieved in FPGAs have been achieved through the wave union method on carry chains [55, 157–159]. The previous chapter [10] showed how Digital Signal Processing (DSP) blocks can be used to improve the resolution for a simple delay line (single edge quantisation), and achieved 5.25 ps cubic mean resolution [11] within 10% of the FPGA's resources per channel but required four parallel delay lines (quantisers) to cover large bins. In this chapter, we will combine the high-resolution DSP blocks with the area-efficient wave union method, with the aim of achieving greater resolution in a lower area footprint. Temperature characterisation will also be performed over a 60°C temperature difference.

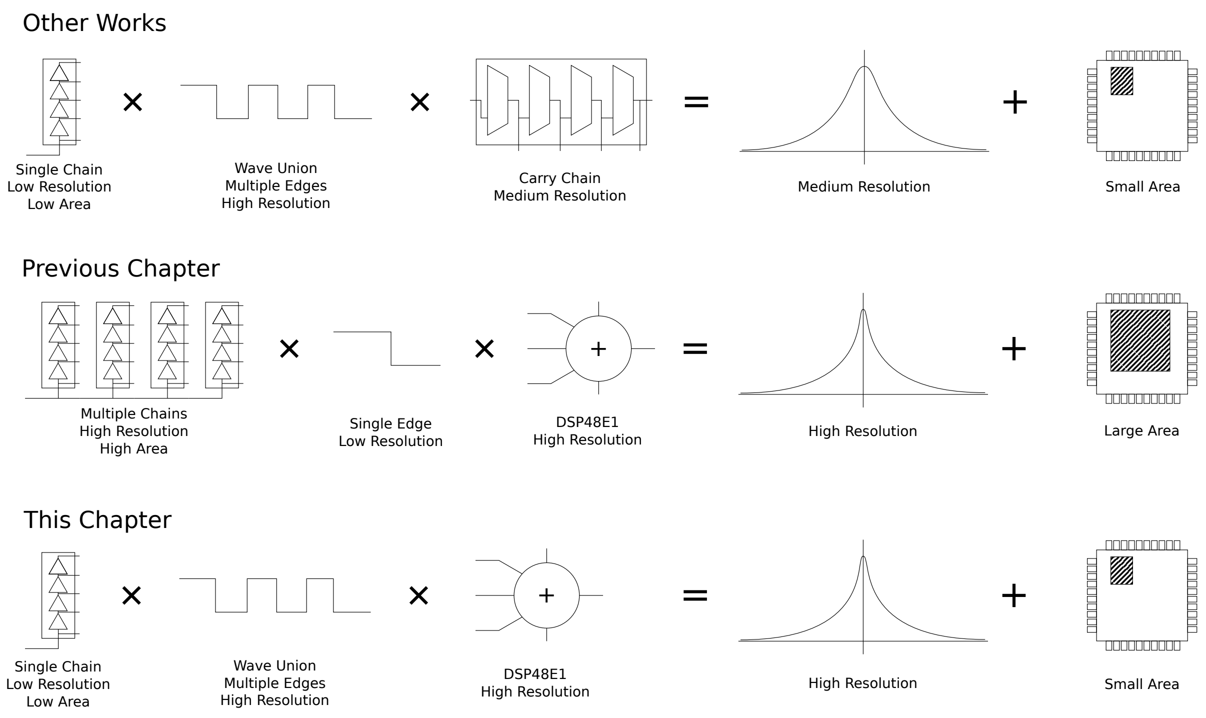


Figure 5.1: Diagram showing the idea to be explored in this chapter. We wish to use the wave union technique with DSP blocks to reduce area while maintaining high resolution.

5.2 DSP blocks

As shown in Chapter 4, our previous papers [10, 11] and other works on DSP blocks [154], modern FPGAs are starting to integrate more complex hard-wired IP (Intellectual Property) blocks into their designs to utilise the number of gates available on modern process nodes more efficiently. One type of block, the DSP block (e.g., DSP48A1, DSP48E1), contains a 48-bit adder (Fig. 5.2), and these can be cascaded to form large adder chains. As the average delay from carry in (CIN) to carry out (COUT) is 690 ps on an Artix-7 DSP48E1 (800 ps for the Spartan-6 DSP48A1), there is an average of 14.4 ps delay for each element, faster than the multiplexers in the CARRY4 elements used in most FPGA designs. This delay mainly consists of one large bin of 200-300 ps and multiple smaller bins of ~5 ps, with the size of the large bin attributed to higher quantities of logic and significantly more clock skew between DSP blocks. However, when encoded with a priority encoder, the bins are missing [10] (bubbles appeared at the output), and this was found to be due to bins transitioning out-of-order [11].

The configuration of DSP blocks as delay lines is described in Chapter 4.

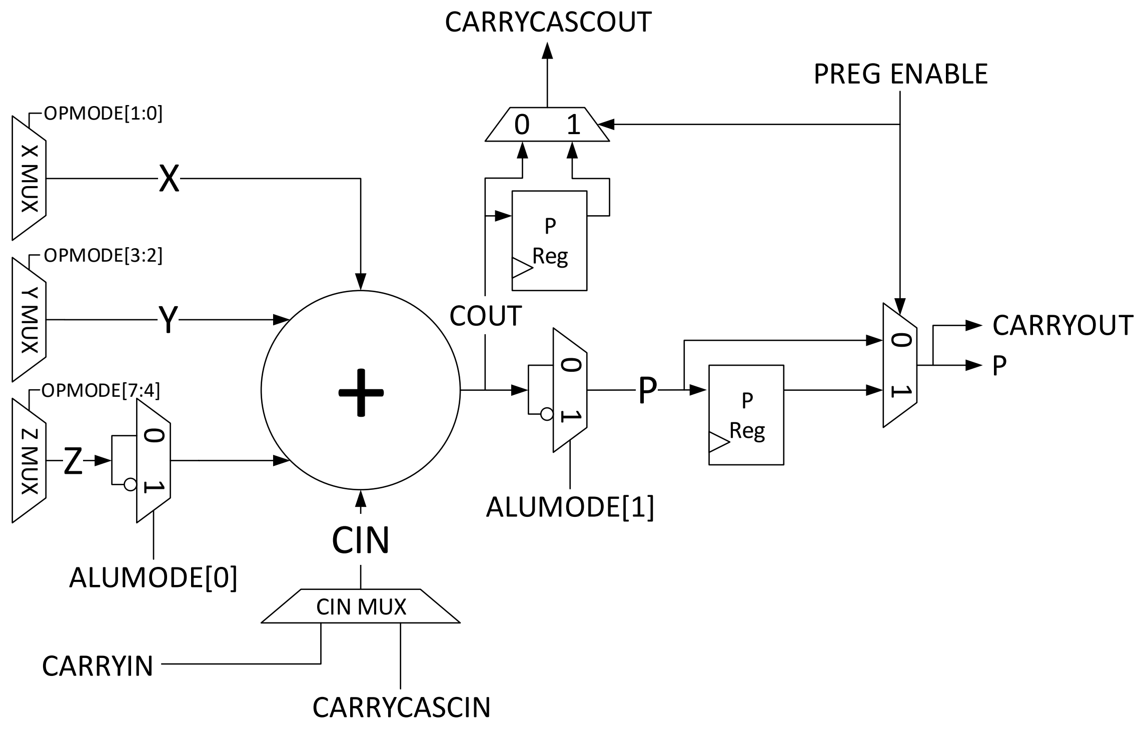


Figure 5.2: DSP add/sub operation, showing the purpose of the ALUMODE[1:0] and OPMODE[7:0] inputs, based on [4].

5.3 Wave Union Methods

Wave union methods are a relatively new advancement in TDC design, first appearing in [1], then later in [55, 160, 161]. This method promises to provide the same level of speed-up as multiple parallel delay lines, but without requiring additional delay line elements and calibration logic (to calibrate the extra delay lines). Rather than duplicating the delay line and all its decode logic, the wave union method instead replaces the first delay element of the delay line with a Finite Step Response (FSR) or Infinite Step Response (ISR) pulse injector. The FSR or ISR generates multiple edges with known relative delays from a single trigger signal and feeds these into the delay line.

The theory of operation for a wave-union TDC is demonstrated in Fig. 5.3. Given a non-linear delay line (no modern TDC has a linear delay line due to process variations and uneven load), there will be some elements (bins) of the delay line which are small and generate a high resolution (locally) and some elements (bins) which are large and generate a low resolution (locally). With only a single edge propagating through the delay line, the overall resolution will be degraded by the large bins.

Previous methods (Averaging and Equivalent Coding Line) solved this by introducing multiple parallel delay lines where one delay line would have its edge propagating through a large bin while other delay lines would have their edges propagating through small bins. When combining the outputs from each delay line, the small bins take precedence, and the large bin does not degrade delay line resolution.

With the wave union method, instead of multiple parallel delay lines, a pulse injector is introduced to the start of the delay line, which generates pulses with some time delay (bin difference) between them. These edges will all propagate through the same delay line, but while some edges are in large bins, others are in small bins. The decoder obtains quantisations of each edge and combines them with precedence given to the edges in smaller bins. This masks the low resolution of the large bins and therefore the resolution is not degraded by these large bins.

Calibration only needs to be performed for a single delay line, reducing area usage, and each of the edges will appear in the delay line in a different position, dependent on its delay from the first edge and the time of the trigger signal. During the calibration phase, we can determine the relative delay, and so we can extract multiple estimates of the trigger time, as in (5.1). The set Ts is the set of estimates of the trigger time, with cardinality N, and $\overline{T}$ is the estimate of the trigger time T based on Ts. T_i is the estimated time of edge i, without edge delay compensation (the 'raw' calibrated value) and T_{i-j} is the delay from edge i to edge j. A diagram of this operation is shown in Fig. 5.4. The time at which different edges arrive at the same bin will vary depending on the delay between the edges. By performing code density calibration, we can determine the delay between edges and therefore only calibrate for a single edge and adjust the calibration values according to the edge delay.

$$Ts = \{\forall i : T_i + T_{1-i}\} \tag{5.1}$$

$$\overline{T} = \frac{\sum_{i=1}^{N} Ts[i]}{N} \tag{5.2}$$

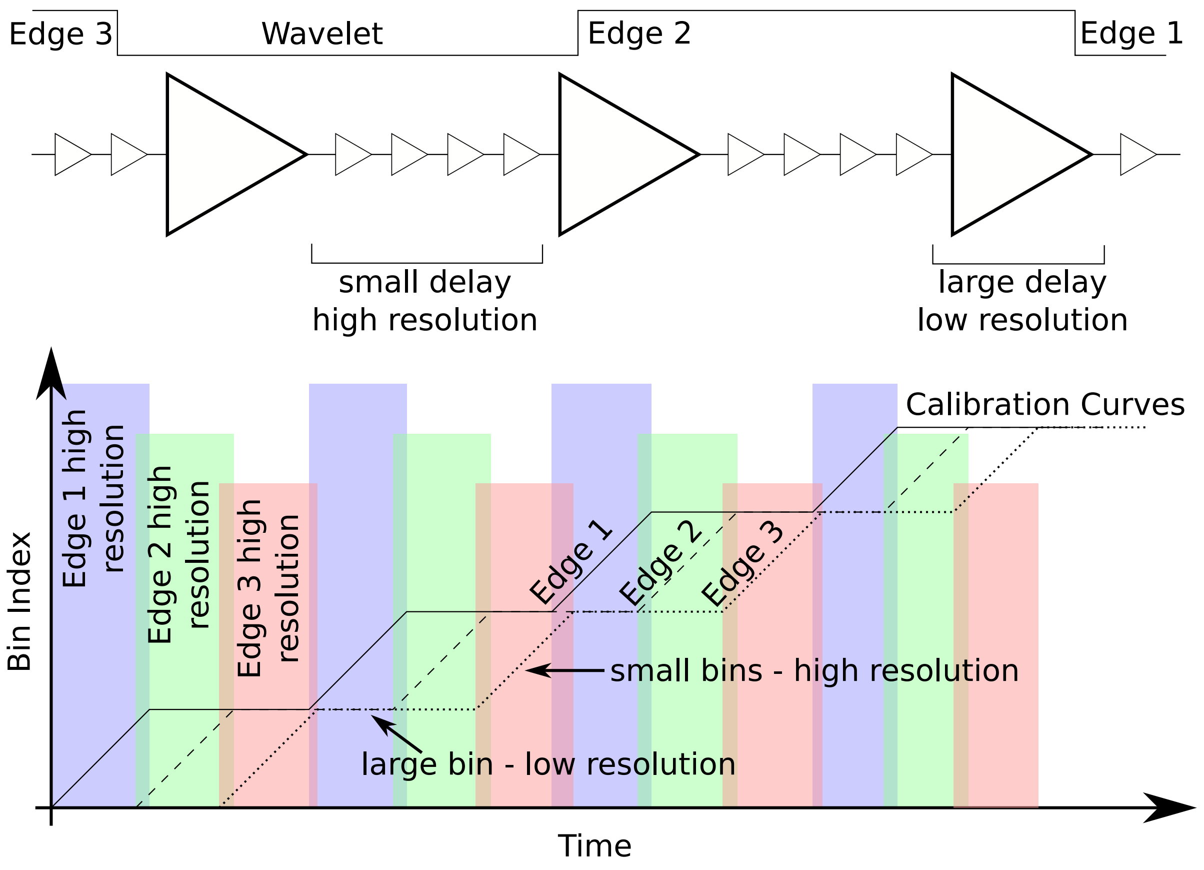


Figure 5.3: Operating principle of a wave-union TDC. At any point in time, at least one edge is in a high-resolution region. Solid line: edge 1; dotted line: edge 2; dashed line: edge 3.

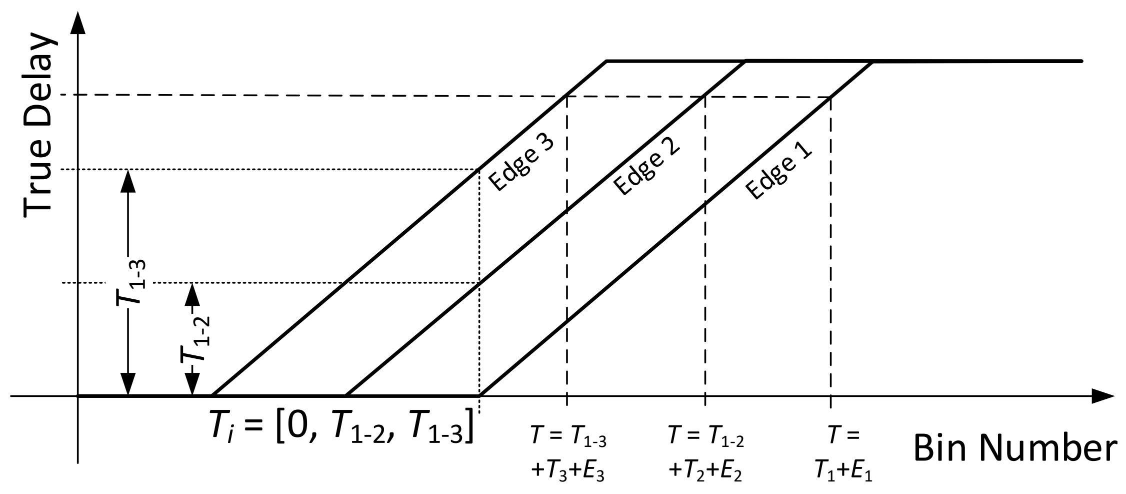

True Delay
Edge 3
Edge 2
Edge 1
$T_{1\text{-}3}$
$T_{1\text{-}2}$
$T_i = [0, T_{1\text{-}2}, T_{1\text{-}3}]$
$T = T_{1\text{-}3} + T_3 + E_3$
$T = T_{1\text{-}2} + T_2 + E_2$
$T = T_1 + E_1$
Bin Number

Figure 5.4: Diagram showing relative wave union operation. T_i = time estimate i. E_i = error in estimate i.

As with any other averaging method, the resolution scales with the inverse square of the number of samples. However, adding another sample is significantly less expensive in a wave union TDC than in a standard delay line as samples are gained simply by adding more stages to the pulse injector. Wave union methods on carry chains have been shown to reach SSPs as low as 3 ps in previous papers [157].

5.4 Combination of the Wave Union Method and DSPs

There are two possible architectures for combining the wave union method with DSPs. The first is to take a simple pulse injector implemented in carry chains or look-up tables (LUTs) and attach it to the front of a DSP delay line. This has the advantage of being a simpler implementation but has two drawbacks: it requires significantly more carry chain or LUT elements (due to each element being smaller than a DSP block) and has a low-resolution period at the start of the delay line where there is not a sufficient quantity of edges in the delay line to obtain a high-resolution result. Additionally, the LUT method results in poor geometry (LUTs cascade horizontally, DSPs cascade vertically, resulting an L-shaped delay line).

The second architecture involves implementing the pulse injector in DSP elements, resulting in a more complex implementation but allowing optimal resolution at the start of the delay line. Due to the more optimal resolution in the initial stages, we chose the second architecture.

To implement the pulse injector in DSP, we require a DSP block configuration that exhibits the following properties:

- When the trigger is asserted, the output must start toggling from 0 to 1, starting at the position the trigger was input and propagating through to the carry output.
- When the carry input toggles, this should also result in the outputs toggling. This must continue to occur every time the carry input toggles.
- Due to the out-of-order toggling of the bins and subsequent requirement for a ones-encoder, there must be a full DSP delay (> 690 ps) between adjacent edges.

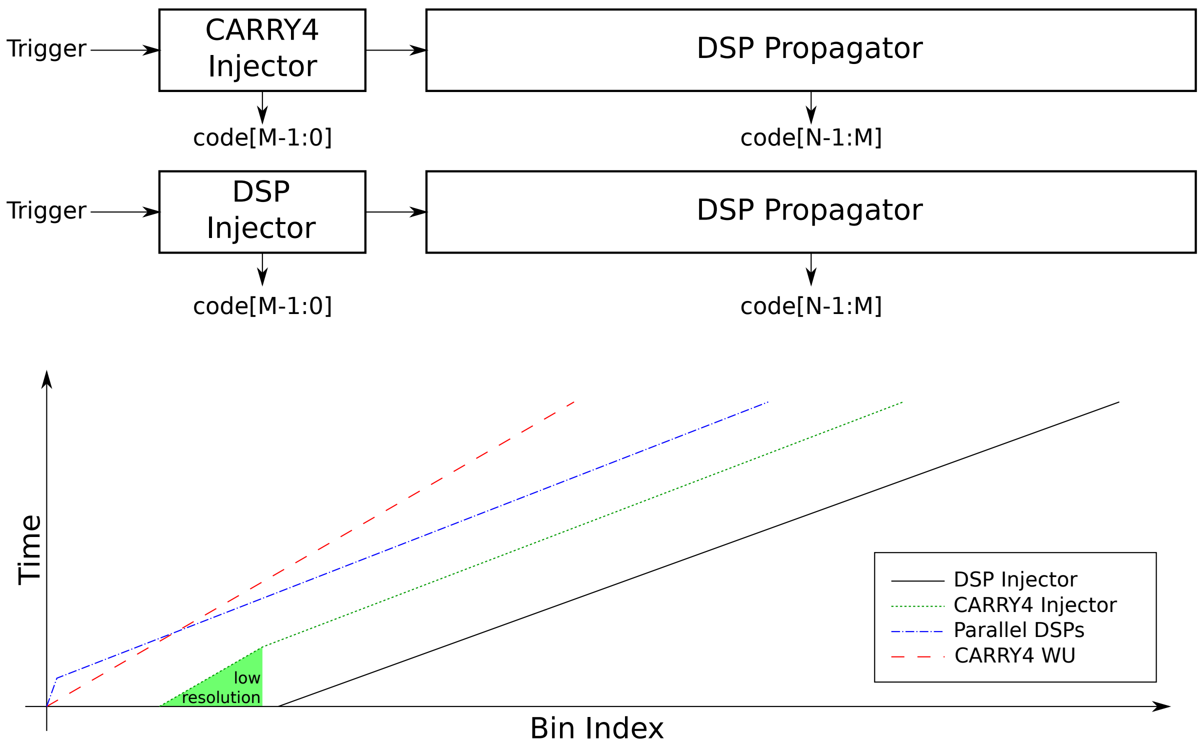


Figure 5.5: Comparison between injector methods for wave union. The CARRY4 injector has the same initial resolution as a CARRY4 WU (Wave Union) method, with the high resolution of the DSP delay line only arriving later.

- Before the trigger has toggled, and after all the edges have propagated through the delay line, the delay line should enter a stable state (not still be oscillating). This requirement only specifies that the delay line must settle if given a sufficient period of time, not that there is a limit to the minimum time provided (thus there are no impacts on dead time).

- On the deassertion of the trigger, the delay line should settle to the same state it had before the assertion of the trigger.

To help us achieve this, the following are open to adjustment:

- It is guaranteed that the carry input will not toggle before the trigger has arrived at the delay line.

- We can choose the initial value of the carry, and this can be guaranteed before the trigger arrives.

- Any set of operations inside the DSP block can be used.
- If a configuration can be found which exhibits these properties, but requires signals to be inverted, we can subsequently invert them again in the decoding process or in the DSP itself (there are input and output inverters available).
- We can use different configurations for alternate DSPs, should this be required.

The configuration we found that satisfies these properties splits the DSP configurations into three categories:

1. Inverting configurations, which cause the edges to be inserted into the delay line (and have the trigger signal connected to them).
2. Non-inverting configurations, which propagate the carry input to the carry output but do not introduce another edge. The configuration of these will change depending on the index of the first edge that will pass through them (and hence the idle-state value of the output).
3. Propagating configurations, which occur after the pulse injector and consist of most of the delay line. They will all have the same configuration and start in the zero state (all outputs 0).

These configurations are connected by their carry-cascade input/output pins (dedicated routes between adjacent DSP blocks) as shown in Fig. 5.6. Fig. 5.6() shows the theoretical implementation of the delay line, with the trigger connected to each inverting element to generate the pulse train, and all other elements being propagation elements that provide a high-resolution delay. However, due to the operation of the carry in the DSP48E1 blocks, the actual configuration is more complex (Fig. 5.6(a)). Due to the nature of the carry, every other inverting DSP block (those with even indices), as well as the following propagators, must have their outputs inverted to create a 'correct' output code.

5.4.1 Inverting Configuration

For the inverting configurations, the DSP blocks are set up to calculate trigger + 0xFFFF FFFF FFFF + 0xFFFF FFFF FFFF + carry (see Fig. 5.7()), with the output optionally inverted depending on

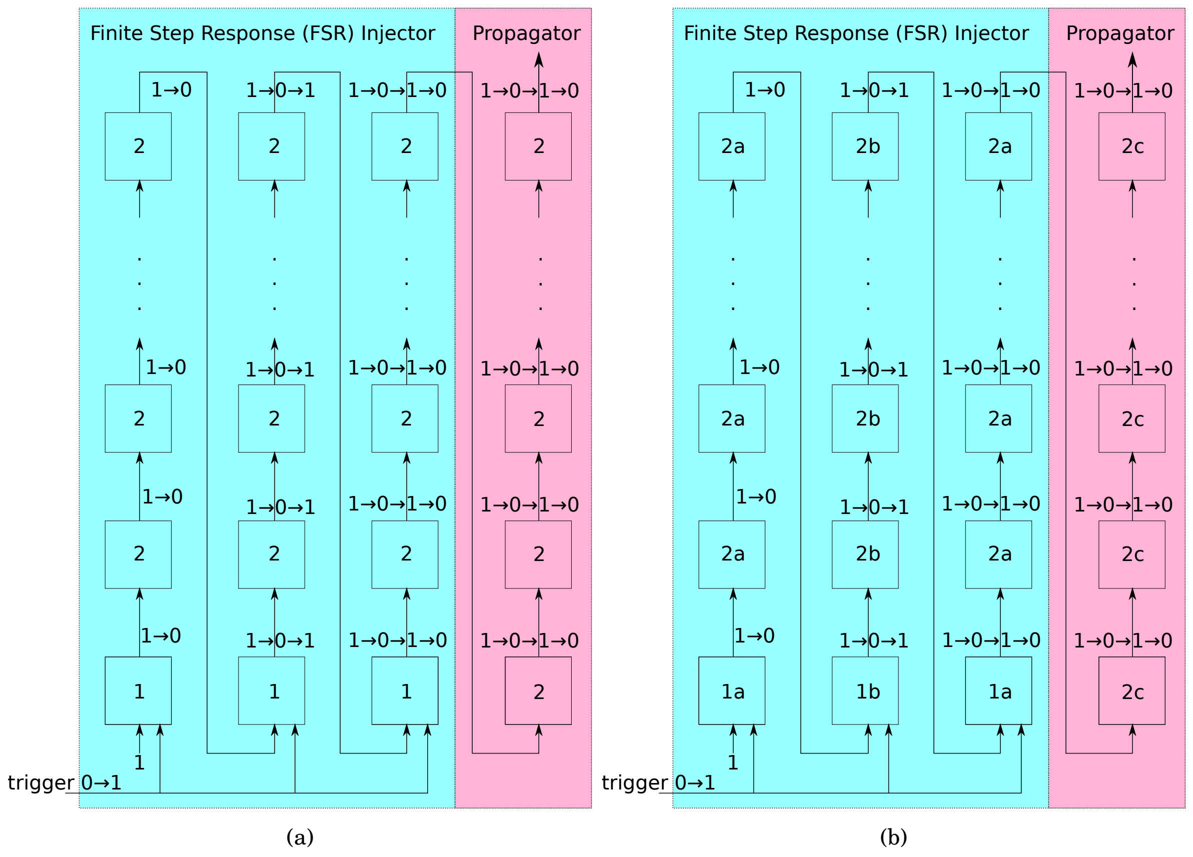


Figure 5.6: (a) Theoretical set-up of a Wave-Union DSP Delay Line (showing carries and trigger only). Each block marked '1' in the FSR adds another edge to the pulse train. (b) Real (on device) set up of the Wave-Union DSP Delay Line (due to differences in architecture).

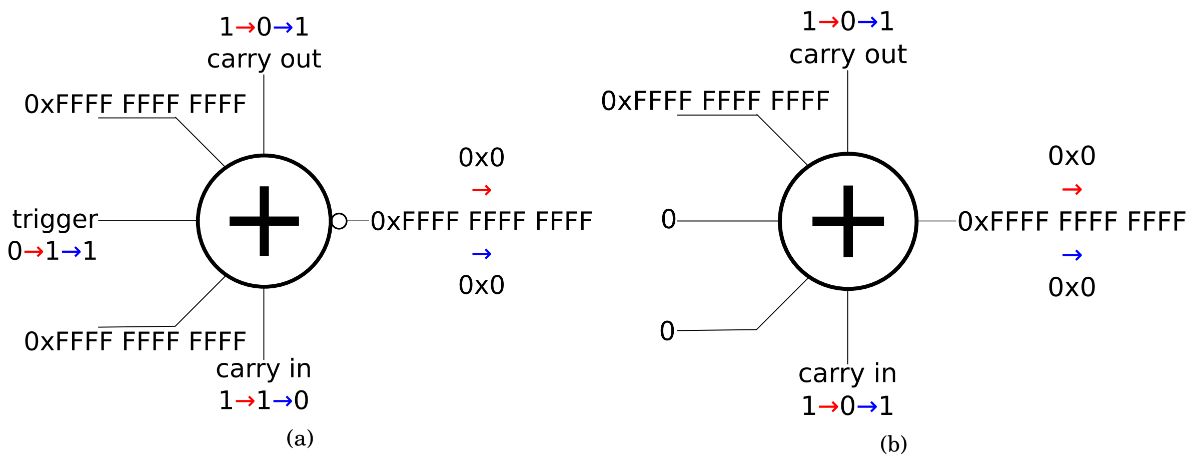


Figure 5.7: (a) Inverting configuration (blocks marked '1'). (b) Non-inverting configuration (blocks marked '2').

the stage of the FSR pulse. The initial state has the carry input as a 1, with a trigger of 0. This results in the computation of $(2^{48}-1)+(2^{48}-1)+1+0$, which has a carry output of 1 and a sum output of $2^{48}-1$ =0xFFFF FFFF FFFF. It can be seen that the carry output is the same as the input, the sum output bits are all the same, and that a small adjustment to the trigger will cause a carry cascade.

When the trigger arrives, the sum will roll round from $2^{49}-1$ to 2^{49}, causing the bits at the sum output to successively toggle. Furthermore, the carry out, which is equivalent to bit 48 (bits 0 to 47 are the sum output) will toggle (deassert), propagating this inversion to the next DSP in the chain. When the carry input deasserts, as would happen when the edge from the previous inverting DSP reaches the current inverting DSP, the output will return to $2^{49}-1$, causing all the output bits to successively toggle, including the carry out, which propagates the received edge to the next DSP. As the carry input continues to toggle, the output bits and carry output will continue to toggle.

At the sum output, without any modification, the initial state would leave all bits from the FSR set to 1. This is not suitable in situations where the trigger and clock signal are close together, as the pulses will not leave the FSR before the clock signal registers them, resulting in poor resolution. To fix this, the output is inverted on even-indexed subsections of the FSR, which results in a signal that starts with edges already present, held in place adjacent to the inverting DSPs.

5.4.2 Non-inverting Configuration

For the non-inverting configuration, the DSP is set to calculate 0xFFFF FFFF FFFF + carry (see Fig. 5.7(a)). As seen from the inverting configuration, the initial carry input has a value of 1, causing a sum output of 0 with a carry output of 1 (output value of 2^{48}). When the carry input deasserts due to a preceding inverting DSP, the output bits will assert and the carry output will follow the carry input, and the opposite will happen when the carry input asserts. Contrary to the inverting DSPs, the outputs of odd-indexed non-inverting DSPs are inverted so that the output pattern will match that of the preceding inverting DSP, resulting in the final state being an all-1 state.

5.4.3 Propagating Configuration

The propagating configuration is similar to the non-inverting configuration, calculating 0xFFFF FFFF FFFF + carry. However, unlike the non-inverting configuration, there are no inverting DSPs between the propagating DSPs, and their initial states should all be 0, so none of them have their outputs inverted. In the initial state, they all have a carry input of 1 (from the inverting and propagating DSPs), with a carry output of 1, resulting in a sum output of 0. When the carry input deasserts on the first propagating DSP, the edge will propagate through that DSP to the next one and eventually to the end of the delay line. Any assertion of the carry input will similarly propagate, even if the previous deassertion has not yet reached the end of the delay line. The final state will be an all-1 state due to a carry input of 0.

5.4.4 Configuration variables

To perform the operations described above, three configuration variables need to be set on the DSP blocks: ALUMODE, OPMODE and CARRYINSEL [4]. In addition, data must be provided to the inputs of the DSP (A, B, C and D). In all DSP blocks except the first one, CARRYINSEL is set to two, which propagates the carry output from the previous DSP block to the carry input of the current DSP block. For the first DSP block (at the start of the wave union launcher), CARRYINSEL is set to zero, which inputs a carry from the fabric, and the carry input is set to one.

ALUMODE is set to either zero or two on all DSP blocks. When ALUMODE is two, the adder operates normally on (calculates the sum of) the X, Y, Z and CIN (carry in) inputs, then the output is inverted before leaving the DSP block. When ALUMODE is zero, the output is not inverted. For the first inverting DSP in the chain (1a, even index), ALUMODE is two. For the subsequent non-inverting DSPs (2a, even index) before the next inverting DSP (1b), the ALUMODE is zero. Then, for the next inverting DSP (1b, odd index), ALUMODE is zero and the subsequent non-inverting DSPs (2b, odd index) before the next inverting DSP (1a again), the ALUMODE is two. In other words, the ALUMODE is two for the 1a and 2b configurations, and zero for the 2a and 1b configurations. For the delay line subsection (2c), the ALUMODE is always zero.

The OPMODE configuration bits change the source of the X, Y and Z inputs to the post-adder. The two configurations of interest are when OPMODE is 0x3B (inverting configurations, 1a and 1b) and 0x08 (non-inverting and propagating configurations, 2a, 2b and 2c). With OPMODE as 0x08, the lowest two bits which control the X input are zero, resulting in the X input being set to zero. The next two bits which control the Y input are one and zero, resulting in the Y input being set to 0xFFFF FFFF FFFF (2^{48}-1). The final three bits are zero, resulting in the Z input being set to zero. This configures the DSP block to add 0xFFFF FFFF FFFF to the input carry.

When OPMODE is 0x3B (011 1011), the lowest two bits are one, resulting in the X input forwarding the concatenated A and B inputs, which are in turn set to 0xFFFF FFFF FFFF. The next two bits are again one and zero, resulting in the Y input being 0xFFFF FFFF FFFF. Finally, the upper three bits are zero and two ones, resulting in the Z input forwarding the C input. The C input is all-zero except for the LSB, which is connected to the trigger and is therefore used to start the wave union launcher. This configures the wave union launcher to add $2^{48} + 2^{48} + \text{trigger} + \text{carry in}$, which inverts the signal relative to the previous DSP block.

5.4.5 Decoding

The decoding of the output bus is not a trivial task. Unlike a carry-chain-based delay line, the bins do not switch in order, meaning the standard technique of performing an exclusive-or (XOR) between adjacent (or close) bins then iteratively picking up and masking off the detected transitions is not feasible. Similarly, applying a ones-encoder to the entire delay line, as in the previous sections, is also unsuitable, as the ones-encoder does not detect edges or their position in the delay line, only the number of ones. Ideally, due to the excessively large bin at the end of each delay line, we would be able to guarantee that an entire delay line will have switched before the subsequent DSP has any bits toggle. Therefore, we would be able to perform a ones-encoding per-DSP, and then use standard edge detection logic between DSPs, with the caveat that the 48 least-significant bits are already encoded.

However, due to the carry-lookahead logic present in the DSP's post-adder, the code spreads itself over multiple bins. Therefore, to decode this pattern correctly, we first need to rearrange the bins near the transitions. To do this, we iterate over the codes with a modified bubble sort which redistributes bits towards the nearest edge. This results in a clean edge which can then be decoded as originally intended.

With this scheme, we cannot have a new edge enter a DSP before the previous edge has left the DSP, as the ones-encoder cannot distinguish between a positive edge at the half-way point with a negative edge at the start, and a positive edge at the end with a negative edge at the half-way point. Providing we satisfy this requirement (leave a suitable time delay between successive edges), we can then detect edges in the output codes from the ones-encoder. Observing three adjacent ones-encoders (- is “Don’t care”), there are four possible scenarios for the output:

1. (48,0,-): There is a positive edge in the final bin of the first DSP. This results in an all-1 outputs for the first DSP and an all-0 outputs for the second and third DSPs.
2. (0,48,-): There is a negative edge in the final bin of the first DSP. This is the inverse of (1).
3. (48,N,0): There is a positive edge in bin **N** of the second DSP. This results in an all-1 output for DSP 1, an output of N for DSP 2, and an all-0 output for DSP 3.
4. (0,N,48): There is a negative edge in bin **48-N** of the second DSP. This is the inverse of (3).

Conversion is applied by iterating through the DSPs from end to start while remembering the expected output. If the current bin is not the same as the expected value (the first expected value is 0), then there is an edge at position ($48 \times i + |48 - expected - N|$), where i is the index of the DSP with the edge. For example, if the converter encounters 0, 12, 48 (LSB on right) with $i = 3$ when observing the ’12’, then the edge position will be $48 \times 3 + |48 - 0 - 12| = 144 + 36 = 180$. However, if it encounters 48, 12, 0, then the edge position will be $48 \times 3 + |0 - 0 - 12| = 144 + 12 = 156$.

In a realistic system, the first edge will propagate through approximately 18 DSPs during a 10 ns clock period (the TDC clock was 100 MHz from the onboard MEMS [Micro Electro-Mechanical System] oscillator). With the addition of a realistic number of edges to test the system (3 for this example), each of which requires at least three DSPs to maintain an edge-to-edge delay greater than a single DSP, there will be >27 DSPs present in the system. Each requires six bits for its ones-encoder, resulting in at least 162 bits for the fine code (28 DSPs spanning 168 bits were used in actuality).

Three DSP blocks are required for each edge due to the length of the bubbles caused by the Carry Look-Ahead (CLA) structure. A bubble may span up to an entire DSP block in length and will be present at every edge. In the worst-case scenario, a bubble will start at bit 24 of DSP i, running to bit 23 of DSP $i+1$, and another bubble will start at bit 24 of DSP$i-k$ and run to bit 23 of DSP $i-k+1$. For the case $k=2$, DSP $i-k+1$ is DSP $i-1$. As both DSPs i and $i-1$ have bubbles in them, there is no completely empty or completely full group of 48 to distinguish one bubble from the other (the bubbles have merged). Therefore, we require at least $k=3$ (three DSP blocks per edge) to ensure decodability of the wave union TDC.

5.4.6 Compression

Due to the large number of bits per time tag (168), the continuous repetition rate of the system is severely reduced due to the output bus being speed-limited. However, compression is relatively easy for this system. As we know that the edges are approximately three DSP delays apart, we can encode the difference between adjacent edges using seven or eight bits. Once this has been encoded, the only other information required is the position of the first or last bin. This can be encoded by dividing the delay line into four or eight chunks (for nine or eight bits bin proximity, respectively), then outputting the chunk where the first bin is located and the distance from the start of the chunk to the first bin.

For example, if the delay line had 1344 bins (28 DSPs), and the edges were in bins 486, 586 and 686, the output would be “010-1001 0110-0110 0100-0110 0100” (2-150-100-100). The meaning of this is:

1. The first bin is in chunk two (“010”), which starts at bin 336, and is 150 (“1001 0110”) ahead of that (bin 486).
2. The second bin is 100 (“0110 0100”) forward of the first bin (bin 586).
3. The third bin is 100 (“0110 0100”) forward of the second bin (bin 686).

This encoding only takes 27 bits, rather than the 168 bits originally required, effectively sextupling the bus-limited repetition rate. Further analysis may yield stronger compression schemes and further increase compression ratios.

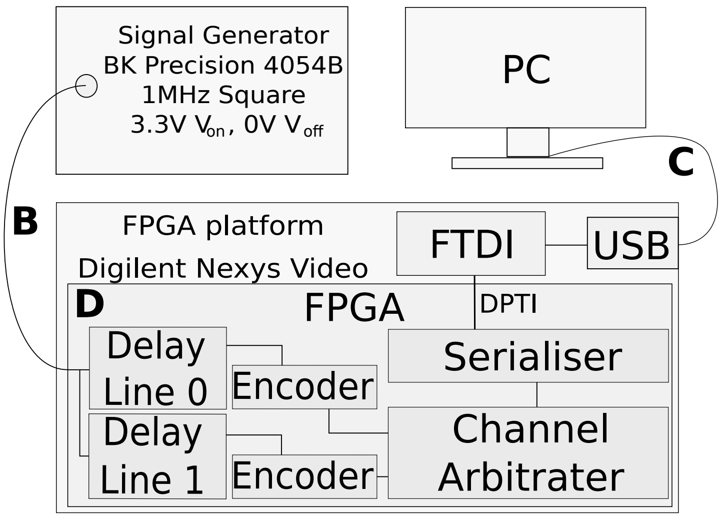
5.5 Test Setup

To test this system, we implemented the wave union DSPs on an Artix-7 200T FPGA (xc7a200tsbg484-1, "Digilent Nexys Video"). Two quantisation channels were instantiated, both being fed by the same input from off-chip. The channels were then multiplexed on to a 60 MHz 8-bit parallel bus (Digilent DPTI) and output to a PC via custom command-line software for calibration and edge quantisation. The calibration and edge quantisation could be performed on-chip [162], but this is not required to test the system. The input was driven by a BK Precision 4054B signal generator using a 100 kHz, 50% duty cycle signal generator, operating at "LVCMOS33" voltages ($3.3V_{pk-pk}$, $1.65V_{off}$). Without a compressor, the maximum sustained output rate is 2.85 Mevents/s, while the maximum burst rate is 50 Mevents/s on each channel (<= 2 clock cycles per channel) for up to 195 events. With a compressor as detailed in the compression subsection, the sustained output rate would be 17.77 Mevents/s and the burst rate would be the same (50 Mevent/s) for up to 1213 events. The test setup is illustrated in Fig. 5.8.

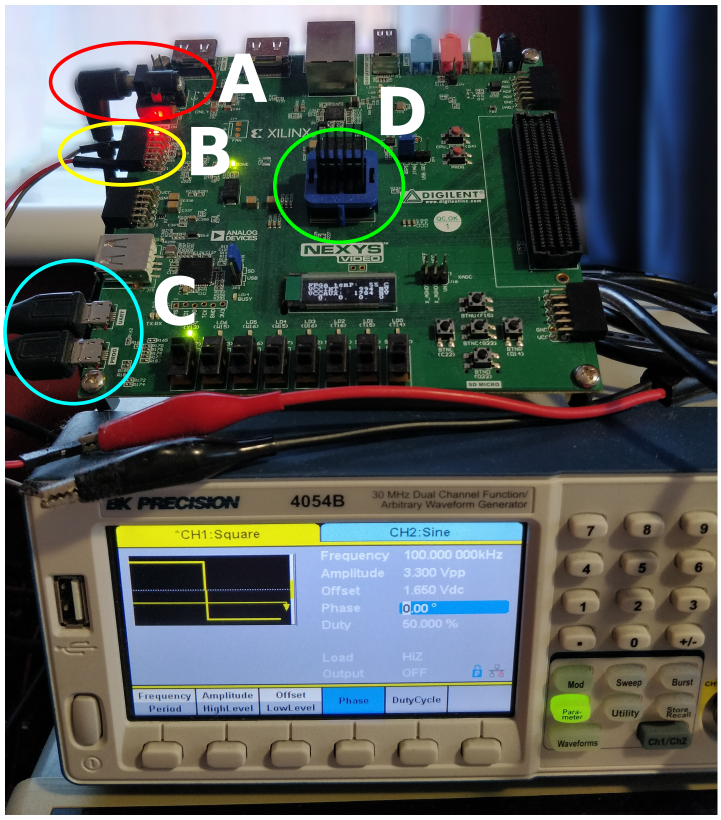
261865 tags were collected, ~130932 from each channel in the normal mode of operation. The first 90% of the tags were used to calibrate the two TDC channels, while pairs of the remaining 10% with the same coarse count were used to determine the Single-Shot Precision (SSP). The SSP is the standard deviation of a single measurement, but we are measuring it by calculating the standard deviation of the difference between the two channels, so we must divide the calculated standard deviation by $\sqrt{2}$ to determine the SSP.

The time difference between the channels was calculated by individually calibrating each edge's progression through the delay line with a code density test, then taking the calibrated offsets and performing a weighted average, with the weight applied to each sample equivalent to the reciprocal of the calculated bin width. With this method, the small (high-resolution) bins have a much larger impact on the result than the large (low-resolution) bins. For comparison, an unweighted average was also calculated. However, it gave worse results in all metrics except temperature dependency, and so is not quoted except during the temperature characterisation.

These experiments were repeated both at ambient temperature and with the supplement of heated air from a hot-air rework station (Yihua 968DA+, resistive heating element with an



(a)



(b)

Figure 5.8: Test setup (a) diagram and (b) picutre. Red circle (A): power in. Yellow circle (B): trigger in. Blue circle (C): data out. Green circle (D): FPGA + heatsink.

axial fan blowing heated air over the FPGA's passive heatsink parallel to the PCB) to perform temperature characterisation. The on-chip temperature diode and XADC (Xilinx Analog-to-Digital Converter) were used to determine the temperature of the chip for a given airflow, and the TDC run simultaneously to collect data at that temperature. At each temperature, the TDC was calibrated as well as tested. Five temperatures were sampled: 39.1°C (ambient, no heated air), 54.2°C, 71.9°C, 85.8°C and 99.6°C. This form of temperature characterisation is quite basic and it would be better to perform these tests in a temperature-controlled chamber if possible.

5.6 Results

Once the results were collated and post-processed, the Cumulative Density Functions (CDFs) related to each edge, bin widths, and edge-to-edge differences were calculated. The CDF of the TDC can be observed in Fig. 5.9. This shows the expected pattern of output codes, with highly accurate linear regions where the DSP delay line is operating as intended, as well as sudden vertical (time-domain) jumps, which demonstrate the excessively large bin at the end of each DSP. Notably, it can be observed that, for any given time offset (observing along a horizontal line of equivalence in the y axis), at least one DSP is in its high-accuracy linear region. This is what enables the sub-dividing of the large bins - as one or two edges are travelling through a large bin, the other edge(s) are in the linear region providing accurate quantisation.

The bin widths can be observed in Fig. 5.10. It can be seen that most of the bins have a width in the region of 0.5 ps to 30 ps (10^{-3} ns to 3×10^{-2} ns), while a few bins are larger than this, (some reaching as high as 500 ps). However, due to the observation from Fig. 5.9 that at least one of the three edges is in the high-resolution region, this large non-linearity can be subdivided using the equivalent coding line method. For this reason, a weighted average was used to prioritise information from the high-resolution bins and de-prioritise low-resolution bins. If it were not the case that at least one edge was always in the high-resolution region, this could be remedied by adding more edges or adjusting the delay of the FSR injector to become asynchronous with the DSP delay.

Fig. 5.11 shows histograms of the measured time differences between two channels (the

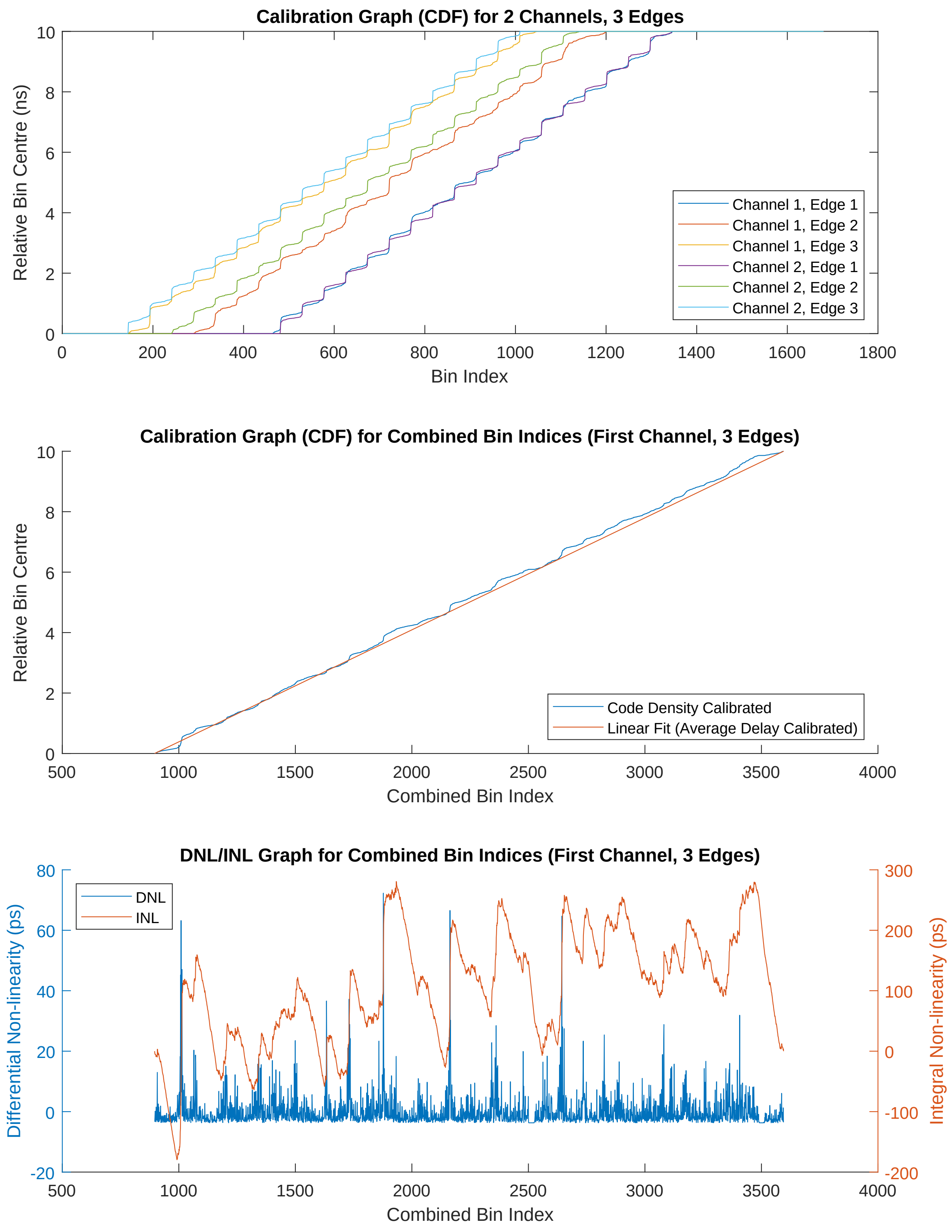


Figure 5.9: Calibration and DNL/INL graph for the wave union method.

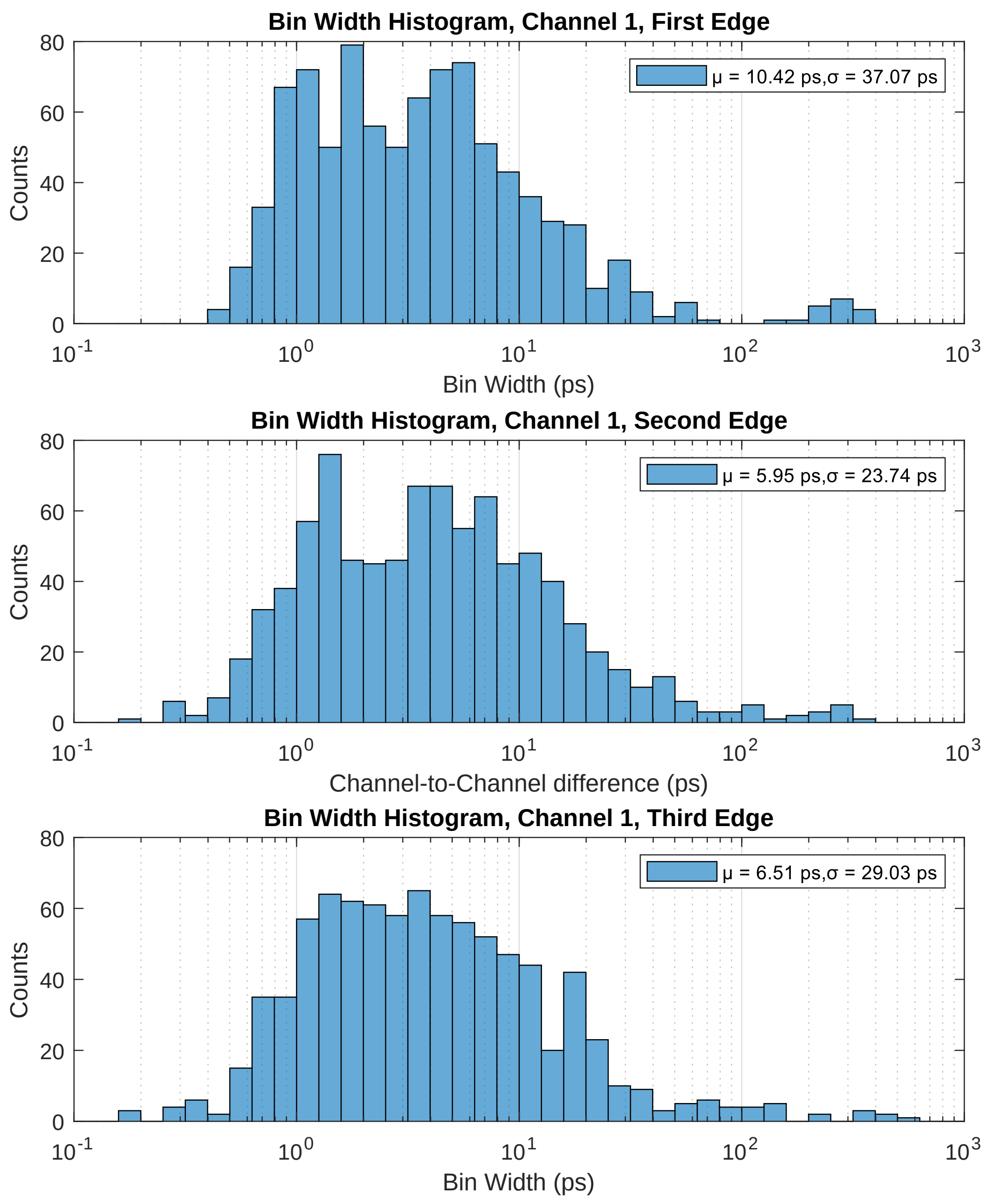


Figure 5.10: Histogram of bin widths for the wave union method. Note the logarithmic x axis.

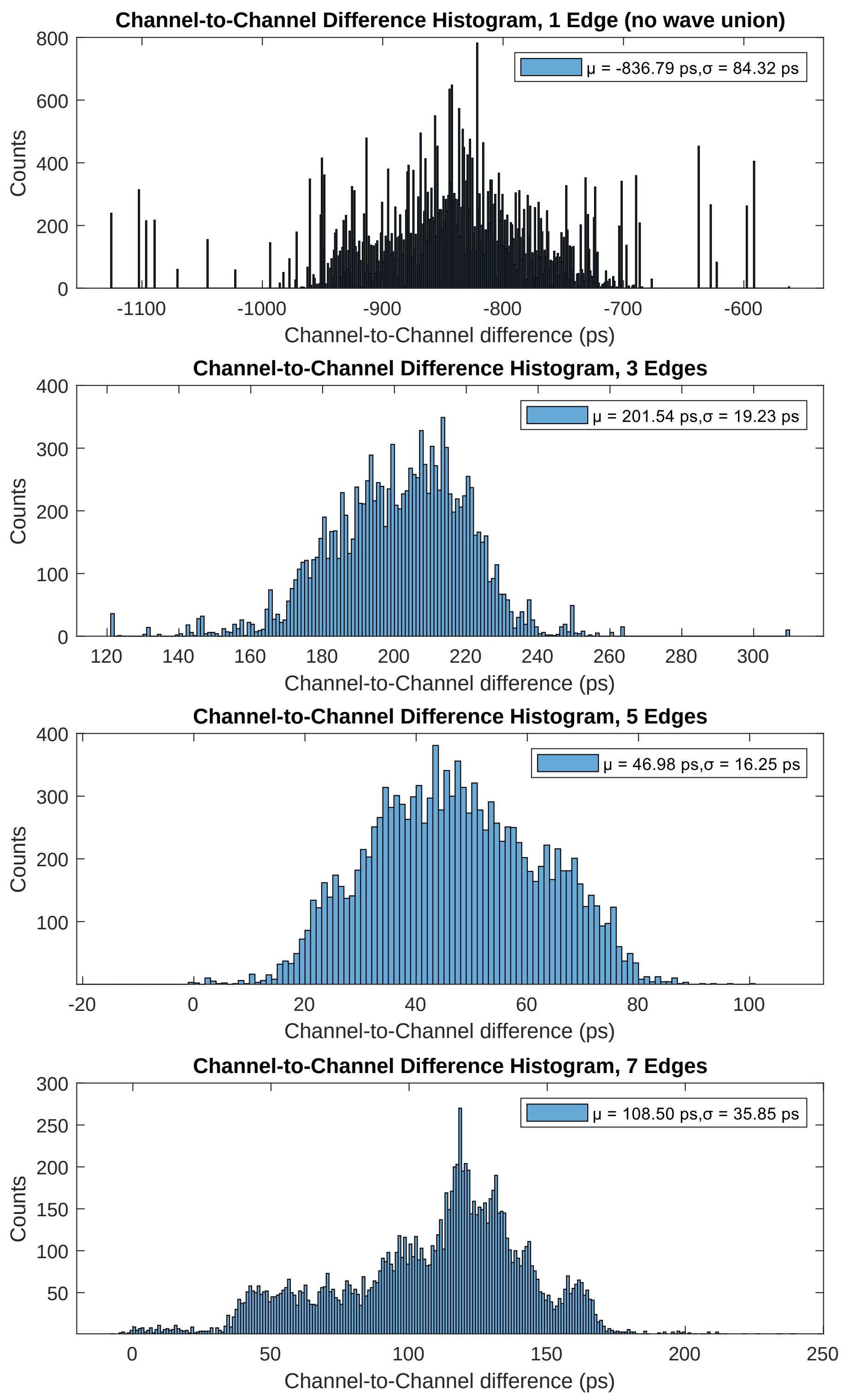


Figure 5.11: Histogram of channel differences (precision) for one edge, three edges, five edges and seven edges. There are three DSPs of delay per edge.

same signal is introduced to both channels and quantised, giving a sample of the channel offset). These histograms provide graphical representations of the precision and accuracy of the TDC (the mean of the histogram is the accuracy, the spread is the precision). There are four graphs, corresponding to implementations with a single edge (no wave union method), three edges, five edges and seven edges (the edge number is odd due to implementation details, not a theoretical limitation). As the channel offset and spread of the time differences is significantly different between the four implementations, the scale on the x axis varies significantly.

The histogram for a single edge (no wave union method) shows a large spread of data, with a standard deviation (double-shot precision) of 84.32 ps (52.65 ps SSP) and significant peaks up to 290 ps away from the mean due to ultra-wide bins at the boundary of DSP blocks. This is massively improved by introducing the wave union technique with three edges, which drops the standard deviation to 19.23 ps (13.60 ps SSP) and exhibits no significant peaks more than 40 ps from the mean. This is due to edges in their high-resolution region sub-dividing the ultra-large bins encountered by other edges, resulting in the TDC always operating in a high-resolution region. In terms of area, a single edge requires 22 DSP blocks (2.97% of the FPGA) per channel, three edges require 28 DSP blocks (6 extra in the FSR injector, 3.78% of the FPGA) per channel, and the parallel implementation in [11] required 74 DSP blocks (10% of the FPGA) per channel for a cubic mean resolution of 5.25 ps.

With five edges, the standard deviation experiences a small improvement to 16.25 ps (11.49 ps SSP). This is because the extra edges are further sub-dividing the available bins to improve the resolution and thus reduce quantisation error. This requires another six DSP blocks (0.81% of the FPGA) per channel for two extra edges compared with 18 DSP blocks (2.43% of the FPGA) per chain in the parallel implementation.

When seven edges are introduced, the standard deviation degrades to 35.85 ps (25.35 ps SSP) rather than improving. This runs contrary to expectations, as it would be expected that the precision would improve with more edges. However, as stated in [163], uncertainties in routing delay (the σ^2_{CY} element) and DSP block PVT variations become a dominant factor at higher edge quantities due to the extensive routing required to distribute signals to the delay line and pulse injector. Experimental adjustment of the placement of the TDC channels confirmed this, as placing the TDC further from the input pins significantly impacted precision (up to three times the SSP). This explanation also accounts for the poor improvement in performance between three and five edges (the expected precision would be 8.16 ps without any other sources of imprecision).

Therefore, we have determined that five edges provide optimal precision in our design, although three edges may be desirable for the reduced area. Clock jitter (~1 ps for the given oscillator) is not a component of the error expressed here, as all samples were taken with identical coarse counts (same clock period), but would be a small contributing factor if larger ranges were measured.

The DSP48E1 blocks were swapped out for CARRY4 blocks to compare the performance between the two delay elements. Without a pulse injector, the CARRY4 implementation performed much better, with an SSP of 9.31 ps (compared to the DSP's 52.65 ps SSP). When moving to three edges, the gap closed somewhat, with the CARRY4 implementation achieving 5.64 ps SSP (compared to the DSP's 13.60 ps). Thus, a wave-union DSP implementation with a 3-edge pulse injector is a good substitute for a CARRY4 delay line without any pulse injection. Due to running calibration on a PC with floating-point arithmetic, the calibrated INL is equivalent to half the DNL at any point in the delay line (since the centre of each bin coincides with the calibration look-up). If the calibration were on-chip, the INL would be slightly worse (most likely using a 1.25 ps LSB). The INL that would be achieved with average delay calibration (significantly worse) is graphed in Fig. 5.9 along with the Differential Non-Linearity (DNL). The minimums and maximums for DNL and INL are $-3.71/+72.28$ ps (DNL), $-179.58/+280.75$ ps (INL, uncalibrated) and $-1.86/+36.14$ ps (INL, calibrated) respectively.

When combining the edges together with the weighted averaging (WAV) method, the temperature characterisation (Fig. 5.12) revealed a 4 ps SSP penalty over the 60°C characterisation range. Specifically, the SSP was 13.50 ps at ambient temperatures (39.1°C T_{diode}) and 17.53 ps at 99.6°C T_{diode}. Comparatively, when using unweighted averaging, the accuracy only varied by 1.3 ps over the 60°C temperature range, but the unweighted average (AV) gave significantly worse results overall, varying from 30.67 ps SSP at 39.1°C T_{diode}(ambient) to 31.98 ps at 99.6°C T_{diode}. This suggests that the temperature is more significantly affecting the small bins separated by logic elements (XOR, AND, MUX etc.) than the large bins separated by wire delay (between DSPs).

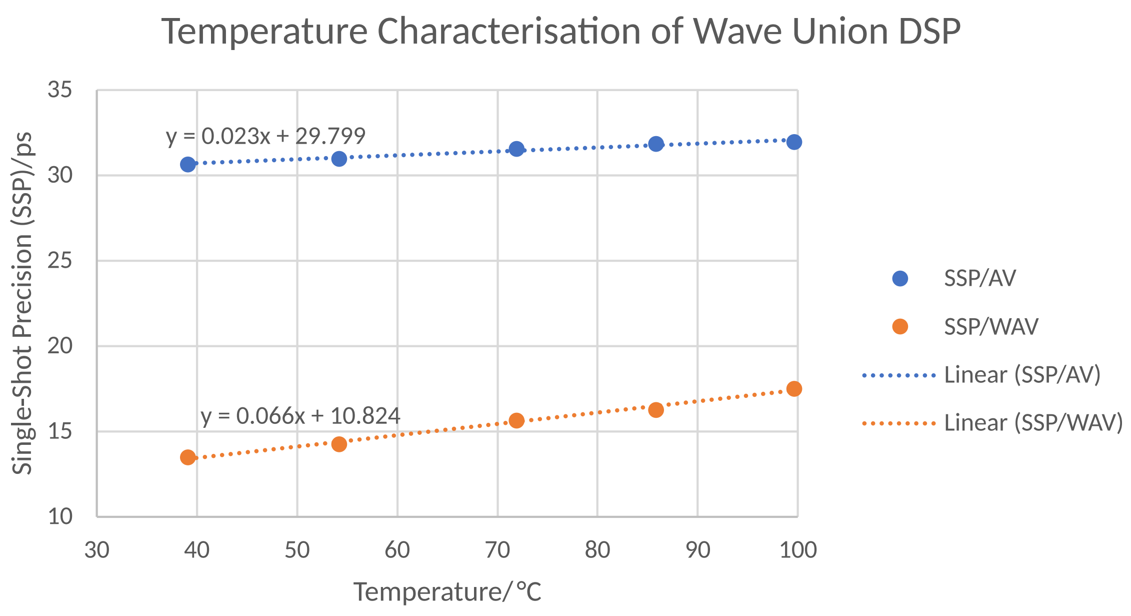


Figure 5.12: SSP vs Temperature characterisation.

A tabulated summary of the comparison between the different DSP delay line architectures is shown as Table 5.1. For all delay line methodologies, the pairing with a coarse counter of arbitrary size allows the range to be effectively infinite, however, the fine part of the delay line is sized according to the system clock, which is 10 ns for all architectures demonstrated in this section. The Effective Number of Bits (ENOB) is therefore the binary log of the clock period divided by the equivalent width of the bins ($\sqrt{12}$ times the SSP), as in (5.3). For the Delay Line, DSP Delay Line and 4x Parallel DSP, the output code is 64 bits, with 18 or 20 bits assigned to the fine count and channel identifier, resulting in a 44 or 46 bit coarse counter, with a system clock of 10 ns. For the Wave Union designs, the coarse counter is up to 64 bits long, with a system clock of 10 ns.

$$\text{ENOB} = log_2\left(\frac{T_{clk}}{\sigma\sqrt{12}}\right) \tag{5.3}$$

Table 5.1: Performance of various TDC implementations. N/A: Not Available.

Author	Delay Elements	Edges per Chain	SSP/ps	Fine ENOB	INL/ps	Fine Range/ns (Coarse)	Area
Wu 2008 [1] (Type-A)	Intel LAB	3	25	4.89	N/A	2.56	1621 LE (20% Cyclone II EP2C8T144C6)
Bayer 2011 [55]	Xilinx CARRY4	2	9	6.50	N/A	2.82 (164 μs)	N/A (Virtex 4 LX40)
Szplet 2013 [37]	Xilinx CARRY4	1	6	5.94	22	1.28 (17.18 s)	13.44 mm^2 (Spartan 6 SLX75)
Szplet 2016 [156]	Xilinx CARRY4	1	4.5	4.50	23	0.352 (4400 s)	42.19 mm^2 (Kintex 7 150T)
Szplet 2016 [164]	Xilinx CARRY4	6	6	4.97	33.93	0.650 (2s)	N/A (Spartan 6)
Wang 2016 [165]	Xilinx CARRY8	2	9.5	5.93	N/A	2	N/A (Kintex Ultra-scale 40T)
Wang 2019 [157]	Xilinx CARRY4	8	3	7.44	88	1.805	1.98% LUTs (Kintex 7 325T)
Lusardi 2019 [166]	Xilinx CARRY4	2	12.5	5.79	8	2.4 (157 μs)	7010 LUTs (Artix 7 200T)
Tancock 2019 [11]	Xilinx DSP48E1	1	N/A (5.25 ps cubic mean)	9.10	N/A	10	10% DSPs (Artix 7 200T)
Kwiatkowski 2020 [158]	Xilinx CARRY4	4	3.3	6.97	55.2	1.43	N/A (Kintex 7)
Xie 2020 [159]	Xilinx CARRY8	2	3.03	7.57	6.14	2	1.75% CLBs (Kin-tex Ultrascale 40T)
Zhu 2020 [167]	Xilinx CARRY4 + DSP48E1	1	3.4	8.4	49.4	4	152 DSP, 2306 Slice (5.43%, 3.04%, Virtex 7 485T)
Tancock 2021 [12]	Xilinx DSP48E1	1	52.56	5.78	296	10	2.97% DSPs (Artix 7 200T)
Zhu 2021 [168]	Xilinx CARRY4 + DSP48E1	1	2.2	7.77	92.5	1.67 (7.16 s)	72 DSP, 2288 Slice (2.57%, 3.01%, Virtex 7 485T)
This Work (Delay Line*)	Xilinx CARRY4	1	13.16	7.78	N/A	10	1.58% Slices (Artix 7 200T)
This Work (Wave Union*)	Xilinx CARRY4	3	7.98	8.50	26.2	10	2.46% Slices (Artix 7 200T)
This Work (WU-DSP)	Xilinx DSP48E1	3	13.60	7.73	36.15	10	3.78% DSPs (Artix 7 200T)
This Work (WU-DSP)	Xilinx DSP48E1	5	11.49	7.97	19.46	10	10% DSPs (Artix 7 200T)

* Implemented for comparison.

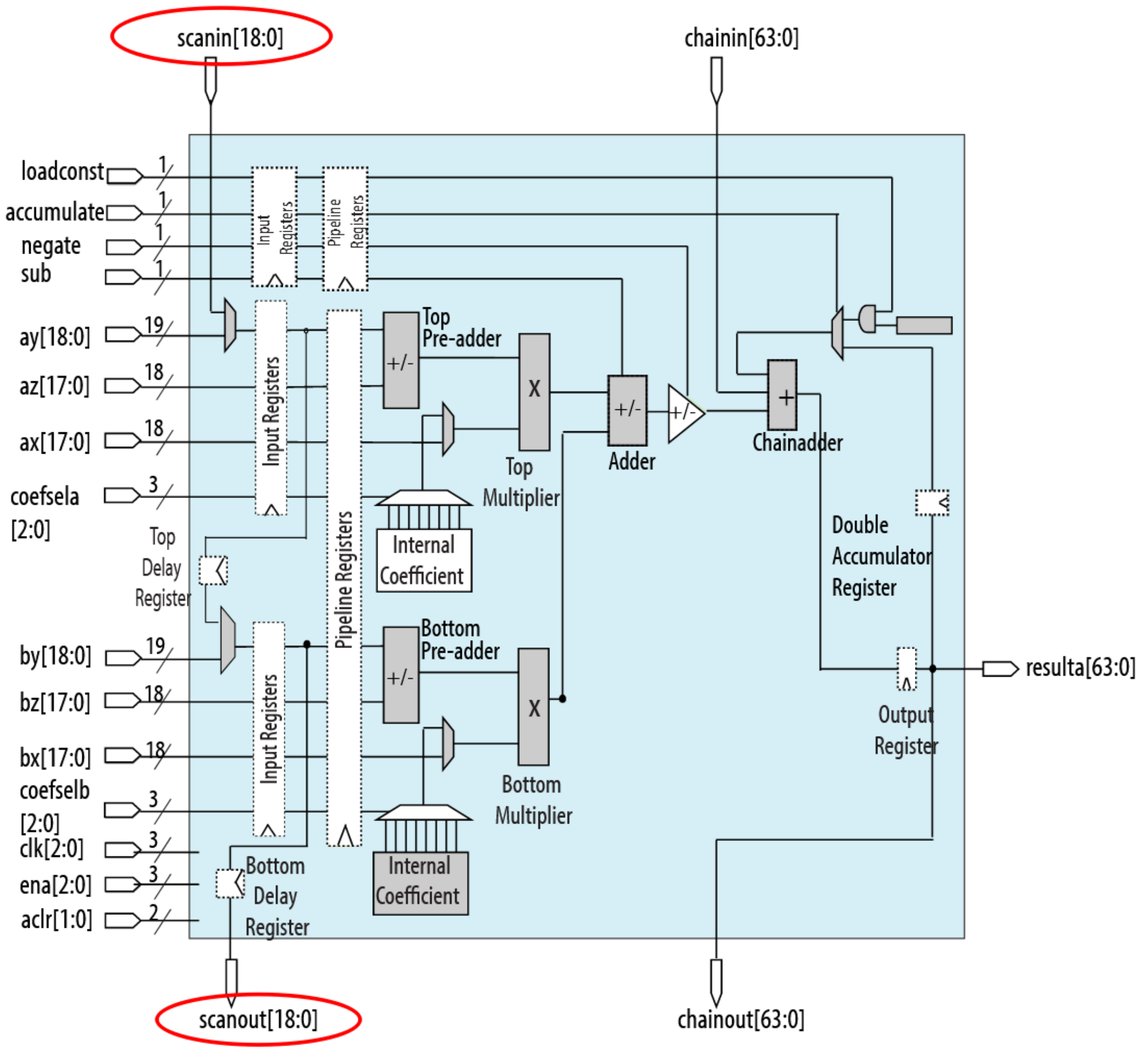


Figure 5.13: Intel DSP Block internals [3]. The 'scanin' and 'scanout' ports are highlighted. Registers with dashed borders are systolic (optional).

5.7 Conclusion

In conclusion, we have adapted the wave union method to utilise the DSP blocks present in Xilinx FPGAs. This was achieved using the DSP blocks' built-in 48-bit adders configured to invert the outputs successively when the carry input or trigger input changed, propagating this toggle to the output. We also suggested a method of compressing the output code to quadruple the bus-limited data rate. While the system was able to obtain five edges with the same area cost as the previous section at four edges, or nine edges with 64% of the area cost of our previous paper at eight edges, the benefits of further increasing the number of edges is minimal.

We performed a two-channel difference test and the performance of the wave union DSP with three edges was on par with a delay line of CARRY4 elements, demonstrating that wave union methods on DSPs are a viable alternative to carry-chain delay lines, allowing for higher channel quantities in multichannel systems or making general-purpose logic available to other applications. Modest gains were made with five edges, though neither outperformed a three-edge wave union on CARRY4 blocks. Temperature characterisation showed a small but not insignificant variation over a 60° C temperature range, to the extent that large temperature changes would require a recalibration for optimal accuracy.

The scheme we have demonstrated is applicable to any Xilinx FPGA from the Spartan-6 series onwards, which covers most of the FPGA market and all Xilinx devices available for non-legacy applications (some older devices are still available to meet the demands of older customers). Given the theoretical basis of the scheme, it could be easily adapted to any device with a large fixed-function adder with asynchronous carry propagation between adder units (i.e., no required register between one adder and the next). For example, the second largest FPGA manufacturer, Intel (formerly Altera), has a DSP block with very little resemblance to the Xilinx DSP block. However, the lowermost bit can be cascaded between the 'scanin' and 'scanout' ports (Fig. 5.13), and this bit can then proceed to trigger the carry cascade in the post-adder.

Chapter 6

Bubble Correction

6.1 Abstract

Bubble detection and correction logic is vital in modern data capture devices to solve bubbles in the output thermometer codes due to non-linearities in the scale causing negative bin widths. Previous bubble correction techniques are either unsuitable for short pulse widths and multiple registration (ones-encoder) or have a very short range (all other methods). In this chapter, we propose a hardware technique to detect and correct bubbles up to the length of the pulse width while preserving position information using a hybrid between the ones-encoder and a single stage of a modified insertion sort. This design was shown to meet timing on a Xilinx Artix-7 FPGA at 100 MHz or above using only 13% of the device, demonstrating hardware-viability. The design is also fully pipelined to demonstrate high bandwidths. The limitations of the algorithm are stated and some possible improvements are suggested.

6.2 Introduction

Process, Voltage and Temperature (PVT) variations in modern Analog-to-Digital Converters (ADCs) and Time-to-Digital Converters (TDCs) are increasing, mainly attributed to the shrinking process node and higher resolution of these devices. This results in larger non-linearities in the output codes of these devices, eventually causing negative-width output bins (lower-valued codes that occur at a higher input value than higher-valued codes). To correct for these disturbances, many bubble detection and correction techniques have been developed [119, 169–173].

However, these techniques all have a very limited scope. The ones-encoder (also known as a population counter) method [170] is intolerant of short pulse widths or double registration, as it does not provide any information about the position of the edge, just the number of ones. In a system where we can guarantee that the lowest bits will not transition on large input values (single-registration, long pulse width), this is equivalent to the position of the leading edge with all bubbles removed. However, for techniques such as double-registration, the sliding scale technique (in some cases) and wave-union TDCs, the later introduction of 0s into the output code results in the ones-encoder failing to differentiate between ones at lower positions and ones at higher positions.

Techniques other than the ones-encoder such as the N-bit NAND technique [171], multiplexer technique [172], Butterfly sorting technique [173] and Stepped-Up Tree Encoder (SUTE) [119] are all very limited in their range and do not scale effectively with the size of the bubble. This is suitable for simple converter architectures such as carry chains (in an FPGA TDC) [174] and resistor ladders (in an ADC) [169], but less useful for more complicated architectures such as 2-D comparative encoders [143] or, as we discovered, in the case of DSP delay lines [11]. These encoders may have prohibitively large bubbles (96 bins in the case of DSP delay lines) due to complex architectural forwarding techniques (e.g., carry look-ahead adders) for techniques other than the ones-encoder.

We used a ones-encoder in section 4.4 on DSP delay lines, but to implement the wave-union technique on the DSP delay lines in chapter 5, we needed a long-range bubble correction technique that preserved edge position. This is because the task of decoding bubbles was significantly more complex with wave-union TDCs due to the long bubbles and multiple edges.

To this end, we suggest a two-stage bubble-correction algorithm consisting of a local ones-encoder to produce partial codes and then a hybrid between a single stage of an insertion sort and a bin-packing algorithm which can separate the ones and zeros around the edge.

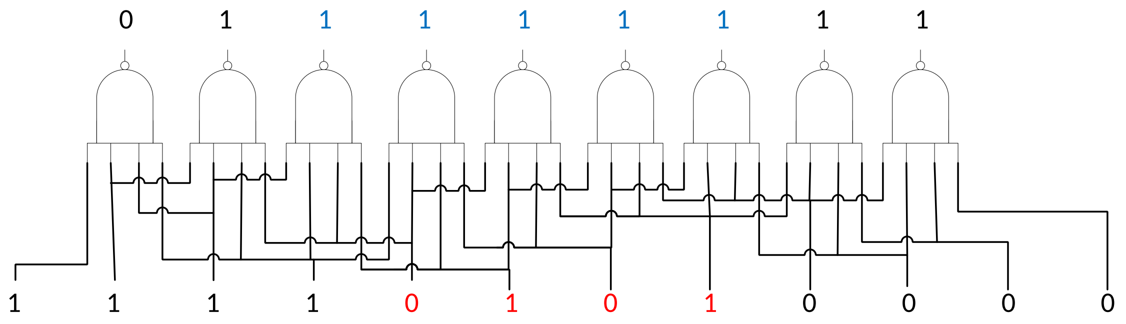


Figure 6.1: N-bit NAND bubble corrector.

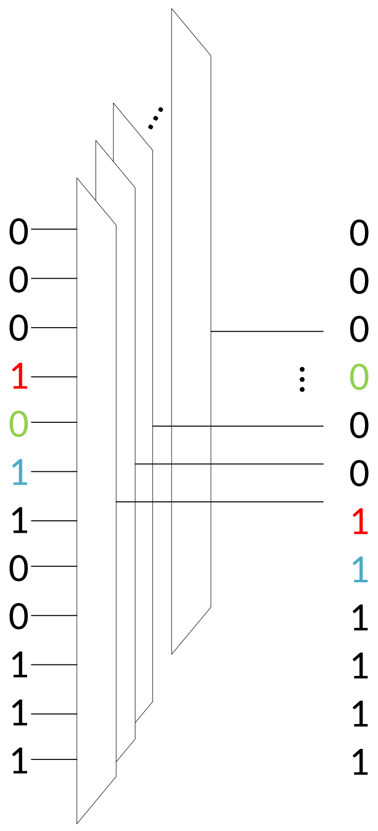


Figure 6.2: Multiplexer bubble corrector.

The rest of this chapter will be structured as follows: Section 6.3 will describe the algorithm to be implemented on the FPGA, Section 6.5 will describe the implementation details, including the pipelining of the core loop to achieve 100 MHz, zero-dead-time operation, Section 6.6 will show the results of the algorithm operating on captured codes from our DSP delay line, Section 6.7

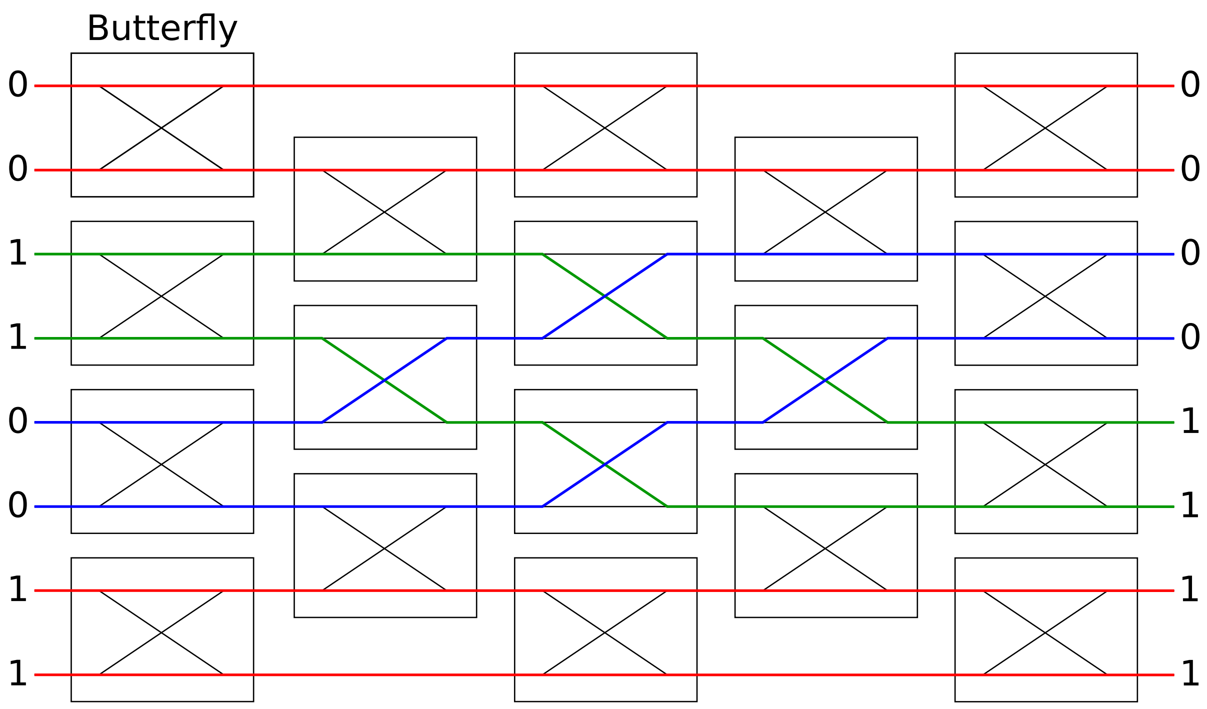


Figure 6.3: Butterfly bubble corrector.

will describe the limitations and extensibility of the algorithm and Section 6.8 will conclude the chapter.

6.3 Algorithm

The proposed algorithm consists of two components: a local ones-encoder stage and a global bin-packing stage. The local ones-encoder stage solves highly out-of-order codes within a defined range to produce monotonically increasing minicodes, while the global bin-packing stage solves for the case of ones bleeding from one local encoder to the next.

Stage 1. The Local Ones-Encoder

The first stage is to implement a ones-encoder that is no longer than half a pulse width. Implementations of ones-encoders are already well documented, with perhaps the most common and efficient method being the Wallace-tree encoder [169], so they will not be discussed here. The choice of ones encoder makes no difference to the performance of the TDC, but is dependent on

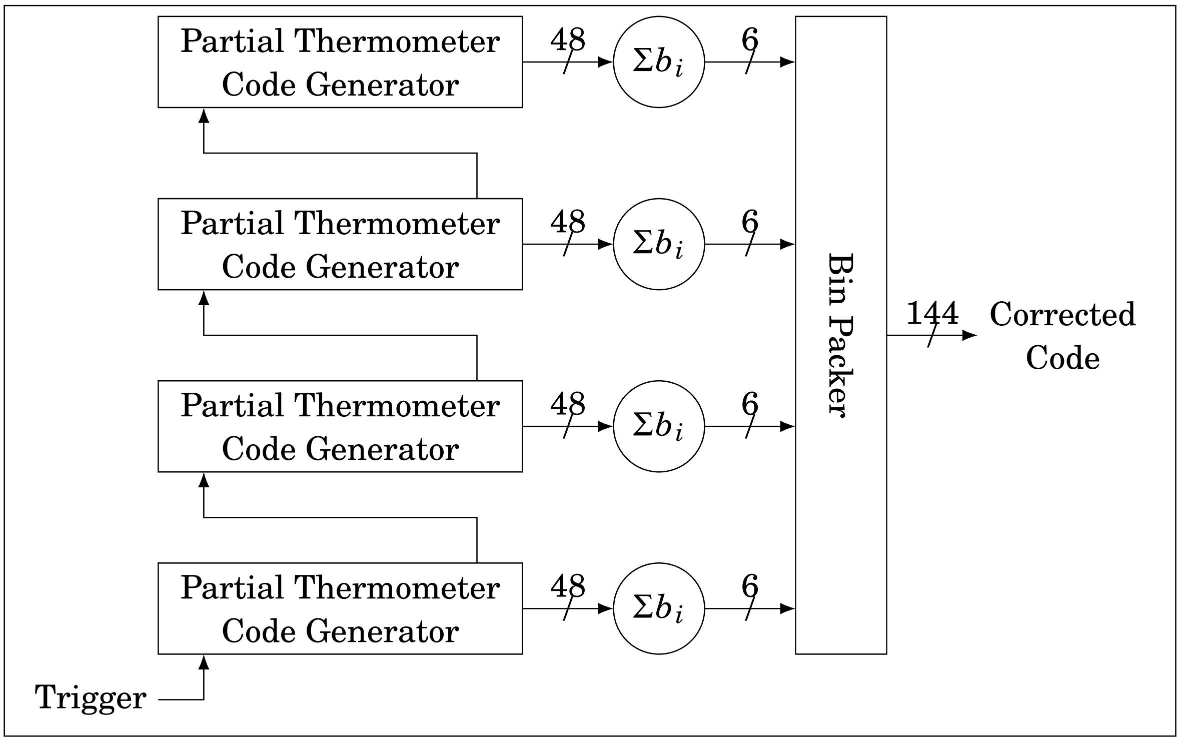

Partial Thermometer Code Generator
Partial Thermometer Code Generator
Partial Thermometer Code Generator
Partial Thermometer Code Generator
48
48
48
48
Σb_i
Σb_i
Σb_i
Σb_i
6
6
6
6
Bin Packer
144
Corrected Code
Trigger

Figure 6.4: Data input from code generator to the ones-encoders then to the bin packer.

the architecture of the device (FPGA architecture, ASIC node etc.) and will impact area and power utilisation. For our purpose, the natural delineation of ones-encoders came from the 48-bin boundary between DSP blocks, but any moderate size is acceptable.

A larger ones-encoder is preferable, as the second stage of the algorithm requires as many cycles latency as there are local ones-encoders. However, if the ones-encoders contain more than half the bins in a minimum-sized pulse, there is no guarantee that there will be a full ones-encoder between two edges, which is required for correct operation of the algorithm, hence the maximum size limitation. In our work on DSP delay lines and the wave union technique, the size of the FSR pulse injector [1] was chosen to be longer than three ones-encoders large.

Stage 2. The Global Bin-Packer

The global bin packer algorithm is a hybrid between a bin packing algorithm and an insertion sort. The kernel of the algorithm takes the data, two pointers (upper and lower) and the expected state of the bins immediately before the lower pointer, then decides on the best distribution of

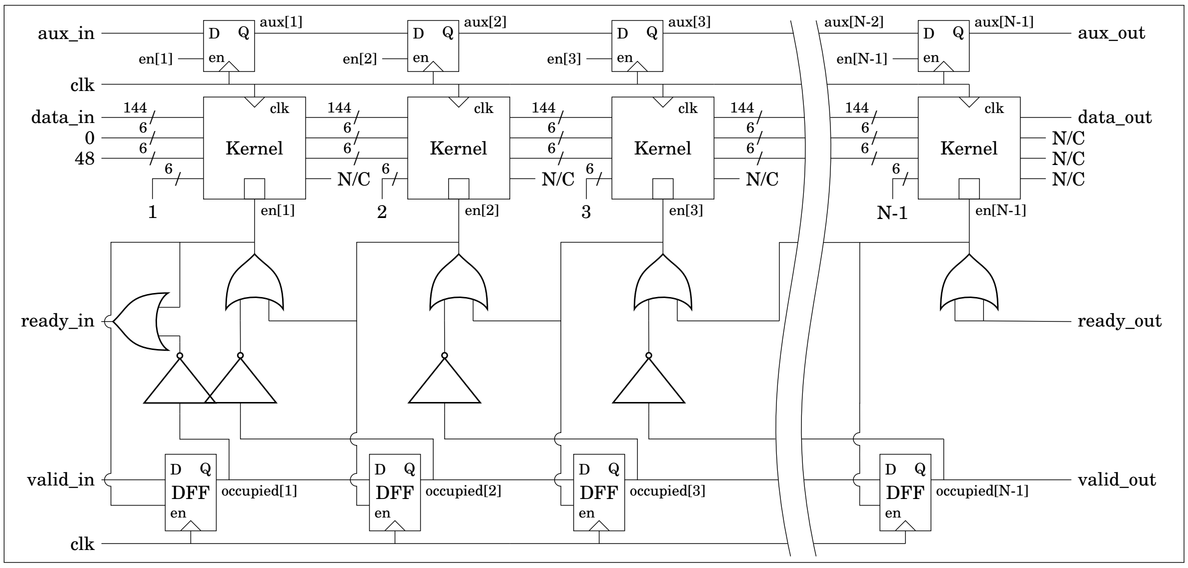


Figure 6.5: Block diagram of the bin packer top level.

data between these bins (and one other 'overflow' bin) to push ones and zeros towards the correct side of the edge. It also decides on the correct next values of the pointers.

If this kernel were iterated $N^2 - N$ times, with a reset of the pointers every N iterations (where N is the number of ones-encoders), then the algorithm would be guaranteed to compress a bubble of any size, in a similar fashion to an insertion or bubble sort. However, to keep runtime and logic requirements sensible, we suggest only performing N iterations, meaning the algorithm can only compress bubbles that span two ones-encoders. This is because the algorithm cannot move in reverse, and so when the expected value is 1s the algorithm cannot pull ones more than two minicodes away to the current location. With the fully pipelined kernel described later, this results in a runtime of $2N$ clock cycles.

Looking at the kernel in detail, we see that the action depends on the expected value of the bins just below the lower pointer. If 0s are expected, then we are expecting a $0 \rightarrow 1$ transition in the future, so we will attempt to push ones forward. Therefore, if the two codes pointed at by the upper and lower pointers have values X and Y respectively, we will attempt to put X+Y in upper code (originally contained Y). If $X + Y$ is greater than the maximum output of a ones-encoder (MAX), then we will put the maximum value in the upper code and put $X + Y - MAX$ in the code below the upper code. We set the lower code to 0 unless it is adjacent to the upper code and

$X+Y > MAX$. This can be seen in Figure 6.6.

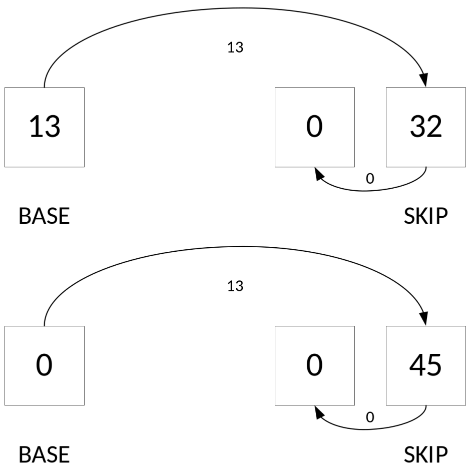


Figure 6.6: Diagram showing how the kernel pushes ones forward.

On the other hand, if the expected value of the bins below the lower pointer is 1, then we wish to pull ones back towards the transition (pushing 0s forward). In this case, $min(X+Y, MAX)$ goes into the lower pointer, 0 goes into the upper pointer unless $X+Y > MAX$ and the codes are adjacent, and $X+Y-MAX$ goes into the bin above the lower bin if $X+Y > MAX$. This can be seen in Figure 6.7.

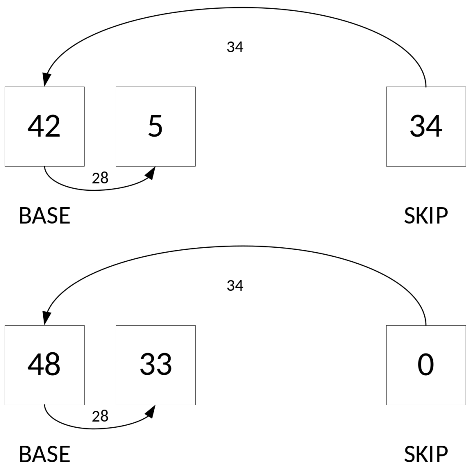


Figure 6.7: Diagram showing how the kernel pulls ones backward.

The expected value for the next iteration of the kernel is also calculated. If the expected value of the bins is 0 and the upper code is MAX, or the expected value of the bins is 1 and the upper code is 0, then the expected value is switched from 0 to 1 and 1 to 0 respectively, otherwise it remains the same. This can be seen in Figure 6.8.

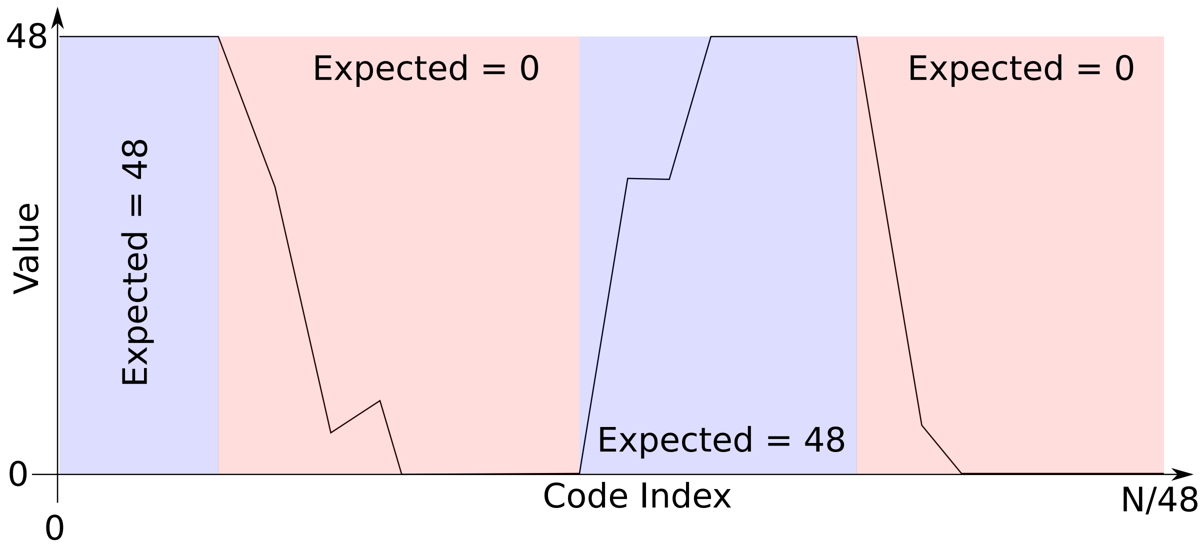


Figure 6.8: Diagram showing how the expected value changes as the algorithm iterates through the codes.

As for the next locations to look at, we define that the upper pointer must increment by 1 each iteration of the kernel, resulting in a tightly bounded runtime of N iterations. The lower pointer is dependent on the values of the codes. If the upper code is 0 or MAX, the lower pointer skips forward to the current value of the upper pointer (one below the next value of the upper pointer). If the expected value of the bins is 0s, the lower pointer will be successively emptying bins as it goes, so the lower pointer will increment by one each iteration. If the expected value of the bins is 1s, then the lower pointer will stay at the current location to receive ones pushed back by the upper pointer, except if the code it points to ‘fills up’ (reaches a value of MAX and starts overflowing to the next bin), at which point it will move forward by 1. This can be seen in Figure 6.9.

6.4 Proof of Correctness (Algorithm)

The proof of correctness for our algorithm will follow that of the bubble sort or insertion sort, but with the fundamental operation (swapping elements in bubble sort) replaced with a bit rearrangement.

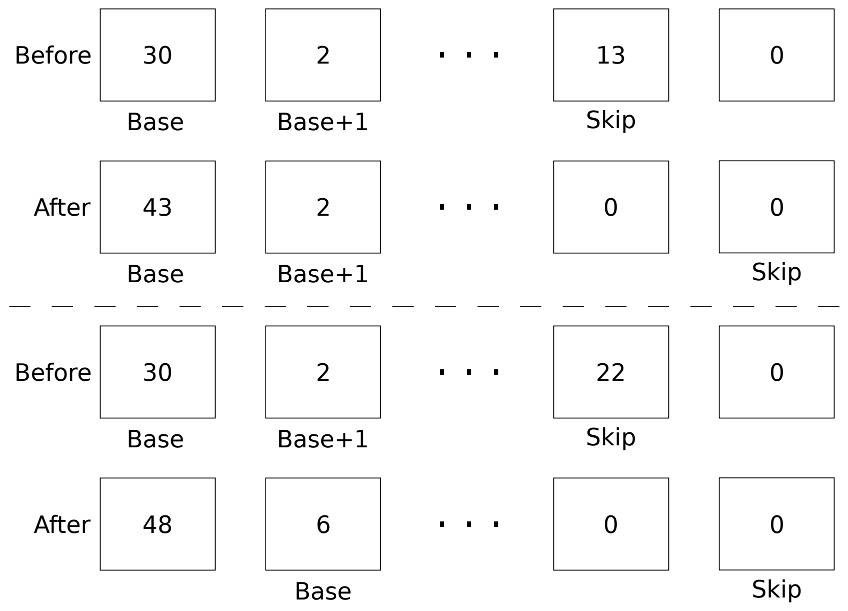


Figure 6.9: Diagram showing the advancement of the 'base' and 'skip' pointers. Skip always advances, while base advances if the current bin is full.

6.4.1 Bit Rearrangement

First, let us look at the bit rearrangement. Bit rearrangement will be an identity function (input = output) if the minicode pointed to by the lower pointer (X) is equal to 0 or MAX, or the minicode pointed to by the upper pointer (Y) is equal to the opposite of the expected value ($EXP = 0$ and $Y = MAX$ or $EXP = MAX$ and $Y = 0$). This covers the following three cases:

1. X is 0 (hence EXP must be 0); we are in a low region of our signal and there are no bits to shuffle forward to the next edge.

2. X is MAX (hence EXP must be MAX); we are in a high region of our signal and there is no space to shuffle bits back.

3. X is not 0 or MAX (covered by the other two cases) and Y is the opposite of EXP; we have encountered a 'solid' edge (no visible bubble) and nothing needs to be done (except inverting EXP).

If none of these three cases are true, we can be sure that:

1. Our lower pointer has high or low bits in it to push forward or pull back into respectively, depending on the value of EXP.

   a) If $EXP = 0$, then we have bits in X which can be pushed forward to Y.

   b) If $EXP = MAX$, then we have space in X where we can pull bits from Y back to.

2. Our upper pointer is not full of the opposite edge type.

   a) If $EXP = 0$, then Y is not full ($Y! = MAX$) hence there must be space to insert bits from X.

   b) If $EXP = MAX$, then Y is not empty ($Y! = 0$) hence there must be bits to insert into X.

If the bit arrangement is not an identity function, then we know our data is of form $\{..., Z*, EXP, X, Z+?, ..., Z+?, Y, MAX - EXP, Z*, ...\}$, where:

- Z is a random number from 1 to MAX-1 (each instance is random).
  - Z0 is a random number from 0 to MAX-1.
  - ZN is a random number from 1 to MAX.
  - Z* is a random number from 0 to MAX.
  - Z+ is a random number from 1 to MAX-1 or EXP.
  - Z- is a random number from 1 to MAX-1 or MAX-EXP.
- EXP is the expected value (either 0 or MAX).
- $MAX - EXP$ is the opposite of the expected value.
- Q? is a possible element equal to Q. In our case this shows the sequence could have any number of Z+s between X and Y.
  - From no Z+s (X and Y adjacent, $upper = lower + 1$)
  - To many Z+s (X and Y far apart, $upper = lower + k$ for some large k).
- ... is a number of elements that match those either side.

- $...,Z$ means any number of Z elements leading up to a Z.
- $Z,...$ means any number of Z elements trailing after a Z.
- $Z,...,Z$ means any number of Z elements between two Zs.
- $Z?,...Z$ means any number of Z elements before a Z (note here this allows for $X,Z?,...,Z$ since $X,...,Z$ is ambiguous).
- $Z,...,Z?$ means any number of Z elements after a Z (note this allows for $Z,...,Z?,Y$ since $Z,...,Y$ is ambiguous).
- $Z?,...,Z?$ means any number of Z elements (note this allows $X,Z?,...,Z?,Y$ since $X,...,Y$ does not imply the intermediate elements are Zs).

Given this, we can observe the effect of the bit rearrangement.

6.4.1.1 $EXP = 0$ case

If $EXP = 0$, then we calculate $Y' = X + Y$ and $X' = X + Y - MAX$. If $Y' > MAX$, then Y becomes MAX and X becomes X'. If $Y' <= MAX$, then Y becomes Y' and X becomes 0. Our data now looks like $...,Z*,0,0,Z+?,...,Z+?,Y',MAX,...$ in the case that $Y' <= MAX$, otherwise $...,Z*,0,X',Z+?,...,Z+?,MAX,MAX,...$.

If $Y' <= MAX$, then the bubble has been shrunk by at least 1 minicode, since the earliest one will be in the first Z+? or the Y'. If $Y' > MAX$, then we have also shrunk the bubble, but in the other direction. The presence of the Z+? elements is problematic, but this will be solved by the iteration.

6.4.1.2 $EXP = MAX$ case

If $EXP = MAX$, we calculate Y' and X' as before. If $Y' > MAX$, then X becomes MAX and W (= element after X) becomes X'. Y becomes 0. If Y' <= MAX, then X becomes Y' and Y becomes 0 (W is also 0). Our data now looks like $...,Z*,MAX,Y',Z+?,...,Z+?,0,Z*,...$ or $...,Z*,MAX,MAX,X',Z+?,...,Z+?,0,Z*,...$. In both cases, the element at Y has been zeroed, with the excess going to X or W. Again, the issue of the intermediate Z+ elements is problematic (perhaps more so given that one is overwritten), but this will be solved by the iteration.

6.4.2 Iteration

As we have seen in the bit rearrangement section, we can shrink a bubble using logic which moves bits from one minicode to another. The question remains, then, how this can be applied repeatedly to fully eliminate bubbles, and this is where the iteration rules are applied.

The initial condition of the algorithm is $EXP = MAX$ (the signal is expected to be high at the LSB of the delay line, we are looking for a high to low transition), with the code being $X,Y,Z+,...$. There are no Z+ elements between X and Y as they are adjacent. From here, iteration is performed as follows:

- If $Y = EXP$, move X to the location of Y and move Y forward by one. The code now looks like $Z+,EXP = X,Y,Z+,...$ (note: $Z+,EXP$ is bubble-free, EXP is MAX).
- If $Y = MAX - EXP$, move X to the location of Y, move Y forward by one and invert EXP. The code now looks like $Z+,MAX-EXP = X,Y,Z+,...$ (note: $Z+,MAX-EXP$ is bubble-free, MAX-EXP is zero).
- If $X = EXP$, move X and Y forward by one. The code now looks like $EXP,X,Y,Z+,...$ (note: EXP is MAX).
- If $X = MAX - EXP$, move X and Y forward by one and invert EXP. The code now looks like $MAX - EXP,X,Y,Z+,...$ (note: MAX-EXP is zero).
- If none of the above, move Y forward by one.
  - If $Y' > MAX$, move X forward by one. The code now looks like $MAX,Y' - MAX = X,Y,Z+,...$.
  - If $Y' <= MAX$, the code now looks like $Y' = X,0,Y,Z+,...$.

As can be observed from the possibilities shown here, the only time the size of the bubble grows is when Y advances and X does not. This only happens when $Y' < MAX$, resulting in zeros occurring between X and Y. Therefore, if $EXP = MAX$ and we start with Y being one ahead of X, we can guarantee that the only thing between X and Y will be zeros. Looking back to the bit rearrangement section, we highlighted the issue of the $Z+$ being overwritten when advancing with $EXP = MAX$. However, if $(\forall Z+),(Z+ = 0)$, then there is no issue with overwriting bits (no bits are disappearing).

We can reach the end of the current edge either by $X = MAX - EXP$ or $Y = MAX - EXP$, with $EXP = MAX$. Other than in the most basic case where EXP is inverted on the first iteration, any minicode that X is pointing to was previously pointed to by Y and therefore would have already shown the end of an edge and triggered the $Y = MAX - EXP$ case. Therefore, other than the first iteration (where Y is exactly one ahead of X), the end of an edge is always detected by $Y = MAX - EXP$ and the distance between X and Y becomes one. Therefore, we can guarantee that the form of the delay line when inverting EXP is $..., MAX, Z+?, X = 0, Y, Z+, ...$ (note: no bubble before the X).

Now considering the $EXP = 0$ case (currently looking for a low to high transition), and knowing that the initial state will be $..., Z+?, X = 0, Y, Z+, ...$, we can consider the loop again:

- If $Y = EXP$, move X to the location of Y and move Y forward by one. The new code looks like $..., Z+?, 0?, ..., 0, X = 0, Y, Z+, ...$. No bubble is present so far.
- If $Y = MAX - EXP$, move X to the location of Y, move Y forward by one and invert EXP. The new code looks like $..., Z+?, 0?, ..., 0, X = MAX, Y, Z+, ...$. No bubble present.
- If $X = EXP$, move X and Y forward by one. The new code looks like $..., Z+?, 0?, ..., 0, X, Y, Z+, ...$. Again, no bubble is present.
- If $X = MAX - EXP$, move X and Y forward by one. The new code looks like $..., Z+?, 0?, ..., 0, MAX, X, Y, Z+, ...$. Note that this cannot occur on the first iteration as $X = 0$ from previously.
- If none of the above, move Y forward by one.
  - If $Y' <= MAX$, move X forward by one. The code now looks like $..., Z+, 0, ..., 0, X, Y' = Y, Z+, ...$. The bubble has been eliminated so far.
  - If $Y' > MAX$, the code now looks like $..., Z+, 0, ..., 0?, X = Y - MAX, MAX, Y, Z+, ...$. The space in between the X and Y has increased, but only a MAX is included.

Similar to the $EXP = MAX$ case, there is only one scenario where the space between X and Y can increase, and in that case the minicodes between X and Y are filled with MAX, meaning the bubble size cannot exceed one minicode within the minicodes observed so far (everything before X is zero, everything after X is MAX). The only scenario where EXP is inverted and we return to the $EXP = MAX$ case is when $Y = MAX - EXP = MAX$. Here, we've observed one of the barrier minicodes that signals we've reached the end of an edge, and so we move X to the current location of Y ($X = MAX$ now) and Y forward by 1. This results in the distance between X and Y being reset to zero with no $Z+$ elements between them, restoring the ideal state from the start.

Because the iteration rules allow the algorithm to retain a "clean" state where the elements between X and Y can only be $MAX - EXP$ and reset the distance back to one when EXP is inverted, the issues we determined with the bit rearrangement step are no longer a problem: the bubble is fully reduced to a single minicode, and no bits are dropped due to being overwritten. Therefore, the correctness of the algorithm has been proven, provided sufficient iteration for Y to reach the end of the code.

By observation, Y is always moved forward by one. Therefore, Y will reach the end of the code after $N-1$ iterations, where N is the number of minicodes per code. After $N-1$ iterations, the bubble will be corrected.

6.5 Implementation

The ones-encoder and bin-packer were implemented in hardware. For simplicity, a binary tree adder was used for the ones-encoder (rather than the more efficient but more complicated Wallace-tree). The bin-packer was implemented as two modules, the bin_pack_block module which implemented the kernel of the bin-packer, and the bin_pack module which implemented the N-iteration loop in the form of an auto-generated pipeline of bin_pack_block modules. This can be seen in Fig. 6.5, where the bin_pack_block modules are denoted as 'Kernel' and connections to the bin_pack_block module are shown in Fig. 6.10b.

The bin_pack_block modules take in the data vector (data_in), the upper and lower pointers

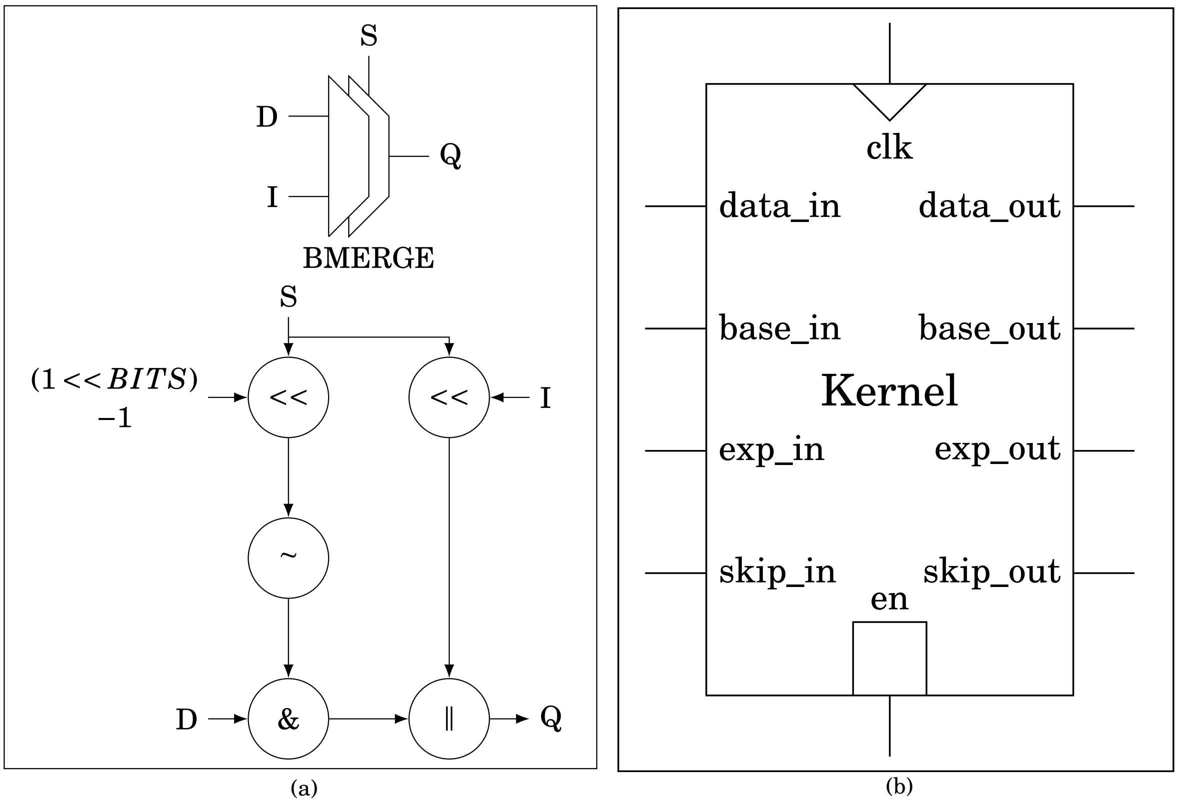


Figure 6.10: (a) Function of a BMERGE block. (b) Pin arrangement on one of the Kernel blocks.

(skip_in and base_in respectively), the expected value of bins (exp_in), an enable signal, an active-high reset (not shown) and a clock signal. The modules output the modified data vector (data_out), the new values of the upper and lower pointers (skip_out and base_out) and the new expected value of the bins (exp_out).

As the value of skip_out is always one higher than skip_in, the skip_in value is hard-coded by the bin_pack module for better logic synthesis.

The bin_pack module manages the movement of data through the pipeline using the occupied, aux and en signals. The occupied signal is a D-type Flip-Flop (DFF) which signals whether there is valid data at a particular position in the pipeline. The en signal determines whether a particular pipeline stage forwards the data to the next stage and is derived from the occupied and en values from the next stage. The aux signal allows auxiliary data to be carried in lock-step with the data to be processed, such as channel or acquisition time information (in our work on DSP delay lines, this was the value of the TDC coarse counter).

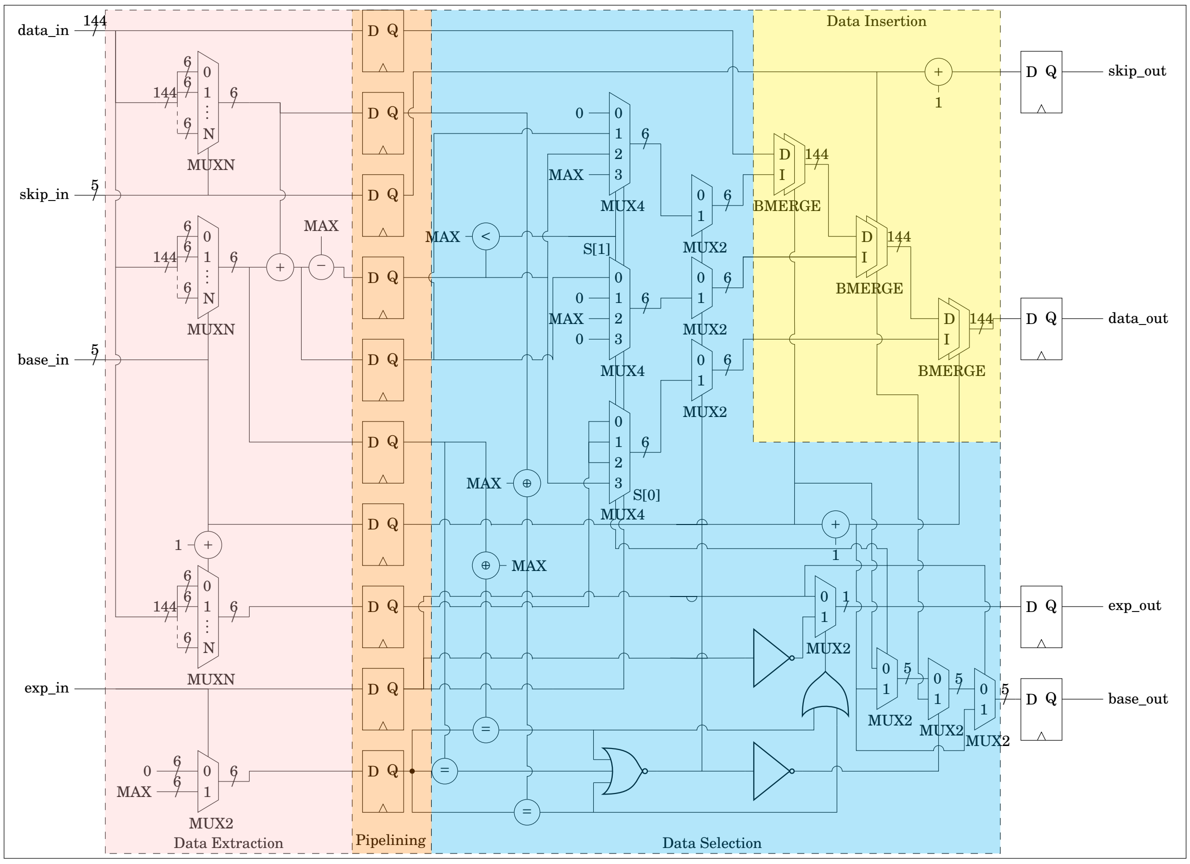


Figure 6.11: Schematic for the kernel, with extra pipeline registers.

The internals of the bin_pack_block module can be seen in Fig. 6.11. The module can be roughly split into three parts: a data extraction part, a data selection part, and a data insertion part.

The extraction part is on the left, before the optional flip-flop stage. This part extracts data from the data_in vector according to the values of base_in and skip_in. If we call the data selected by base_in ‘data_base’ and the data selected by skip_ in ‘data_skip’, then the algorithm also calculates data_base + data_skip and data_base + data_skip - MAX (MAX = 48 in our work). It also calculates the expected value of the delay line before base_in from the 1 bit signal exp_in.

This data is then optionally registered by a stage of D-type Flip-Flops (DFFs). Doing this cuts the critical path in half, resulting in significantly higher clock speeds (not quite double, as there is overhead from the flip-flop and net fanout).

The next part is the data selection part, where several multiplexers select the correct data to insert into the output data vector. This corresponds to several if statements in the source code. The conditions for the data selection are computed near the bottom diagram using the equality and less than operators, then some gates to combine the results. The outputs of the gates serve as the select inputs to the multiplexers, with the data to be inserted on each branch of the condition as the data inputs. Similar selections also occur for the exp_in and base_in signals.

The final part is data insertion. Here, the data selected earlier is inserted into the output data vector, overwriting a group of six bits. This is accomplished using the BMERGE block (Fig. 6.10a), which takes an offset at the S input and some data at the I input. The Q output vector has the bit values of I[5:0] at bit locations Q[S+5:S], while the rest of the bits are the same as the D input vector.

The outputs of the BMERGE blocks are cascaded and the final one is registered before leaving the module. The base_out and exp_out signals simply register the selected data without any merging. As skip_out is always skip_in + 1, the logic is very simple here.

By splitting the internals of the kernel into two stages, one which looks-up the required data from the input vector and calculates the required sums ($X+Y$ and $X+Y-MAX$), and another which selectively writes to the output vector, we were able to significantly reduce the latency and hence improve the clock frequency. The source code of the implementation can be found at [175].

6.6 In-Situ Results

The design was instantiated with typical parameters for our use case: 24 codes, each six bits wide with a maximum value of 48 (due to each code relating to a DSP block). This results in 144-bit input and output vectors.

With a single clock cycle kernel (the series of registers in the centre of Fig. 6.11 removed), the design takes 11 ns cycle-to-cycle on an Artix-7 FPGA (with aggressive optimisation settings), meaning it would not reach the target 100 MHz clock frequency. With a two-cycle kernel, the cycle-to-cycle latency was reduced to 7.668 ns, which could achieve a 130 MHz clock frequency if required.

The area utilisation on the XC7A200T chip used for this work was 13% of the slice LUTs (Look-

Up Tables) and 3.1% of the slice FFs, which is not small, but given that the logic is outputting 144 bits of data at 100 MHz (14.4 Gbit/s), this is well above the speed of most interfaces, and so reducing the depth of the pipeline (requiring more dead-time) is entirely possible to reduce area utilisation.

Due to non-optimal packing of logic, the number of slices occupied is 21% of the total. However, if the FPGA was pressed for space, the implementation tools would fill the gaps with other logic or the design's floorplan would be more tightly compressed, at the expense of timing efficiency.

The addition of the bubble corrector to our wave union design (Chapter 5) improved the DNL from 188 LSBs to 14.5 LSBs, a 13x improvement. Without the bubble corrector, our bubbles could extend for 2 DSP blocks (96 bins, 940 ps) length, while the result with the bubble corrector was accurate to a single bin.

Table 6.1: Sample corrections applied by the bubble corrector to the DSP wave union TDC.

Time (cycle)	State
0-13	`0, 0, 0, 1, 48, 0, 1, 47, 48, 48`
14-end	`0, 0, 0, 1, 48, 0, 0, 48, 48, 48`
0-5	`0, 0, 1, 43, 48, 0, 12, 40, 48, 48`
6-13	`0, 0, 0, 44, 48, 0, 12, 40, 48, 48`
14-end	`0, 0, 0, 44, 48, 0, 4, 48, 48, 48`
0-7	`0, 0, 48, 37, 13, 0, 44, 48, 48, 48`
8-end	`0, 0, 48, 48, 2, 0, 44, 48, 48, 48`

6.7 Limitations and Extensibility

Our design does have some significant limitations. One of the most significant is that there must always be at least one full ones-encoder between two edges, placing an upper limit on the size of the ones-encoders of half the pulse width (so that, regardless of the phase relationship between the pulse and ones-encoders, there will always be a full code).

Significant margin must also be added on top of this to ensure that the assorted ones and zeros around the edge do not spill into the guaranteed bin. In our application, nominally three codes (i.e., three DSP blocks) between edges was sufficient to ensure a full bin between each edge.

In addition, with only N iterations, the algorithm can only guarantee to deal with bubbles of two ones-encoders width. If the bubble can extend further than this (i.e., split itself over 3+ consecutive codes), more iterations of the algorithm are required to deal with the $0 \rightarrow 1$ transition case. $2N$ iterations are sufficient for bubbles split over three codes, $3N$ for four codes, etc. Repeating the algorithm for more cycles is a sufficiently simple process.

In the future, it may be appropriate to investigate operating the algorithm in two directions (top to bottom, bottom to top), requiring $2N$ iterations. We see that when pulling bins back ($1 \rightarrow 0$ transitions), the algorithm perfectly retrieves all 1s regardless of the bubble length. When iterating in the other direction, $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions swap, so these two directions should be sufficient to fix all bubbles, regardless of size.

This is somewhat similar to the bubble and insertion sort, where moving data from one end to the other is efficient but moving data in the other direction is not. Two-way bubble and insertion sorts solve this problem, and the same can be said here.

If the number of edges and maximum edge length were known in advance, then an alteration to the algorithm could be applied where it alternates between finding the next edge in the delay line and compressing the bubbles around that edge. For very long delay lines or low numbers of edges (edges are relatively sparse, or all clustered in a small area), this could significantly improve the runtime of the algorithm, and is suggested for future work.

6.8 Conclusion

In this chapter, we proposed a hardware bubble correction algorithm capable of dealing with very large bubbles in a priority code while maintaining position information, making it suitable for multiple registration and short pulse width applications. This makes it ideal for implementing wave union TDCs with highly non-linear delay lines.

The algorithm was implemented on an Artix-7 200T FPGA and utilised 13% of the LUTs and 3.1% of the registers. It achieved a worst-case cycle-to-cycle latency of 11 ns with a single-cycle kernel and 7.668 ns when fully pipelined (with a two-cycle kernel), allowing for a theoretical maximum frequency of 130 MHz. This was performed with 24 codes, each six bits wide, with a maximum value per-code of 48, meaning it was capable of processing 144 bits per cycle at the target speed of 100 MHz (14.4 Gb/s) or higher.

We have highlighted the limitations of the algorithm, including the requirement for a full ones-encoder between each edge, which provides an upper bound for the size of a code. This was shown to constrain the runtime and area of the algorithm. Methods were suggested to improve the range of the bubble compression, if required.

Appendix A

Multi-Chain Vernier TDCs

A.1 Abstract

One of the most common techniques for high-resolution time measurement (at the expense of repetition rate) is the Vernier TDC, consisting of two delay lines with a small delay difference that can be used to quantify time difference at sub-gate-delay resolutions. Other TDC architectures have shown compatibility with multi-chain measurement techniques such as multi-chain averaging and equivalent coding lines, but this has not yet been demonstrated with Vernier TDCs. In this section, we adapt multi-chain measurement to the Vernier architecture using a multiple-start scheme with a pre-offset. Using Cadence Design Systems' GPDK 180 nm simulation, we demonstrate a two-chain Vernier TDC with an LSB of 12.5 ps, half that of an equivalent single-chain Vernier TDC (25 ps), showing that the resolution may be able to be doubled in an Equivalent Coding Line scheme. A Monte-Carlo simulation is run to observe the variation between devices, and it is confirmed that the variation in the sizes of the initial offset is sufficiently small (±1.58 ps) for this scheme to work on semi-real devices. Future work would use a more realistic PDK, increase the number of chains, optimise the Vernier elements to use less silicon and power and add calibration, binary decoding, and readout.

A.2 Introduction

Time-to-Digital Converters (TDCs) are important components in frequency locking, rangefinding and nuclear applications [8, 40, 176]. Depending on the application, the acceptable trade-off between resolution, repetition rate and cost will change. For many imaging systems, the aim is to achieve a high resolution and low area or power (cost), at the expense of a lower repetition rate (which is acceptable due to the relatively low rates at which events can occur). For these systems, Vernier TDCs (VTDCs), Pulse Shrinking TDCs (PS-TDCs) and Successive Approximation TDCs (SA-TDCs) are ideal candidates as they have much lower repetition rates but much higher resolutions for a given area.

Of these, the Vernier TDC [177] is the simplest to implement, consisting of two delay lines with slightly different delays per element. The two events (start, stop) are inserted into these delay lines, with the former (start) put in the delay line with the longer delay, and the latter (stop) put in the delay line with the shorter delay. Due to the difference in delays between the two delay lines, the stop pulse will 'catch up' with and 'overtake' the start pulse. Discriminator elements in the Vernier TDC determine whether the start or stop pulse arrived at their position first, and this generates a thermometer code according to the time delay between start and stop. For two delays, T_f and T_s, where $T_f < T_s$, each stage of the Vernier TDC will decrease the delay between the start and stop signals by $T_s - T_f$, which is therefore the resolution.

Other TDC techniques such as Flash TDCs, Counter (Coarse) TDCs and Gated Ring Oscillator TDCs (GRO-TDCs) have all benefitted from multi-chain techniques [37, 178, 179]. In these schemes, multiple parallel sets of delay elements characterise the same signal simultaneously, occasionally with a pre-applied offset which varies per-chain. The individual per-chain measurements are individually decoded and then combined using a scheme such as Equivalent Coding Line (ECL) or multi-chain averaging. The differences in the quantisations by the different delay elements are exploited by the combination scheme to provide higher resolution.

However, until now, multi-chain techniques have not been applied to Vernier TDCs. Therefore, in this chapter, we will demonstrate a scheme to apply multi-chain techniques to Vernier TDCs. In Section A.3, we describe the design of the multi-chain Vernier TDC. Then, in Section A.4 we describe how we tested the TDC core and present the results in Section A.5. Finally, we conclude and suggest future work in Section A.6.

A.3 System Design

A.3.1 Overview

To demonstrate multi-chain techniques on Vernier delay lines, we determined that we would need to implement the core of the TDC up to the point of generating a thermometer code output. For this, we would need the Vernier delay elements, discriminators, and an initial offset to stagger the chains relative to each other. For the Vernier delay elements, CMOS inverters with varying channel lengths were chosen as they are simple, reliable, and relatively performant. For the discriminators, sense amplifiers were chosen as D-type flip-flops are much more sensitive to metastability conditions, have larger blackout times and have greater individual variation. As for the initial offset, Vernier elements with varying channel lengths were once again chosen as the most reliable option, considering that they could be scaled relative to the Vernier elements to maintain specific relative performance.

A.3.2 PDK

To test our implementation, we made use of Cadence Design Systems' [180] GPDK 180 nm process. This process was chosen due to its simplicity and lack of non-disclosure agreements, while still being semi-realistic. With a gate length of 180 nm, the performance of the TDC cannot be expected to compete with modern TDC designs on smaller process nodes. However, it is sufficient to demonstrate the concept of multi-chain Vernier TDC operation, which is the main purpose of this chapter.

A.3.3 Discriminators

As time sensitivity is an important metric for discriminators in TDCs, using a more advanced discriminator often causes significant improvements in linearity and resolution. Therefore, instead of using a flip-flop as a discriminator, we opted for a discriminator based on a sense amplifier as described in [181]. These discriminators offer smaller metastability periods and less variance in blackout times compared to flip-flops, which is preferable for time-sensitive circuits. The circuit schematic can be seen in Fig. A.1.

A.3.4 Vernier Elements

Once the discriminator was built, design could proceed to the Vernier element. This element should implement a very small number of taps of the delay line (one or two) and be capable of chaining with other elements.

Inverters I0 to I5 in Fig. A.2 implement the main Vernier delay chain. I0, I2, I3 and I5 are sized at NMOS=1u/260n and PMOS=2u/260n, while I1 and I4 are sized at NMOS=1u/180n and PMOS=2u/180n. The difference in channel lengths between the start chains (I0, I2, I3, I5) and the stop chain (I1, I4) cause the start signal to propagate slower than the stop signal and thus causes the two signals to converge, eventually changing the order of signals incident at the sense amplifiers and creating an edge in the thermometer code output at the corresponding position.

I6 through to I9 are discriminators (sense amplifiers) to determine whether the start or stop signal reaches the tap first. If the start signal reaches first, it outputs a logic high (‘1’), else it outputs a logic low (‘0’).

The sense amplifiers are triggered by a low-to-high transition, so for half the taps (outputs of I0 through I2), the polarity of the signal in the VDL is opposite to the value required by the discriminators. This is fixed using the inverters I10 through I12. I10 and I12 are sized with NMOS=1u/180n and PMOS=2u/180n, while I11 has double the width to account for driving two sense amplifier inputs. However, if I3 through I5 were to drive the sense amplifiers directly, the fanout would be much higher for even-numbered taps (due to the greater quantity of transistors in a sense amplifier) than odd numbered taps, degrading the linearity of the TDC. To counteract this, two inverters (I13-I18, with I13-I14 being double-sized) were added to each of I3-I5, disconnecting the sense amplifier load from the delay line. With this, the load at each tap in the delay line is always two inverters, each of them the same size as the inverter driving them.

The outputs of the Vernier elements are two bits of quantisation per chain in the delay line (d0, d1), one chain output for the stop signal and one chain output per start signal to allow

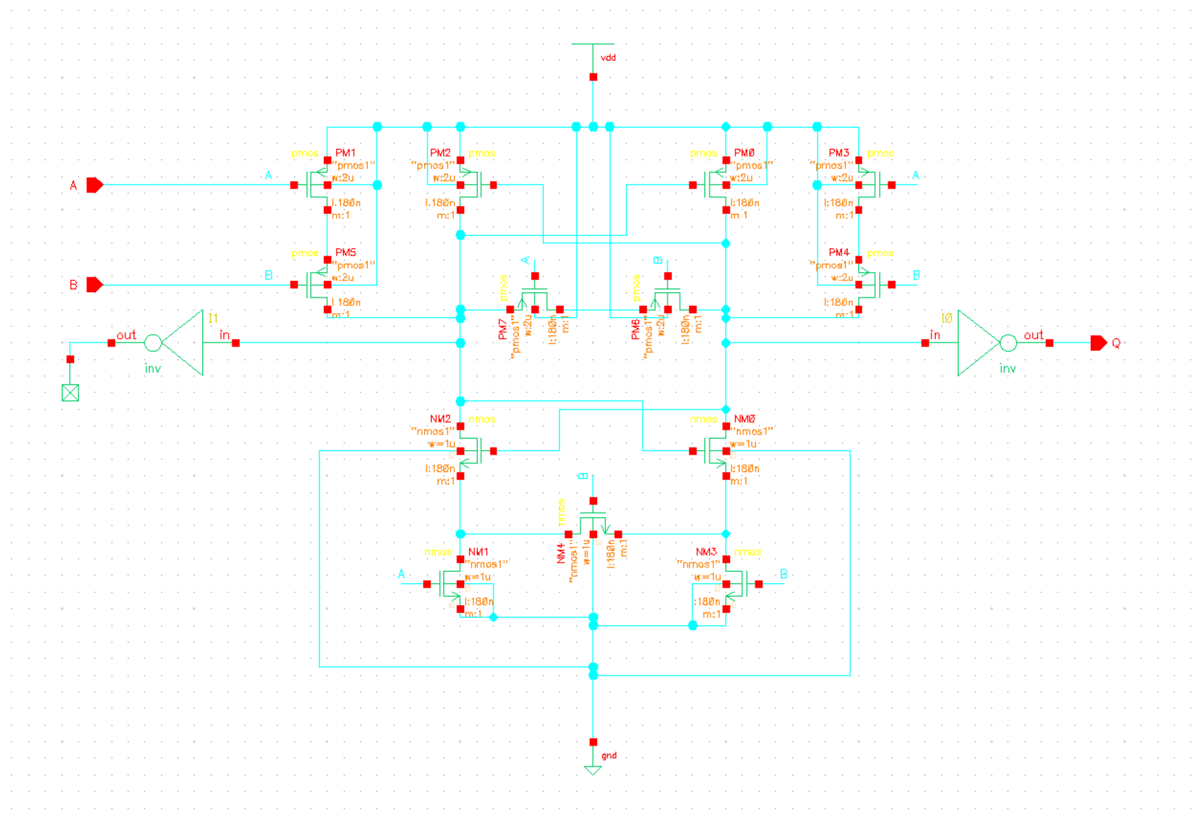


Figure A.1: Sense amplifier used as a discriminator in the TDC core. Cell name ‘sense’. The ‘inv’ cell is a simple CMOS inverter.

multiple Vernier elements to be cascaded.

A.3.5 Initial Time Offset

Once the Vernier element was complete, building the entire delay line was as simple as cascading several Vernier elements. However, to obtain multi-chain performance benefits, an offset between the initial start signals needs to be formed. For this purpose, the cluster of inverters on the left of Fig. A.5 was introduced. The inverters are implemented as pairs to avoid changing the polarity of the signal and to ensure the input drive strength is equal. The first inverter in each pair is always NMOS=1u/180n and PMOS=2u/180n, which decouples the next stage from the input and normalises the drive strength of the signal. The second inverter in each pair has a length of 180n for the stop channel and first start channel, with the length slightly longer (220n) on the second start channel. This reduces the drive strength and increases the input and output node

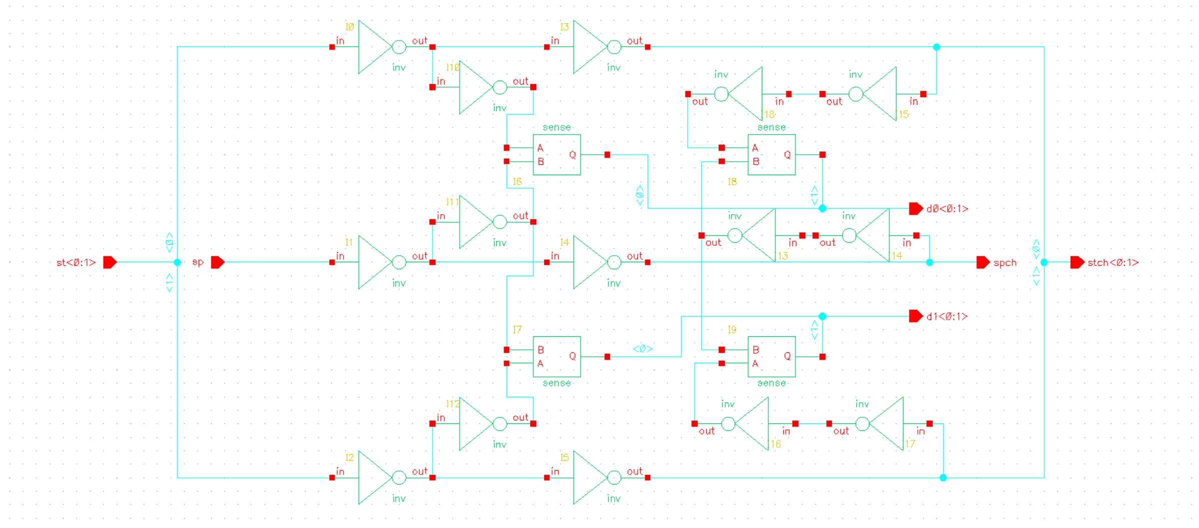


Figure A.2: Vernier element used in the TDC core. Implements two taps of the Vernier delay line. The top and bottom chains are 'start' chains, while the middle chain is the 'stop' chain.

capacitance slightly, causing a relative delay half the resolution of the single-channel equivalent Vernier delay line $((T_{slow} - T_{fast})/2)$.

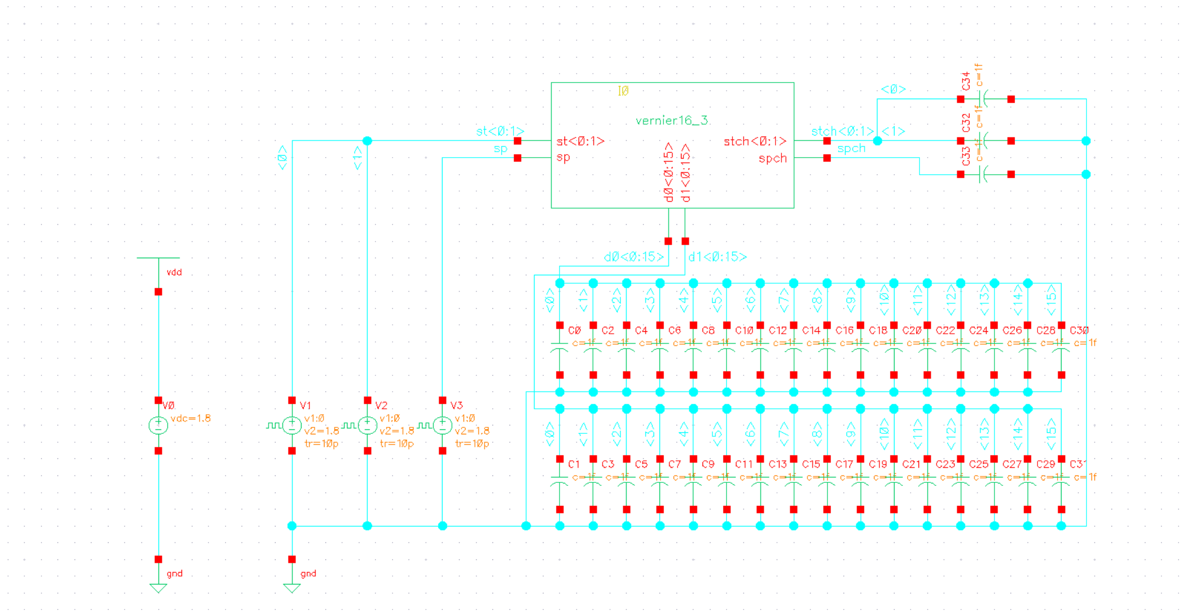
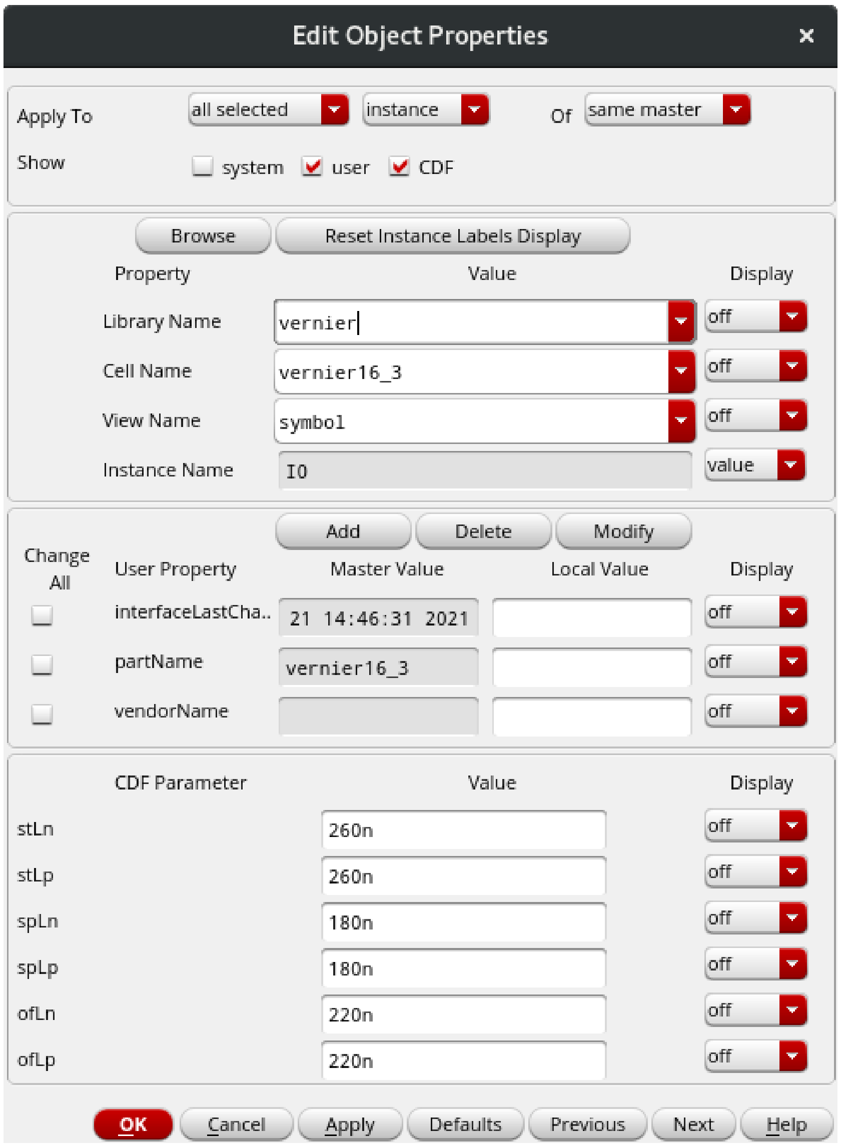


Figure A.3: Testbench used to confirm correct operation of the TDC.



Edit Object Properties

Apply To: all selected, instance; Of: same master

Show: [ ] system [✓ user] [✓ CDF]

Property	Value	Display
Library Name	vernier	off
Cell Name	vernier16_3	off
View Name	symbol	off
Instance Name	I0	value

Change All	User Property	Master Value	Local Value	Display
[ ]	interfaceLastCha..	21 14:46:31 2021		off
[ ]	partName	vernier16_3		off
[ ]	vendorName			off

CDF Parameter	Value	Display
stLn	260n	off
stLp	260n	off
spLn	180n	off
spLp	180n	off
ofLn	220n	off
ofLp	220n	off

Figure A.4: Parameters of the 16 element two-chain Vernier delay line. st = start chains, sp = stop chain, of = offset transistors on second start chain, Ln = Length of NMOS transistors, Lp = Length of PMOS trnasistors.

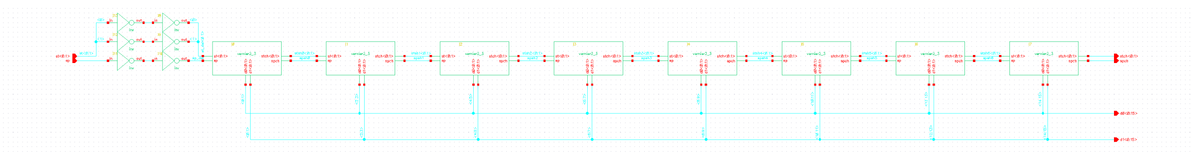


Figure A.5: Vernier delay line for 16 taps.

A.4 Test Methodology

A.4.1 Testbench

To test the TDC core, a testbench was created for simulation by Cadence's Virtuoso tool. The supply rail Vcc was supplied with 1.8 V via a DC voltage source. The two input pins, start and stop, were fed by square wave generators (vpulse) with a 10 ns period, 5 ns pulse width, 100 ps rise/fall times, 1.8 V on-voltage and 0 V (relative to the system ground) off-voltage. The delay for the start signal was set at 1 ns and the delay for the stop signal was set at 1.1 ns, 100ps retarded compared to the start signal. All output signals were connected to 1 fF loads for measurement. This is all shown in Fig. A.3

A.4.2 Model Parameters

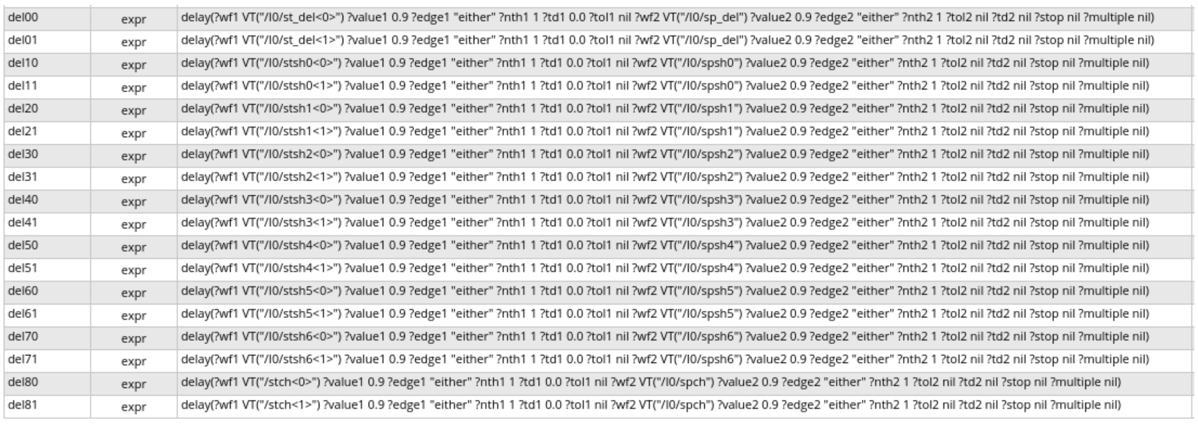
The Cadence simulation was set up for the GPDK 180 nm “reference” process node distributed by Cadence Design Systems. The simulations were performed at a temperature of 27° C with, where not otherwise specified, the width of PMOS transistors being 2 um, the width of NMOS transistors being 1 um, and the length of all transistors being 180 nm. Particular exceptions to this were the length of transistors in the ‘stop’ chain, which were 160 nm in length, and the length of the transistors which were used to offset the second start channel compared to the first, which were 220 nm in length. This is shown in Fig. A.4.

A.4.3 Typical Mean and Sweeps

The first simulation to be run was a typical mean simulation, to determine the average case performance and configure model parameters correctly. A 20 ns transient simulation was set up with the delay between the start and stop signals measured at each tap in the delay line (see Fig. A.6).

A.4.4 Monte-Carlo Analysis

Once the typical mean was shown to be operating correctly, a Monte-Carlo simulation was set up to determine the variability in the design. A 200-point Monte-Carlo simulation was set up,



del00	expr	delay(?wf1 VT("/I0/st_del<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/sp_del") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del01	expr	delay(?wf1 VT("/I0/st_del<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/sp_del") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del10	expr	delay(?wf1 VT("/I0/stsh0<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh0") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del11	expr	delay(?wf1 VT("/I0/stsh0<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh0") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del20	expr	delay(?wf1 VT("/I0/stsh1<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh1") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del21	expr	delay(?wf1 VT("/I0/stsh1<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh1") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del30	expr	delay(?wf1 VT("/I0/stsh2<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh2") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del31	expr	delay(?wf1 VT("/I0/stsh2<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh2") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del40	expr	delay(?wf1 VT("/I0/stsh3<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh3") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del41	expr	delay(?wf1 VT("/I0/stsh3<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh3") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del50	expr	delay(?wf1 VT("/I0/stsh4<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh4") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del51	expr	delay(?wf1 VT("/I0/stsh4<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh4") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del60	expr	delay(?wf1 VT("/I0/stsh5<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh5") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del61	expr	delay(?wf1 VT("/I0/stsh5<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh5") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del70	expr	delay(?wf1 VT("/I0/stsh6<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh6") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del71	expr	delay(?wf1 VT("/I0/stsh6<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spsh6") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del80	expr	delay(?wf1 VT("/stch<0>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spch") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)
del81	expr	delay(?wf1 VT("/stch<1>") ?value1 0.9 ?edge1 "either" ?nth1 1 ?td1 0.0 ?tol1 nil ?wf2 VT("/I0/spch") ?value2 0.9 ?edge2 "either" ?nth2 1 ?tol2 nil ?td2 nil ?stop nil ?multiple nil)

Figure A.6: Expressions used to determine the tap delays. delXY = Delay tap X of chain Y. Notice how two start channel taps (/I0/stshX<0> and /I0/stshX<1>) map to the same stop channel tap (/I0/spshX).

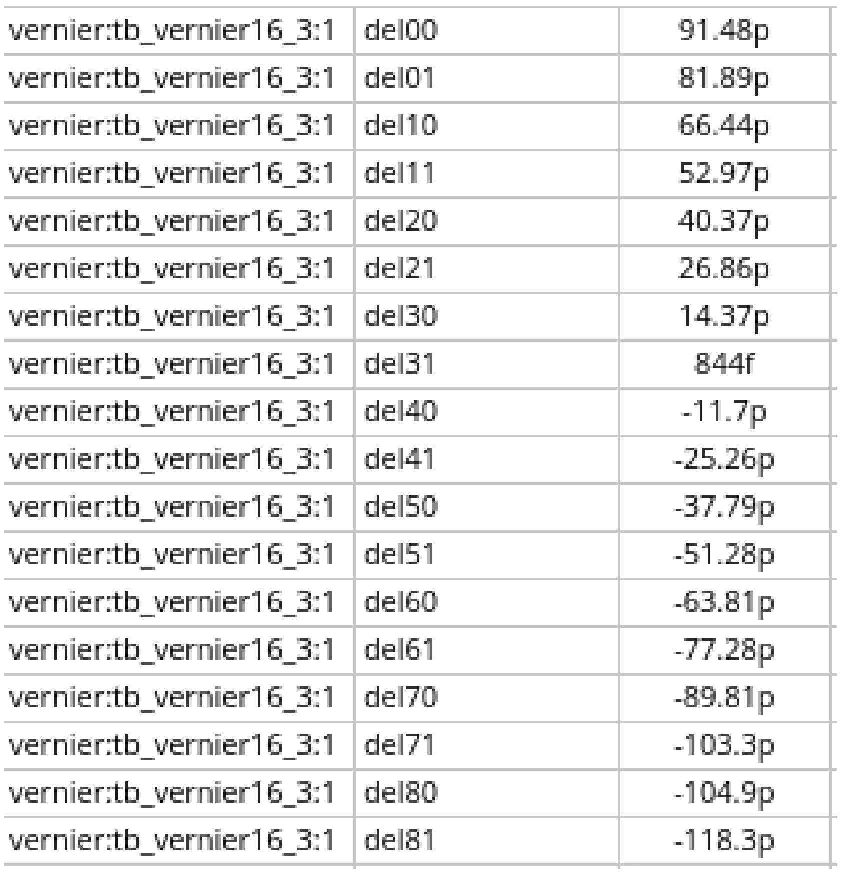
with variation in the entire delay line design being considered (not in the temperature or voltage source).

A.5 Results

A.5.1 Typical Mean and Sweeps

The results of the typical mean simulation can be seen in Fig. A.7. We see that the phase difference between the two signals inverts between delay taps 'del31', the eighth tap, and 'del40', the ninth tap. With some variability, this shows that there is approximately 12.5 ps between each tap in the combined delay line, compared to approximately 25 ps between each tap in each individual delay line, an optimal 2x improvement for two delay lines. Presuming that an accurate initial offset delay could be set up, this scheme could be extended to significantly more chains for further improvements.

One of the most significant challenges when implementing the TDC was maintaining the difference between the start chains. Even a slight mismatch between the load on the start and stop chains resulted in a drift in the LSB size which severely degraded TDC linearity and therefore performance. To counteract this, the aforementioned inverters were put in at every output of the delay line to normalise the load. However, a more appropriate solution would be to



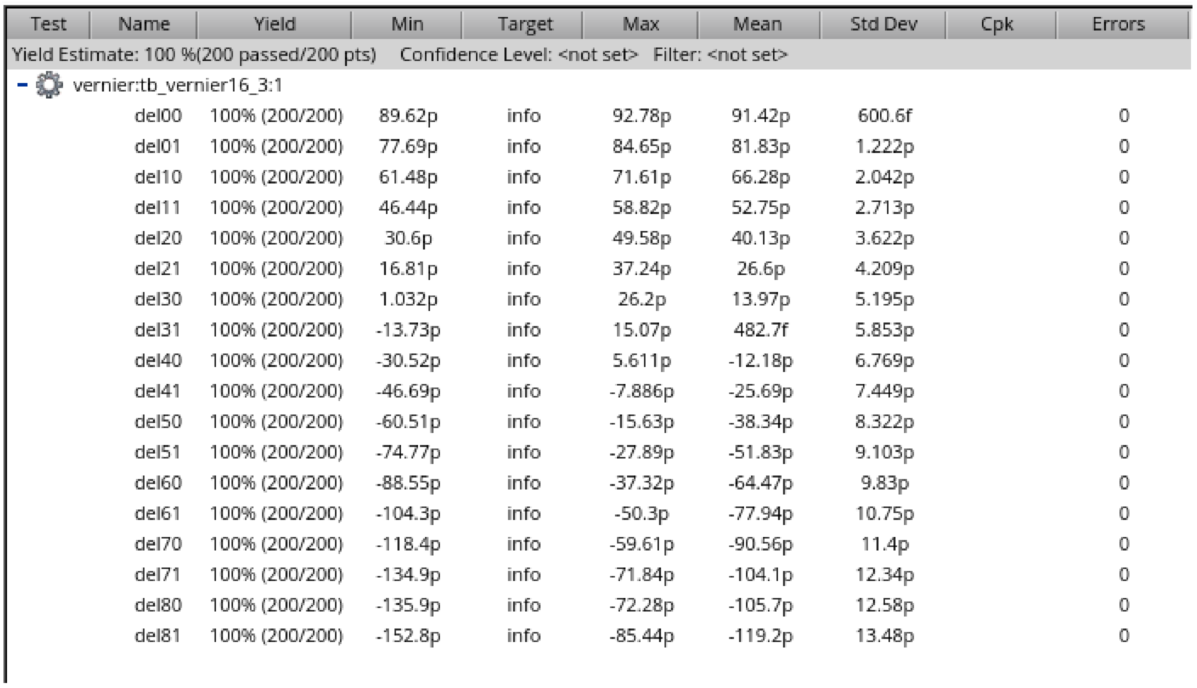
vernier:tb_vernier16_3:1	del00	91.48p
vernier:tb_vernier16_3:1	del01	81.89p
vernier:tb_vernier16_3:1	del10	66.44p
vernier:tb_vernier16_3:1	del11	52.97p
vernier:tb_vernier16_3:1	del20	40.37p
vernier:tb_vernier16_3:1	del21	26.86p
vernier:tb_vernier16_3:1	del30	14.37p
vernier:tb_vernier16_3:1	del31	844f
vernier:tb_vernier16_3:1	del40	-11.7p
vernier:tb_vernier16_3:1	del41	-25.26p
vernier:tb_vernier16_3:1	del50	-37.79p
vernier:tb_vernier16_3:1	del51	-51.28p
vernier:tb_vernier16_3:1	del60	-63.81p
vernier:tb_vernier16_3:1	del61	-77.28p
vernier:tb_vernier16_3:1	del70	-89.81p
vernier:tb_vernier16_3:1	del71	-103.3p
vernier:tb_vernier16_3:1	del80	-104.9p
vernier:tb_vernier16_3:1	del81	-118.3p

Figure A.7: Delays of the Vernier taps under typical mean scenarios. delXY = Delay tap X of chain Y.

size the delay chain elements to account for the difference. This would require significantly more effort and is not required to demonstrate multi-chain techniques on Vernier delay lines, so it was not implemented in this chapter.

A.5.2 Monte-Carlo

The Monte-Carlo simulation results can be observed in Fig. A.8. We see that over the course of 17 taps in the delay line, the length can vary from 175.1 ps (89.62 ps to -85.44 ps) to 245.6 ps (92.78 ps to -152.8 ps). This would correspond to an LSB varying from 10.30 ps to 14.45 ps. However, this could be minimised using delay-locked loop techniques to calibrate the length to a known value (e.g., 15 ps) using an external reference and current-starving transistors. The standard deviation also increases by 800 ps per stage of the delay line due to the lack of calibration.



Yield Estimate: 100 %(200 passed/200 pts) Confidence Level: <not set> Filter: <not set>

− vernier:tb_vernier16_3:1

Test	Name	Yield	Min	Target	Max	Mean	Std Dev	Cpk	Errors
	del00	100% (200/200)	89.62p	info	92.78p	91.42p	600.6f		0
	del01	100% (200/200)	77.69p	info	84.65p	81.83p	1.222p		0
	del10	100% (200/200)	61.48p	info	71.61p	66.28p	2.042p		0
	del11	100% (200/200)	46.44p	info	58.82p	52.75p	2.713p		0
	del20	100% (200/200)	30.6p	info	49.58p	40.13p	3.622p		0
	del21	100% (200/200)	16.81p	info	37.24p	26.6p	4.209p		0
	del30	100% (200/200)	1.032p	info	26.2p	13.97p	5.195p		0
	del31	100% (200/200)	-13.73p	info	15.07p	482.7f	5.853p		0
	del40	100% (200/200)	-30.52p	info	5.611p	-12.18p	6.769p		0
	del41	100% (200/200)	-46.69p	info	-7.886p	-25.69p	7.449p		0
	del50	100% (200/200)	-60.51p	info	-15.63p	-38.34p	8.322p		0
	del51	100% (200/200)	-74.77p	info	-27.89p	-51.83p	9.103p		0
	del60	100% (200/200)	-88.55p	info	-37.32p	-64.47p	9.83p		0
	del61	100% (200/200)	-104.3p	info	-50.3p	-77.94p	10.75p		0
	del70	100% (200/200)	-118.4p	info	-59.61p	-90.56p	11.4p		0
	del71	100% (200/200)	-134.9p	info	-71.84p	-104.1p	12.34p		0
	del80	100% (200/200)	-135.9p	info	-72.28p	-105.7p	12.58p		0
	del81	100% (200/200)	-152.8p	info	-85.44p	-119.2p	13.48p		0

Figure A.8: Delays of the Vernier taps under a Monte Carlo simulation.

A.6 Conclusion

In this chapter, we have demonstrated that it may be possibile to implement multi-chain techniques such as multi-chain averaging and equivalent coding lines on a Vernier TDC. Using a common stop chain and multiple start chains on Cadence's GPDK 180 nm process, we were able to create differential offsets at the start of the chains to stagger the transition time of the start chains and hence sub-divide the fundamental resolution of the Vernier TDC. Variance in transistor sizes was shown to cause a $\pm 1.58\ ps$ variability for the initial 12.5 ps differential offset on a native Vernier resolution of 25 ps, well within tolerances for multi-chain techniques. Without calibration, the LSB was shown to vary between 10.30 ps and 14.45 ps, with the average being 12.5 ps. This could be improved using the delay-locked loop techniques present in most Vernier TDCs to normalise the resolution to, e.g., 15 ps.

Our implementation of the Vernier delay line was not particularly efficient, focusing more on ensuring accuracy than chip area or power. For that purpose, we used an excessive quantity of inverters to decouple components from each other when it would have been sufficient to tune the sizes of the elements to match propagation delays. Our implementation also demonstrated only the TDC core and its proximal components; a real TDC would likely implement calibration and a thermometer-to-binary decoder in addition, but this is not directly attached to the TDC core and so would not impact performance.

Future work on this topic could take the demonstrated multi-chain technique, implement it on a more realistic PDK, extend it to four or more chains, and implement it in a full TDC core with calibration, delay-locked loop techniques, data decoding and read-out. It could then be manufactured on a real process node to demonstrate the viability in silicon. Of particular interest would be attempting to determine the limit to the resolution in the differential offsets at the start of the delay line and the correlation between the delays in parallel chains, since this determines the maximum gain that can be obtained from the multi-chain method.

Appendix B

Other Publications

B.1 Multi-Channel TDCs

A paper on multi-stop TDCs for photonics applications was published alongside the main author, Ekin Arabul [17]. I was mainly responsible for the data readout and dual-sampling aspects of this paper.

B.2 Machine Vision

I have co-authored three papers relating to machine vision. [182] is a paper on real-time stereovision on an unmanned (autonomous) airborne vehicle (UAV). [183] is a related paper on ground object tracking using templates on the same UAV platform. [184] is a paper on the use of stereovision for real-time road surface estimation in autonomous vehicles.

B.3 E-Learning

I have authored four papers related to the teaching and learning of Engineering. In [185], we discuss the challenges associated with project handover for long-running projects between Masters students, as well as the techniques we use to enable these projects and quickly induct new students. [186] goes into more detail on how students are prepared for the final-year project, including the resources we prepare beforehand to create a working base environment for the students to develop their project in.

[187] describes a real-time feedback tool for use in lectures to assist lecturers in determining engagement and understanding without interrupting the flow of the lecture. [188] extends this by suggesting a machine learning system that can digest student feedback and behaviours to determine engagement and understanding directly, without requiring the interpretation of the lecturer. This provides at-a-glance feedback on the progress of the lecture and allows the lecturer to dynamically adapt the lecture to better involve and motivate the students.

APPENDIX C

INSTANTIATION OF DSP BLOCKS

C.1 DSP48A1 Blocks in VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

library unisim;
use unisim.vcomponents.all;

entity DSP_DL is
port (
  trigger : in std_logic;
  code : out std_logic_vector(20*48-1 downto 0);
  clk : in std_logic
  );
```

```

end entity;
--A, B and D will be set to 0 so they don't affect anything.
--C will be set to 0xFFFF FFFF FFFF to enable full propagation of the carry
↪  bit.
--C Reg wil be bypassed
--Z mux will be set to act on C input.
--X mux will be set to act on 0 input.
--Carry cascade cmux will be set to CARRYIN input.
--CYI reg will be bypassed.
--CYO reg will be bypassed.
--P reg will be used to register DL state.
--CARRYOUT will be connected to next CARRYIN.

--OPMODE:
--[1:0] 0 for outputting zeros
--[3:2] 3 for using the C port
-- [4]  X (use 0)
-- [5]  X (use 0)
-- [6]  X (use 0)
-- [7]  0 for addition

--Clock Enables (CEx)
--CEP: 1 for enabling clock on PREG
--Others: 0

--Resets (RSTx):
-- None used?

```

```vhdl
architecture primitive of DSP_DL is
  signal carrys : std_logic_vector(20 downto 0);
begin


carrys(0) <= '0';

carry48: DSP48A1
  generic map (
    A0REG => 0,
    A1REG => 0,
    B0REG => 0,
    B1REG => 0,
    CARRYINREG => 0,
    CARRYINSEL => "OPMODE5",
    CARRYOUTREG => 0,
    CREG => 0,
    DREG => 0,
    MREG => 0,
    OPMODEREG => 0,
    PREG => 1,
    RSTTYPE => "ASYNC"
  ) port map (
    BCOUT => open,
    PCOUT => open,
    CARRYOUT => carrys(1),
    CARRYOUTF => open,
    M => open,
    P => code(47 downto 0),
    PCIN => X"0000_0000_0000",
    CLK => clk,
    OPMODE => "00001111",
    A => "00" & X"0000",
    B => "0" & X"0000" & trigger,
    C => X"FFFF_FFFF_FFFF",
    CARRYIN => carrys(0),
    D => "00" & X"0000",
    CEA => '0',
    CEB => '0',
    CEC => '0',
    CECARRYIN => '0',
    CED => '0',
    CEM => '0',
    CEOPMODE => '0',
    CEP => '1',
    RSTA => '0',
    RSTB => '0',
    RSTC => '0',
    RSTCARRYIN => '0',
    RSTD => '0',
    RSTM => '0',
    RSTOPMODE => '0',
    RSTP => '0'
  );

gen_dsps : for i in 1 to 19 generate
  carry48: DSP48A1
  generic map (
    A0REG => 0,
    A1REG => 0,
    B0REG => 0,
    B1REG => 0,
    CARRYINREG => 0,
    CARRYINSEL => "CARRYIN",
    CARRYOUTREG => 0,
    CREG => 0,
    DREG => 0,
    MREG => 0,
    OPMODEREG => 0,
    PREG => 1,
    RSTTYPE => "ASYNC"
  ) port map (
    BCOUT => open,
    PCOUT => open,
    CARRYOUT => carrys(i+1),
    CARRYOUTF => open,
    M => open,
    P => code((48*(i+1))-1 downto 48*i),
    PCIN => X"0000_0000_0000",
    CLK => clk,
    OPMODE => "00001100",
    A => "00" & X"0000",
    B => "00" & X"0000",
    C => X"FFFF_FFFF_FFFF",
    CARRYIN => carrys(i),
    D => "00" & X"0000",
    CEA => '0',
    CEB => '0',
    CEC => '0',
    CECARRYIN => '0',
    CED => '0',
    CEM => '0',
    CEOPMODE => '0',
    CEP => '1',
    RSTA => '0',
    RSTB => '0',
    RSTC => '0',
    RSTCARRYIN => '0',
    RSTD => '0',
    RSTM => '0',
    RSTOPMODE => '0',
    RSTP => '0'
  );
end generate;
end architecture primitive;
```

C.2 DSP48E1 Blocks in SystemVerilog

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 17.07.2017 15:18:57
// Design Name:
// Module Name: dl
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module semifine_dl #(
  parameter L_SEMIFINE = 9,
  parameter L_FINE     = 6,
  parameter SEMIFINE   = 2 ** L_SEMIFINE,
  parameter FINE       = 2  ** L_FINE
) (
  input  wire                clk     ,
  input  wire                trigger ,
  output wire [SEMIFINE-1:0] semifine,
  output wire [    FINE-1:0] fine    ,
  output wire                valid
);

  reg[SEMIFINE-1:0] r_sline_0 = 0, r_sline_1 = 0;
  reg r_valid  = 0               ;
  reg r_trig_0 = 0, r_trig_1 = 0;

  assign semifine = r_sline_1;
  assign valid    = r_valid;

  always @ (posedge clk) begin
    r_trig_1  <= r_trig_0;
    r_trig_0  <= trigger;
    r_valid   <= r_trig_0 & !r_trig_1;
    r_sline_1 <= ~r_sline_0;
    r_sline_0 <= semifine_line;
  end

  //Semifine components.
  wire[SEMIFINE-1:0] semifine_line;
  wire[SEMIFINE-1:0] cascade_carries;

  DSP48E1 #(
    // Feature Control Attributes: Data Path Selection
    .A_INPUT           ("DIRECT"        ), // Selects A input source, "DIRECT"
     ↪  (A port) or "CASCADE" (ACIN port)
    .B_INPUT           ("DIRECT"        ), // Selects B input source, "DIRECT"
     ↪  (B port) or "CASCADE" (BCIN port)
    .USE_DPORT         ("FALSE"         ), // Select D port usage (TRUE or
     ↪  FALSE)
    .USE_MULT          ("NONE"          ), // Select multiplier usage
     ↪  ("MULTIPLY", "DYNAMIC", or "NONE")
    .USE_SIMD          ("ONE48"         ), // SIMD selection ("ONE48", "TWO24",
     ↪  "FOUR12")
```

```
    // Pattern Detector Attributes: Pattern Detection Configuration
    .AUTORESET_PATDET   ("NO_RESET"      ), // "NO_RESET", "RESET_MATCH",
    ↪  "RESET_NOT_MATCH"
    .MASK               (48'hffffffffffff), // 48-bit mask value for pattern
    ↪  detect (1=ignore)
    .PATTERN            (48'h000000000000), // 48-bit pattern match for pattern
    ↪  detect
    .SEL_MASK           ("MASK"          ), // "C", "MASK", "ROUNDING_MODE1",
    ↪  "ROUNDING_MODE2"
    .SEL_PATTERN        ("PATTERN"       ), // Select pattern value ("PATTERN"
    ↪  or "C")
    .USE_PATTERN_DETECT("NO_PATDET"      ), // Enable pattern detect ("PATDET"
    ↪  or "NO_PATDET")
    // Register Control Attributes: Pipeline Register Configuration
    .ACASCREG           (0               ), // Number of pipeline stages between
    ↪  A/ACIN and ACOUT (0, 1 or 2)
    .ADREG              (0               ), // Number of pipeline stages for
    ↪  pre-adder (0 or 1)
    .ALUMODEREG         (0               ), // Number of pipeline stages for
    ↪  ALUMODE (0 or 1)
    .AREG               (0               ), // Number of pipeline stages for A
    ↪  (0, 1 or 2)
    .BCASCREG           (0               ), // Number of pipeline stages between
    ↪  B/BCIN and BCOUT (0, 1 or 2)
    .BREG               (0               ), // Number of pipeline stages for B
    ↪  (0, 1 or 2)
    .CARRYINREG         (0               ), // Number of pipeline stages for
    ↪  CARRYIN (0 or 1)
    .CARRYINSELREG     (0                ), // Number of pipeline stages for
    ↪  CARRYINSEL (0 or 1)
    .CREG              (0                ), // Number of pipeline stages for C
    ↪  (0 or 1)
    .DREG              (0                ), // Number of pipeline stages for D
    ↪  (0 or 1)
    .INMODEREG         (0                ), // Number of pipeline stages for
    ↪  INMODE (0 or 1)
    .MREG              (0                ), // Number of multiplier pipeline
    ↪  stages (0 or 1)
    .OPMODEREG         (0                ), // Number of pipeline stages for
    ↪  OPMODE (0 or 1)
    .PREG              (0                )  // Number of pipeline stages for P
    ↪  (0 or 1)
  ) DSP48E1_head (
    // Cascade: 30-bit (each) output: Cascade Ports
    //.ACOUT         (ACOUT         ), // 30-bit output: A port cascade output
    //.BCOUT         (BCOUT         ), // 18-bit output: B port cascade output
    .CARRYCASCOUT (cascade_carries[0]    ), // 1-bit output: Cascade carry
    ↪  output
    //.MULTSIGNOUT   (MULTSIGNOUT   ), // 1-bit output: Multiplier sign cascade
    ↪  output
    //.PCOUT         (      ), // 48-bit output: Cascade output
    // Control: 1-bit (each) output: Control Inputs/Status Bits
    //.OVERFLOW      (OVERFLOW              ), // 1-bit output: Overflow in
    ↪  add/acc output
    //.PATTERNBDETECT(PATTERNBDETECT        ), // 1-bit output: Pattern bar
    ↪  detect output
    //.PATTERNDETECT (PATTERNDETECT        ), // 1-bit output: Pattern detect
    ↪ output
    //.UNDERFLOW     (UNDERFLOW            ), // 1-bit output: Underflow in
    ↪ add/acc output
    // Data: 4-bit (each) output: Data Ports
    //.CARRYOUT      (CARRYOUT     ), // 4-bit output: Carry output
    .P             (semifine_line[0 +: 48]), // 48-bit output: Primary data
    ↪ output
    // Cascade: 30-bit (each) input: Cascade Ports
    .ACIN          (0                    ), // 30-bit input: A cascade data
    ↪ input
    .BCIN          (0                    ), // 18-bit input: B cascade input
    .CARRYCASCIN   (0                    ), // 1-bit input: Cascade carry
    ↪ input
    .MULTSIGNIN    (0                    ), // 1-bit input: Multiplier sign
    ↪ input
    .PCIN          (0                    ), // 48-bit input: P cascade input
    // Control: 4-bit (each) input: Control Inputs/Status Bits
    .ALUMODE       (4'h0                 ), // 4-bit input: ALU control input
    .CARRYINSEL    (3'h0                 ), // 3-bit input: Carry select input
    .CLK           (clk                  ), // 1-bit input: Clock input
    .INMODE        (4'h0                 ), // 5-bit input: INMODE control
    ↪ input
    .OPMODE        (7'h08                ), // 7-bit input: Operation mode
    ↪ input
    // Data: 30-bit (each) input: Data Ports
    .A             (0                    ), // 30-bit input: A data input
    .B             (0                    ), // 18-bit input: B data input
    .C             (0                    ), // 48-bit input: C data input
    .CARRYIN      (trigger                ), // 1-bit input: Carry input signal
    .D            (0                      ), // 25-bit input: D data input
    // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
    .CEA1         (0                      ), // 1-bit input: Clock enable input
    ↪  for 1st stage AREG
    .CEA2         (0                      ), // 1-bit input: Clock enable input
    ↪  for 2nd stage AREG
    .CEAD         (0                      ), // 1-bit input: Clock enable input
    ↪  for ADREG
    .CEALUMODE    (0                      ), // 1-bit input: Clock enable input
    ↪  for ALUMODE
    .CEB1         (0                      ), // 1-bit input: Clock enable input
    ↪  for 1st stage BREG
    .CEB2         (0                      ), // 1-bit input: Clock enable input
    ↪  for 2nd stage BREG
    .CEC          (0                      ), // 1-bit input: Clock enable input
    ↪  for CREG
    .CECARRYIN    (0                      ), // 1-bit input: Clock enable input
    ↪  for CARRYINREG
    .CECTRL       (0                      ), // 1-bit input: Clock enable input
    ↪  for OPMODEREG and CARRYINSELREG
    .CED          (0                      ), // 1-bit input: Clock enable input
    ↪  for DREG
    .CEINMODE     (0                      ), // 1-bit input: Clock enable input
    ↪  for INMODEREG
    .CEM          (0                      ), // 1-bit input: Clock enable input
    ↪  for MREG
    .CEP          (0                      ), // 1-bit input: Clock enable input
    ↪  for PREG
    .RSTA        (0                    ), // 1-bit input: Reset input for
    ↪ AREG
    .RSTALLCARRYIN(0                   ), // 1-bit input: Reset input for
    ↪ CARRYINREG
    .RSTALUMODE  (0                    ), // 1-bit input: Reset input for
    ↪ ALUMODEREG
    .RSTB        (0                    ), // 1-bit input: Reset input for
    ↪ BREG
    .RSTC        (0                    ), // 1-bit input: Reset input for
    ↪ CREG
    .RSTCTRL     (0                    ), // 1-bit input: Reset input for
    ↪ OPMODEREG and CARRYINSELREG
    .RSTD        (0                    ), // 1-bit input: Reset input for
    ↪ DREG and ADREG
    .RSTINMODE   (0                    ), // 1-bit input: Reset input for
    ↪ INMODEREG
    .RSTM        (0                    ), // 1-bit input: Reset input for
    ↪ MREG
    .RSTP        (0                    )  // 1-bit input: Reset input for
    ↪ PREG
  );

  genvar g1;
  generate
    for(g1 = 1; g1 < SEMIFINE/48; g1 = g1 + 1) begin : gen_semifines
      DSP48E1 #(
        // Feature Control Attributes: Data Path Selection
        .A_INPUT          ("DIRECT"      ), // Selects A input source,
        ↪ "DIRECT" (A port) or "CASCADE" (ACIN port)
        .B_INPUT           ("DIRECT"        ), // Selects B input source,
        ↪ "DIRECT" (B port) or "CASCADE" (BCIN port)
        .USE_DPORT         ("FALSE"         ), // Select D port usage (TRUE or
        ↪ FALSE)
        .USE_MULT          ("NONE"          ), // Select multiplier usage
        ↪ ("MULTIPLY", "DYNAMIC", or "NONE")
        .USE_SIMD          ("ONE48"         ), // SIMD selection ("ONE48",
        ↪ "TWO24", "FOUR12")
        // Pattern Detector Attributes: Pattern Detection Configuration
        .AUTORESET_PATDET  ("NO_RESET"      ), // "NO_RESET", "RESET_MATCH",
        ↪ "RESET_NOT_MATCH"
        .MASK              (48'hffffffffffff), // 48-bit mask value for pattern
        ↪ detect (1=ignore)
        .PATTERN           (48'h000000000000), // 48-bit pattern match for
        ↪ pattern detect
        .SEL_MASK          ("MASK"          ), // "C", "MASK",
        ↪ "ROUNDING_MODE1", "ROUNDING_MODE2"
        .SEL_PATTERN       ("PATTERN"       ), // Select pattern value
        ↪ ("PATTERN" or "C")
        .USE_PATTERN_DETECT("NO_PATDET"     ), // Enable pattern detect
        ↪ ("PATDET" or "NO_PATDET")
        // Register Control Attributes: Pipeline Register Configuration
        .ACASCREG          (0               ), // Number of pipeline stages
        ↪ between A/ACIN and ACOUT (0, 1 or 2)
        .ADREG             (0               ), // Number of pipeline stages for
        ↪ pre-adder (0 or 1)
        .ALUMODEREG        (0               ), // Number of pipeline stages for
        ↪ ALUMODE (0 or 1)
```

```
        .AREG           (0              ), // Number of pipeline stages for
        ↪  A (0, 1 or 2)
        .BCASCREG       (0              ), // Number of pipeline stages
        ↪  between B/BCIN and BCOUT (0, 1 or 2)
        .BREG           (0              ), // Number of pipeline stages for
        ↪  B (0, 1 or 2)
        .CARRYINREG     (0              ), // Number of pipeline stages for
        ↪  CARRYIN (0 or 1)
        .CARRYINSELREG  (0              ), // Number of pipeline stages for
        ↪  CARRYINSEL (0 or 1)
        .CREG           (0              ), // Number of pipeline stages for
        ↪  C (0 or 1)
        .DREG           (0              ), // Number of pipeline stages for
        ↪  D (0 or 1)
        .INMODEREG      (0              ), // Number of pipeline stages for
        ↪  INMODE (0 or 1)
        .MREG           (0              ), // Number of multiplier pipeline
        ↪  stages (0 or 1)
        .OPMODEREG      (0              ), // Number of pipeline stages for
        ↪  OPMODE (0 or 1)
        .PREG           (0              )  // Number of pipeline stages for
        ↪  P (0 or 1)
      ) DSP48E1_tail (
        // Cascade: 30-bit (each) output: Cascade Ports
        //.ACOUT          (ACOUT         ), // 30-bit output: A port cascade
        ↪  output
        //.BCOUT          (BCOUT         ), // 18-bit output: B port cascade
        ↪  output
        .CARRYCASCOUT (cascade_carries[g1]       ), // 1-bit output: Cascade carry output
        //.MULTSIGNOUT   (MULTSIGNOUT   ), // 1-bit output: Multiplier sign cascade output
        //.PCOUT         (      ), // 48-bit output: Cascade output
        // Control: 1-bit (each) output: Control Inputs/Status Bits
        //.OVERFLOW      (OVERFLOW                 ), // 1-bit output: Overflow in add/acc output
        //.PATTERNBDETECT(PATTERNBDETECT           ), // 1-bit output: Pattern bar detect output
        //.PATTERNDETECT (PATTERNDETECT            ), // 1-bit output: Pattern detect output
        //.UNDERFLOW     (UNDERFLOW                ), // 1-bit output: Underflow in add/acc output
        // Data: 4-bit (each) output: Data Ports
        //.CARRYOUT      (CARRYOUT      ), // 4-bit output: Carry output
        .P            (semifine_line[g1*48 +: 48]), // 48-bit output: Primary data output
        // Cascade: 30-bit (each) input: Cascade Ports
        .ACIN         (0                         ), // 30-bit input: A cascade data input
        .BCIN         (0                         ), // 18-bit input: B cascade input
        .CARRYCASCIN  (cascade_carries[g1-1]     ), // 1-bit input: Cascade carry input
        .MULTSIGNIN   (0                         ), // 1-bit input: Multiplier sign input
        .PCIN         (0                         ), // 48-bit input: P cascade input
        // Control: 4-bit (each) input: Control Inputs/Status Bits
        .ALUMODE      (4'h0                        ), // 4-bit input: ALU control
        ↪ input
        .CARRYINSEL   (3'h2                        ), // 3-bit input: Carry
        ↪ select input
        .CLK          (clk                         ), // 1-bit input: Clock
        ↪ input
        .INMODE       (4'h0                        ), // 5-bit input: INMODE
        ↪ control input
        .OPMODE       (7'h08                       ), // 7-bit input: Operation
        ↪ mode input
        // Data: 30-bit (each) input: Data Ports
        .A            (0                           ), // 30-bit input: A data
        ↪ input
        .B            (0                           ), // 18-bit input: B data
        ↪ input
        .C            (0                           ), // 48-bit input: C data
        ↪ input
        .CARRYIN      (0                           ), // 1-bit input: Carry input
        ↪ signal
        .D            (0                           ), // 25-bit input: D data
        ↪ input
        // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
        .CEA1         (0                           ), // 1-bit input: Clock
        ↪ enable input for 1st stage AREG
        .CEA2         (0                           ), // 1-bit input: Clock
        ↪ enable input for 2nd stage AREG
        .CEAD         (0                           ), // 1-bit input: Clock
        ↪ enable input for ADREG
        .CEALUMODE    (0                        ), // 1-bit input: Clock
        ↪  enable input for ALUMODE
        .CEB1         (0                        ), // 1-bit input: Clock
        ↪  enable input for 1st stage BREG
        .CEB2         (0                        ), // 1-bit input: Clock
        ↪  enable input for 2nd stage BREG
        .CEC          (0                        ), // 1-bit input: Clock
        ↪  enable input for CREG
        .CECARRYIN    (0                        ), // 1-bit input: Clock
        ↪  enable input for CARRYINREG
        .CECTRL       (0                        ), // 1-bit input: Clock
        ↪  enable input for OPMODEREG and CARRYINSELREG
        .CED          (0                        ), // 1-bit input: Clock
        ↪  enable input for DREG
        .CEINMODE     (0                        ), // 1-bit input: Clock
        ↪  enable input for INMODEREG
        .CEM          (0                        ), // 1-bit input: Clock
        ↪  enable input for MREG
        .CEP          (0                        ), // 1-bit input: Clock
        ↪  enable input for PREG
        .RSTA         (0                        ), // 1-bit input: Reset input
        ↪  for AREG
        .RSTALLCARRYIN(0                        ), // 1-bit input: Reset input
        ↪  for CARRYINREG
        .RSTALUMODE   (0                        ), // 1-bit input: Reset input
        ↪  for ALUMODEREG
        .RSTB         (0                        ), // 1-bit input: Reset input
        ↪  for BREG
        .RSTC         (0                        ), // 1-bit input: Reset input
         ↪ for CREG
        .RSTCTRL      (0                        ), // 1-bit input: Reset input
         ↪ for OPMODEREG and CARRYINSELREG
        .RSTD         (0                        ), // 1-bit input: Reset input
         ↪ for DREG and ADREG
        .RSTINMODE    (0                        ), // 1-bit input: Reset input
         ↪ for INMODEREG
        .RSTM         (0                        ), // 1-bit input: Reset input
         ↪ for MREG
        .RSTP         (0                        )  // 1-bit input: Reset input
         ↪ for PREG
      );
    end // gen_semifines
  endgenerate


endmodule // dl
```

C.3 DSP48E1 Wave Union in SystemVerilog

C.3.1 Pulse Injector

```
`timescale 1ns / 1ps

module dsp_inj #(
  parameter W_HEAD        = 2             ,
  parameter L_HEAD        = 3             ,
  parameter S_HEAD        = 48            ,
  parameter A_HEAD        = L_HEAD * W_HEAD,
  parameter HEAD          = A_HEAD * S_HEAD,
  parameter FIRST_ALU     = 4'h2           ,
  parameter INV_HEAD_ALU  = 4'h2           ,
  parameter PROP_HEAD_ALU = 4'h2
) (
  input  wire            clk     ,
  input  wire            trigger ,
  output wire [HEAD-1:0] line    ,
  output wire            pulse
);

  wire[A_HEAD-1:0] cascade_carries;
  assign pulse = cascade_carries[A_HEAD-1];

  genvar g2;
  genvar g3;

  generate
    for(g2 = 0; g2 < L_HEAD; g2 += 1) begin : gen_head
      if(g2 == 0) begin : first_dsp48
        DSP48E1 #(
          // Feature Control Attributes: Data Path Selection
          .A_INPUT          ("DIRECT"        ), // Selects A input source,
          ↪  "DIRECT" (A port) or "CASCADE" (ACIN port)
          .B_INPUT          ("DIRECT"        ), // Selects B input source,
          ↪  "DIRECT" (B port) or "CASCADE" (BCIN port)
          .USE_DPORT        ("FALSE"         ), // Select D port usage (TRUE
          ↪  or FALSE)
          .USE_MULT          ("NONE"          ), // Select multiplier usage
          ↪ ("MULTIPLY", "DYNAMIC", or "NONE")
          .USE_SIMD          ("ONE48"         ), // SIMD selection ("ONE48",
          ↪ "TWO24", "FOUR12")
          // Pattern Detector Attributes: Pattern Detection Configuration
          .AUTORESET_PATDET  ("NO_RESET"      ), // "NO_RESET", "RESET_MATCH",
          ↪ "RESET_NOT_MATCH"
          .MASK              (48'hffffffffffff), // 48-bit mask value for
          ↪ pattern detect (1=ignore)
          .PATTERN           (48'h000000000000), // 48-bit pattern match for
          ↪ pattern detect
          .SEL_MASK          ("MASK"          ), // "C", "MASK",
          ↪ "ROUNDING_MODE1", "ROUNDING_MODE2"
          .SEL_PATTERN       ("PATTERN"       ), // Select pattern value
          ↪ ("PATTERN" or "C")
          .USE_PATTERN_DETECT("NO_PATDET"     ), // Enable pattern detect
          ↪ ("PATDET" or "NO_PATDET")
          // Register Control Attributes: Pipeline Register Configuration
          .ACASCREG          (0               ), // Number of pipeline stages
          ↪ between A/ACIN and ACOUT (0, 1 or 2)
          .ADREG             (0               ), // Number of pipeline stages
          ↪ for pre-adder (0 or 1)
          .ALUMODEREG        (0               ), // Number of pipeline stages
          ↪ for ALUMODE (0 or 1)
          .AREG              (0               ), // Number of pipeline stages
          ↪ for A (0, 1 or 2)
          .BCASCREG          (0               ), // Number of pipeline stages
          ↪ between B/BCIN and BCOUT (0, 1 or 2)
```

```
      .BREG              (0                ), // Number of pipeline stages
      ↪ for B (0, 1 or 2)
      .CARRYINREG        (0                ), // Number of pipeline stages
      ↪ for CARRYIN (0 or 1)
      .CARRYINSELREG     (1                ), // Number of pipeline stages
      ↪ for CARRYINSEL (0 or 1)
      .CREG              (0                ), // Number of pipeline stages
      ↪ for C (0 or 1)
      .DREG              (0                ), // Number of pipeline stages
      ↪ for D (0 or 1)
      .INMODEREG         (0                ), // Number of pipeline stages
      ↪ for INMODE (0 or 1)
      .MREG              (0                ), // Number of multiplier
      ↪ pipeline stages (0 or 1)
      .OPMODEREG         (1                ), // Number of pipeline stages
      ↪ for OPMODE (0 or 1)
      .PREG              (0                )  // Number of pipeline stages
      ↪ for P (0 or 1)
    ) DSP48E1_first (
      // Cascade: 30-bit (each) output: Cascade Ports
      //.ACOUT          (ACOUT          ), // 30-bit output: A port cascade
      ↪ output
      //.BCOUT          (BCOUT          ), // 18-bit output: B port cascade
      ↪ output
      .CARRYCASCOUT (cascade_carries[0]), // 1-bit output: Cascade carry
      ↪ output
      //.MULTSIGNOUT    (MULTSIGNOUT    ), // 1-bit output: Multiplier sign
      ↪ cascade output
      //.PCOUT          (       ), // 48-bit output: Cascade output
```

```
            // Control: 1-bit (each) output: Control Inputs/Status Bits
            //.OVERFLOW      (OVERFLOW              ), // 1-bit output: Overflow
             ↪ in add/acc output
            //.PATTERNBDETECT(PATTERNBDETECT        ), // 1-bit output: Pattern
             ↪ bar detect output
            //.PATTERNDETECT (PATTERNDETECT         ), // 1-bit output: Pattern
             ↪ detect output
            //.UNDERFLOW     (UNDERFLOW             ), // 1-bit output: Underflow
             ↪ in add/acc output
            // Data: 4-bit (each) output: Data Ports
            //.CARRYOUT      (CARRYOUT      ), // 4-bit output: Carry output
            .P             (line[0 +: S_HEAD] ), // 48-bit output: Primary data
             ↪ output
            // Cascade: 30-bit (each) input: Cascade Ports
            .ACIN          (0                 ), // 30-bit input: A cascade data
             ↪ input
            .BCIN          (0                 ), // 18-bit input: B cascade input
            .CARRYCASCIN   (0                 ), // 1-bit input: Cascade carry
             ↪ input
            .MULTSIGNIN    (0                 ), // 1-bit input: Multiplier sign
             ↪ input
            .PCIN          (0                 ), // 48-bit input: P cascade input
            // Control: 4-bit (each) input: Control Inputs/Status Bits
            .ALUMODE       (FIRST_ALU         ), // 4-bit input: ALU control
             ↪ input
            .CARRYINSEL    (3'h0              ), // 3-bit input: Carry select
             ↪ input
            .CLK           (clk               ), // 1-bit input: Clock input
            .INMODE       (5'h0               ), // 5-bit input: INMODE control
            ↪  input
            .OPMODE       (7'h3B              ), // 7-bit input: Operation mode
            ↪  input
            // Data: 30-bit (each) input: Data Ports
            .A            (30'h3FFFFFFF       ), // 30-bit input: A data input
            .B            (18'h3FFFF          ), // 18-bit input: B data input
            .C            ({47'h0,trigger}    ), // 48-bit input: C data input
            .CARRYIN      (1                  ), // 1-bit input: Carry input
            ↪  signal
            .D            (0                  ), // 25-bit input: D data input
            // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
            .CEA1         (0                  ), // 1-bit input: Clock enable
            ↪  input for 1st stage AREG
            .CEA2         (0                  ), // 1-bit input: Clock enable
            ↪  input for 2nd stage AREG
            .CEAD         (0                  ), // 1-bit input: Clock enable
            ↪  input for ADREG
            .CEALUMODE    (0                  ), // 1-bit input: Clock enable
            ↪  input for ALUMODE
            .CEB1         (0                  ), // 1-bit input: Clock enable
            ↪  input for 1st stage BREG
            .CEB2         (0                  ), // 1-bit input: Clock enable
            ↪  input for 2nd stage BREG
            .CEC          (0                  ), // 1-bit input: Clock enable
            ↪  input for CREG
            .CECARRYIN    (0                  ), // 1-bit input: Clock enable
            ↪  input for CARRYINREG
```

```
        .CECTRL       (1                  ), // 1-bit input: Clock enable
         ↪ input for OPMODEREG and CARRYINSELREG
        .CED          (0                  ), // 1-bit input: Clock enable
         ↪ input for DREG
        .CEINMODE     (0                  ), // 1-bit input: Clock enable
         ↪ input for INMODEREG
        .CEM          (0                  ), // 1-bit input: Clock enable
         ↪ input for MREG
        .CEP          (0                  ), // 1-bit input: Clock enable
         ↪ input for PREG
        .RSTA         (0                  ), // 1-bit input: Reset input for
         ↪ AREG
        .RSTALLCARRYIN(0                  ), // 1-bit input: Reset input for
         ↪ CARRYINREG
        .RSTALUMODE   (0                  ), // 1-bit input: Reset input for
         ↪ ALUMODEREG
        .RSTB         (0                  ), // 1-bit input: Reset input for
         ↪ BREG
        .RSTC         (0                  ), // 1-bit input: Reset input for
         ↪ CREG
        .RSTCTRL      (0                  ), // 1-bit input: Reset input for
         ↪ OPMODEREG and CARRYINSELREG
        .RSTD         (0                  ), // 1-bit input: Reset input for
         ↪ DREG and ADREG
        .RSTINMODE    (0                  ), // 1-bit input: Reset input for
         ↪ INMODEREG
        .RSTM         (0                  ), // 1-bit input: Reset input for
         ↪ MREG
```

```
      .RSTP         (0                  )  // 1-bit input: Reset input for
      ↪  PREG
    );
  end // first_dsp48
  else begin : other_dsp48s
    DSP48E1 #(
      // Feature Control Attributes: Data Path Selection
      .A_INPUT           ("DIRECT"        ), // Selects A input source,
      ↪  "DIRECT" (A port) or "CASCADE" (ACIN port)
      .B_INPUT           ("DIRECT"        ), // Selects B input source,
      ↪  "DIRECT" (B port) or "CASCADE" (BCIN port)
      .USE_DPORT         ("FALSE"         ), // Select D port usage (TRUE
      ↪  or FALSE)
      .USE_MULT          ("NONE"          ), // Select multiplier usage
      ↪  ("MULTIPLY", "DYNAMIC", or "NONE")
      .USE_SIMD          ("ONE48"         ), // SIMD selection ("ONE48",
      ↪  "TWO24", "FOUR12")
      // Pattern Detector Attributes: Pattern Detection Configuration
      .AUTORESET_PATDET  ("NO_RESET"      ), // "NO_RESET", "RESET_MATCH",
      ↪  "RESET_NOT_MATCH"
      .MASK              (48'hffffffffffff), // 48-bit mask value for
      ↪  pattern detect (1=ignore)
      .PATTERN           (48'h000000000000), // 48-bit pattern match for
      ↪  pattern detect
      .SEL_MASK          ("MASK"          ), // "C", "MASK",
      ↪  "ROUNDING_MODE1", "ROUNDING_MODE2"
      .SEL_PATTERN       ("PATTERN"       ), // Select pattern value
      ↪  ("PATTERN" or "C")
```

```
        .USE_PATTERN_DETECT("NO_PATDET"     ), // Enable pattern detect
        ↪  ("PATDET" or "NO_PATDET")
        // Register Control Attributes: Pipeline Register Configuration
        .ACASCREG          (0                ), // Number of pipeline stages
        ↪  between A/ACIN and ACOUT (0, 1 or 2)
        .ADREG             (0                ), // Number of pipeline stages
        ↪  for pre-adder (0 or 1)
        .ALUMODEREG        (0                ), // Number of pipeline stages
        ↪  for ALUMODE (0 or 1)
        .AREG              (0                ), // Number of pipeline stages
        ↪  for A (0, 1 or 2)
        .BCASCREG          (0                ), // Number of pipeline stages
        ↪  between B/BCIN and BCOUT (0, 1 or 2)
        .BREG              (0                ), // Number of pipeline stages
        ↪  for B (0, 1 or 2)
        .CARRYINREG        (0                ), // Number of pipeline stages
        ↪  for CARRYIN (0 or 1)
        .CARRYINSELREG     (1                ), // Number of pipeline stages
        ↪  for CARRYINSEL (0 or 1)
        .CREG              (0                ), // Number of pipeline stages
        ↪  for C (0 or 1)
        .DREG              (0                ), // Number of pipeline stages
        ↪  for D (0 or 1)
        .INMODEREG         (0                ), // Number of pipeline stages
        ↪  for INMODE (0 or 1)
        .MREG              (0                ), // Number of multiplier
        ↪  pipeline stages (0 or 1)
        .OPMODEREG         (1                ), // Number of pipeline stages
        ↪  for OPMODE (0 or 1)
```

```
      .PREG              (0               )  // Number of pipeline stages
      ↪  for P (0 or 1)
    ) DSP48E1_inv_head (
      // Cascade: 30-bit (each) output: Cascade Ports
      //.ACOUT         (ACOUT         ), // 30-bit output: A port cascade
      ↪  output
      //.BCOUT         (BCOUT         ), // 18-bit output: B port cascade
      ↪  output
      .CARRYCASCOUT  (cascade_carries[g2*W_HEAD]      ), // 1-bit output:
      ↪  Cascade carry output
      //.MULTSIGNOUT   (MULTSIGNOUT   ), // 1-bit output: Multiplier sign
      ↪  cascade output
      //.PCOUT         (      ), // 48-bit output: Cascade output
      // Control: 1-bit (each) output: Control Inputs/Status Bits
      //.OVERFLOW      (OVERFLOW              ), // 1-bit output: Overflow
      ↪  in add/acc output
      //.PATTERNBDETECT(PATTERNBDETECT        ), // 1-bit output: Pattern
      ↪  bar detect output
      //.PATTERNDETECT (PATTERNDETECT         ), // 1-bit output: Pattern
      ↪  detect output
      //.UNDERFLOW     (UNDERFLOW             ), // 1-bit output: Underflow
      ↪  in add/acc output
      // Data: 4-bit (each) output: Data Ports
      //.CARRYOUT      (CARRYOUT      ), // 4-bit output: Carry output
      .P             (line[g2*W_HEAD*S_HEAD +: S_HEAD]), // 48-bit output:
      ↪  Primary data output
      // Cascade: 30-bit (each) input: Cascade Ports
      .ACIN          (0                               ), // 30-bit input: A
      ↪  cascade data input
      .BCIN         (0                              ), // 18-bit input: B
      ↪ cascade input
      .CARRYCASCIN  (cascade_carries[g2*W_HEAD-1]   ), // 1-bit input:
      ↪ Cascade carry input
      .MULTSIGNIN   (0                              ), // 1-bit input:
      ↪ Multiplier sign input
      .PCIN         (0                              ), // 48-bit input: P
      ↪ cascade input
      // Control: 4-bit (each) input: Control Inputs/Status Bits
      .ALUMODE      ((g2[0])?4'h0:INV_HEAD_ALU      ), // 4-bit input: ALU
      ↪ control input
      .CARRYINSEL   (3'h2                           ), // 3-bit input:
      ↪ Carry select input
      .CLK          (clk                            ), // 1-bit input:
      ↪ Clock input
      .INMODE       (5'h0                           ), // 5-bit input:
      ↪ INMODE control input
      .OPMODE       (7'h3B                          ), // 7-bit input:
      ↪ Operation mode input
      // Data: 30-bit (each) input: Data Ports
      .A            (30'h3FFFFFFF                   ), // 30-bit input: A
      ↪ data input
      .B            (18'h3FFFF                      ), // 18-bit input: B
      ↪ data input
      .C            ({47'h0,trigger}                ), // 48-bit input: C
      ↪ data input
      .CARRYIN      (0                              ), // 1-bit input:
      ↪ Carry input signal
```

```
          .D            (0                                      ), // 25-bit input: D
          ↪  data input
          // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
          .CEA1         (0                                      ), // 1-bit input:
          ↪  Clock enable input for 1st stage AREG
          .CEA2         (0                                      ), // 1-bit input:
          ↪  Clock enable input for 2nd stage AREG
          .CEAD         (0                                      ), // 1-bit input:
          ↪  Clock enable input for ADREG
          .CEALUMODE    (0                                      ), // 1-bit input:
          ↪  Clock enable input for ALUMODE
          .CEB1         (0                                      ), // 1-bit input:
          ↪  Clock enable input for 1st stage BREG
          .CEB2         (0                                      ), // 1-bit input:
          ↪  Clock enable input for 2nd stage BREG
          .CEC          (0                                      ), // 1-bit input:
          ↪  Clock enable input for CREG
          .CECARRYIN    (0                                      ), // 1-bit input:
          ↪  Clock enable input for CARRYINREG
          .CECTRL       (1                                      ), // 1-bit input:
          ↪  Clock enable input for OPMODEREG and CARRYINSELREG
          .CED          (0                                      ), // 1-bit input:
          ↪  Clock enable input for DREG
          .CEINMODE     (0                                      ), // 1-bit input:
          ↪  Clock enable input for INMODEREG
          .CEM          (0                                      ), // 1-bit input:
          ↪  Clock enable input for MREG
          .CEP          (0                                      ), // 1-bit input:
          ↪  Clock enable input for PREG
```

```
        .RSTA         (0                                  ), // 1-bit input:
        ↪  Reset input for AREG
        .RSTALLCARRYIN(0                                  ), // 1-bit input:
        ↪  Reset input for CARRYINREG
        .RSTALUMODE   (0                                  ), // 1-bit input:
        ↪  Reset input for ALUMODEREG
        .RSTB         (0                                  ), // 1-bit input:
        ↪  Reset input for BREG
        .RSTC         (0                                  ), // 1-bit input:
        ↪  Reset input for CREG
        .RSTCTRL      (0                                  ), // 1-bit input:
        ↪  Reset input for OPMODEREG and CARRYINSELREG
        .RSTD         (0                                  ), // 1-bit input:
        ↪  Reset input for DREG and ADREG
        .RSTINMODE    (0                                  ), // 1-bit input:
        ↪  Reset input for INMODEREG
        .RSTM         (0                                  ), // 1-bit input:
        ↪  Reset input for MREG
        .RSTP         (0                                  )  // 1-bit input:
        ↪  Reset input for PREG
      );
    end // other_dsp48s
    for(g3 = 1; g3 < W_HEAD; g3 += 1) begin : gen_head_prop
      DSP48E1 #(
        // Feature Control Attributes: Data Path Selection
        .A_INPUT           ("DIRECT"        ), // Selects A input source,
        ↪  "DIRECT" (A port) or "CASCADE" (ACIN port)
        .B_INPUT           ("DIRECT"        ), // Selects B input source,
        ↪  "DIRECT" (B port) or "CASCADE" (BCIN port)
```

```
           .USE_DPORT          ("FALSE"          ), // Select D port usage (TRUE
           ↪  or FALSE)
           .USE_MULT           ("NONE"           ), // Select multiplier usage
           ↪  ("MULTIPLY", "DYNAMIC", or "NONE")
           .USE_SIMD           ("ONE48"          ), // SIMD selection ("ONE48",
           ↪  "TWO24", "FOUR12")
           // Pattern Detector Attributes: Pattern Detection Configuration
           .AUTORESET_PATDET   ("NO_RESET"       ), // "NO_RESET", "RESET_MATCH",
           ↪  "RESET_NOT_MATCH"
           .MASK               (48'hffffffffffff), // 48-bit mask value for
           ↪  pattern detect (1=ignore)
           .PATTERN            (48'h000000000000), // 48-bit pattern match for
           ↪  pattern detect
           .SEL_MASK           ("MASK"           ), // "C", "MASK",
           ↪  "ROUNDING_MODE1", "ROUNDING_MODE2"
           .SEL_PATTERN        ("PATTERN"        ), // Select pattern value
           ↪  ("PATTERN" or "C")
           .USE_PATTERN_DETECT("NO_PATDET"      ), // Enable pattern detect
           ↪  ("PATDET" or "NO_PATDET")
           // Register Control Attributes: Pipeline Register Configuration
           .ACASCREG           (0                ), // Number of pipeline stages
           ↪  between A/ACIN and ACOUT (0, 1 or 2)
           .ADREG              (0                ), // Number of pipeline stages
           ↪  for pre-adder (0 or 1)
           .ALUMODEREG         (0                ), // Number of pipeline stages
           ↪  for ALUMODE (0 or 1)
           .AREG               (0                ), // Number of pipeline stages
           ↪  for A (0, 1 or 2)
```

```
        .BCASCREG         (0                ), // Number of pipeline stages
         ↪ between B/BCIN and BCOUT (0, 1 or 2)
        .BREG             (0                ), // Number of pipeline stages
         ↪ for B (0, 1 or 2)
        .CARRYINREG       (0                ), // Number of pipeline stages
         ↪ for CARRYIN (0 or 1)
        .CARRYINSELREG    (1                ), // Number of pipeline stages
         ↪ for CARRYINSEL (0 or 1)
        .CREG             (0                ), // Number of pipeline stages
         ↪ for C (0 or 1)
        .DREG             (0                ), // Number of pipeline stages
         ↪ for D (0 or 1)
        .INMODEREG        (0                ), // Number of pipeline stages
         ↪ for INMODE (0 or 1)
        .MREG             (0                ), // Number of multiplier
         ↪ pipeline stages (0 or 1)
        .OPMODEREG        (1                ), // Number of pipeline stages
         ↪ for OPMODE (0 or 1)
        .PREG             (0                )  // Number of pipeline stages
         ↪ for P (0 or 1)
      ) DSP48E1_prop_head (
        // Cascade: 30-bit (each) output: Cascade Ports
        //.ACOUT          (ACOUT         ), // 30-bit output: A port cascade
         ↪ output
        //.BCOUT          (BCOUT         ), // 18-bit output: B port cascade
         ↪ output
        .CARRYCASCOUT (cascade_carries[g2*W_HEAD + g3]      ), // 1-bit
         ↪ output: Cascade carry output
        //.MULTSIGNOUT   (MULTSIGNOUT   ), // 1-bit output: Multiplier sign
        ↪  cascade output
        //.PCOUT         (        ), // 48-bit output: Cascade output
        // Control: 1-bit (each) output: Control Inputs/Status Bits
        //.OVERFLOW      (OVERFLOW             ), // 1-bit output: Overflow
        ↪  in add/acc output
        //.PATTERNBDETECT(PATTERNBDETECT       ), // 1-bit output: Pattern
        ↪  bar detect output
        //.PATTERNDETECT (PATTERNDETECT        ), // 1-bit output: Pattern
        ↪  detect output
        //.UNDERFLOW     (UNDERFLOW            ), // 1-bit output: Underflow
        ↪  in add/acc output
        // Data: 4-bit (each) output: Data Ports
        //.CARRYOUT      (CARRYOUT      ), // 4-bit output: Carry output
        .P             (line[(g2*W_HEAD+g3)*S_HEAD +: S_HEAD]), // 48-bit
        ↪  output: Primary data output
        // Cascade: 30-bit (each) input: Cascade Ports
        .ACIN          (0                                  ), // 30-bit
        ↪  input: A cascade data input
        .BCIN          (0                                  ), // 18-bit
        ↪  input: B cascade input
        .CARRYCASCIN   (cascade_carries[g2*W_HEAD + g3-1]    ), // 1-bit
        ↪  input: Cascade carry input
        .MULTSIGNIN    (0                                  ), // 1-bit
        ↪  input: Multiplier sign input
        .PCIN          (0                                  ), // 48-bit
        ↪  input: P cascade input
        // Control: 4-bit (each) input: Control Inputs/Status Bits
        .ALUMODE       ((g2[0])?PROP_HEAD_ALU:4'h0             ), // 4-bit
         ↪  input: ALU control input
        .CARRYINSEL    (3'h2                                   ), // 3-bit
         ↪  input: Carry select input
        .CLK           (clk                                    ), // 1-bit
         ↪  input: Clock input
        .INMODE        (5'h0                                   ), // 5-bit
         ↪  input: INMODE control input
        .OPMODE        (7'h08                                  ), // 7-bit
         ↪  input: Operation mode input
        // Data: 30-bit (each) input: Data Ports
        .A             (0                                      ), // 30-bit
         ↪  input: A data input
        .B             (0                                      ), // 18-bit
         ↪  input: B data input
        .C             (0                                      ), // 48-bit
         ↪  input: C data input
        .CARRYIN       (0                                      ), // 1-bit
         ↪  input: Carry input signal
        .D             (0                                      ), // 25-bit
         ↪  input: D data input
        // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
        .CEA1          (0                                      ), // 1-bit
         ↪  input: Clock enable input for 1st stage AREG
        .CEA2          (0                                      ), // 1-bit
         ↪  input: Clock enable input for 2nd stage AREG
        .CEAD          (0                                      ), // 1-bit
         ↪  input: Clock enable input for ADREG
```

```
    .CEALUMODE    (0                                     ), // 1-bit
    ↪ input: Clock enable input for ALUMODE
    .CEB1         (0                                     ), // 1-bit
    ↪ input: Clock enable input for 1st stage BREG
    .CEB2         (0                                     ), // 1-bit
    ↪ input: Clock enable input for 2nd stage BREG
    .CEC          (0                                     ), // 1-bit
    ↪ input: Clock enable input for CREG
    .CECARRYIN    (0                                     ), // 1-bit
    ↪ input: Clock enable input for CARRYINREG
    .CECTRL       (1                                     ), // 1-bit
    ↪ input: Clock enable input for OPMODEREG and CARRYINSELREG
    .CED          (0                                     ), // 1-bit
    ↪ input: Clock enable input for DREG
    .CEINMODE     (0                                     ), // 1-bit
    ↪ input: Clock enable input for INMODEREG
    .CEM          (0                                     ), // 1-bit
    ↪ input: Clock enable input for MREG
    .CEP          (0                                     ), // 1-bit
    ↪ input: Clock enable input for PREG
    .RSTA         (0                                     ), // 1-bit
    ↪ input: Reset input for AREG
    .RSTALLCARRYIN(0                                     ), // 1-bit
    ↪ input: Reset input for CARRYINREG
    .RSTALUMODE   (0                                     ), // 1-bit
    ↪ input: Reset input for ALUMODEREG
    .RSTB         (0                                     ), // 1-bit
    ↪ input: Reset input for BREG
```

```
          .RSTC         (0                                    ), // 1-bit
           ↪  input: Reset input for CREG
          .RSTCTRL      (0                                    ), // 1-bit
           ↪  input: Reset input for OPMODEREG and CARRYINSELREG
          .RSTD         (0                                    ), // 1-bit
           ↪  input: Reset input for DREG and ADREG
          .RSTINMODE    (0                                    ), // 1-bit
           ↪  input: Reset input for INMODEREG
          .RSTM         (0                                    ), // 1-bit
           ↪  input: Reset input for MREG
          .RSTP         (0                                    )  // 1-bit
           ↪  input: Reset input for PREG
        );
      end // gen_head_prop
    end // gen_head
  endgenerate

endmodule // dsp_inj
```

C.3.2 Delay Line

```
`timescale 1ns / 1ps

module dsp_prop #(
  parameter L_TAIL   = 20             ,
  parameter CASCIN   = 0              ,
  parameter S_TAIL   = 48             ,
  parameter TAIL     = S_TAIL * L_TAIL,
  parameter TAIL_ALU = 4'h0           ,
  parameter LOC_NEXT = "DSP48_X0Y42"
```

```
) (
  input  wire              clk  ,
  input  wire              pulse,
  output wire [TAIL-1:0] line
);

  wire[L_TAIL:0] cascade_carries;
  assign cascade_carries[0] = pulse;

  genvar g1;
  generate
    for(g1 = 0; g1 < L_TAIL; g1 = g1 + 1) begin : gen_tail
      if(g1 == 0 && CASCIN == 0) begin : first_tail
        (* LOC=LOC_NEXT, DONT_TOUCH="true" *)
        DSP48E1 #(
          // Feature Control Attributes: Data Path Selection
          .A_INPUT          ("DIRECT"        ), // Selects A input source,
          ↪ "DIRECT" (A port) or "CASCADE" (ACIN port)
          .B_INPUT          ("DIRECT"        ), // Selects B input source,
          ↪ "DIRECT" (B port) or "CASCADE" (BCIN port)
          .USE_DPORT        ("FALSE"         ), // Select D port usage (TRUE
          ↪ or FALSE)
          .USE_MULT         ("NONE"          ), // Select multiplier usage
          ↪ ("MULTIPLY", "DYNAMIC", or "NONE")
          .USE_SIMD         ("ONE48"         ), // SIMD selection ("ONE48",
          ↪ "TWO24", "FOUR12")
          // Pattern Detector Attributes: Pattern Detection Configuration
          .AUTORESET_PATDET ("NO_RESET"      ), // "NO_RESET", "RESET_MATCH",
          ↪ "RESET_NOT_MATCH"
          .MASK              (48'hffffffffffff), // 48-bit mask value for pattern detect (1=ignore)
          .PATTERN           (48'h000000000000), // 48-bit pattern match for pattern detect
          .SEL_MASK          ("MASK"           ), // "C", "MASK", "ROUNDING_MODE1", "ROUNDING_MODE2"
          .SEL_PATTERN       ("PATTERN"        ), // Select pattern value ("PATTERN" or "C")
          .USE_PATTERN_DETECT("NO_PATDET"      ), // Enable pattern detect ("PATDET" or "NO_PATDET")
          // Register Control Attributes: Pipeline Register Configuration
          .ACASCREG          (0                ), // Number of pipeline stages between A/ACIN and ACOUT (0, 1 or 2)
          .ADREG             (0                ), // Number of pipeline stages for pre-adder (0 or 1)
          .ALUMODEREG        (0                ), // Number of pipeline stages for ALUMODE (0 or 1)
          .AREG              (0                ), // Number of pipeline stages for A (0, 1 or 2)
          .BCASCREG          (0                ), // Number of pipeline stages between B/BCIN and BCOUT (0, 1 or 2)
          .BREG              (0                ), // Number of pipeline stages for B (0, 1 or 2)
          .CARRYINREG        (0                ), // Number of pipeline stages for CARRYIN (0 or 1)
          .CARRYINSELREG     (1                ), // Number of pipeline stages for CARRYINSEL (0 or 1)
          .CREG              (0                ), // Number of pipeline stages for C (0 or 1)
```

```
      .DREG            (0              ), // Number of pipeline stages
      ↪  for D (0 or 1)
      .INMODEREG       (0              ), // Number of pipeline stages
      ↪  for INMODE (0 or 1)
      .MREG            (0              ), // Number of multiplier
      ↪  pipeline stages (0 or 1)
      .OPMODEREG       (1              ), // Number of pipeline stages
      ↪  for OPMODE (0 or 1)
      .PREG            (0              )  // Number of pipeline stages
      ↪  for P (0 or 1)
    ) DSP48E1_first (
      // Cascade: 30-bit (each) output: Cascade Ports
      //.ACOUT          (ACOUT         ), // 30-bit output: A port cascade
      ↪  output
      //.BCOUT          (BCOUT         ), // 18-bit output: B port cascade
      ↪  output
      .CARRYCASCOUT (cascade_carries[1]), // 1-bit output: Cascade carry
      ↪  output
      //.MULTSIGNOUT    (MULTSIGNOUT   ), // 1-bit output: Multiplier sign
      ↪  cascade output
      //.PCOUT          (      ), // 48-bit output: Cascade output
      // Control: 1-bit (each) output: Control Inputs/Status Bits
      //.OVERFLOW       (OVERFLOW                   ), // 1-bit output:
      ↪  Overflow in add/acc output
      //.PATTERNBDETECT(PATTERNBDETECT              ), // 1-bit output:
      ↪  Pattern bar detect output
      //.PATTERNDETECT (PATTERNDETECT               ), // 1-bit output:
      ↪  Pattern detect output
```

```
        //.UNDERFLOW     (UNDERFLOW                ), // 1-bit output:
         ↪ Underflow in add/acc output
        // Data: 4-bit (each) output: Data Ports
        //.CARRYOUT      (CARRYOUT      ), // 4-bit output: Carry output
        .P             (line[0 +: S_TAIL]  ), // 48-bit output: Primary data
         ↪ output
        // Cascade: 30-bit (each) input: Cascade Ports
        .ACIN          (0                  ), // 30-bit input: A cascade data
         ↪ input
        .BCIN          (0                  ), // 18-bit input: B cascade input
        .CARRYCASCIN   (0                  ), // 1-bit input: Cascade carry
         ↪ input
        .MULTSIGNIN    (0                  ), // 1-bit input: Multiplier sign
         ↪ input
        .PCIN          (0                  ), // 48-bit input: P cascade input
        // Control: 4-bit (each) input: Control Inputs/Status Bits
        .ALUMODE       (TAIL_ALU           ), // 4-bit input: ALU control
         ↪ input
        .CARRYINSEL    (3'h0               ), // 3-bit input: Carry select
         ↪ input
        .CLK           (clk                ), // 1-bit input: Clock input
        .INMODE        (5'h0               ), // 5-bit input: INMODE control
         ↪ input
        .OPMODE        (7'h08              ), // 7-bit input: Operation mode
         ↪ input
        // Data: 30-bit (each) input: Data Ports
        .A             (0                  ), // 30-bit input: A data input
        .B             (0                  ), // 18-bit input: B data input
        .C             (0                  ), // 48-bit input: C data input
        .CARRYIN      (cascade_carries[0]), // 1-bit input: Carry input
           ↪ signal
        .D            (0                 ), // 25-bit input: D data input
        // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
        .CEA1         (0                 ), // 1-bit input: Clock enable
           ↪ input for 1st stage AREG
        .CEA2         (0                 ), // 1-bit input: Clock enable
           ↪ input for 2nd stage AREG
        .CEAD         (0                 ), // 1-bit input: Clock enable
           ↪ input for ADREG
        .CEALUMODE    (0                 ), // 1-bit input: Clock enable
           ↪ input for ALUMODE
        .CEB1         (0                 ), // 1-bit input: Clock enable
           ↪ input for 1st stage BREG
        .CEB2         (0                 ), // 1-bit input: Clock enable
           ↪ input for 2nd stage BREG
        .CEC          (0                 ), // 1-bit input: Clock enable
           ↪ input for CREG
        .CECARRYIN    (0                 ), // 1-bit input: Clock enable
           ↪ input for CARRYINREG
        .CECTRL       (1                 ), // 1-bit input: Clock enable
           ↪ input for OPMODEREG and CARRYINSELREG
        .CED          (0                 ), // 1-bit input: Clock enable
           ↪ input for DREG
        .CEINMODE     (0                 ), // 1-bit input: Clock enable
           ↪ input for INMODEREG
        .CEM          (0                 ), // 1-bit input: Clock enable
           ↪ input for MREG
        .CEP          (0                  ), // 1-bit input: Clock enable
        ↪  input for PREG
        .RSTA         (0                  ), // 1-bit input: Reset input for
        ↪  AREG
        .RSTALLCARRYIN(0                  ), // 1-bit input: Reset input for
        ↪  CARRYINREG
        .RSTALUMODE   (0                  ), // 1-bit input: Reset input for
        ↪  ALUMODEREG
        .RSTB         (0                  ), // 1-bit input: Reset input for
        ↪  BREG
        .RSTC         (0                  ), // 1-bit input: Reset input for
        ↪  CREG
        .RSTCTRL      (0                  ), // 1-bit input: Reset input for
        ↪  OPMODEREG and CARRYINSELREG
        .RSTD         (0                  ), // 1-bit input: Reset input for
        ↪  DREG and ADREG
        .RSTINMODE    (0                  ), // 1-bit input: Reset input for
        ↪  INMODEREG
        .RSTM         (0                  ), // 1-bit input: Reset input for
        ↪  MREG
        .RSTP         (0                  )  // 1-bit input: Reset input for
        ↪  PREG
      );
    end // first_tail
    else begin : subsequent_tails
      DSP48E1 #(
        // Feature Control Attributes: Data Path Selection
        .A_INPUT           ("DIRECT"         ), // Selects A input source,
        ↪  "DIRECT" (A port) or "CASCADE" (ACIN port)
```

```
          .B_INPUT          ("DIRECT"        ), // Selects B input source,
          ↪  "DIRECT" (B port) or "CASCADE" (BCIN port)
          .USE_DPORT        ("FALSE"         ), // Select D port usage (TRUE
          ↪  or FALSE)
          .USE_MULT         ("NONE"          ), // Select multiplier usage
          ↪  ("MULTIPLY", "DYNAMIC", or "NONE")
          .USE_SIMD         ("ONE48"         ), // SIMD selection ("ONE48",
          ↪  "TWO24", "FOUR12")
          // Pattern Detector Attributes: Pattern Detection Configuration
          .AUTORESET_PATDET ("NO_RESET"      ), // "NO_RESET", "RESET_MATCH",
          ↪  "RESET_NOT_MATCH"
          .MASK             (48'hffffffffffff), // 48-bit mask value for
          ↪  pattern detect (1=ignore)
          .PATTERN          (48'h000000000000), // 48-bit pattern match for
          ↪  pattern detect
          .SEL_MASK         ("MASK"          ), // "C", "MASK",
          ↪  "ROUNDING_MODE1", "ROUNDING_MODE2"
          .SEL_PATTERN      ("PATTERN"       ), // Select pattern value
          ↪  ("PATTERN" or "C")
          .USE_PATTERN_DETECT("NO_PATDET"    ), // Enable pattern detect
          ↪  ("PATDET" or "NO_PATDET")
          // Register Control Attributes: Pipeline Register Configuration
          .ACASCREG         (0               ), // Number of pipeline stages
          ↪  between A/ACIN and ACOUT (0, 1 or 2)
          .ADREG            (0               ), // Number of pipeline stages
          ↪  for pre-adder (0 or 1)
          .ALUMODEREG       (0               ), // Number of pipeline stages
          ↪  for ALUMODE (0 or 1)
```

```
        .AREG          (0                ), // Number of pipeline stages
         ↪ for A (0, 1 or 2)
        .BCASCREG      (0                ), // Number of pipeline stages
         ↪ between B/BCIN and BCOUT (0, 1 or 2)
        .BREG          (0                ), // Number of pipeline stages
         ↪ for B (0, 1 or 2)
        .CARRYINREG    (0                ), // Number of pipeline stages
         ↪ for CARRYIN (0 or 1)
        .CARRYINSELREG (1                ), // Number of pipeline stages
         ↪ for CARRYINSEL (0 or 1)
        .CREG          (0                ), // Number of pipeline stages
         ↪ for C (0 or 1)
        .DREG          (0                ), // Number of pipeline stages
         ↪ for D (0 or 1)
        .INMODEREG     (0                ), // Number of pipeline stages
         ↪ for INMODE (0 or 1)
        .MREG          (0                ), // Number of multiplier
         ↪ pipeline stages (0 or 1)
        .OPMODEREG     (1                ), // Number of pipeline stages
         ↪ for OPMODE (0 or 1)
        .PREG          (0                )  // Number of pipeline stages
         ↪ for P (0 or 1)
      ) DSP48E1_tail (
        // Cascade: 30-bit (each) output: Cascade Ports
        //.ACOUT        (ACOUT        ), // 30-bit output: A port cascade
         ↪ output
        //.BCOUT        (BCOUT        ), // 18-bit output: B port cascade
         ↪ output
```

```
    .CARRYCASCOUT (cascade_carries[g1+1]    ), // 1-bit output: Cascade
    ↪  carry output
    //.MULTSIGNOUT   (MULTSIGNOUT   ), // 1-bit output: Multiplier sign
    ↪  cascade output
    //.PCOUT          (      ), // 48-bit output: Cascade output
    // Control: 1-bit (each) output: Control Inputs/Status Bits
    //.OVERFLOW      (OVERFLOW                  ), // 1-bit output:
    ↪  Overflow in add/acc output
    //.PATTERNBDETECT(PATTERNBDETECT            ), // 1-bit output:
    ↪  Pattern bar detect output
    //.PATTERNDETECT (PATTERNDETECT             ), // 1-bit output:
    ↪  Pattern detect output
    //.UNDERFLOW     (UNDERFLOW                 ), // 1-bit output:
    ↪  Underflow in add/acc output
    // Data: 4-bit (each) output: Data Ports
    //.CARRYOUT      (CARRYOUT      ), // 4-bit output: Carry output
    .P            (line[g1*S_TAIL +: S_TAIL]), // 48-bit output: Primary
    ↪  data output
    // Cascade: 30-bit (each) input: Cascade Ports
    .ACIN         (0                        ), // 30-bit input: A cascade
    ↪  data input
    .BCIN         (0                        ), // 18-bit input: B cascade
    ↪  input
    .CARRYCASCIN  (cascade_carries[g1]      ), // 1-bit input: Cascade
    ↪  carry input
    .MULTSIGNIN   (0                        ), // 1-bit input: Multiplier
    ↪  sign input
    .PCIN         (0                        ), // 48-bit input: P cascade
    ↪  input
```

```
        // Control: 4-bit (each) input: Control Inputs/Status Bits
        .ALUMODE       (TAIL_ALU                  ), // 4-bit input: ALU
         ↪  control input
        .CARRYINSEL    (3'h2                      ), // 3-bit input: Carry
         ↪  select input
        .CLK           (clk                       ), // 1-bit input: Clock
         ↪  input
        .INMODE        (5'h0                      ), // 5-bit input: INMODE
         ↪  control input
        .OPMODE        (7'h08                     ), // 7-bit input: Operation
         ↪  mode input
        // Data: 30-bit (each) input: Data Ports
        .A             (0                         ), // 30-bit input: A data
         ↪  input
        .B             (0                         ), // 18-bit input: B data
         ↪  input
        .C             (0                         ), // 48-bit input: C data
         ↪  input
        .CARRYIN       (0                         ), // 1-bit input: Carry
         ↪  input signal
        .D             (0                         ), // 25-bit input: D data
         ↪  input
        // Reset/Clock Enable: 1-bit (each) input: Reset/Clock Enable Inputs
        .CEA1          (0                         ), // 1-bit input: Clock
         ↪  enable input for 1st stage AREG
        .CEA2          (0                         ), // 1-bit input: Clock
         ↪  enable input for 2nd stage AREG
        .CEAD          (0                         ), // 1-bit input: Clock
         ↪  enable input for ADREG
```

```
          .CEALUMODE    (0                        ), // 1-bit input: Clock
           ↪  enable input for ALUMODE
          .CEB1         (0                        ), // 1-bit input: Clock
           ↪  enable input for 1st stage BREG
          .CEB2         (0                        ), // 1-bit input: Clock
           ↪  enable input for 2nd stage BREG
          .CEC          (0                        ), // 1-bit input: Clock
           ↪  enable input for CREG
          .CECARRYIN    (0                        ), // 1-bit input: Clock
           ↪  enable input for CARRYINREG
          .CECTRL       (1                        ), // 1-bit input: Clock
           ↪  enable input for OPMODEREG and CARRYINSELREG
          .CED          (0                        ), // 1-bit input: Clock
           ↪  enable input for DREG
          .CEINMODE     (0                        ), // 1-bit input: Clock
           ↪  enable input for INMODEREG
          .CEM          (0                        ), // 1-bit input: Clock
           ↪  enable input for MREG
          .CEP          (0                        ), // 1-bit input: Clock
           ↪  enable input for PREG
          .RSTA         (0                        ), // 1-bit input: Reset
           ↪  input for AREG
          .RSTALLCARRYIN(0                        ), // 1-bit input: Reset
           ↪  input for CARRYINREG
          .RSTALUMODE   (0                        ), // 1-bit input: Reset
           ↪  input for ALUMODEREG
          .RSTB         (0                        ), // 1-bit input: Reset
           ↪  input for BREG
          .RSTC         (0                        ), // 1-bit input: Reset
           ↪ input for CREG
          .RSTCTRL      (0                        ), // 1-bit input: Reset
           ↪ input for OPMODEREG and CARRYINSELREG
          .RSTD         (0                        ), // 1-bit input: Reset
           ↪ input for DREG and ADREG
          .RSTINMODE    (0                        ), // 1-bit input: Reset
           ↪ input for INMODEREG
          .RSTM         (0                        ), // 1-bit input: Reset
           ↪ input for MREG
          .RSTP         (0                        )  // 1-bit input: Reset
           ↪ input for PREG
        );
      end // subsequent_tails
    end // gen_tail
  endgenerate

endmodule
```

BIBLIOGRAPHY

[1] J. Wu and Z. Shi, "The 10-ps wave union TDC: Improving FPGA TDC resolution beyond its cell delay," in *2008 IEEE Nuclear Science Symposium Conference Record*, 2008, pp. 3440–3446.

[2] H. Ma, J. He, Y. Liu, J. Kuai, H. Li, L. Liu, and Y. Zhao, "On-chip trust evaluation utilizing tdc-based parameter-adjustable security primitive," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020.

[3] Intel Corp. (2020, Sep) Intel Cyclone 10 GX Core Fabric and General Purpose I/Os Handbook. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/vua1487061384661.html#sam1403481435443

[4] Xilinx Inc., "7 Series DSP48E1 Slice User Guide (UG479)," March 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[5] P. Keränen and J. Kostamovaara, "Algorithmic time-to-digital converter," in *2013 NORCHIP*, Nov 2013, pp. 1–4.

[6] Xilinx. (2014) Spartan-6 FPGA DSP48A1 Slice User Guide (UG389). [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug389.pdf

[7] S. Tancock and N. Dahnoun, "Digital Signal Processing Systems: Choosing a Processor," in *2020 International Conference Engineering and Telecommunication (En&T)*, Nov 2020, Plenary Talk.

[8] S. Tancock, E. Arabul, and N. Dahnoun, "A Review of New Time-to-Digital Conversion Techniques," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 10, pp. 3406–3417, 2019.

[9] S. Tancock, J. Rarity, and N. Dahnoun, "Developments in Time-to-Digital Converters during 2020," in *5th International Nordic-Mediterranean Workshop on Time-to-Digital Converters and Applications NoMe–TDC 2021*, 2021.

[10] S. Tancock, E. Arabul, N. Dahnoun, and S. Mehmood, "Can DSP48A1 adders be used for high-resolution delay generation?" in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, IEEE. Institute of Electrical and Electronics Engineers (IEEE), Aug 2018, pp. 1–6.

[11] S. Tancock and N. Dahnoun, "A 5.25 ps-resolution TDC on FPGA using DSP blocks," in *Proceedings of the Digital Image & Signal Processing'19 conference*, October 2019.

[12] S. Tancock, J. Rarity, and N. Dahnoun, "Temperature Characterisation of the DSP Delay Line," in *5th International Nordic-Mediterranean Workshop on Time-to-Digital Converters and Applications NoMe–TDC 2021*, 2021.

[13] ——, "The Wave-Union Method on DSP Blocks: Improving FPGA-based TDC resolutions by 3x with a 1.5x area increase," *IEEE Transactions on Instrumentation and Measurement (submitted with corrections)*, 2021.

[14] ——, "A Long-Range Hardware Bubble Corrector Technique for Short-Pulse-Width and Multiple-Registration Encoders," in *5th International Nordic-Mediterranean Workshop on Time-to-Digital Converters and Applications NoMe–TDC 2021*, 2021.

[15] ——, "Improving TDC Resolution with the Multi-Chain Vernier Method," *IEEE Access (submitted)*, 2021.

[16] E. Arabul, J. Rarity, and N. Dahnoun, "FPGA based fast integrated real-time multi coincidence counter using a time-to-digital converter," *2018 7th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–4, 2018.

[17] E. Arabul, S. Paesani, S. Tancock, J. Rarity, and N. Dahnoun, “A Precise High Count-Rate FPGA Based Multi-Channel Coincidence Counting System for Quantum Photonics Applications,” *IEEE Photonics Journal*, vol. 12, no. 2, pp. 1–14, 2020.

[18] F. ming Zhu, S. Hsieh, W. Yen, and H. Chou, “A digital coincidence measurement system using fpga techniques,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 652, no. 1, pp. 454–457, 2011, symposium on Radiation Measurements and Applications (SORMA) XII 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016890021100252X

[19] A. Aguilar, A. J. González, J. Torres, R. García-Olcina, J. Martos, J. Soret, P. Conde, L. Hernández, F. Sánchez, and J. M. Benlloch, “Timing results using an fpga-based tdc with large arrays of 144 sipms,” *IEEE Transactions on Nuclear Science*, vol. 62, no. 1, pp. 12–18, 2015.

[20] J. Shida, E. Spieglan, B. Adams, E. Angelico, K. Domurat-Sousa, A. Elagin, H. Frisch, P. La Riviere, and A. Squires, “Low-dose high-resolution tof-pet using ionization-activated multi-state low-z detector media,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1017, p. 165801, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900221007865

[21] “Time-of-Flight Fundamentals,” Nov 2013. [Online]. Available: https://msr.dom.wustl.edu/time-of-flight-fundamentals/

[22] K. Naresh, “Applications of Fluorescence Spectroscopy,” *National Level Workshop on Spectroscopic Techniques in Structural Elucidation Journal of Chemical and Pharmaceutical Sciences*, 2014. [Online]. Available: https://www.jchps.com/specialissues/Specialissue5/05jchpssi5knaresh18-21.pdf

[23] M. Wahl, “Time-Correlated Single Photon Counting,” 2014. [Online]. Available: https://www.picoquant.com/images/uploads/page/files/7253/technote_tcspc.pdf

[24] P. Vines, K. Kuzmenko, J. Kirdoda, D. C. Dumas, M. M. Mirza, R. W. Millar, D. J. Paul, and G. S. Buller, "High performance planar germanium-on-silicon single-photon avalanche diode detectors," *Nature communications*, vol. 10, no. 1, p. 1086, 2019.

[25] OSI Optoelectronics. (2013) 1.25Gbps Silicon Photodiodes. [Online]. Available: http://osioptoelectronics.com/standard-products/silicon-photodiodes/high-speed-silicon-photodiodes/1-25gbps-photodiodes.aspx

[26] G.-C. Hsieh and J. Hung, "Phase-locked loop techniques. a survey," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 6, pp. 609–615, 1996.

[27] V. Prasad and C. Sharma, "A review of phase locked loop," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 6, pp. 98–104, 2012.

[28] A. Singhal, C. Madhu, and V. Kumar, "Designs of all digital phase locked loop," in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, 2014, pp. 1–5.

[29] W. H. Chen, C. C. Hsu, and S. Y. Huang, "Rapid pll monitoring by a novel min-max time-to-digital converter," in *2020 IEEE International Test Conference (ITC)*, 2020, pp. 1–8.

[30] C. C. Chen, C. L. Chen, W. Fang, and Y. C. Chu, "All-digital cmos time-to-digital converter with temperature-measuring capability," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 9, pp. 2079–2083, 2020.

[31] M. S. Rostami, M. Saberi, M. Maymandi-Nejad, and M. Sawan, "A low-power time-to-digital converter for sensor interface circuits," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 2853–2857, 2020.

[32] Texas Instruments Incorporated, "TDC7201 (ACTIVE)." [Online]. Available: http://www.ti.com/product/TDC7201

[33] Swabian Instruments, "Time Tagger Series." [Online]. Available: https://www.swabianinstruments.com/time-tagger/

[34] P. GmbH, "TCSPC and Time Tagging Electronics." [Online]. Available: https://www.picoquant.com/products/category/tcspc-and-time-tagging-modules

[35] B. . H. GmbH, "SPC-150NXX." [Online]. Available: https://www.becker-hickl.com/products/spc-150nxx/#product-specs

[36] "ID900 Time Controller." [Online]. Available: https://www.idquantique.com/single-photon-systems/products/id900-time-controller/

[37] R. Szplet, Z. Jachna, P. Kwiatkowski, and K. Rozyc, "A 2.9 ps equivalent resolution interpolating time counter based on multiple independent coding lines," *Measurement Science and Technology*, vol. 24, no. 3, p. 035904, 2013.

[38] L. Perktold and J. Christiansen, "A fine time-resolution (< 3 ps-rms) time-to-digital converter for highly integrated designs," in *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 2013, pp. 1092–1097.

[39] J. Wu, "Uneven bin width digitization and a timing calibration method using cascaded PLL," in *2014 19th IEEE-NPSS Real Time Conference*, 2014, pp. 1–4.

[40] S. Henzler, *Time-to-digital converters*. Springer Science & Business Media, 2010, vol. 29.

[41] P. Toth and H. Ishikuro, "A Wide Input-Range, Low-Power and Highly Flexible 18 Bit Time-to-Digital Converter with Compact Differential Circuit Topology," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020, pp. 937–940.

[42] J. Ch, "Arduino Playground - Ultrasonic Sensor," http://playground.arduino.cc/Main/UltrasonicSensor, accessed 25/04/2017.

[43] K. Maatta and J. Kostamovaara, "A high-precision time-to-digital converter for pulsed time-of-flight laser radar applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 47, no. 2, pp. 521–536, 1998.

[44] S. Alahdab, A. Mantyniemi, and J. Kostamovaara, "A time-to-digital converter (TDC) with a 13-bit cyclic time domain successive approximation interpolator with sub-ps-level resolution using current DAC and differential switch," in *Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on*. IEEE, 2013, pp. 828–831.

[45] ——, "Review of a time-to-digital converter (TDC) based on Cyclic Time Domain Successive Approximation interpolator method with sub-ps-level resolution," in *Time-to-Digital Converters (NoMe TDC), 2013 IEEE Nordic-Mediterranean Workshop on*. IEEE, 2013, pp. 1–5.

[46] S. Al-Ahdab, A. Mäntyniemi, and J. Kostamovaara, "Cyclic time domain successive approximation time-to-digital converter (TDC) with sub-ps-level resolution," in *Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE*. IEEE, 2011, pp. 1–4.

[47] M. Fishburn, L. H. Menninga, C. Favi, and E. Charbon, "A 19.6 ps, FPGA-based TDC with multiple channels for open source applications," *IEEE Transactions on Nuclear Science*, vol. 60, no. 3, pp. 2203–2208, 2013.

[48] D. Kościelnik, J. Szyduczyński, D. Rzepka, W. Andrysiewicz, and M. Miśkowicz, "Optimized design of successive approximation time-to-digital converter with single set of delay lines," in *Event-based Control, Communication, and Signal Processing (EBCCSP), 2016 Second International Conference on*. IEEE, June 2016, pp. 1–8.

[49] A. Mantyniemi, T. Rahkonen, and J. Kostamovaara, "A CMOS time-to-digital converter (TDC) based on a cyclic time domain successive approximation interpolation method," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 11, pp. 3067–3078, 2009.

[50] A. Mäntyniemi and J. Kostamovaara, "Time-to-digital converter (TDC) based on startable ring oscillators and successive approximation," in *NORCHIP, 2014*. IEEE, 2014, pp. 1–4.

[51] A. S. Yousif and J. W. Haslett, "A fine resolution TDC architecture for next generation PET imaging," *IEEE Transactions on Nuclear Science*, vol. 54, no. 5, pp. 1574–1582, 2007.

[52] C. Favi and E. Charbon, “A 17Ps Time-to-digital Converter Implemented in 65Nm FPGA Technology,” in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA ’09. New York, NY, USA: ACM, 2009, pp. 113–120. [Online]. Available: http://doi.acm.org/10.1145/1508128.1508145

[53] R. Pelka, J. Kalisz, and R. Szplet, “Nonlinearity correction of the integrated time-to-digital converter with direct coding,” *IEEE Transactions on Instrumentation and Measurement*, vol. 46, no. 2, pp. 449–453, Apr 1997.

[54] J. Wu, Z. Shi, and I. Y. Wang, “Firmware-only implementation of time-to-digital converter (TDC) in field-programmable gate array (FPGA),” in *2003 IEEE Nuclear Science Symposium. Conference Record (IEEE Cat. No.03CH37515)*, vol. 1, Oct 2003, pp. 177–181 Vol.1.

[55] E. Bayer and M. Traxler, “A High-Resolution ($<$ 10 ps RMS) 48-Channel Time-to-Digital Converter (TDC) Implemented in a Field Programmable Gate Array (FPGA),” *IEEE Transactions on Nuclear Science*, vol. 58, no. 4, pp. 1547–1552, Aug 2011.

[56] A. M. Amiri, M. Boukadoum, and A. Khouas, “A Multihit Time-to-Digital Converter Architecture on FPGA,” *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 3, pp. 530–540, March 2009.

[57] D. M. Santos, S. F. Dow, J. M. Flasck, and M. E. Levi, “A CMOS delay locked loop and sub-nanosecond time-to-digital converter chip,” *IEEE Transactions on Nuclear Science*, vol. 43, no. 3, pp. 1717–1719, Jun 1996.

[58] V. Dhanasekaran, M. Gambhir, M. M. Elsayed, E. Sanchez-Sinencio, J. Silva-Martinez, C. Mishra, L. Chen, and E. Pankratz, “A 20MHz BW 68dB DR CT $\Delta\Sigma$ ADC based on a multi-bit time-domain quantizer and feedback element,” in *2009 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2009, pp. 174–175,175a.

[59] C. Niclass, C. Favi, T. Kluter, M. Gersbach, and E. Charbon, “A 128 $\times$ 128 Single-Photon Image Sensor With Column-Level 10-Bit Time-to-Digital Converter Array,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 12, pp. 2977–2989, Dec 2008.

[60] M. Mota and J. Christiansen, “A four-channel self-calibrating high-resolution time to digital converter,” in *1998 IEEE International Conference on Electronics, Circuits and Systems. Surfing the Waves of Science and Technology (Cat. No.98EX196)*, vol. 1, 1998, pp. 409–412 vol.1.

[61] I. Nissinen and J. Kostamovaara, “Time-to-Digital Converter based on an On-chip Voltage Reference Locked Ring Oscillator,” in *2006 IEEE Instrumentation and Measurement Technology Conference Proceedings*, April 2006, pp. 250–254.

[62] T. Watanabe, T. Mizuno, and Y. Makino, “An all-digital analog-to-digital converter with 12- mu;V/LSB using moving-average filtering,” *IEEE Journal of Solid-State Circuits*, vol. 38, no. 1, pp. 120–125, Jan 2003.

[63] R. B. Staszewski, K. Muhammad, D. Leipold, C.-M. Hung, Y.-C. Ho, J. L. Wallberg, C. Fernando, K. Maggio, R. Staszewski, T. Jung, J. Koh, S. John, I. Y. Deng, V. Sarda, O. Moreira-Tamayo, V. Mayega, R. Katz, O. Friedman, O. E. Eliezer, E. de Obaldia, and P. T. Balsara, “All-digital TX frequency synthesizer and discrete-time receiver for Bluetooth radio in 130-nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 39, no. 12, pp. 2278–2291, Dec 2004.

[64] P. Dudek, S. Szczepanski, and J. V. Hatfield, “A high-resolution CMOS time-to-digital converter utilizing a Vernier delay line,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 2, pp. 240–247, Feb 2000.

[65] A. H. Chan and G. W. Roberts, “A jitter characterization system using a component-invariant Vernier delay line,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, pp. 79–95, Jan 2004.

[66] ——, “A synthesizable, fast and high-resolution timing measurement device using a component-invariant vernier delay line,” in *Proceedings International Test Conference 2001 (Cat. No.01CH37260)*, 2001, pp. 858–867.

[67] ——, “A deep sub-micron timing measurement circuit using a single-stage Vernier delay line,” in *Proceedings of the IEEE 2002 Custom Integrated Circuits Conference (Cat. No.02CH37285)*, 2002, pp. 77–80.

[68] T. Hashimoto, H. Yamazaki, A. Muramatsu, T. Sato, and A. Inoue, “Time-to-digital converter with vernier delay mismatch compensation for high resolution on-die clock jitter measurement,” in *2008 IEEE Symposium on VLSI Circuits*, June 2008, pp. 166–167.

[69] G. H. Li and H. P. Chou, “A high resolution time-to-digital converter using two-level vernier delay line technique,” in *2007 IEEE Nuclear Science Symposium Conference Record*, vol. 1, Oct 2007, pp. 276–280.

[70] V. Ramakrishnan and P. T. Balsara, “A wide-range, high-resolution, compact, CMOS time to digital converter,” in *19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID’06)*, Jan 2006, pp. 6 pp.–.

[71] G. Li, Y. M. Tousi, A. Hassibi, and E. Afshari, “Delay-Line-Based Analog-to-Digital Converters,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 6, pp. 464–468, June 2009.

[72] C.-S. Hwang, P. Chen, and H.-W. Tsao, “A high-precision time-to-digital converter using a two-level conversion scheme,” *IEEE Transactions on Nuclear Science*, vol. 51, no. 4, pp. 1349–1352, Aug 2004.

[73] B. Markovic, S. Tisa, F. A. Villa, A. Tosi, and F. Zappa, “A High-Linearity, 17 ps Precision Time-to-Digital Converter Based on a Single-Stage Vernier Delay Loop Fine Interpolation,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, no. 3, pp. 557–569, March 2013.

[74] D. K. Xie, Q. C. Zhang, G. S. Qi, and D. Y. Xu, “Cascading delay line time-to-digital converter with 75 ps resolution and a reduced number of delay cells,” *Review of Scientific Instruments*, vol. 76, no. 1, p. 014701, 2005. [Online]. Available: http://dx.doi.org/10.1063/1.1829931

[75] L. Vercesi, A. Liscidini, and R. Castello, "Two-Dimensions Vernier Time-to-Digital Converter," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 8, pp. 1504–1512, Aug 2010.

[76] M. C. Lin, G. R. Tsai, C. Y. Liu, and S. S. Chu, "FPGA-Based High Area Efficient Time-To-Digital IP Design," in *TENCON 2006 - 2006 IEEE Region 10 Conference*, Nov 2006, pp. 1–4.

[77] A. Aloisio, P. Branchini, R. Cicalese, R. Giordano, V. Izzo, and S. Loffredo, "FPGA implementation of a high-resolution time-to-digital converter," in *2007 IEEE Nuclear Science Symposium Conference Record*, vol. 1, Oct 2007, pp. 504–507.

[78] P. Chen, P. Y. Chen, J. S. Lai, and Y. J. Chen, "FPGA Vernier Digital-to-Time Converter With 1.58 ps Resolution and 59.3 Minutes Operation Range," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 6, pp. 1134–1142, June 2010.

[79] K. Karadamoglou, N. P. Paschalidis, E. Sarris, N. Stamatopoulos, G. Kottaras, and V. Paschalidis, "An 11-bit high-resolution and adjustable-range CMOS time-to-digital converter for space science instruments," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 1, pp. 214–222, Jan 2004.

[80] C.-C. Chen, P. Chen, C.-S. Hwang, and W. Chang, "A precise cyclic CMOS time-to-digital converter with low thermal sensitivity," *IEEE Transactions on Nuclear Science*, vol. 52, no. 4, pp. 834–838, Aug 2005.

[81] P. Chen and S.-I. Liu, "A cyclic CMOS time-to-digital converter with deep sub-nanosecond resolution," in *Proceedings of the IEEE 1999 Custom Integrated Circuits Conference (Cat. No.99CH36327)*, 1999, pp. 605–608.

[82] S. Tisa, A. Lotito, A. Giudice, and F. Zappa, "Monolithic time-to-digital converter with 20ps resolution," in *ESSCIRC 2004 - 29th European Solid-State Circuits Conference (IEEE Cat. No.03EX705)*, Sept 2003, pp. 465–468.

[83] Y. Liu, U. Vollenbruch, Y. Chen, C. Wicpalek, L. Maurer, Z. Boos, and R. Weigel, "Multistage pulse shrinking time-to-digital converter for time interval measurements," in *2007 European Microwave Integrated Circuit Conference*, Oct 2007, pp. 267–270.

[84] Y. Liu, U. Vollenbruch, Y. Chen, C. Wicpalek, L. Maurer, T. Mayer, Z. Boos, and R. Weigel, "A 6ps resolution pulse shrinking Time-to-Digital Converter as phase detector in multimode transceiver," in *2008 IEEE Radio and Wireless Symposium*, Jan 2008, pp. 163–166.

[85] P. Chen, S.-L. Liu, and J. Wu, "A CMOS pulse-shrinking delay element for time interval measurement," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 9, pp. 954–958, Sep 2000.

[86] P. Chen, S.-I. Liu, and J. Wu, "A low power high accuracy CMOS time-to-digital converter," in *Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on*, vol. 1, Jun 1997, pp. 281–284 vol.1.

[87] S. Henzler, S. Koeppe, W. Kamp, H. Mulatz, and D. Schmitt-Landsiedel, "90nm 4.7ps-Resolution 0.7-LSB Single-Shot Precision and 19pJ-per-Shot Local Passive Interpolation Time-to-Digital Converter with On-Chip Characterization," in *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, Feb 2008, pp. 548–635.

[88] S. Henzler, S. Koeppe, D. Lorenz, W. Kamp, R. Kuenemund, and D. Schmitt-Landsiedel, "A Local Passive Time Interpolation Concept for Variation-Tolerant High-Resolution Time-to-Digital Conversion," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 7, pp. 1666–1676, July 2008.

[89] ——, "Variation tolerant high resolution and low latency time-to-digital converter," in *ESSCIRC 2007 - 33rd European Solid-State Circuits Conference*, Sept 2007, pp. 194–197.

[90] M. S. Kim, Y. B. Kim, and K. K. Kim, "All-digital phased-locked loop with local passive interpolation time-to-digital converter based on a tristate inverter," in *2012 IEEE 55th*

International Midwest Symposium on Circuits and Systems (MWSCAS), Aug 2012, pp. 326–329.

[91] A. Samarah and A. C. Carusone, “A Digital Phase-Locked Loop With Calibrated Coarse and Stochastic Fine TDC,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1829–1841, Aug 2013.

[92] V. Kratyuk, P. K. Hanumolu, K. Ok, U. K. Moon, and K. Mayaram, “A Digital PLL With a Stochastic Time-to-Digital Converter,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 8, pp. 1612–1621, Aug 2009.

[93] M. Zanuso, S. Levantino, A. Puggelli, C. Samori, and A. L. Lacaita, “Time-to-digital converter with 3-ps resolution and digital linearization algorithm,” in *2010 Proceedings of ESSCIRC*, Sept 2010, pp. 262–265.

[94] J. Wu, Z. Wang, C. Chen, C. Huang, and M. Zhang, “A 2.4-GHz All-Digital PLL With a 1-ps Resolution 0.9-mW Edge-Interchanging-Based Stochastic TDC,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 10, pp. 917–921, Oct 2015.

[95] S. Pellerano, P. Madoglio, and Y. Palaskas, “A 4.75-GHz Fractional Frequency Divider-by-1.25 With TDC-Based All-Digital Spur Calibration in 45-nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 12, pp. 3422–3433, Dec 2009.

[96] C. W. Fan and J. T. Wu, “Jitter Measurement and Compensation for Analog-to-Digital Converters,” *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 11, pp. 3874–3884, Nov 2009.

[97] P. K. Hanumolu, G. Y. Wei, U. K. Moon, and K. Mayaram, “Digitally-Enhanced Phase-Locking Circuits,” in *2007 IEEE Custom Integrated Circuits Conference*, Sept 2007, pp. 361–368.

[98] J. S. Tandon, T. J. Yamaguchi, S. Komatsu, and K. Asada, “A stochastic sampling time-to-digital converter with tunable 180-770fs resolution, INL less than 0.6LSB, and

selectable dynamic range offset," in *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*, Sept 2013, pp. 1–4.

[99] S. Ito, S. Nishimura, H. Kobayashi, S. Uemori, Y. Tan, N. Takai, T. J. Yamaguchi, and K. Niitsu, "Stochastic TDC architecture with self-calibration," in *2010 IEEE Asia Pacific Conference on Circuits and Systems*, Dec 2010, pp. 1027–1030.

[100] S.-J. Kim, W. Kim, M. Song, J. Kim, T. Kim, and H. Park, "15.5 A 0.6V 1.17ps PVT-tolerant and synthesizable time-to-digital converter using stochastic phase interpolation with 16×spatial redundancy in 14nm FinFET technology," in *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, Feb 2015, pp. 1–3.

[101] O. C. Akgun, "An Asynchronous Pipelined Time-to-Digital Converter Using Time-Domain Subtraction," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[102] A. Mantyniemi, T. Rahkonen, and J. Kostamovaara, "A CMOS Time-to-Digital Converter (TDC) Based On a Cyclic Time Domain Successive Approximation Interpolation Method," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 11, pp. 3067–3078, Nov 2009.

[103] S. Al-Ahdab, A. Mäntyniemi, and J. Kostamovaara, "Cyclic time domain successive approximation time-to-digital converter (TDC) with sub-ps-level resolution," in *2011 IEEE International Instrumentation and Measurement Technology Conference*, May 2011, pp. 1–4.

[104] S. Alahdab, A. Mäntyniemi, and J. Kostamovaara, "A time-to-digital converter (TDC) with a 13-bit cyclic time domain successive approximation interpolator with sub-ps-level resolution using current DAC and differential switch," in *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2013, pp. 828–831.

[105] ——, "A 12-bit digital-to-time converter (DTC) with sub-ps-level resolution using current DAC and differential switch for time-to-digital converter (TDC)," in *2012 IEEE Inter-*

national Instrumentation and Measurement Technology Conference Proceedings, May 2012, pp. 2668–2671.

[106] H. Chung, H. Ishikuro, and T. Kuroda, “A 10-Bit 80-MS/s Decision-Select Successive Approximation TDC in 65-nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 47, no. 5, pp. 1232–1241, May 2012.

[107] R. Jiang, C. Li, M. Yang, H. Kobayashi, Y. Ozawa, N. Tsukiji, M. Hirano, R. Shiota, and K. Hatayama, “Successive approximation time-to-digital converter with vernier-level resolution,” in *2016 IEEE 21st International Mixed-Signal Testing Workshop (IMSTW)*, July 2016, pp. 1–6.

[108] D. Kościelnik, M. Miśkowicz, J. Szyduczyński, and D. Rzepka, “Optimizing time-to-digital converter architecture for successive approximation time measurements,” in *2013 IEEE Nordic-Mediterranean Workshop on Time-to-Digital Converters (NoMe TDC)*, Oct 2013, pp. 1–8.

[109] P. Keränen and J. Kostamovaara, “A Wide Range, 4.2 ps(rms) Precision CMOS TDC With Cyclic Interpolators Based on Switched-Frequency Ring Oscillators,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 12, pp. 2795–2805, Dec 2015.

[110] J. Richardson, R. Walker, L. Grant, D. Stoppa, F. Borghetti, E. Charbon, M. Gersbach, and R. K. Henderson, “A 32×32 50ps resolution 10 bit time to digital converter array in 130nm CMOS for time correlated imaging,” in *2009 IEEE Custom Integrated Circuits Conference*, Sept 2009, pp. 77–80.

[111] Y. Cao, P. Leroux, W. D. Cock, and M. Steyaert, “A 1.7mW 11b 1-1-1 MASH $\Delta\Sigma$ time-to-digital converter,” in *2011 IEEE International Solid-State Circuits Conference*, Feb 2011, pp. 480–482.

[112] A. Elshazly, S. Rao, B. Young, and P. K. Hanumolu, “A 13b 315fsrms 2mW 500MS/s 1MHz bandwidth highly digital time-to-digital converter using switched ring oscillators,” in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 464–466.

[113] M. Z. Straayer and M. H. Perrott, “A Multi-Path Gated Ring Oscillator TDC With First-Order Noise Shaping,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1089–1098, April 2009.

[114] ——, “An efficient high-resolution 11-bit noise-shaping multipath gated ring oscillator TDC,” in *2008 IEEE Symposium on VLSI Circuits*, June 2008, pp. 82–83.

[115] C. M. Hsu, M. Z. Straayer, and M. H. Perrott, “A Low-Noise Wide-BW 3.6-GHz Digital ΔΣ Fractional-N Frequency Synthesizer With a Noise-Shaping Time-to-Digital Converter and Quantization Noise Cancellation,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 12, pp. 2776–2786, Dec 2008.

[116] P. Lu and P. Andreani, “A high-resolution Vernier Gated-Ring-Oscillator TDC in 90-nm CMOS,” in *NORCHIP 2010*, Nov 2010, pp. 1–4.

[117] P. Lu, A. Liscidini, and P. 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*, vol. 47, no. 7, pp. 1626–1635, July 2012.

[118] K. Gammoh, C. K. Peterson, D. A. Penry, and S.-H. W. Chiang, “Linearity Theory of Stochastic Phase-Interpolation Time-to-Digital Converter,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4348–4359, 2020.

[119] X. Hu, L. Zhao, S. Liu, J. Wang, and Q. An, “A Stepped-Up Tree Encoder for the 10-ps Wave Union TDC,” *IEEE Transactions on Nuclear Science*, vol. 60, no. 5, pp. 3544–3549, Oct 2013.

[120] O. Sasaki, T. Taniguchi, T. K. Ohska, and H. Kurashige, “A high resolution TDC in TKO box system,” *IEEE Transactions on Nuclear Science*, vol. 35, no. 1, pp. 342–347, Feb 1988.

[121] D. Stoppa, F. Borghetti, J. Richardson, R. Walker, L. Grant, R. K. Henderson, M. Gersbach, and E. Charbon, “A 32x32-pixel array with in-pixel photon counting and arrival time

measurement in the analog domain," in *2009 Proceedings of ESSCIRC*, Sept 2009, pp. 204–207.

[122] B. K. Swann, B. J. Blalock, L. G. Clonts, D. M. Binkley, J. M. Rochelle, E. Breeding, and K. M. Baldwin, "A 100-ps time-resolution CMOS time-to-digital converter for positron emission tomography imaging applications," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 11, pp. 1839–1852, Nov 2004.

[123] K. Maatta and J. Kostamovaara, "A high-precision time-to-digital converter for pulsed time-of-flight laser radar applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 47, no. 2, pp. 521–536, Apr 1998.

[124] J. Kostamovaara and R. Myllylä, "Time-to-digital converter with an analog interpolation circuit," *Review of Scientific Instruments*, vol. 57, no. 11, pp. 2880–2885, 1986. [Online]. Available: http://dx.doi.org/10.1063/1.1139008

[125] M. Bogdan, H. Frisch, M. Heintz, A. Paramonov, H. Sanders, S. Chappa, R. DeMaat, R. Klein, T. Miao, P. Wilson, and T. J. Phillips, "A 96-channel FPGA-based Time-to-Digital Converter (TDC) and fast trigger processor module with multi-hit capability and pipeline," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 554, no. 1, pp. 444–457, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900205016797

[126] A. Elkholy, T. Anand, W. S. Choi, A. Elshazly, and P. K. Hanumolu, "A 3.7 mW Low-Noise Wide-Bandwidth 4.5 GHz Digital Fractional-N PLL Using Time Amplifier-Based TDC," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 867–881, April 2015.

[127] M. Lee and A. A. Abidi, "A 9b, 1.25ps Resolution Coarse-Fine Time-to-Digital Converter in 90nm CMOS that Amplifies a Time Residue," in *2007 IEEE Symposium on VLSI Circuits*, vol. 43, no. 4, June 2007, pp. 168–169.

[128] ——, “A 9 b, 1.25 ps Resolution Coarse-Fine Time-to-Digital Converter in 90 nm CMOS that Amplifies a Time Residue,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 769–777, April 2008.

[129] M. Lee, M. E. Heidari, and A. A. Abidi, “A Low-Noise Wideband Digital Phase-Locked Loop Based on a Coarse-Fine Time-to-Digital Converter With Subpicosecond Resolution,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 10, pp. 2808–2816, Oct 2009.

[130] Y. H. Seo, J. S. Kim, H. J. Park, and J. Y. Sim, “A 1.25 ps Resolution 8b Cyclic TDC in 0.13 mum CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 736–743, March 2012.

[131] S. K. Lee, Y. H. Seo, H. J. Park, and J. Y. Sim, “A 1 GHz ADPLL With a 1.25 ps Minimum-Resolution Sub-Exponent TDC in 0.18 mu m CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 45, no. 12, pp. 2874–2881, Dec 2010.

[132] Y. H. Seo, J. S. Kim, H. J. Park, and J. Y. Sim, “A 0.63ps resolution, 11b pipeline TDC in 0.13 μm CMOS,” in *2011 Symposium on VLSI Circuits - Digest of Technical Papers*, June 2011, pp. 152–153.

[133] K. Kim, Y. H. Kim, W. Yu, and S. Cho, “A 7 bit, 3.75 ps Resolution Two-Step Time-to-Digital Converter in 65 nm CMOS Using Pulse-Train Time Amplifier,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 4, pp. 1009–1017, April 2013.

[134] H. J. Kwon, J. s. Lee, J. Y. Sim, and H. J. Park, “A high-gain wide-input-range time amplifier with an open-loop architecture and a gain equal to current bias ratio,” in *IEEE Asian Solid-State Circuits Conference 2011*, Nov 2011, pp. 325–328.

[135] P. Chen, C. C. Chen, and Y. S. Shen, “A Low-Cost Low-Power CMOS Time-to-Digital Converter Based on Pulse Stretching,” *IEEE Transactions on Nuclear Science*, vol. 53, no. 4, pp. 2215–2220, Aug 2006.

[136] E. Raisanen-Ruotsalainen, T. Rahkonen, and J. Kostamovaara, “An integrated time-to-digital converter with 30-ps single-shot precision,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 10, pp. 1507–1510, Oct 2000.

[137] J. S. Teh, L. Siek, A. M. Alonso, A. Firdauzi, and A. Matsuzawa, “A 14-b, 850fs Fully Synthesizable Stochastic-Based Branching Time-to-Digital Converter in 65nm CMOS,” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.

[138] P. Keranen, K. Maatta, and J. Kostamovaara, “Wide-Range Time-to-Digital Converter With 1-ps Single-Shot Precision,” *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 9, pp. 3162–3172, Sept 2011.

[139] M. Safi-Harb and G. W. Roberts, “Embedded Narrow Pulse Measurement in Digital CMOS,” in *2006 IEEE Instrumentation and Measurement Technology Conference Proceedings*, April 2006, pp. 1195–1200.

[140] J.-P. Jansson, A. Mantyniemi, and J. Kostamovaara, “A CMOS time-to-digital converter with better than 10 ps single-shot precision,” *IEEE Journal of Solid-State Circuits*, vol. 41, no. 6, pp. 1286–1296, June 2006.

[141] M. Takayama, S. Dosho, N. Takeda, M. Miyahara, and A. Matsuzawa, “A time-domain architecture and design method of high speed A-to-D converters with standard cells,” in *IEEE Asian Solid-State Circuits Conference 2011*, Nov 2011, pp. 353–356.

[142] R. B. Staszewski, S. Vemulapalli, P. Vallur, J. Wallberg, and P. 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*, vol. 53, no. 3, pp. 220–224, March 2006.

[143] A. Mantyniemi, T. Rahkonen, and J. Kostamovaara, “An integrated 9-channel time digitizer with 30 ps resolution,” in *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, vol. 1, Feb 2002, pp. 266–465 vol.1.

[144] M. Rashdan, A. Yousif, J. Haslett, and B. Maundy, "A new time-based architecture for serial communication links," in *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*, Dec 2009, pp. 531–534.

[145] B. M. Helal, M. Z. Straayer, G.-Y. Wei, and M. H. Perrott, "A Highly Digital MDLL-Based Clock Multiplier That Leverages a Self-Scrambling Time-to-Digital Converter to Achieve Subpicosecond Jitter Performance," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 4, pp. 855–863, April 2008.

[146] C. Veerappan, J. Richardson, R. Walker, D. Li, M. W. Fishburn, Y. Maruyama, D. Stoppa, F. Borghetti, M. Gersbach, R. K. Henderson, and E. Charbon, "A 160×128 single-photon image sensor with on-pixel 55ps 10b time-to-digital converter," in *2011 IEEE International Solid-State Circuits Conference*, Feb 2011, pp. 312–314.

[147] K. Cui, Z. Ren, X. Li, Z. Liu, and R. Zhu, "A High-Linearity, Ring-Oscillator-Based, Vernier Time-to-Digital Converter Utilizing Carry Chains in FPGAs," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 697–704, Jan 2017.

[148] R. Narasimman, A. Prabhakar, and N. Chandrachoodan, "Implementation of a 30 ps resolution time to digital converter in FPGA," in *2015 International Conference on Electronic Design, Computer Networks Automated Verification (EDCAV)*, Jan 2015, pp. 12–17.

[149] Xilinx Inc. (2010, February) Spartan-6 FPGA Configurable Logic Block (UG384). [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug384.pdf

[150] Xilinx. (2015) Spartan-6 FPGA Data Sheet: DC and Switching Characteristics (DS162). [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds162.pdf

[151] Xilinx Inc., "Xilinx Spartan-6 Libraries Guide for HDL Designers (UG615)," pp. 92–98, October 2013. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/spartan6_hdl.pdf

[152] Opal Kelly. (2014) Opal Kelly Front Panel SDK. [Online]. Available: https://www.opalkelly.com/products/frontpanel/

[153] ——. (2014) Opal Kelly XEM6010. [Online]. Available: https://www.opalkelly.com/products/xem6010/

[154] P. Kwiatkowski, "Employing fpga dsp blocks for time-to-digital conversion," *Metrology and Measurement Systems*, vol. vol. 26, no. No 4, pp. 631–643, 2019. [Online]. Available: http://journals.pan.pl/Content/114023/PDF/04_MMS_4_INTERNET.pdf

[155] Xilinx Inc. (2018, Jul) 7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf

[156] R. Szplet, P. Kwiatkowski, Z. Jachna, and K. Różyc, "An Eight-Channel 4.5-ps Precision Timestamps-Based Time Interval Counter in FPGA Chip," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 9, pp. 2088–2100, Sep. 2016.

[157] Y. Wang, X. Zhou, Z. Song, J. Kuang, and Q. Cao, "A 3.0-ps rms precision 277-msamples/s throughput time-to-digital converter using multi-edge encoding scheme in a kintex-7 fpga," *IEEE Transactions on Nuclear Science*, vol. 66, no. 10, pp. 2275–2281, 2019.

[158] P. Kwiatkowski and R. Szplet, "Multisampling wave union time-to-digital converter," in *2020 6th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*, 2020, pp. 1–5.

[159] W. Xie, H. Chen, and D. D.-U. Li, "Efficient time-to-digital converters in 20 nm fpgas with wave union methods," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 1, pp. 1021–1031, 2022.

[160] J. Qi, Z. Deng, H. Gong, and Y. Liu, "A 20ps resolution wave union FPGA TDC with on-chip real time correction," in *IEEE Nuclear Science Symposuim Medical Imaging Conference*, Oct 2010, pp. 396–399.

[161] J. Wang, S. Liu, L. Zhao, X. Hu, and Q. An, “The 10-ps Multitime Measurements Averaging TDC Implemented in an FPGA,” *IEEE Transactions on Nuclear Science*, vol. 58, no. 4, pp. 2011–2018, Aug 2011.

[162] K. Katoh, Y. Doi, S. Ito, H. Kobayashi, E. Li, N. Takai, and O. Kobayashi, “An Analysis of Stochastic Self-Calibration of TDC Using Two Ring Oscillators,” in *2013 22nd Asian Test Symposium*, Nov 2013, pp. 140–146.

[163] R. Szplet, R. Szymanowski, and D. Sondej, “Measurement uncertainty of precise interpolating time counters,” *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 11, pp. 4348–4356, 2019.

[164] R. Szplet, D. Sondej, and G. Grzęda, “High-precision time digitizer based on multiedge coding in independent coding lines,” *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1884–1894, 2016.

[165] Y. Wang and C. Liu, “A 3.9 ps time-interval rms precision time-to-digital converter using a dual-sampling method in an ultrascale fpga,” *IEEE Transactions on Nuclear Science*, vol. 63, no. 5, pp. 2617–2621, 2016.

[166] N. Lusardi, F. Garzetti, and A. Geraci, “Digital instrument with configurable hardware and firmware for multi-channel time measures,” *Review of Scientific Instruments*, vol. 90, no. 5, p. 055113, 2019. [Online]. Available: https://doi.org/10.1063/1.5028131

[167] X. Qin, M.-D. Zhu, W.-Z. Zhang, Y.-H. Lin, Y. Rui, X. Rong, and J. Du, “A high resolution time-to-digital-convertor based on a carry-chain and dsp48e1 adders in a 28-nm field-programmable-gate-array,” *Review of Scientific Instruments*, vol. 91, no. 2, p. 024708, 2020. [Online]. Available: https://doi.org/10.1063/1.5141391

[168] M.-D. Zhu, X. Qin, L. Wang, W.-Z. Zhang, Y. Lin, X. Rong, and J. Du, “A time-to-digital-converter utilizing bits-counters to decode carry-chains and DSP48e1 slices in a field-programmable-gate-array,” *Journal of Instrumentation*, vol. 16, no. 02, pp. P02 009–P02 009, feb 2021. [Online]. Available: https://doi.org/10.1088/1748-0221/16/02/p02009

[169] E. Sall and M. Vesterbacka, “Thermometer-to-binary decoders for flash analog-to-digital converters,” in *2007 18th European Conference on Circuit Theory and Design*, Aug 2007, pp. 240–243.

[170] F. Kaess, R. Kanan, B. Hochet, and M. Declercq, “New encoding scheme for high-speed flash ADC’s,” in *Proceedings of 1997 IEEE International Symposium on Circuits and Systems. Circuits and Systems in the Information Age ISCAS ’97*, vol. 1, June 1997, pp. 5–8 vol.1.

[171] K. Uyttenhove, A. Marques, and M. Steyaert, “A 6-bit 1 GHz acquisition speed CMOS flash ADC with digital error correction,” in *Proceedings of the IEEE 2000 Custom Integrated Circuits Conference (Cat. No.00CH37044)*, May 2000, pp. 249–252.

[172] E. Sail and M. Vesterbacka, “A multiplexer based decoder for flash analog-to-digital converters,” in *2004 IEEE Region 10 Conference TENCON 2004.*, vol. D, Nov 2004, pp. 250–253 Vol. 4.

[173] J. Terada, Y. Matsuya, F. Morisawa, and Y. Kado, “8-mW, 1-V, 100-Msps, 6-bit A/D converter using a transconductance latched comparator,” in *Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No.00EX434)*, Aug 2000, pp. 53–56.

[174] Y. Wang, J. Kuang, C. Liu, and Q. Cao, “A 3.9-ps RMS Precision Time-to-Digital Converter Using Ones-Counter Encoding Scheme in a Kintex-7 FPGA,” *IEEE Transactions on Nuclear Science*, vol. 64, no. 10, pp. 2713–2718, Oct 2017.

[175] S. Tancock. (2019, July) lr-bubble-detect git repository. [Online]. Available: https://gitlab.com/scott-tancock/lr-bubble-correct/tree/rev2/src

[176] A. El-Hadbi, O. Elissati, and L. Fesquet, “Time-to-digital converters: A literature review and new perspectives,” in *2019 5th International Conference on Event-Based Control, Communication, and Signal Processing (EBCCSP)*. IEEE, 2019, pp. 1–8.

[177] J. Kalisz, “Review of methods for time interval measurements with picosecond resolution,” *Metrologia*, vol. 41, no. 1, p. 17, 2003.

[178] M. Z. Straayer and M. H. Perrott, “A multi-path gated ring oscillator TDC with first-order noise shaping,” *IEEE Journal of Solid-State Circuits*, vol. 44, no. 4, pp. 1089–1098, 2009.

[179] P. Kwiatkowski and R. Szplet, “Efficient implementation of multiple time coding lines-based TDC in an FPGA device,” *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 10, pp. 7353–7364, 2020.

[180] Cadence Design Systems Inc. (2021) Cadence - Computational Software for Intelligent System Design. [Online]. Available: https://www.cadence.com/en_US/home.html

[181] F. Villa, R. Lussana, D. Bronzi, S. Tisa, A. Tosi, F. Zappa, A. Dalla Mora, D. Contini, D. Durini, and S. Weyers, “CMOS imager with 1024 SPADs and TDCs for single-photon timing and 3D time-of-flight,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 20, 09 2014.

[182] A. Stanoev, N. Audinet, S. Tancock, and N. Dahnoun, “Real-time stereo vision for collision detection on autonomous uavs,” in *2017 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2017, pp. 1–6.

[183] F. Mendez, M. Ferguson, N. Dahnoun, and S. Tancock, “Real-time user selected dynamic template tracking for uav,” in *2017 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2017, pp. 1–6.

[184] R. Fan, Y. Liu, X. Yang, M. J. Bocus, N. Dahnoun, and S. Tancock, “Real-time stereo vision for road surface 3-d reconstruction,” in *2018 IEEE International Conference on Imaging Systems and Techniques (IST)*. IEEE, 2018, pp. 1–6.

[185] S. Tancock and N. Dahnoun, “Project Handover in Undergraduate Projects-Efficient Handover for Increased Learning Opportunities,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 6463–6467.

[186] ——, “Preparing Undergraduates for the Final-Year Project,” in *Enhancing Student Learning Through Innovative Scholarship Conference*, 2018.

[187] S. Tancock, Y. Dahnoun, and N. Dahnoun, “Real-Time and Non-digital Feedback E-Learning Tool,” in *2018 International Symposium on Educational Technology (ISET)*, 2018, pp. 57–59.

[188] ——, “Machine Learning Based Real-Time and Non-Digital Feedback E-Learning Tool,” in *Enhancing Student Learning Through Innovative Scholarship Conference*, 2018, p. 43.