SAND2017-10097

SANDIA REPORT
SAND#: 659334
Unlimited Release
August 2017

D-DMTD: Digital Dual Mixer Time Difference

Dustin Tso, Sarina Kapai, Marc R. Feldman

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

SAND#: 659334
Unlimited Release
August 2017

D-DMTD: Digital Dual Mixer Time Difference

Dustin Tso, Sarina Kapai, Marc Feldman, Yalin Hu
Telemetry and R&D
Sandia National Laboratories
Livermore, CA 94550

Abstract

The digital design discussed in the following document is inspired by the digital dual mixer time difference circuit and based on the white paper, *Digital femtosecond time difference circuit for CERN's timing system* [5].

DMTD is originally an analog technique that measures the time difference between two events with high precision using a commercial time interval counter. Our project applies this analog concept to a digital system programmed onto a Microsemi ProASIC3E FPGA (Field-Programmable Gate Array).

D-DMTD is a digital system theoretically capable of measuring the time difference between two digital clock signals with very fine resolution (sub-picosecond) using a relatively low frequency counter. The system essentially acts as a digital phase detector with femtosecond time resolution. The main concern with this processing technique is its feasibility and accuracy when implemented on an FPGA. Another concern is the environmentally-induced, horizontal phase noise in the digital signals; this "jitter" jeopardizes the fidelity of the clocks and generates glitches in the signals. Thus, the majority of the work was focused on determining under what conditions the digital design performs optimally and generates the most accurate estimations for the phase shift.

This work was funded by Dept. 8736, Telemetry and R&D at Sandia National Laboratories.

ACKNOWLEDGEMENTS

The authors would like to thank the Telemetry group at Sandia National Laboratories (California site) for providing this project with funding, resources, and feedback.

We would also like to thank Robert J. Mariano for his managerial support, Joni Dahl for her administrative support, and the engineers in Mobile 44 for their generous donations of equipment and technical expertise in the fields of FPGA programming and digital design.

Table of Contents

Figures

Tables

NOMENCLATURE

ASIC	Application Specific Integration Circuit
FPGA	Field-Programmable Gate Array
DMTD	Dual Mixer Time Difference
D-DMTD	Digital Dual Mixer Time Difference
PLL	Phase-Locked Loop
VHDL	VHSIC Hardware Description Language
RMS	Root-Mean-Square

1. INTRODUCTION

Recent developments in technology have made FPGAs highly applicable in a wide spectrum of application fields due to their re-configurability, fast processing capabilities, and ability to simultaneously handle a large number of electronic signals [1]. In particular, FPGAs with precise time difference capabilities are increasingly utilized for time-based signal processing and nuclear medical imaging [6][7]. In other cases, such as high-energy physical experiments, high precision timing is used to explore subatomic level fine structure in fixed target and collision experiments, which often demands picosecond resolution [3]. Finally, the implementation of D-DMTD is of special interest to the field of nuclear electronics, where high precision timing is required, despite high temperatures and unstable environments [2].

With these applications in mind, this paper builds and expands upon P. Moreira's theoretical digital dual mixer time difference (D-DMTD) circuit for CERN's (a European Organization for Nuclear Research) timing system, and implements this design onto a Microsemi ProASIC3E FPGA device.

D-DMTD is capable of measuring the time difference between two out-of-phase digital clock signals with sub-picosecond resolution using relatively low frequency counters [5]. Current phase-difference detectors mostly use time-to-digital converter (TDC) design techniques, where propagation delay structures in tapped delay line architectures are sensitive to temperature and power supply voltage [2]. This creates the need for constant calibration, which can consume valuable resources on FPGAs [4]. Even with these limitations, TDC designs thus far have only achieved sub-nanosecond resolution in controlled environments. D-DMTD, like TDCs, have the advantage in that it can achieve precision well below the sampling rate, allowing for use of lower frequency components, which are cheaper, require less power, and simplify circuit board designs [2].

The focus of this paper will be to confirm the capabilities of D-DMTD, theorized by P. Moreira, and to qualify under what conditions this digital design optimally performs.

2. HIGH LEVEL DESIGN DESCRIPTION

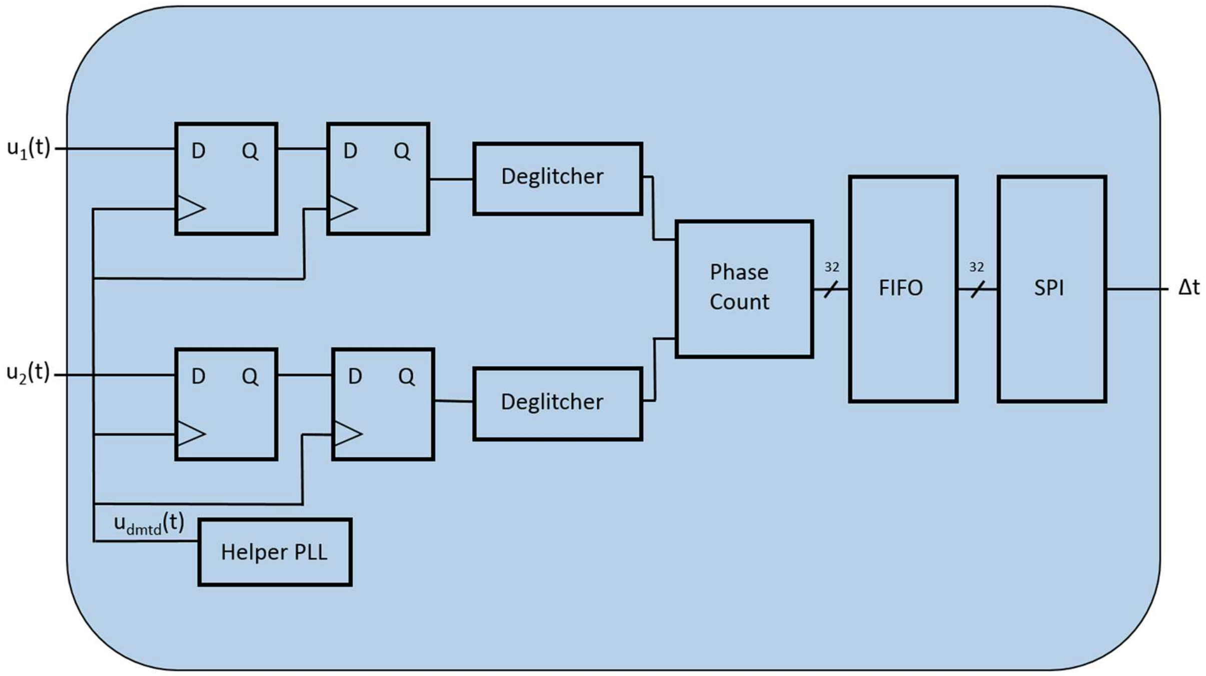


Figure 1: **Block diagram of D-DMTD**

The overall schematic of D-DMTD is shown above in *Figure 1*. The design consists of two input digital clock signals—$u_1(t)$ and $u_2(t)$—both of which have a base frequency of v_n and are separated by a phase difference called $\Delta t(t)$ or simply Δt. The Helper PLL (phase locked loop) is set by the user at a specific clock frequency in order to produce the $u_{dmtd}(t)$ clock signal; the purpose of $u_{dmtd}(t)$ is to sample $u_1(t)$ and $u_2(t)$ at a slightly lower frequency than v_n in order to spread out the two phase-shifted signals in the time domain. The value of v_{dmtd} is limited by the capabilities of the FPGA used; for example, the ProASIC3E an internal clock of 40 MHz and is unable to produce clock signals greater than 350 MHz. As discussed in a later section, the clock frequency, or v_{dmtd}, correlates to the level of resolution allowed in the estimation of Δt. The user specifies the frequency of $u_{dmtd}(t)$ using the following relationship:

$$v_{dmtd} = \left[\frac{N}{N+1}\right] v_n \qquad (1)$$

The 'N' value also affects v_{dmtd} and the level of resolution possible in the estimation of Δt, and is selected based on the FPGA used. The ProASIC3E has PLLs capable of achieving $\frac{127}{128}$, which yields an 'N' value of 127; this is the 'N' value used for all simulation and hardware testing.

2.1 Selection of ‘N’

The value designated as ‘N’—along with v_n—determines the precision of the D-DMTD estimation for Δt. Below, *Table 1* displays the calculated values of Δt_{min} for various frequency and ‘N’ value combinations, where Δt_{min} specifies the minimum resolution or the smallest phase shift that can be detected by the system. It is given by the following formula:

$$\Delta t_{\min} = \frac{1}{[N \cdot v_n]} \quad (2)$$

***Table 1:* Resolution achieved by D-DMTD with various ‘N’ values and different clock frequencies. The higher the ‘N’ values and clock frequency, the more precise the resolution.**

N →	*11*	*63*	*127*	*255*	*511*	*1023*
1 MHz	90.91	15.87	7.87	3.92	1.96	977.52
2.5 MHz	36.36	6.35	3.15	1.57	782.78	391.01
5 MHz	18.18	3.17	1.58	784.31	391.39	195.50
10 MHz	9.09	1.59	787.40	392.16	195.70	97.75
20 MHz	4.55	793.65	393.70	196.08	97.85	48.88
40 MHz	2.27	396.83	196.85	98.04	48.92	24.44
50 MHz	1.82	317.46	157.48	78.43	39.14	19.55
100 MHz	909.09	158.73	78.74	39.22	19.57	9.78
500 MHz	181.82	31.75	15.75	7.84	3.91	1.96
1.25 GHz	72.73	12.70	6.30	3.14	1.57	782

🟦 Nano-second ▒ Pico-second ■ Femto-second

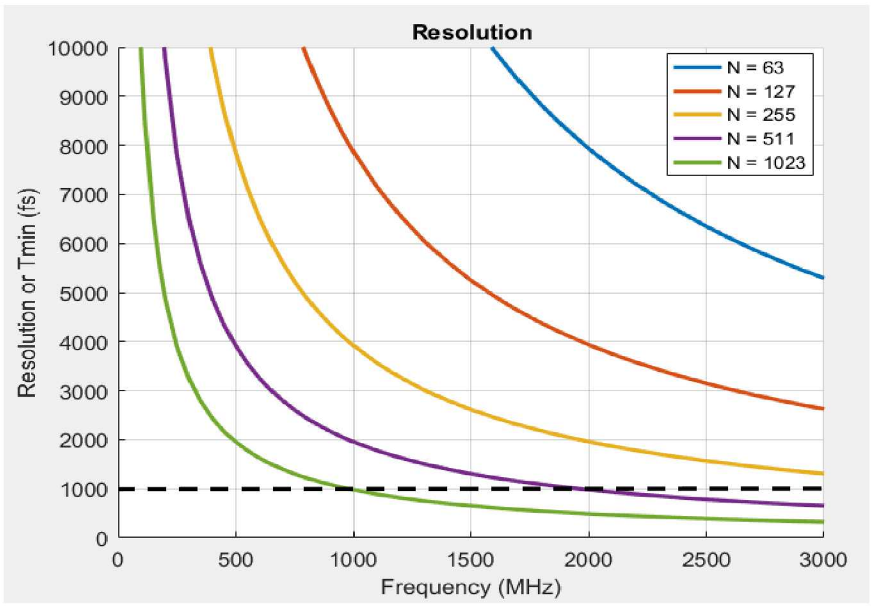


Figure 2: Graphical Representation of *Table 1* results. The threshold for sub-picosecond resolution is indicated by the dotted line.

In general, higher 'N' values and higher frequencies allow for finer resolution, which is shown in *Figure 2* above. This has primarily been a discussion of the variables that need to be set before running any simulation or test. The next sections detail how the digital design system physically manipulates the input signals and produces an estimation for Δt.

2.2 Beating Two Signals

As shown in *Figure 1*, $u_1(t)$ and $u_2(t)$ first pass through a set of D-flip-flops. On the rising edges of $u_{dmtd}(t)$, the D-flip flops sample the two input clocks, creating beat signals called $u_{beat1}(t)$ and $u_{beat2}(t)$. These are shown below in *Figure 3*.

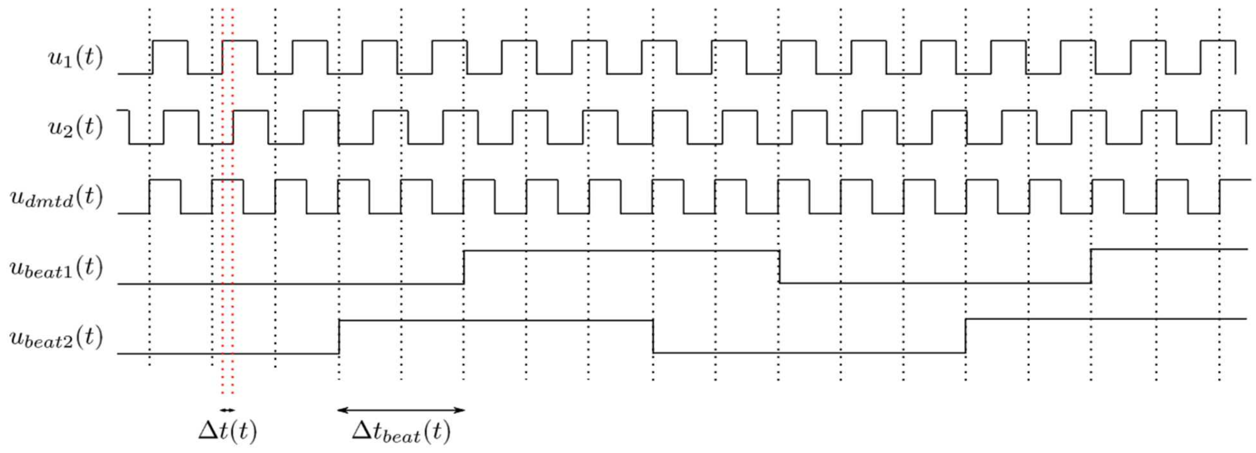


***Figure 3:* Sampling $u_1(t)$ and $u_2(t)$ with $u_{dmtd}(t)$ yields two elongated beat signals that are later processed to calculate the phase shift.**

A half period of the beat signal is equivalent to 'N' periods of $u_{dmtd}(t)$, since $u_{beat1}(t)$ and $u_{beat2}(t)$ each have 50% duty cycles. The beat signals above were created with an 'N' value of 5.

Essentially, by sampling a specific frequency signal with a slightly slower clock, D-DMTD stretches the signal out in the time domain, allowing for heavy aliasing of $u_1(t)$ and $u_2(t)$. It should also be noted that *Figure 1* indicates that the two clock signals each pass through two D-flip-flops that are tied together. This was done to address the issue of metastability in the hardware.

2.3 Deglitching the Signals

Uncontrollable environmental noise often propagates in the digital domain as horizontal phase noise, or jitter, which refers to the variation in period of an oscillatory signal. Depending on the amount of jitter, the act of passing of $u_1(t)$ and $u_2(t)$ through the D-flip flops can produce an output signal with an unpredictable number of glitches. One such signal is shown in *Figure 4*.

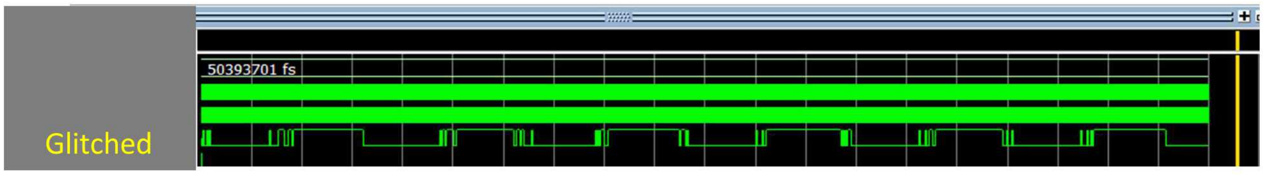


Figure 4: **Glitched signals arise from horizontal jitter in the input signals, which makes rising and falling edges indistinguishable.**

This presents a problem: the variable pattern of glitches masks where the rising and falling edges of the beat signals truly are, threatening the integrity of the original input signals and making an accurate phase shift detection unlikely. This uncertainty creates the need for a deglitcher mechanism.

The Deglitching Algorithm implemented in D-DMTD uses a shifting array that takes in a new value from a beat signal on every rising edge of $u_{dmtd}(t)$, while continuously evaluating whether the number of ones and zeros in the array are equal. When the number of ones and zeros become equal, the algorithm initiates a transition in the output "deglitched" signal: a 1→0 transition or a 0→1 transition. The results of the Deglitcher Algorithm are shown below in *Figure 5*.

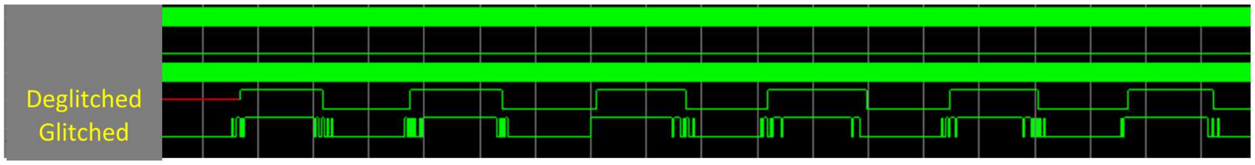


Figure 5: **Beat signals fed through the deglitcher block of the D-DMTD no longer have random spikes near rising and falling edges.**

There is one more variable of importance in this algorithm: the size of the shifting array, which can be determined by the user. If the shifting array size is too small, then a random pattern of glitches could be interpreted as a pulse incorrectly. If the shifting array is too large, then the array is rarely filled with an equal number of ones and zeros. Simulations for various window sizes and levels of jitter indicated that a window of size 20 yielded the most accurate estimations for Δt; the data that drove this decision is discussed in the next section.

2.4 Selection of Window Size

The relationship between window size, jitter, and standard deviation of Δt is of particular interest. Using simulation data to populate *Figure 6*, we measured our confidence in the estimation of Δt—a value dependent on the amount of environmental noise, or RMS jitter, and the size of our sampling window. The input signals were generated at 20 MHz, and separated by a phase shift of 1.3 ns, which is larger than the t_{min} value given by *Table 1* for a frequency of 20 MHz and an *N* value of 127.

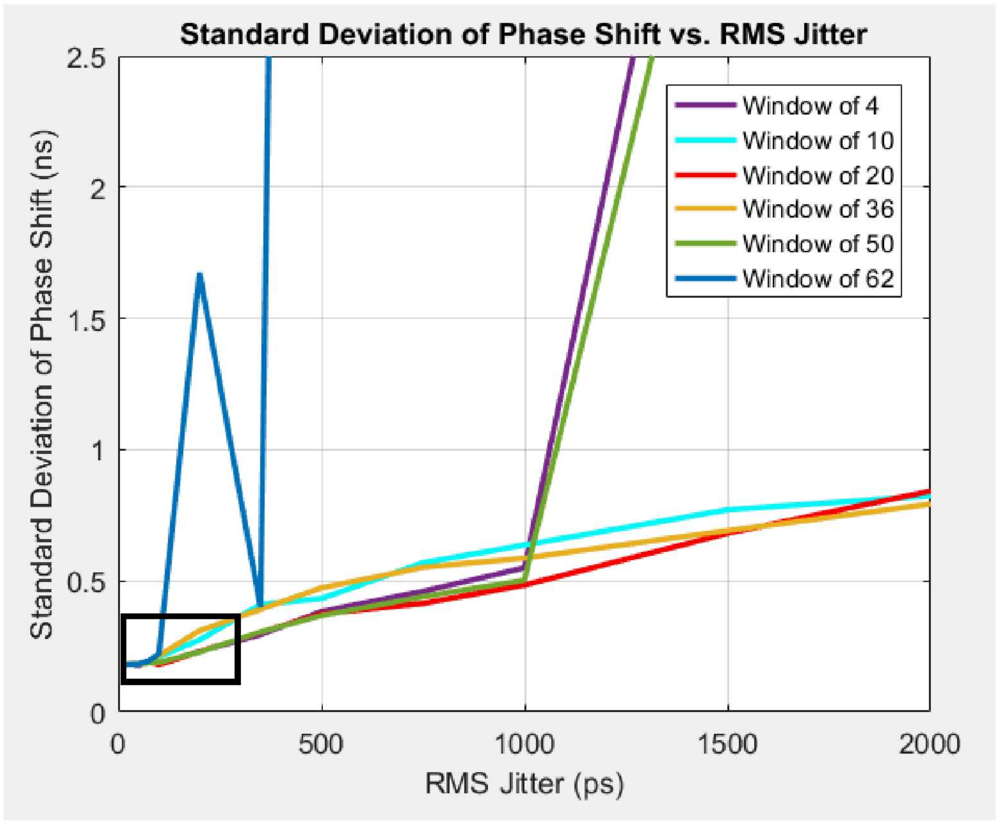


Figure 6: **Graph displaying the standard deviation of Δt from the ideal Δt value at varying jitter/window sizes.**

As one would expect, higher values of jitter or phase noise, correspond to less accuracy in the estimations for Δt. Thus, there is a positive, linear trend between RMS jitter and the standard deviation of Δt for most window sizes.

Of particular interest in *Figure 6* is how the relationship between jitter and the accuracy of Δt varies with window size. For some window sizes, the standard deviation-RMS jitter relationship follows fairly linear trends (i.e. 10, 20, 36), while others are much more unpredictable (i.e. 4) and highly sensitive to changes in RMS jitter (i.e. 50, 62).

However, environmental RMS jitter is rarely as high as 1500 ps; jitter on the scale of tens of picoseconds or less is a much more reasonable estimation. Thus, we focus in on the portion of the graph within the black box. *Figure 7* shows the zoomed-in area within the outlined region in *Figure 6*.

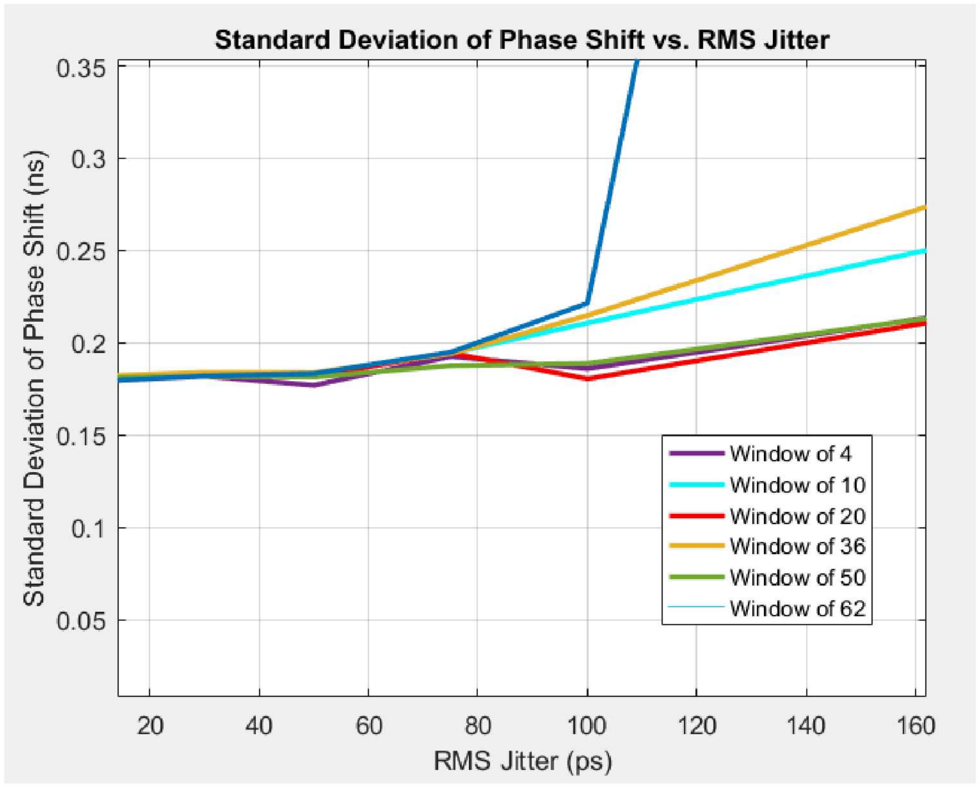


***Figure 7*: Zoomed in version of Figure 6, focusing on resolutions that may be of interest in real world applications.**

For more realistic values of jitter, the standard deviation of Δt is relatively independent of window size. The lines start to diverge as RMS jitter increases, with a window of 20 consistently yielding the least amount of uncertainty (red-colored line). This is the window size used for simulation and testing purposes.

2.5 XORing the Beat Signals

After deciding on a specific window size and running the deglitcher algorithm to generate two deglitched beat signals, the phase shift can be determined by looking at the XOR results of the two signals.

- XOR_result = 1 – signals are out of phase
- XOR_result = 0 – signals are in phase

The number of continuous '1's in the XOR result—the number of rising edges of $u_{dmtd}(t)$ occur when XOR_result = '1'—is related to Δt_{beat}. This XOR result is referred to as 'count', and its value represents the amount of time that the two deglitched *beat* signals are out of phase. This is proportional to the phase difference between the two *original* signals.

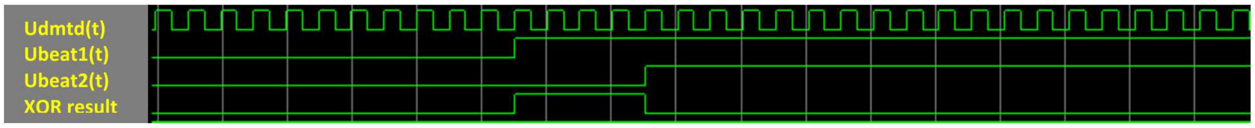


Figure 8: **XORing the two beat signals together produces a logic '1' when the beat signals are misaligned.**

As shown in Figure 8, the XOR result is '1' for 4 clock periods of $u_{dmtd}(t)$, each of which lasts 50.394 ns (based on 20 MHz input signals). With 'count' = 4, Δt_{beat} = 4 * 50.394 ns = 201.576 ns. It is then possible to calculate Δt using a ratio of frequencies. Using the equation below, Δt = 201.576 ns * (1 – 127/128) = 1.5748125 ns for the phase shift between the original two signals $u_1(t)$ and $u_2(t)$.

$$\Delta t = \Delta t_{beat}\left(\frac{v_{beat}}{v_n}\right) = (\Delta t_{beat})\left[1 - \left(\frac{N}{N+1}\right)\right] \tag{3}$$

The simulation stimulus set $u_1(t)$ and $u_2(t)$ to have a relative phase shift of 1.3 ns, so the estimation of 1.5748125 ns for Δt is slightly off. This is probably due to the high level of jitter induced in the input signals, and thus simply highlights need for to record many values of 'count', average them, and then calculate Δt_{beat} and Δt to get a more accurate estimation for the original phase shift between $u_1(t)$ and $u_2(t)$.

3. DESIGN VERIFICATION AND TESTING

3.1 Simulation Set-Up

Figure 9 shows the complete simulation for significant signals within the D-DMTD system. The first four signals at the top of the waveform were signals generated from the testbench (ddmtd_tb.vhd). The signals, input_signal_1_tb and input_signal_2_tb, were generated using a Matlab script (IdealClockNormalizedJitter.m).

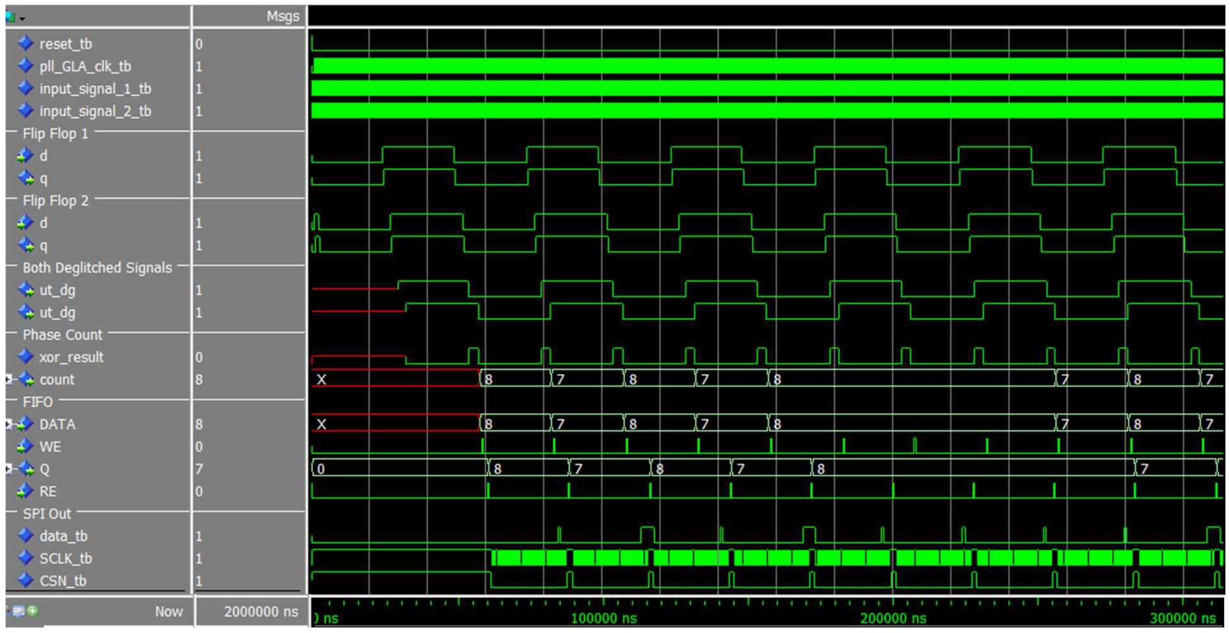


***Figure 9:* D-DMTD signals from input to D-flip-flops, deglitcher, XORing, FIFO, and SPI Interface.**

Pll_GLA_clk_tb is an output that is grabbed from the PLL IP Core instantiated in the top file. The PLL takes in a 40 MHz signal—the board clock frequency produced by ProASIC3E—and outputs a 2.480 MHz signal, which is the calculated $u_{dmtd}(t)$ value for a system where v_n is 2.5 MHz.

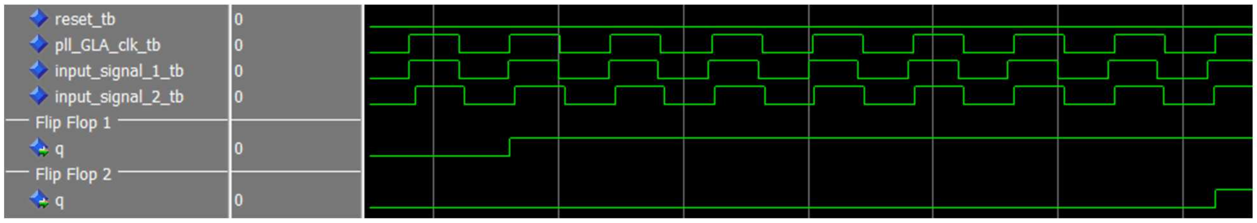


***Figure 10:* Out of phase beat signals are produced when $u_1(t)$ and $u_2(t)$ are sampled on rising edges of $u_{dmtd}(t)$**

Figure 10 is a zoomed in version of the simulation, with the inputs to the flip flops removed for clarity. The q's represent the beat signals that are produced by sampling the input signals with $u_{dmtd}(t)$—pll_GLA_clk_tb in *Figure 10*. In D-DMTD, there are actually a total of four D-flip flops (two for each input signal for metastability purposes), but only two are shown in simulation to avoid visual redundancy.

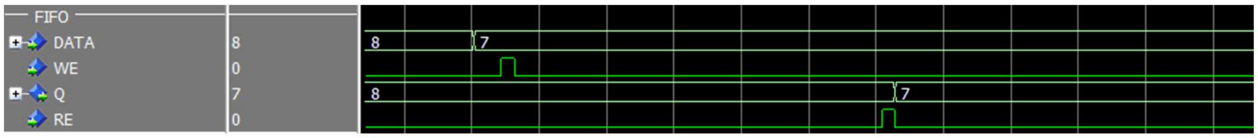


Figure 11: **Data being written to, and read from, the FIFO along with corresponding write enable and read enable signals**

Figure 11 is a zoomed in version of the FIFO results for the simulation. D-DMTD writes to the FIFO immediately after a new value is produced (from phase count algorithm). RE is enabled by the SPI interface to grab a value from the FIFO to output serially when SPI is not outputting data and the FIFO is not empty.

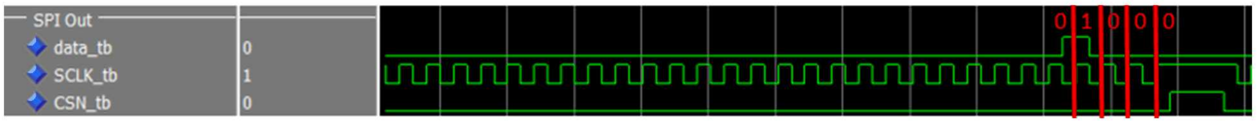


Figure 12: **Each SPI transaction consists of 32 output bits; in this figure, a binary 8 is produced**

Figure 12 is a zoomed in version of the SPI output signals. SCLK is running continuously in the background at a frequency of about 1000 KHz. Towards the end of the transaction, there is an extended pulse for SCLK that wraps the signal due to a COUNT_MAX calculation in the code that rounds our $u_{dmtd}(t)$ decimal value to an integer, causing truncation error. Data_tb in *Figure 12* indicates that the output value of this transaction is 8. It is determined by looking at data_tb relative to the rising edges of SCLK_tb. Each transaction is 32-bits, with the LSB (least significant byte) all the way to the right and ending with the extended SCLK_tb transaction. CSN_tb represents a framing bit that spikes in between different transactions.

3.2 Hardware Set-Up

After D-DMTD was successfully simulated in ModelSim 10.4, it was programmed onto the ProASIC3E FPGA. The DG645 Stanford Research Systems pulse generator was used to generate the two 2.5 MHz input signals, and the LogicPort Analyzer extracted the data from the SPI interface. The Logic Port would receive data each time the reset button on the FPGA was pressed, and would then write that information, in the form of 'count' values, to an excel file.

To compare how the physical lab testing compared to the simulations in ModelSim 10.4, the phase shift was varied between [0, 100] ns and the absolute error between the data and the input phase shift was observed. Theoretically, the simulation and hardware data should reveal no apparent trend between absolute error and phase shift; relative to each other, one might be offset from the other by a constant amount, information that when taken into account, would help increase the accuracy of later simulations by adding that offset to ModelSim's estimations.

The upper limit of the phase shift range for lab testing was set by the pulse generator; the DG645 is incapable of creating a 2.5 MHz signal with a 50% duty cycle at phase shifts greater than ¼ the period. After gathering a few hundred data points (Δt estimations) for each phase shift, the time-stamped excel files were post-processed using a Matlab script (read_process_HW_files.m). The results of this comparison between hardware and simulation absolute error in calculating Δt are shown in *Figure 13*.

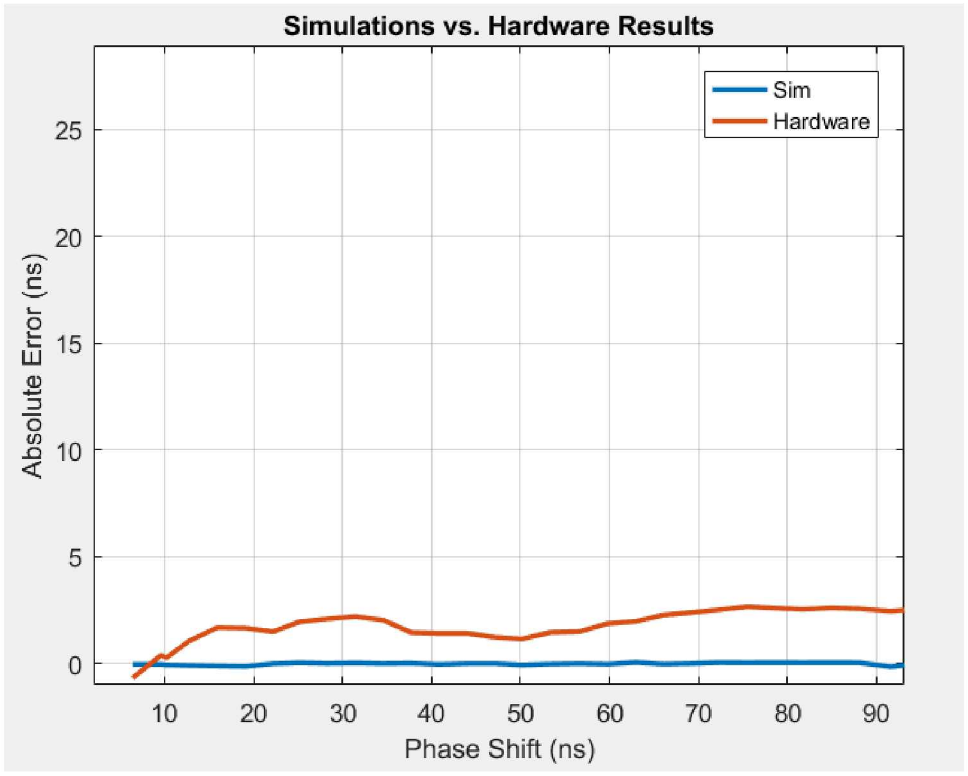


***Figure 13*: Comparison between hardware and simulation absolute error versus phase shift.**

Most noticeably in *Figure 13* is the fact that with a period of 400 ns for $u_1(t)$ and $u_2(t)$, the absolute error for hardware was only ~2 ns, and only 1 ns away from the simulation predictions. This is an absolute error of only 0.5%. Compared to the resolution allowed with a pure 2.5 MHz clock (400 ns), D-DMTD successfully provided resolution down to 3.14 ns (t_{min}) with minimal error.

Figure 13 also confirms that there is no notable trend between the absolute error of Δt and the phase shift between $u_1(t)$ and $u_2(t)$; this applies to both hardware and simulation estimations of Δt. Additionally, there is a fairly constant offset between the two lines, allowing for the accuracy of ModelSim's simulations to be improved upon by adding that difference in error to simulation estimations.

Again, due to limitations of the equipment available, the comparison study between the simulation and hardware results was conducted using only 2.5 MHz signals for $u_1(t)$ and $u_2(t)$. Even so, there was a clear correlation between the two sets of data, and proof that a physical implementation of D-DMTD can accurately estimate Δt. *Figure 13* also demonstrates that the difference in performance between hardware and simulation data will be minimal.

4. VHDL IMPLEMENTATION

4.1 Code Hierarchy

The code hierarchy is as shown in Figure 14.

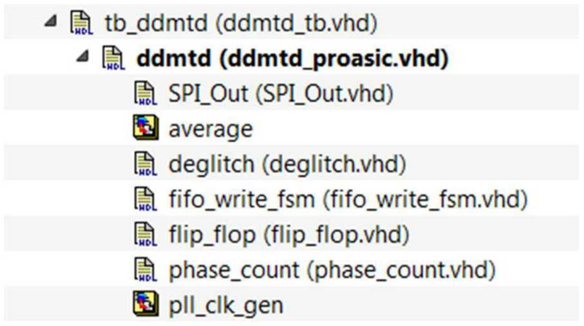

tb_ddmtd (ddmtd_tb.vhd)
ddmtd (ddmtd_proasic.vhd)
SPI_Out (SPI_Out.vhd)
average
deglitch (deglitch.vhd)
fifo_write_fsm (fifo_write_fsm.vhd)
flip_flop (flip_flop.vhd)
phase_count (phase_count.vhd)
pll_clk_gen

Figure 14: **Design hierarchy of D-DMTD including testbench wrapper**

- ddmtd_tb.vhd
    - ddmtd_proasic.vhd
        - SPI_Out.vhd
        - average.vhd (FIFO IP CORE)
        - deglitch.vhd
        - fifo_write_fsm.vhd
        - flip_flop.vhd
        - phase_count.vhd
        - pll_clk_gen (PLL IP CORE)

4.2 VHDL Files

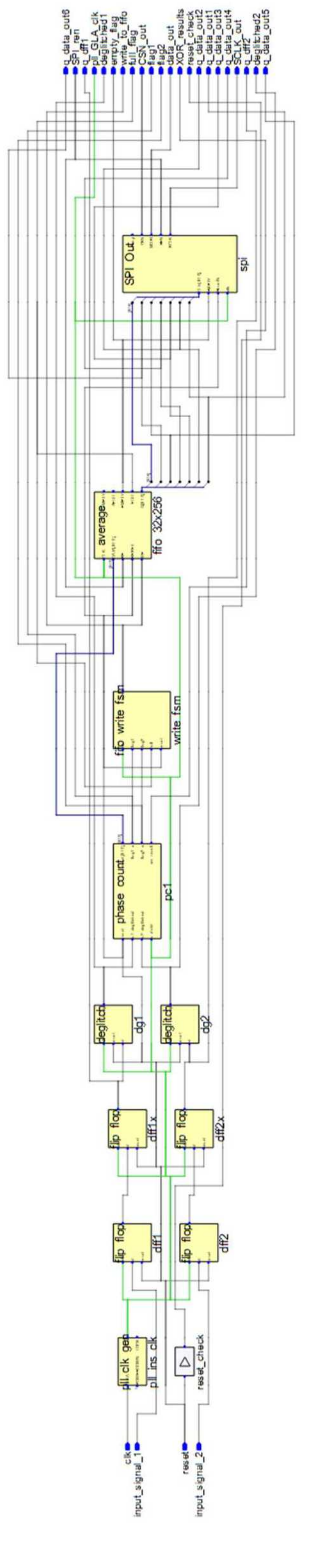
4.1.1 ddmtd_proasic.vhd

This is the main top level file in which we declare and instantiate all of our components. It is based on the design described in the Design section.

4.1.2 pll_clk_gen.vhd

This is an IP core that we used within Libero SOC to generate the proper clock frequency we needed for $u_{dmtd}(t)$. It has two outputs: “lock” and “pll_clk”.

Figure 15: Schematic of the components within D-DMTD



4.1.3 flip_flop.vhd

This entity is a regular D flip flop with an additional input, called “lock”, that indicates when the PLL clock is generating a valid signal that will allow us to sample our inputs correctly.

4.1.4 deglitch.vhd

This is an algorithm that we developed to accurately determine when the start and end of a signal should occur given glitches in the input signal that could arise. We used the zero count method, which involves the use of a shifting array (with a length that can be determined by the user) that is initially filled with zeros and shifts once to the left each clock of udmtd during which it takes in one new input value. When the number of 1’s and 0’s in the shifting array was equal, we flipped the signal from 1 to 0 or 0 to 1 based on what the signal was previously (kept track of this with two flags), producing deglitched beat signals. Depending on the window size used, the two deglitched signals would be slightly delayed from its original input. However, this is okay because the delays to both of the signals are equal. After running simulations, we determined a window size of 20 was optimal.

4.1.5 phase_count.vhd

We determined the Δt_{beat} by counting the number of clock cycles of u_{dmtd} that the two beat signals were out of phase. We determined this Δt_{beat} by XORing the two signals together and storing this value into a count variable that would be outputted from this block. This block would reset itself after each iteration, allowing us to get as much phase_count data as we wanted. The count was outputted as a 32-bit integer.

4.1.6 average.vhd

Average.vhd is a CoreFIFO instantiated within Libero SOC. This allows us to store values from phase count before sending them out of the FPGA. This is needed because phase_count.vhd produces data much faster than our SPI_Out.vhd can ship data out. The FIFO is 32x256. We arbitrarily decided to utilize 70 results from phase count.

4.1.7 fifo_write_fsm.vhd

This is a state machine that determines when to write to the FIFO by spiking the WE (write enable) signal. The state machine is controlled by the two flags from phase_count.vhd, which indicate when a signal is valid or about to change.

4.1.8 SPI_Out.vhd

We used SPI protocol to send out our data. This file utilizes a state machine that looks for when the FIFO is no longer empty before spiking the REN signal to grab a value and transmit out of the FPGA a single bit at a time. The user specifies the clock frequency used in the design (in our case udmtd), which helps determine the SCLK.

5. MATLAB ANALYSIS AND DATA PROCESSING

5.1 Resolution Limitation

tmin_resolution.m

This script calculates the values of t_{min} for 'N' values ranging from 63 to 1023, and frequencies (V_n) ranging from 0.1 MHz to 5 GHz. The resulting t_{min} values—calculated using Formula (2)—are plotted on a graph (*Figure 2*) and are given on the scale of femto-seconds.

5.2 Window Size

stdev_RMS_Jitter_ALL.m

This script takes the already processed data from ModelSim's simulations, in the form of standard deviation and RMS Jitter, and displays the points on a graph (Figure 6).

5.3 Simulation Signal Generation

IdealClockNormalizedJitter.m

This script generates two output .txt files that contain the data points for signals $u_1(t)$ and $u_2(t)$. The text of these files can then be transferred to the simulation testbench file to generate clock signals. The user can specify the frequency, relative phase shift, and amount of RMS jitter these signals inherently have. The script calculates the ideal data points for a signal with the frequency specified, and then adds a random amount of variation to each point that falls in the range [0, RMS jitter].

5.4 Data Processing and Graphing

5.4.1 Process_5_files.m

The purpose of this script is to process the five .txt files that are filled with the output of five different simulations run in ModelSim 10.4; the values stored in these files are various values of 'count'. This script parses the files, converts these values of 'count' to Δt, and then calculates the standard deviation and average in the time domain. The absolute error is calculated separately in excel.

5.4.2 read_process_HW_files.m

The purpose of this script is to process the data files obtained from in-lab or hardware testing. Data is collected from the FPGA using the Logic Port Analyzer, which then exports the values of 'count' to an excel file. This script opens the 100+ excel files, reads in the values for 'count', converts to the time domain, and calculates the average and standard deviation for the Δt value given by these files. This is the hardware equivalent of the Process_5_files.m, which processes the data gathered from ModelSim 10.4.

5.4.3 graphingHW_Sim.m

This script graphs the absolute error between the Δt estimated by both the hardware and simulations, and the Δt input to the system in the beginning. The results are shown in *Figure 13*.

6. LIMITATIONS AND FURTHER WORK

6.1 Limitations

6.1.1 D-DMTD Design

D-DMTD is unable to detect phase shifts greater than half the period of the input signals; phase shifts greater than this amount result in a loss of information. To calculate the total phase difference between the two input signals (including those greater than half the period), multiple instances of D-DMTD running at different frequencies (v_{dmtd}) can be implemented on an FPGA. Another option would be to obtain a coarse count for the number of periods that elapse between the two input signals that are fed into D-DMTD using the FPGA board clock.

6.1.2 Analog Frontend

A fully-functional, sub-clock digital phase-detector (D-DMTD) requires analog circuitry capable of converting two environmental triggers into two clocks separated by a relative phase difference equivalent to the time elapsed between the two events. Extensive research was done to see if conventional, industry-standard IC oscillators can accomplish this; however, while many off-the-shelf products offer the desirable frequency range and level of precision, they are not designed to work for this purpose. Dual oscillator ICs were considered, but they would need to have deterministic power up times, so that the same power-up time delay would be applied to both trigger signals and not affect the natural temporal separation.

A possible alternative to generating the two square wave input signals for D-DMTD, is an analog or digital form of a ring oscillator. The limitation here would be the frequency of the square wave that can be generated, which correlates to the propagation delay of the circuitry (not gates) needed to make the signal oscillate.

6.2 Further Work

The digital implementation of the project was limited by the equipment available. The Stanford DG645 pulse generator is incapable of producing multiple phase-shifted clock signals with 50% duty cycles at frequencies greater than 2.5 MHz. Other industry-available pulse generators are capable of much higher frequencies, which, as has been discussed earlier, affects the resolution of the digital phase detector. Thus, future work would entail increasing test equipment capability and conduct tests to provide more data points for analysis.

Jitter is another issue that needs to be considered as the frequency of $u_{dmtd}(t)$ increases to obtain more precise timing resolution. As the sampling rate increases, $u_{dmtd}(t)$ will be sampling closer to the rising and falling edges of the input signals, causing more glitches. Within the D-DMTD design, the deglitcher mechanism has the most room for improvement. By analyzing the results of different deglitching methods/algorithms, the accuracy of D-DMTD can be improved. The zero count detection method was employed for the current D-DMTD design, but there may be better alternatives.

7. REFERENCES

[1] Torres, J., et al. "Time-to-Digital Converter Based on FPGA With Multiple Channel Capability." *IEEE Transactions on Nuclear Science*, vol. 61, no. 1, 2014, pp. 107–114., doi:10.1109/tns.2013.2283196.

[2] Wang, Yonggang, et al. "A Multi-Chain Merged Tapped Delay Line for High Precision Time-to-Digital Converters in FPGAs." *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2017, pp. 1–1., doi:10.1109/tcsii.2017.2698479.

[3] K. Karadamoglou, N. Paschalidis, , N. Stamatopoulos, et al., A cmos time to digital converter for space science instruments., Proceedings of the 28th European Solid-State Circuits Conference (ESSCIRC 2002), pp. 707–710, 2002

[4] Wu, Jinyuan, and Zonghan Shi. "The 10-Ps Wave Union TDC: Improving FPGA TDC Resolution beyond Its Cell Delay." *2008 IEEE Nuclear Science Symposium Conference Record*, 2008, doi:10.1109/nssmic.2008.4775079.

[5] Moreira , P., and I. Darwazeh. "Digital Femtosecond Time Difference Circuit for CERN's Timing System ." *London Communications Symposium 2011*.

[6] Roberts, Gordon W., et al. "A Brief Introduction to Time-to-Digital and Digital-to-Time Converters." *IEEE Transactions on Circuits and Systems - II: Express Briefs* , vol. 57, no. 3, Mar. 2010.

[7] Moses, W.w. "Time of Flight in Pet Revisited." *IEEE Transactions on Nuclear Science*, vol. 50, no. 5, 2003, pp. 1325–1330., doi:10.1109/tns.2003.817319.

[8] Microsemi. "ProASIC3/E Proto Kit User's Guide." https://www.microsemi.com/document-portal/doc_view/130776-proasic3-e-proto-kit-user-s-guide-and-tutorial.

8. APPENDIX

DMTD	dual mixer time difference
DDMTD	Digital- dual mixer time difference
$\Delta t(t)$ or Δt	Phase shift between two digital clock signals; represents time between two initial trigger events
Δt_{min}	The minimum phase shift we can detect between two digital clock signals; measure of resolution
N	Variable that establishes ratio between vn and vbeat
$U_1(t)$	Input signal with frequency vn
$U_2(t)$	Input signal with frequency vn; phase shifted relative to U1(t)
$U_{dmtd}(t)$	Clock signal that samples u1(t) and u2(t); set at a slightly lower frequency so as to spread the two signals out in the time domain
$U_{1beat}(t)$	Result of passing U1(t) through d-flip-flops; stretched out in time domain
$U_{2beat}(t)$	Result of passing U2(t) through d-flip-flops; stretched out in time domain
V_n	Frequency of input signals: U1(t) and U2(t)
V_{beat}	Frequency of beat signals: U1beat(t) and U2beat(t)
V_{dmtd}	Frequency of Udmtd(t); slightly less than Vn
Window	Size of shifting array in deglitching algorithm; affects how
PLL	Phase-locked loop; responsible for creating udmtd(t) signal
SPI	Serial Peripheral Interface; used for synchronous serial communication