Time to Digital Converter

a Vernier Implementation on Field Programmable Gate Arrays

Bachelor Thesis Electrical Engineering SS2019

Authors
Noah Kälin, Michael Schmid
Supervisor
Prof. Dr. Paul Zbinden, IMES
Assistant Supervisor
Dorian Amiet, IMES
Subject
Digital Microelectronics

HSR Hochschule Für Technik Rapperswil
December 3, 2025

This document is created with LaTeX and Ti*k*Z

The layout is based on Prof. Dr. Andreas Müller's *Partial Differential Equations* lecture notes

If you are interested in one of the tikz images contact michaelschmid13@hotmail.com

Abstract

Introduction With the ongoing development in various scientific fields like nuclear physics, biomedical engineering and time-of-flight applications, the need to measure short time intervals and convert these into a digital format is higher than ever before. Time-to-Digital Converters (TDCs) are capable of measuring time intervals in the picosecond (1ps = 10^{-12} s) region. The Institute for Microelectronics and Embedded Systems has used Tapped Delay Line (TDL) TDCs successfully for many years. The downside of these TDLs is, that they rely heavily on constant delay times. For this thesis, new implementation methods should be evaluated and implemented. Special attention shall be paid to the Vernier principle.

Approach In the literature study, the most appealing papers were summarized. In particular, two subsequent publications from the same research group caught our attention. The centerpiece of both is a counter matrix. One of them is based solely on the use of the Phase-Locked Loop (PLL) hardware of the Field Programmable Gate Array (FPGA). These PLLs are phase shifting and overclocking the system clock in order to generate a discrete uniform distributed clock network for the counter matrix.

The second one uses a hybrid structure that utilizes the PLL in combination with delay elements. Both variants were implemented as single-core and multi-core TDCs on several Xilinx FPGAs using Very High Speed Integrated Circuit Hardware Description Language (VHDL).

Conclusion The achieved standard deviation for the best possible implementation with a stochastic multi-core approach is as low as 13 ps. Even with an implementation on an inexpensive FPGA, good results can be achieved. The performance of the implemented TDCs is in the same range as their counterparts in the preceding papers. However, a properly calibrated TDL TDC is superior regarding the standard deviation. Nevertheless, this thesis shows an easy to handle and capable alternative.

Acknowledgement

We want to express our great appreciation to our supervisor, Prof. Dr. Paul Zbinden and assistant supervisor Dorian Amiet for their valuable and constructive support. They helped us in many ways throughout the whole project with their extensive experience.

Finally, we want to thank our families for supporting us in writing a Bachelor's thesis at home during this special time due to the lockdown in spring 2020.

Contents

Abbreviations

ADC Analog-to-Digital Converter

BUFG Global Clock Buffer

BUFH Horizontal Clock Buffer

CDF Cumulative Distribution Function

CLB Configurable Logic Block

DDR Double Data Rate

DNL Differential Nonlinearity

FF Flip Flop

FPGA Field Programmable Gate Array

GRO Gated Ring Oscillator

INL Integral Nonlinearity

LSB Least Significant Bit

LUT Look-Up Table

MMCM Mixed-Mode Clock Manager

PDF Probability Density Function

PLL Phase-Locked Loop

RTL Register Transfer Level

TDC Time-to-Digital Converter

TDL Tapped Delay Line

TNS Total Negative Slack

UART Universal Asynchronous Receiver Transmitter

USB Universal Serial Bus

VCO Voltage Controlled Oscillator

VDL Vernier delay line

VHDL Very High Speed Integrated Circuit Hardware Description Language

WNS Worst Negative Slack

Chapter 1

Introduction

With the ongoing development in various scientific fields like nuclear physics, biomedical engineering and time-of-flight applications, the need to measure short time intervals and convert these into a digital format is higher than ever before TDCs are capable of measuring time intervals in the picosecond ($10^{-12}s$) region. The Institute for Microelectronics and Embedded Systems has used TDL TDCs successfully for many years. The downside of these TDLs are, that they rely heavily on constant delay times. Prior to this thesis, one was conducted that successfully calibrated such a TDL TDC [**?**]. The drawback of the calibration just mentioned is that it is very hardware-intensive and the calibration is dependent on the measurement data. For this thesis, new methods should be evaluated and implemented. Special attention should be given to the Vernier principle.

The Vernier principle is not a well-defined method, but more a general term for a measuring method with multiple frequencies. A promising approach is the utilization of the onboard PLLs of an FPGA. These PLLs allow to build a complex clock network with various frequencies and phases. It is expected that a method with the Vernier principle is much more reliable and that a possible calibration could be carried out with ease.

In the last couple of years, many papers that utilize some sort of a Vernier TDC have been published. The first task of this thesis is to carry out a broad literature study, where recent papers in the field shall be studied and the most appealing ones shall be summarized. This document is divided into five main parts together with an appendix. After the current chapter 1 Introduction follows the introduction to the topic of TDCs in chapter 2 Fundamentals. The main part with the chosen methods and their implementation descriptions is in chapter 3 Methods. Following, chapter 4 Results shows the achieved results with this work. In chapter 5 Conclusion the achieved results are put into relation and the general outcome of the thesis is discussed.

Chapter 2

Fundamentals

2.1 Literature Study

TDCs can be implemented in various ways. The simplest method for a time measurement is counting with a certain frequency, mostly the system clock. To get the desired resolution of more or less 10 ps, a clock frequency of 100 GHz would be needed. Such high frequencies are rather impractical. Therefore, a system, which is capable of measuring times shorter than one clock cycle is required. A wide literature study in the field of TDCs on FPGA is conducted. Several key ideas for the implementation of TDCs appear repeatedly in scientific publications. Besides the basic principle, the exact implementation differs from paper to paper. Following, the most important principles are shown.

The images in this section are all self-drawn with Ti*k*Z, but are inspired by images in the referenced papers.

2.1.1 Tapped Delay Line TDC

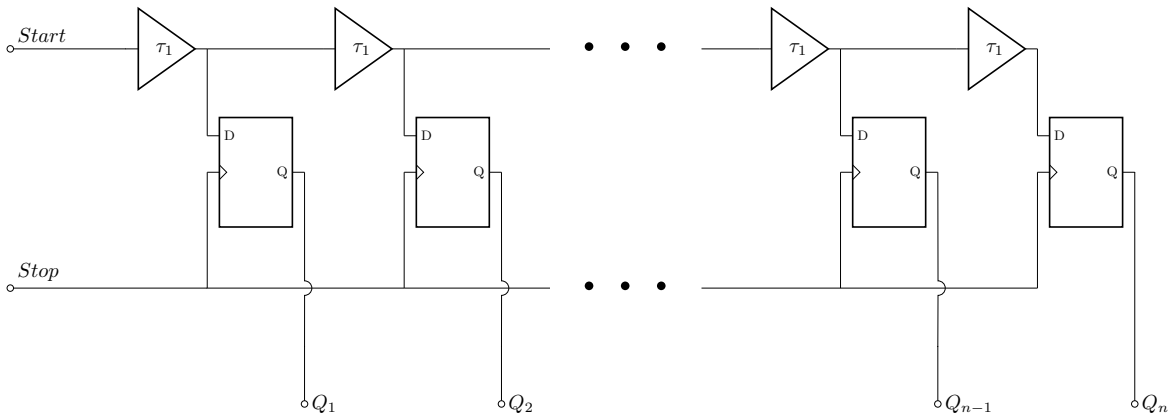


Figure 2.1: Block diagram Tapped Delay Line

The TDL is a very simple implementation method of a TDC whose block diagram can be seen in Figure Fig. 2.1. The principle is that a chain of delay elements delays the start signal. The start signal is a unit step and every output of a delay element goes to a D-Flip Flop (FF).

The step propagates through the delay line. The stop signal, also a unit step, is started when the stop event occurs. Because there are no delay elements in the stop signal line, the stop signal appears (theoretically) instantaneously at the clock input of every FF. The output vector Q is a thermometer code that represents, how far the start signal has propagated. The time T between the start and stop event can be computed with the delay time τ_1 and the number of activated FFs n:

$$T = \tau_1 \sum_{n=0}^{N} Q_n \tag{2.1}$$

2.1.1.1 Challenges

The main challenge of the TDL is that the delay time τ has to be known and that they should be identical throughout the entire delay line.

With this method, a calibration is unavoidable for a good resolution. Several methods have been proposed by a previous student project [**?**].

2.1.1.2 FPGA Implementation

The programmable logic of a standard FPGA contains arrays of Configurable Logic Blocks (CLBs) and switch matrices. These CLBs differ from manufacturer to manufacturer. In the Xilinx 7 Series, it contains two slices with a separate carry chain. Whereas a slice contains logic function generators or Look-Up Tables (LUTs), several FFs, multiplexers and a Carry logic. [**?**]

For the implementation of the TDL, the carry chain of the slice structure in an FPGA can be used. The signal propagates through the multiplexers in the carry chain. These multiplexers have a more or less consistent delay time.

2.1.1.3 Stochastic approach

The mentioned challenge of the TDL is that the delay time of a delay element should be known. A rather simple way to avoid variation in the delay time is to measure with multiple TDC cores and take the average. In Fig. 2.2, the principle schematics are shown.

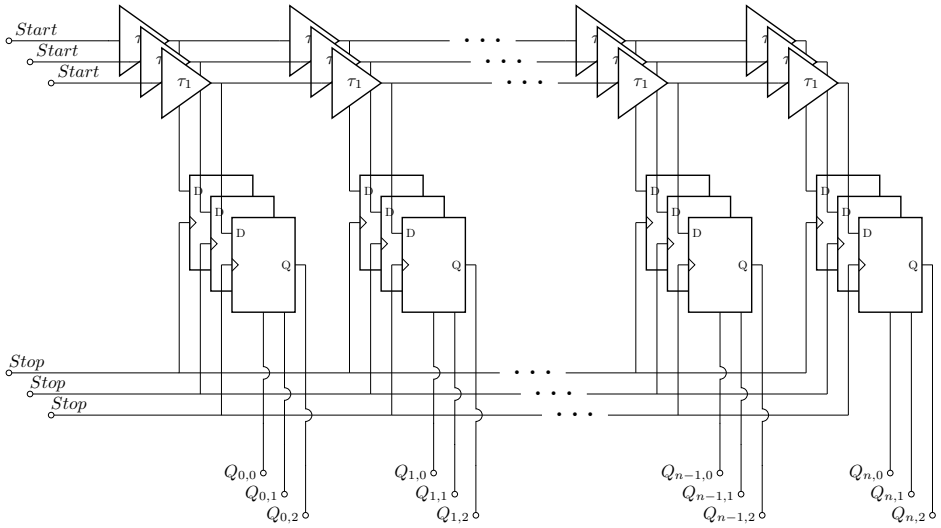


Figure 2.2: Block diagram Stochastic Tapped Delay Line

2.1.2 Vernier Delay Line TDC

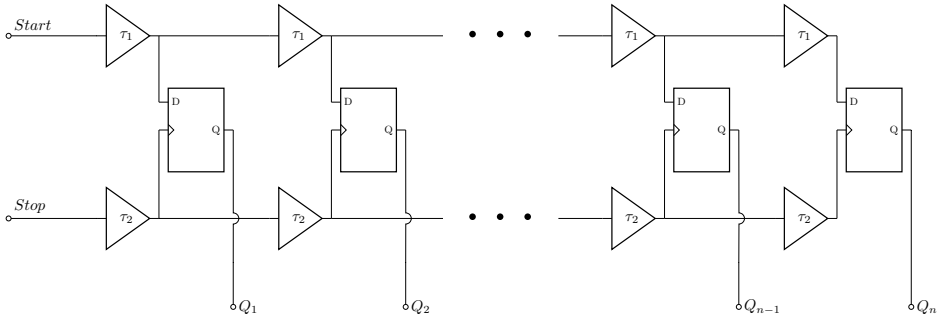


Figure 2.3: Block diagram Vernier Delay Line

The Vernier delay line (VDL) works according to the Vernier principle. A more self-explanatory name would be differential delay line. This additional name already suggests that this method uses a differential approach to achieve sub-gate delay resolution. The basic functionality can be seen in Fig. 2.3. The start signal that propagates through the start signal line from FF to FF is delayed by a delay time τ_1. Additionally, the stop signal that is connected to the clock input of every FF, gets delayed by a different delay time τ_2. The resulting timing diagram can be seen in Fig. 2.4. The (theoretically) shortest measurable time is the difference between the two delay times [**?**]:

$$T_{min} = \tau_1 - \tau_2 \tag{2.2}$$

2.1.2.1 Challenges

The same applies as for the TDL. Another challenge is to have two slightly different delay elements, where τ_2 is marginally shorter than τ_1.

2.1.2.2 FPGA Implementation

One cannot generate two slightly different delay elements with ease on an FPGA that could be used for such a delay line. There are several ways around this problem, for instance, the one described in section 2.1.4.

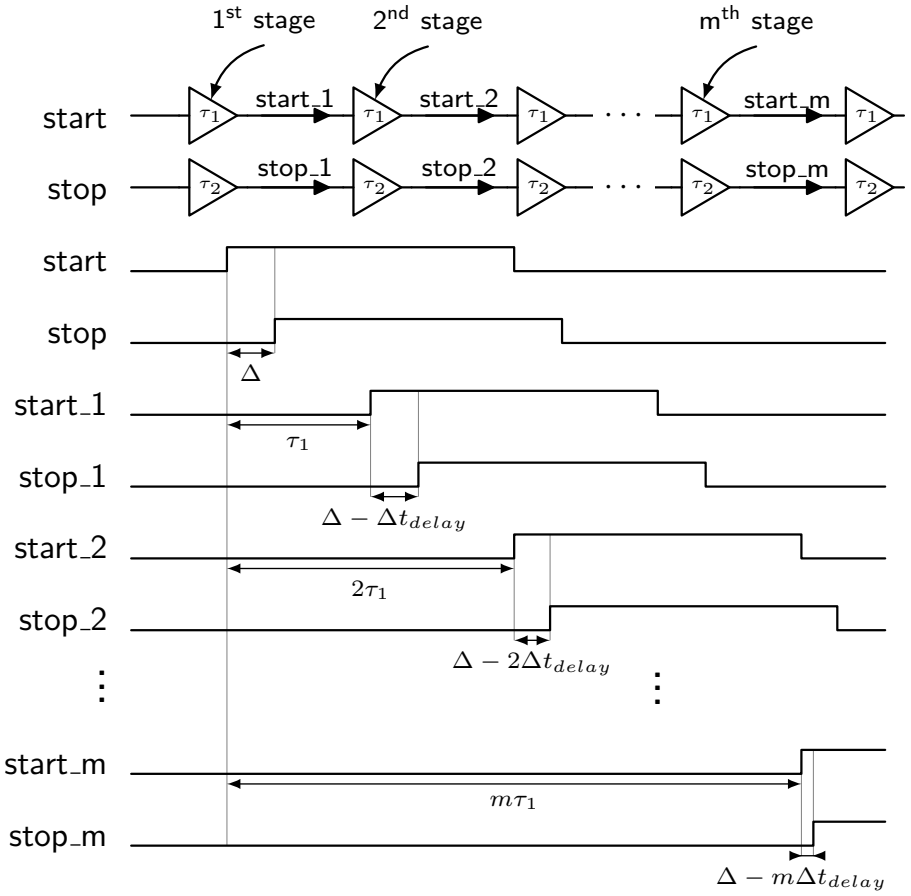


Figure 2.4: Timing diagram Vernier Delay line

2.1.3 Gated Ring Oscillator

The Gated Ring Oscillator (GRO) is a frequently mentioned method to implement a TDC. Several recent papers are based on a principle using a GRO, one of them is [**?**]. Usually, gated inverter elements are used for the implementation. As soon as the input pulse gets high, the ring begins to oscillate. As long as there is an odd amount of inverter blocks, the ring will always be able to start oscillating. The counters in Fig. 2.5 count how many times the inverter has switched its state. With this information and the propagation delay of the inverter, the 'on' time can be computed.

With some further improvements, TDCs with a high performance can be implemented [**?**]. The FPGA implementation is rather difficult, due to the lack of fast inverters.

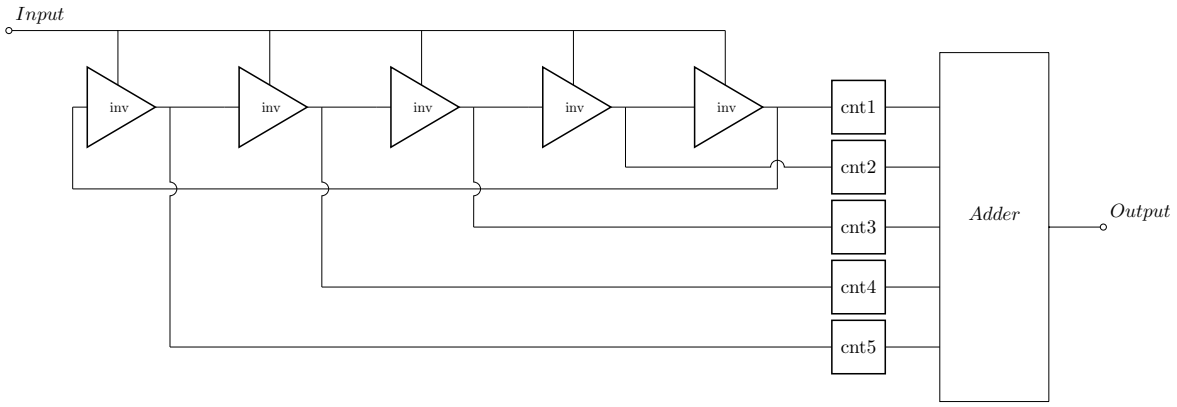


Figure 2.5: Block diagram Gated Ring Oscillator

2.1.4 Delay Wrapping

This method [?], [?] is a modified version of the VDL, which overcomes the problem that two different delay times are needed. First, the reason, why it uses a 2D arrangement is explained and then it will be evolved further so that it is implementable on an FPGA and only needs one delay time.

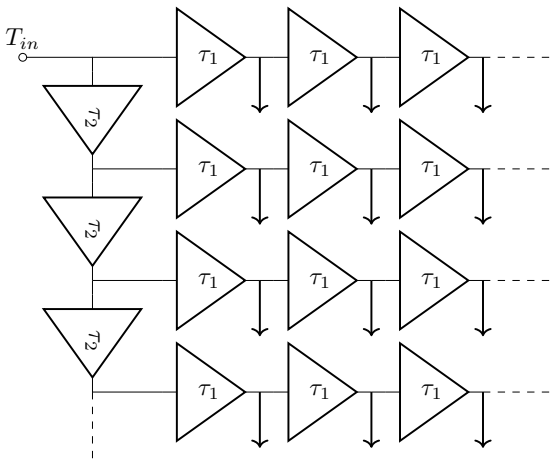


Figure 2.6: Delay matrix with buffers

In Fig. 2.6, the basic idea is shown. The signal that needs to be measured is delayed not only in a one-dimensional delay line, but in a two-dimensional delay array, which is called delay matrix. With this delay matrix the signal gets shifted many times and with a much finer resolution than the system clock. All these shifted signals (downwards pointing arrows) are being summed up by a counter each. Those counters are counting for how many clock edges their corresponding T_{in} is high. The width of T_{in} can be computed with this sum and the achieved resolution. The resolution is given by

$$\mathrm{LSB} = \frac{1}{H\,\nu\,f}, \tag{2.3}$$

where H is the number of vertical delay elements and v is the number of horizontal delay elements. Furthermore, it is important that the total delay of the delay matrix results in a full clock period.

The advantage is that the (horizontal) delay lines do not need to be as long as in a conventional VDL. Still, this configuration cannot be implemented easily on an FPGA because it needs to have two different delay elements. Before we solve this fundamental problem, we will improve the structure of the array.

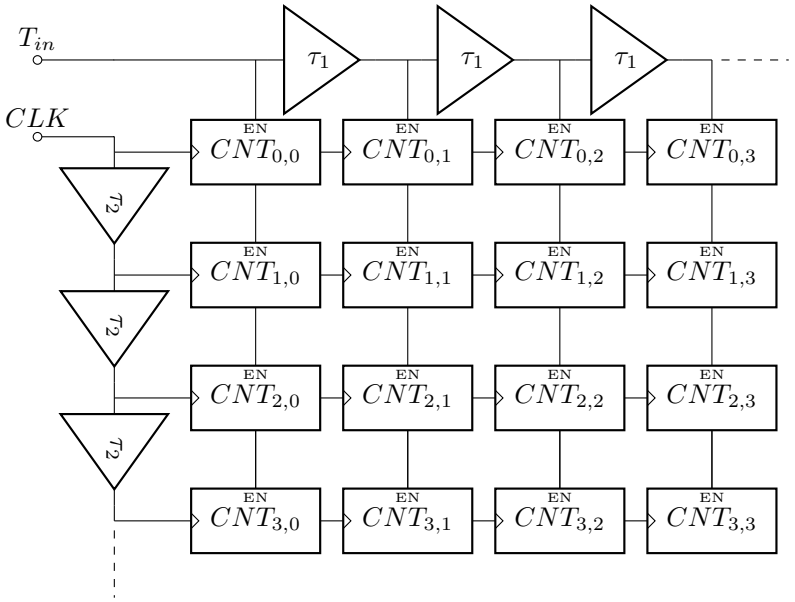


Figure 2.7: Hybrid delay matrix with counting elements

The buffer elements in the array can be substituted with counting elements, as can be seen in Fig. 2.7. Each element ($CNT_{0,0}, CNT_{0,1}, \ldots, CNT_{n,m}$) represents a counter. As in the paragraph above, all output values of these counting digits are summed and the pulse width can be computed.

Before, we only had a delay line/matrix for the input signal, but now we split it up and have one delay line for a clock signal and one for the input signal. Both delay lines combined control the matrix. This is the reason, why the design is called a **hybrid delay matrix**. Even though there are now two separate delay lines, still fewer delay elements are needed, because the whole matrix structure got replaced by counting elements.

The next step is to eliminate the need for two different delay times.

2.1.4.1 PLL and Delay Element Delay Matrix

The delay elements in the clock signal path can be replaced by a PLL block. The design is shown in Fig. 2.8. The PLL has the clock signal as an input and generates output clocks ($C_0, C_1, C_2, \ldots$) which all have different phase shifts. Then each one is routed to a row of counting elements. The resolution of the measured time is limited by the combination of the delay time of a delay element in the signal line and the phase shift in the clock signal line. If we have eight PLL outputs and the phase shifts are evenly distributed, we will have a resolution of 1/8-th of a clock period in the clock signal line. On top of that, if we have eight delay elements in the signal

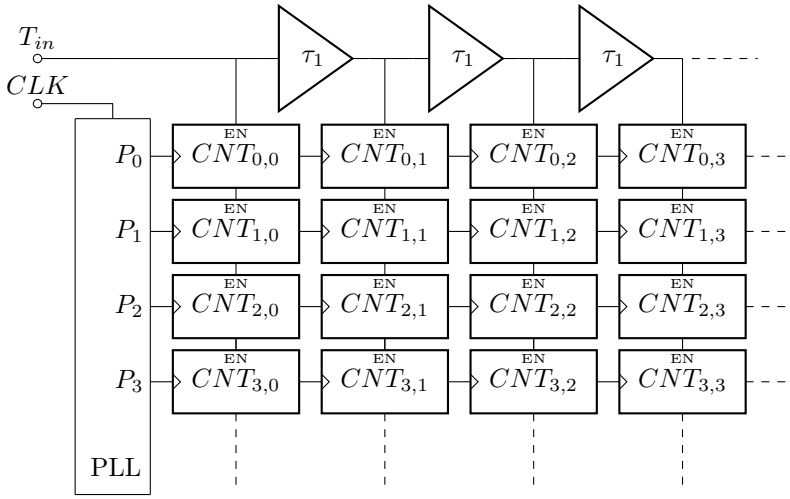


Figure 2.8: Hybrid delay matrix with counting elements and PLL

line and they fit together perfectly with the clocks, the resolution in the signal line will also be 1/8-th of a clock period. In this example, the resulting (theoretical) resolution could get down to $1/8 \cdot 1/8 = 1/64$th of a clock period. But only in a theoretical sense, the practical resolution would be worse because of uncertainties in the PLL frequency and the variation in delay time of each delay element.

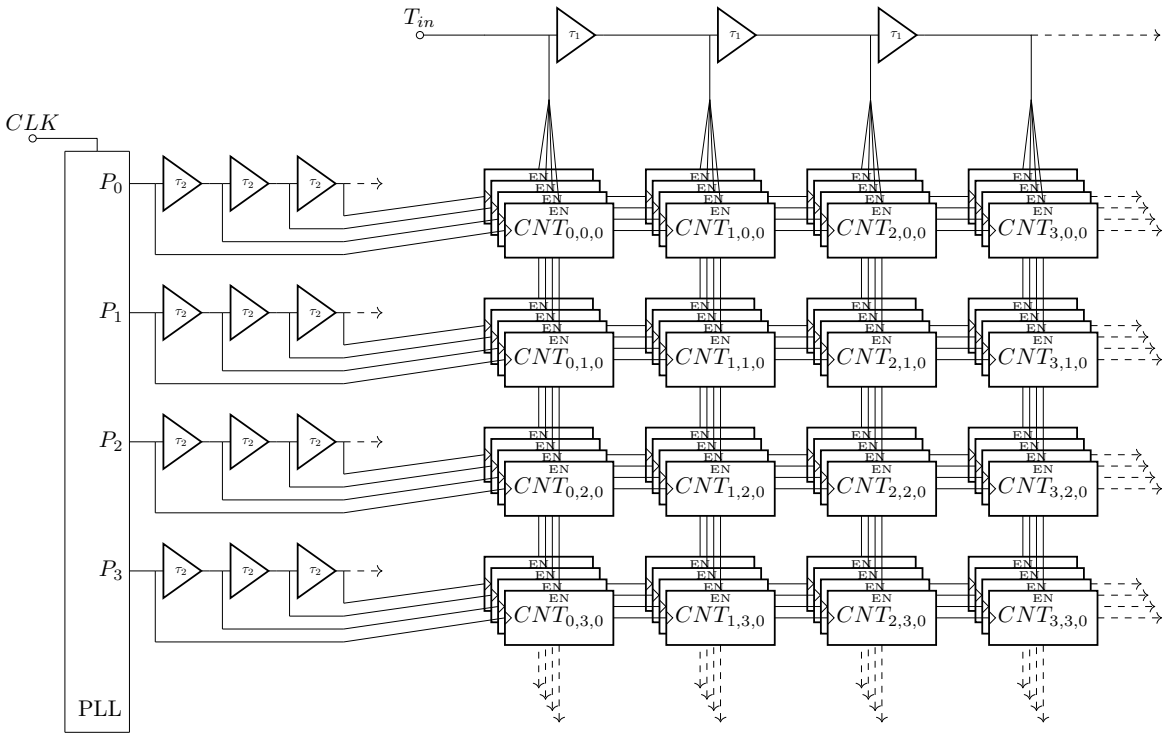


Figure 2.9: 3D hybrid delay matrix with counting elements and PLL

To achieve even higher resolutions, we can build more than just one delay matrix and stack

them. This is now called a **3D hybrid delay matrix** and can be seen in Fig. 2.9. Each layer (one layer = one delay matrix) needs a clock signal that is different from all of the others. In order to get many different clock signals, we need to delay each PLL output with a delay element. Then they all are connected to their own row and layer in the delay matrix.

Resolution For the computation of the theoretical resolution, the characteristics of this implementation have been put into variables. The used notation is the same as in [**?**].

- P stands for the number of PLL outputs
- ν stands for the number of Delay elements after the PLL
- H stands for the number of Delay elements after the input T_{in}
- f stands for the frequency of the output clock of the PLL

The resolution is therefore

$$\text{LSB} = \frac{1}{H \nu P f}. \tag{2.4}$$

When using the same example as in the paper, $H = 5, \nu = 10, P = 10$ and $f = 800$ MHz

$$\text{LSB} = \frac{1}{5 \cdot 10 \cdot 4 \cdot 800 \cdot 10^6\,\text{Hz}} = 2.5\,\text{ps} \tag{2.5}$$

a final resolution of 2.5 ps is achieved.

2.1.4.2 PLL Only, Delay Matrix

We can still evolve this design further and eliminate the use of delay elements completely. We remove the delay elements in the input signal line and connect it directly to the enable input of all counting elements. The delay elements in the clock signal line are replaced by PLL blocks. These changes are shown in Fig. 2.10.

The resolution is computed similarly to section 2.1.4.1. Again, the same notation as in the corresponding paper [**?**] is used.

- M stands for the number of PLL outputs of the first stage
- N stands for the number of PLL outputs of the secondary stage
- f stands for the frequency of the first PLL stage

The resolution can therefore be described as

$$\text{LSB} = \frac{1}{f M N}. \tag{2.6}$$

Double Data Rate One way to enhance the resolution of the structure even further is to use the Double Data Rate (DDR) technique, where there are FFs in the counting matrix that react to the falling and the rising edge of the clock. The new resolution is doubled.

$$\text{LSB} = \frac{1}{f M N 2} \tag{2.7}$$

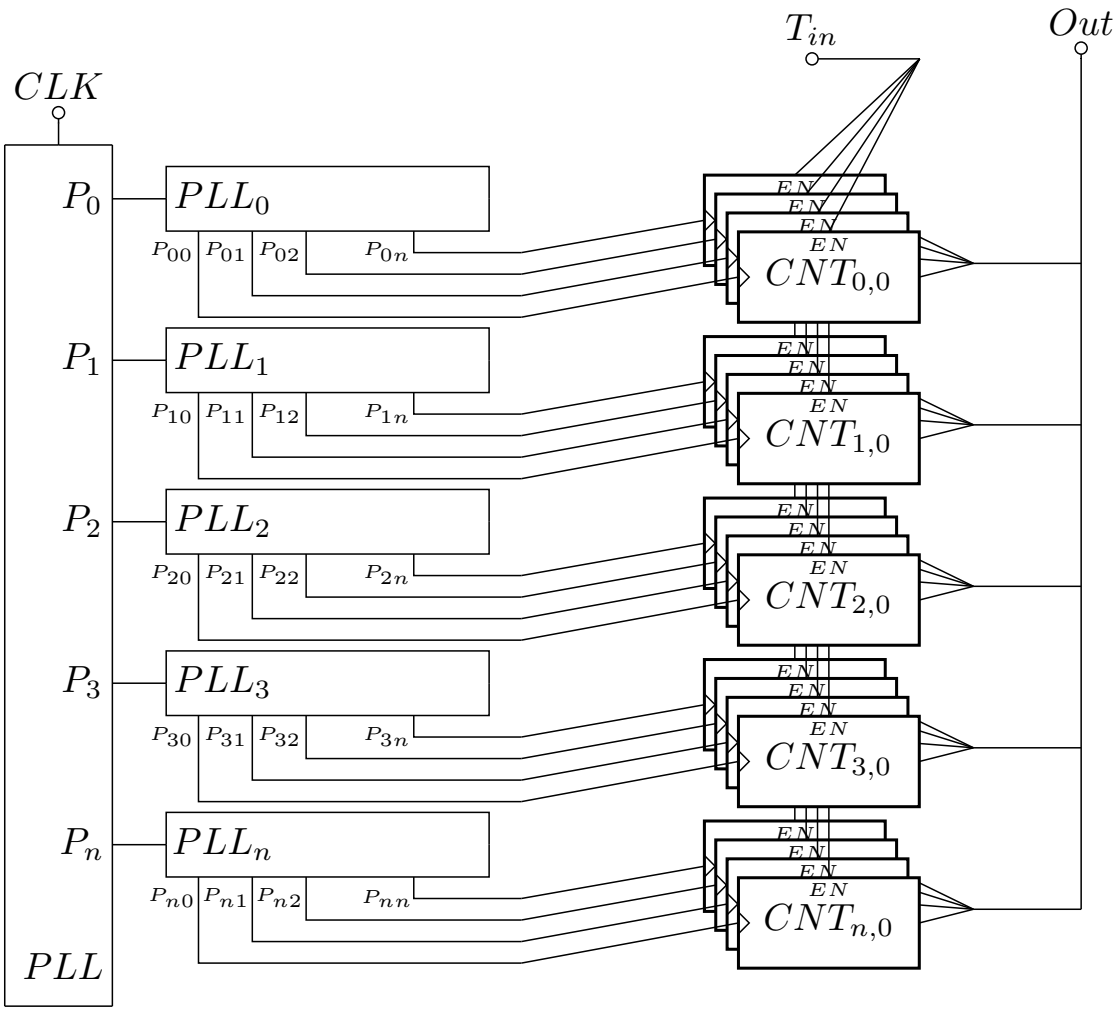
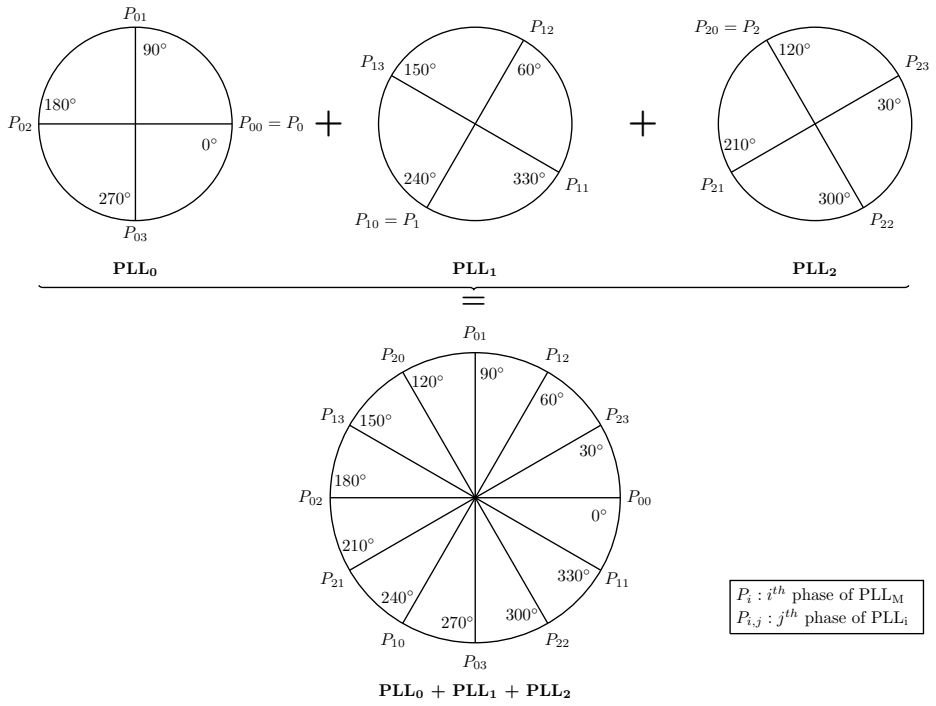


Figure 2.10: 3D hybrid delay matrix without delay elements

Integer Overclocking To improve the resolution of the structure further, the frequency of the secondary stage PLLs can be overclocked with the factor O in respect to the first stage. The new resolution is thereby

$$\text{LSB} = \frac{1}{f\,M\,N\,2\,O}. \tag{2.8}$$

As an example, in Fig. 2.11, the first stage PLL has a period of 600 ps and the period of the secondary stage is 120 ps, thus the overclocking rate O is 5. The phase value shown in the figure is in relation to the output P_{00} of PLL_0. The other parameters are $M = 3$, $N = 4$ and no DDR is applied. In the upper part of Fig. 2.11, the phase distribution is shown.



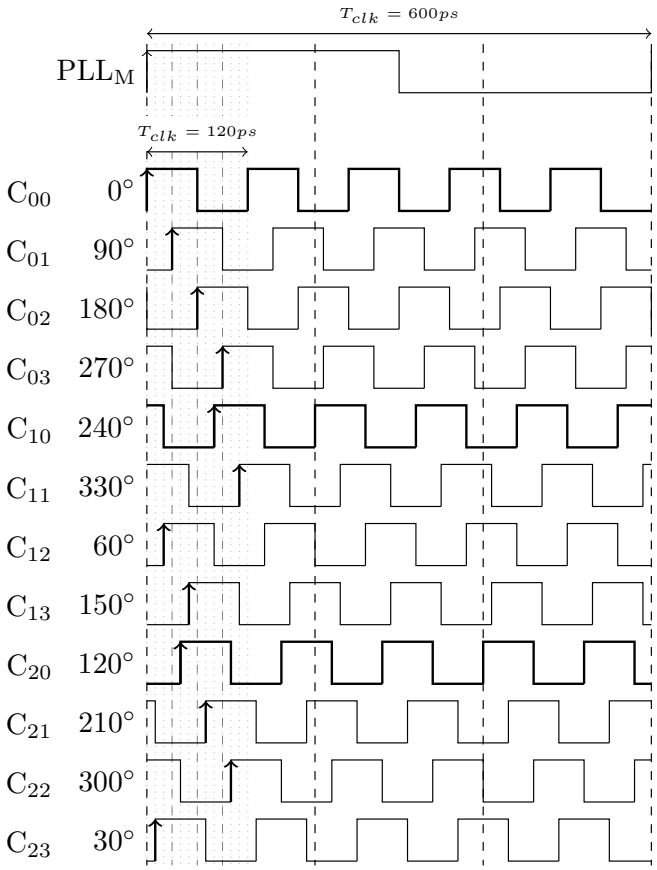


Figure 2.11: Phase and timing diagram 3D hybrid delay matrix with PLLs

Similarly, to Fig. 2.11, in Fig. 2.12 the timing and phase diagrams with applied DDR are shown. The phase of the secondary stage PLLs is now distributed differently and is again shown in the upper part of the image.

Phase Overlapping One can clearly see in Fig. 2.11 that caution with the overclocking rate is advisable, not every rate is suitable. The general aim is to construct a clock network with uniformly distributed phase-shifts. To make sure that no phase overlapping occurs, the parameters must be coprime [**?**]

$$(M, 2\,N\,O) = 1. \tag{2.9}$$

Two integers are coprime, if the only positive integer that divides both of them is one. In the following example the coprime property is violated.

$$(4, 2 \cdot 6 \cdot 5) \Rightarrow (4, 60) \neq 1 \tag{2.10}$$

Fractional Overclocking As an addition to the integer overclocking scheme, fractional overclocking can be applied. The parameter O is rewritten with $\frac{d}{c}$. Important is, that c must be a divisor of $2\,N$. Similar to integer overclocking, the coprime property

$$\left(M, \frac{2\,N\,d}{c}\right) \Rightarrow \left(M, d\,\frac{2\,N}{c}^{\xi}\right) \Rightarrow (M, d\,\xi) = 1 \tag{2.11}$$

must hold.

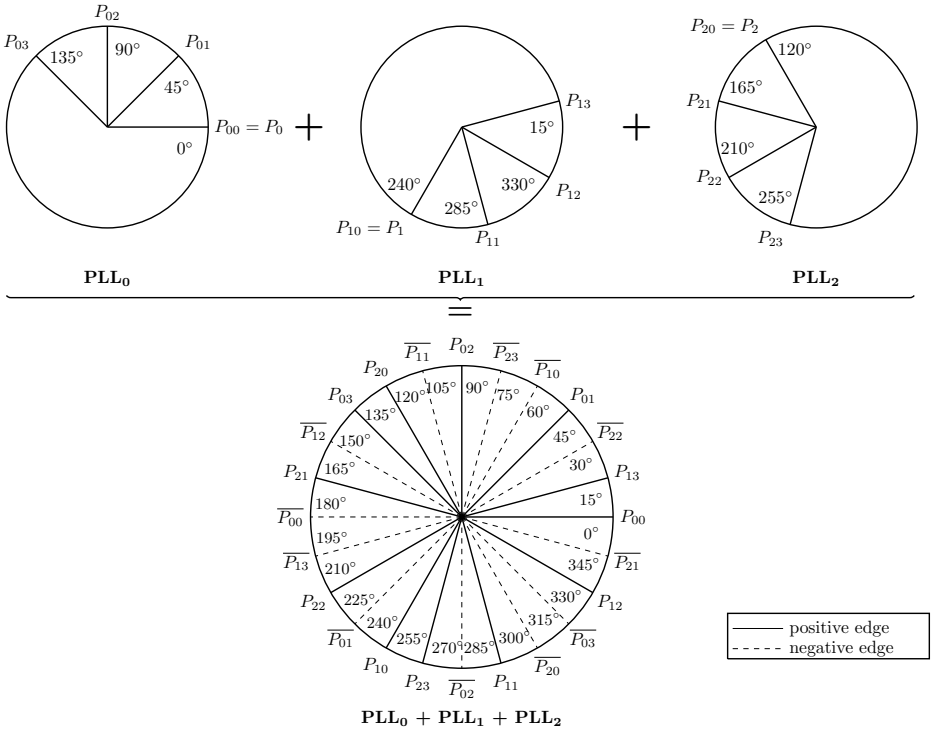
The resolution with fractional overclocking is therefore

$$\mathrm{LSB} = \frac{1}{f\,M\,N\,2\,\frac{d}{c}}. \tag{2.12}$$

2.1.5 Conclusion Literature Study

In conclusion, many possible TDC implementations allow to measure short time intervals. A next step in the completion of this thesis is to select a suitable system to rebuild.

As mentioned before, the utilization of delay lines is a rather difficult task and is heavily dependable on the consistency of the delay lines. Therefore, an implementation without delay lines should be chosen, or at least one where the consistency does not have to be given. The proposed structures 'PLL and Delay Element Delay Matrix' in section 2.1.4.1 and 'PLL Only, Delay Matrix' in section 2.1.4.2 are therefore methods well suited for this thesis to be rebuilt. The general level of detail in [**?**] and its predecessor [**?**] is sufficient to be able to implement the proposed TDC. Even though the papers have a high level of detail, there are neither highly detailed schematics available nor is code included. For this reason, the TDC has to be built from scratch.



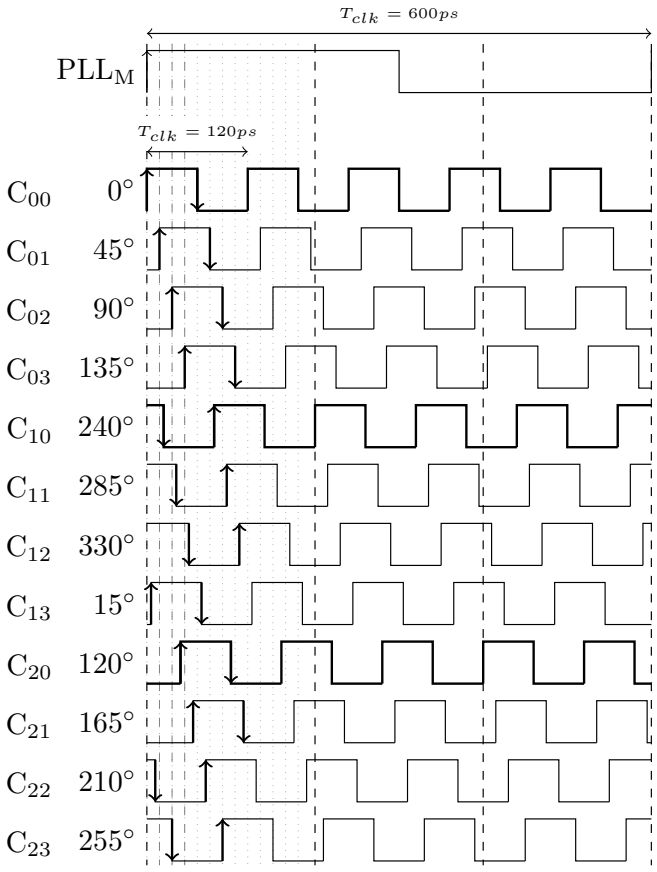


Figure 2.12: Phase and timing diagram 3D hybrid delay matrix with PLLs and DDR

Chapter 3

Methods

As mentioned in section 2.1.5, the practical part of this thesis is to rebuild two of the proposed structures of section 2.1.4. In this chapter, the actual implementation will be described meticulously. The following implementations are developed in the Xilinx Vivado environment on various FPGA boards. The newer Xilinx FPGAs includes Mixed-Mode Clock Managers (MMCMs) and PLLs. A PLL is a subset of an MMCM regarding its functions but in general, they have the same functionality [**?**]. However, MMCMs have more outputs and better phase shift capabilities, which is why they are used instead of PLLs.

First, the simulation system is described in section 3.1. Then, in section 3.2, the transformation of the two chosen principles into a complete measuring system is shown. In section 3.3 and section 3.4, the two different variants and their implementations are described. To understand the difficulties encountered during implementation better, they are discussed in section 3.5. The last section of this chapter, section 3.6, is about the measuring system.

3.1 System Simulation

Before the system implementation, an abstract system simulation has been carried out. Because the abstraction level of the leading paper is not as detailed, it has to be assured that the principals have been understood properly. With a system wide simulation in a high-level programming language, the next implementation step with a hardware description language should be less complicated.

The chosen simulation language is `Python`. All the given parameters from the theoretical part in section 2.1.4 have been included in the simulation. Even though the real life application will work with continuous signals, the signals in the simulation are discrete. A time series is constructed as an array, whereas the difference between two samples is specified as 1 ps. Such an array is displayed in Fig. 3.1.

Only the integer overclocking scheme has been implemented. The problem is that the chosen time resolution is 1 ps. If a simulated clock has a period, which is not an integer multiple of one picosecond, the simulation system would be inaccurate. This means, that fractional overclocking cannot be simulated, because normally, the clock periods they generate are not integer multiples of one picosecond. Nevertheless, even without these extensions the simulation helps to improve understanding.

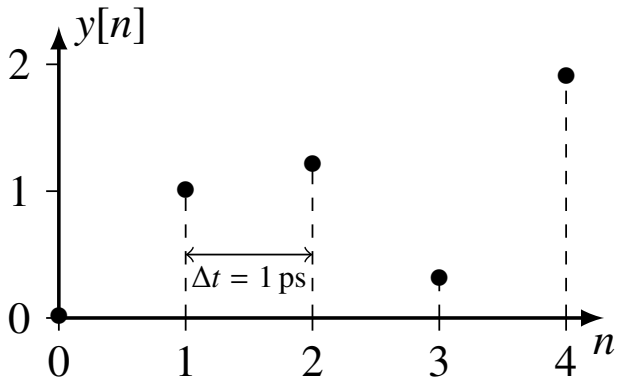


Figure 3.1: Time series

3.2 Measuring Time

After talking about measuring short time intervals in section 2.1.4, this section addresses the whole system for measuring a time pulse. Theoretically, the structure with a counter matrix is capable of measuring long time pulses by itself, only limited by the maximum counter value of the matrix elements. However, the uncertainty and small variations in the clock network would result in inaccurate measurements. The Nutt method [**?**] proposes a scheme to divide the input pulse T_{in} into three separate pulses.

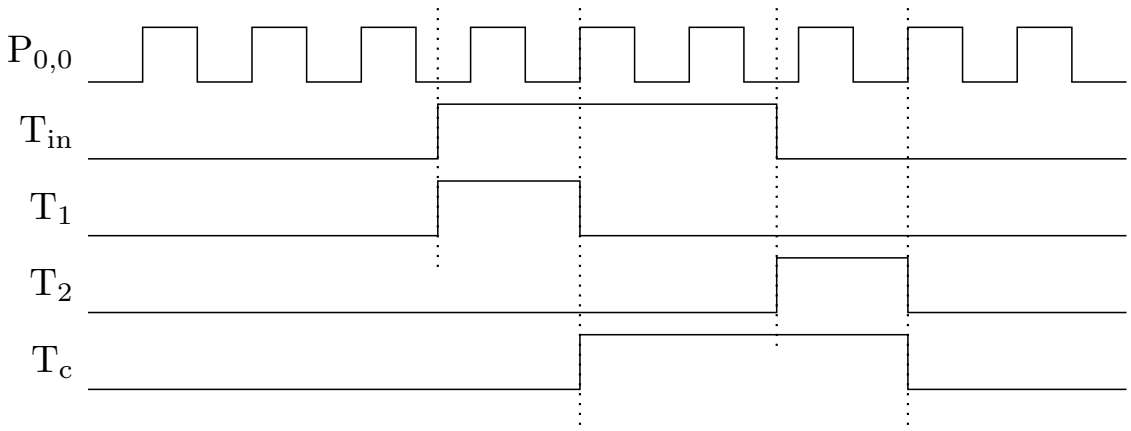


Figure 3.2: Timing diagram pulse measurement

There are two short pulses T_1 and T_2 for the fine measurement and a long pulse T_c for the coarse measurement. This so-called fine measurement is done with one of the two chosen TDC variants, whereas the coarse measurement is done by a simple counter that counts the positive clock edges.

As can be seen in Fig. 3.2, T_1 starts with the input pulse and stops aligned with the clock $P_{0,0}$ after the second rising clock edge. This results in a pulse length of at least one period of $P_{0,0}$

and a maximum of two periods of $P_{0,0}$. T_2 starts when the input pulse ends and stops aligned with the clock $P_{0,0}$ after the second rising clock edge, exactly like T_1 does. T_c on the other hand starts when T_1 ends and ends together with T_2. As a result, T_c starts and ends synchronous with the clock; therefore it can be precisely measured with a simple counter.

These three pulses can be generated with the circuit in Fig. 3.3 [**?**].

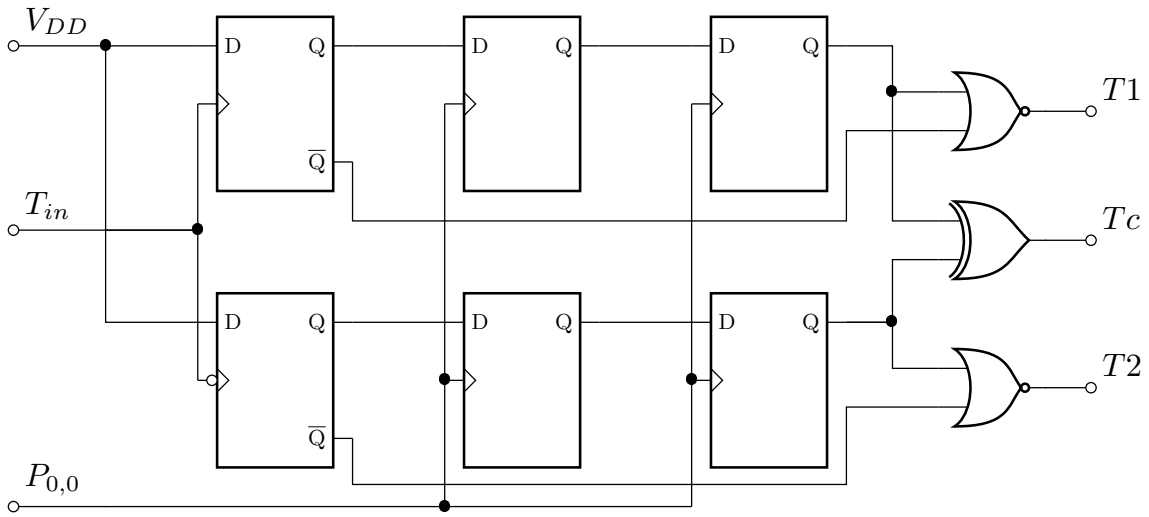


Figure 3.3: Schematics pulse generator

If one wanted to change the time T_1 and T_2 stay high, the amount of FFs stages after the first column of FFs can be adjusted. In this implementation with two additional rows of FFs, the pulses T_1 and T_2 will end with the second following positive clock edge and behave as in Fig. 3.2.

If an input pulse is measured with the described single pulses T_1, T_2 and T_c, the length of the input pulse can then be computed with Equation 3.1.

$$T_{in} = T_1 + T_c - T_2 \tag{3.1}$$

3.3 Variant 1

Fig. 2.10 already gives the theoretical structure of the TDC core, whereas the logical structure of the whole measurement system can be seen in Fig. 3.4. The two MMCM stages generate the clock network; the TDC-Core measures the input pulse and prepares the data for the Universal Asynchronous Receiver Transmitter (UART) transmission.

First, a general note on the structure of the block diagrams. They are generated directly from the written VHDL files within Sigasi Studio. With the very handy *Graphics Configuration Language* the block diagrams can be grouped, filtered and colored. [**?**]

Normally, a block is collapsed and given a suitable name, like `UART` for the UART block in Fig. 3.4. Unfortunately, sometimes important information gets lost through collapsing. Therefore, these blocks are then not collapsed and the name stands as it is given in the VHDL file. The nomenclature for such a non-collapsed block is: `name_of_instance:name_of_entity`.

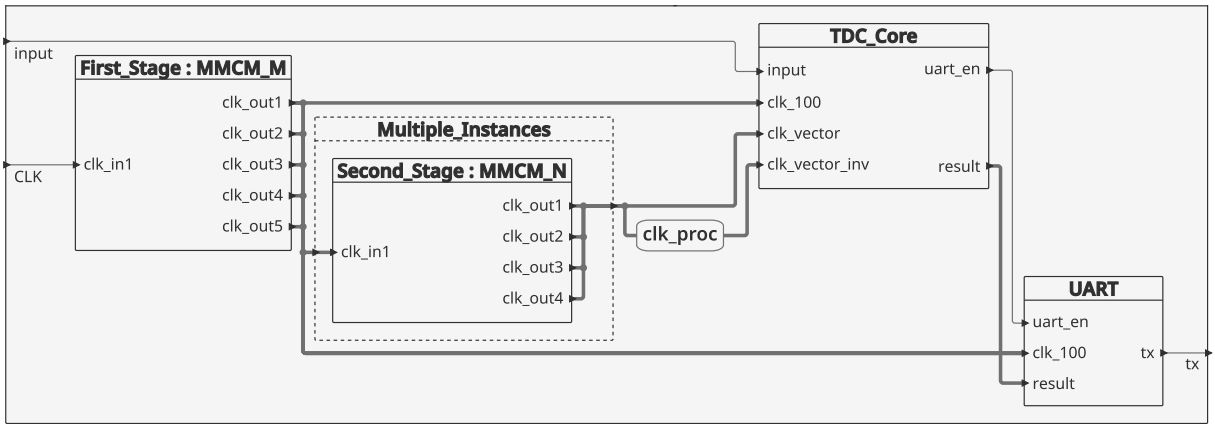


Figure 3.4: Top Schematics

An example is the block `First_Stage:MMCM_M` in Fig. 3.4. Another special block in Fig. 3.4 is the block `Second_Stage:MMCM_N` with the dashed box. This one represents a useful VHDL statement called `for generate`.

```
1     Multiple_Instances : for i in 0 to M - 1 generate
2         Second_Stage : component MMCM_N
3             port map(
4                 clk_in1  => clk_M(i),
5                 reset    => rst,
6                 clk_out1 => clk_vector(i * N),
7                 clk_out2 => clk_vector(i * N + 1),
8                 clk_out3 => clk_vector(i * N + 2),
9                 clk_out4 => clk_vector(i * N + 3));
10    end generate Multiple_Instances;
```

A short explanation about these `for generate` statements. In the given example, every output of the first stage MMCM has a second stage MMCM. The statement places M instances of the secondary stage and connects its outputs to their corresponding index in the array `clk_vector`. This helps to reduce the amount of code tremendously and it also supports a generic structure of the whole project.

3.3.1 MMCM Configuration

The configuration described in this section refers to the 'NEXYS 4 DDR' FPGA, more details in Table 4.1. Other limitations than the ones mentioned may apply if other FPGAs are used.

There are certain restrictions that must be respected to find a suitable configuration. First, the described conditions in section 2.1.4.2 have to be met. Furthermore, there are also some limitations regarding the configurability of the MMCMs included in FPGAs. Those that we need to consider are

$$F_{VCO} = F_{CLKIN} \cdot \frac{M}{D} \tag{3.2}$$

$$F_{OUT} = F_{CLKIN} \cdot \frac{M}{D \cdot O} \tag{3.3}$$

and

$$\text{SPS(frac)} = \frac{360}{8 \cdot O} = \frac{45}{O} \tag{3.4}$$

which all can be found in the Xilinx documentation, more specifically in the '7 Series FPGAs Clocking Resources User Guide' [**?**]. Equation 3.2 calculates the resulting Voltage Controlled

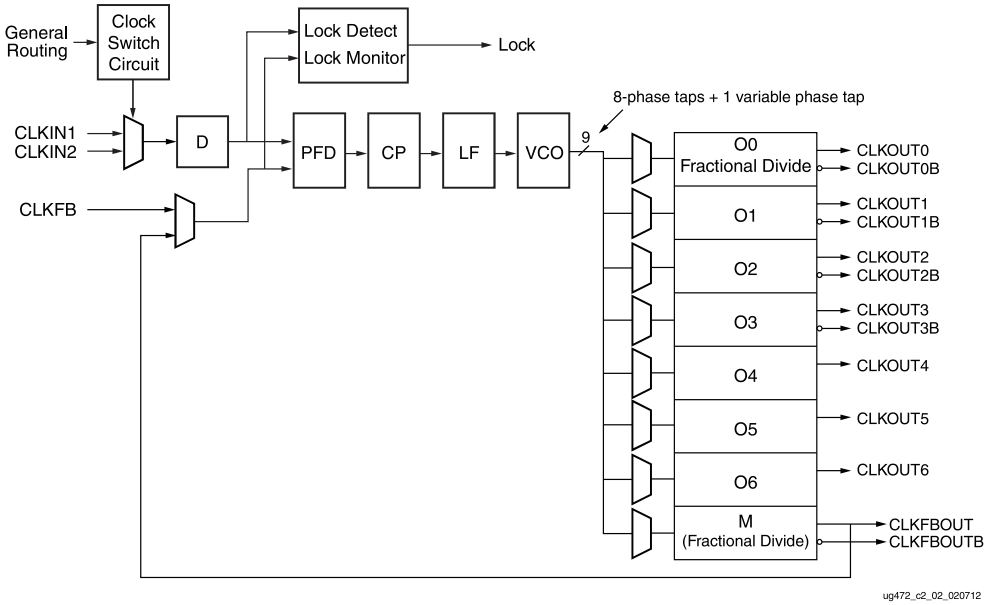


Figure 3.5: Detailed MMCM Block Diagram [**?**]

Oscillator (VCO) frequency, which needs to be in the range of 600 to 1200MHz, according to the datasheet of the used chip [**?**]. Unfortunately, the VCO frequency cannot always be directly used at the output, because the output frequency of the MMCM needs to lie within the range of 4.69 to 800MHz. Equation 3.3 computes the frequency of an output clock. Equation 3.4 describes the step size of the phase shift capability for one output. The block diagram of an MMCM device inside a '7 series' FPGA can be seen in Fig. 3.5.

An optimal solution has to be found to obtain a high resolution in the resulting measuring system. Such a solution needs as many differently shifted outputs as possible in combination with as high as possible output frequencies. The one we found has a first stage MMCM with five outputs and, therefore, five second stage MMCMs with 4 outputs each. It uses the onboard oscillator with a frequency of 100MHz for the input clock of the first MMCM stage and has following parameters (the parameter names from here on correspond to the names in Fig. 3.5):

- $M = 10$
- $D = 1$
- $O0 = O1 = O2 = O3 = O4 = 10$

which results in an output frequency of

$$F_{OUT} = 100\text{MHz} \cdot \frac{10}{1 \cdot 10} = 100\text{MHz} \tag{3.5}$$

for all five outputs. The VCO frequency lies within its boundaries:

$$600\text{MHz} \leq F_{VCO} = 100\text{MHz} \cdot \frac{10}{1} = 1000\text{MHz} \leq 1200\text{MHz} \tag{3.6}$$

The five output clocks need to be uniformly shifted clocks, therefore their phase shifts need to be 0°, 72°, 144°, 216° and 288° in relation to the input clock. In order to be able to achieve

these phase shifts, they need to be integer multiples of

$$\text{SPS(frac)} = \frac{45}{10} = 4.5, \tag{3.7}$$

which they all are.

Each of these five output clocks with different phases is now used as an input clock for one MMCM in the second stage. This totals to $5 \cdot 4 = 20$ different output clocks in the resulting clocking network that can be used for the `counter matrix`.

The parameters for all four second stage MMCMs are identical and are:

- $M = 9.25$
- $D = 1$
- $O0 = O1 = O2 = O3 = 2$

so the output frequency of all 20 output clocks is

$$F_{OUT} = 100\text{MHz} \cdot \frac{9.25}{1 \cdot 2} = 462.5\text{MHz}. \tag{3.8}$$

With the chosen parameters, the VCO frequency of all second stage MMCMs is within its limits:

$$600\text{MHz} \leq F_{VCO} = 100\text{MHz} \cdot \frac{9.25}{1} = 925\text{MHz} \leq 1200\text{MHz} \tag{3.9}$$

The output clocks of each second stage MMCM need to be shifted to 0°, 45°, 90° and 135°. These phases are multiples of

$$\text{SPS(frac)} = \frac{45}{2} = 22.5 \tag{3.10}$$

which means, they can be realized by the MMCMs.

	clkout0	**clkout1**	**clkout2**	**clkout3**	**clkout4**
MMCM0	0°	72°	144°	216°	288°
MMCM1	0° + 0°	0° + 45°	0° + 90°	0° + 135°	
MMCM2	72° + 0°	72° + 45°	72° + 90°	72° + 135°	
MMCM3	144° + 0°	144° + 45°	144° + 90°	144° + 135°	
MMCM4	216° + 0°	216° + 45°	216° + 90°	216° + 135°	
MMCM5	288° + 0°	288° + 45°	288° + 90°	288° + 135°	

Table 3.1: Phases of output clocks

The phases of the 20 output clocks are, with the use of the DDR concept that is described in section 2.1.4.2, evenly distributed over 360°, because all of the four second stage MMCMs

shift their output clocks relative to their input clock and, as mentioned above, the input clock is different for each of them. They are listed in Table 3.1.

In order to use the DDR concept, all of these clocks need to be inverted to be able to use the negative edge too and can be seen in Fig. 3.4, named `clk_proc`. To better understand what we gain from the use of DDR, the phase distribution is graphically shown in Fig. 3.6.

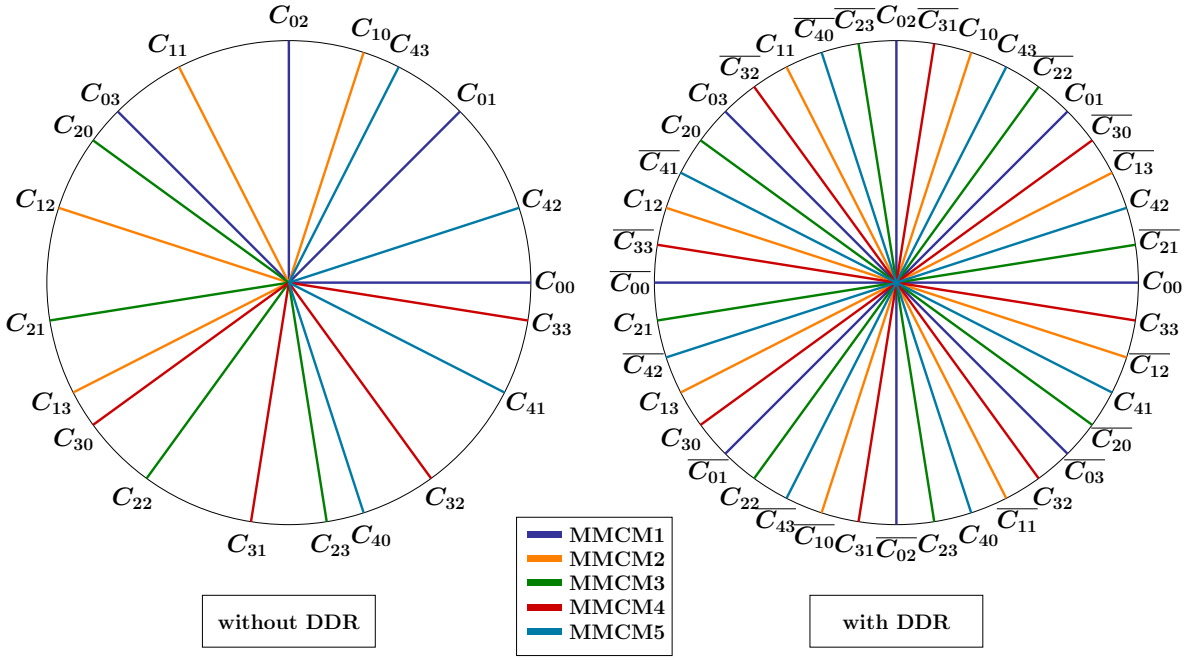


Figure 3.6: Phases of output clocks graphically

To sum up, we now have 20 (respectively 40, after they got inverted) output clocks with a frequency of 462.5MHz and, because we use DDR, a regular phase shift difference of 9° between each of them. The theoretical resolution of this implementation results to

$$\text{LSB} = \frac{1}{100\text{MHz} \cdot 5 \cdot 4 \cdot 2 \cdot \frac{9.25}{2}} = 54.054\text{ps} \tag{3.11}$$

according to Equation 2.12.

3.3.2 TDC Core

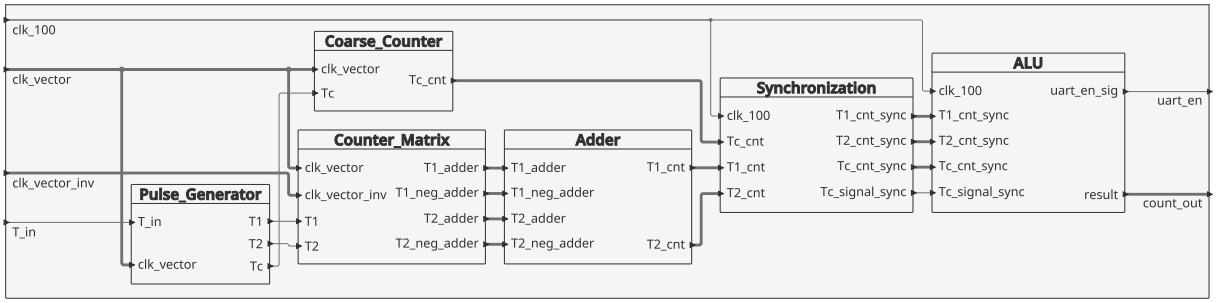


Figure 3.7: `TDC-Core` Schematics

The TDC-Core carries out the actual measurement. It is split up into various blocks which are all shown in Fig. 3.7.

3.3.2.1 Pulse Generator

The `pulse generator` is implemented the same way as it is shown in Fig. 3.3. The only difference is, that in the here described Variant 1, it helped to instantiate Global Clock Buffers (BUFGs) at the output ports of the block, but only in the single-core configuration (more to the multi-core configuration in section 3.3.4). This adjustment was found through trial and error and allows to get better measuring results when the implementation was tested.

3.3.2.2 Coarse Counter

As mentioned in section 3.2, the T_c signal is used to count the clock periods for which the input signal is 'high'. This counter is very simple but compared to a standard counter there is a small adjustment in order to be able to count higher frequencies. The Least Significant Bit (LSB) is toggled with every positive clock edge to get a frequency divider. As a result, the main part of the counter only runs on half the input frequency and its reliability when used with higher frequencies is improved.

3.3.2.3 Counter Matrix

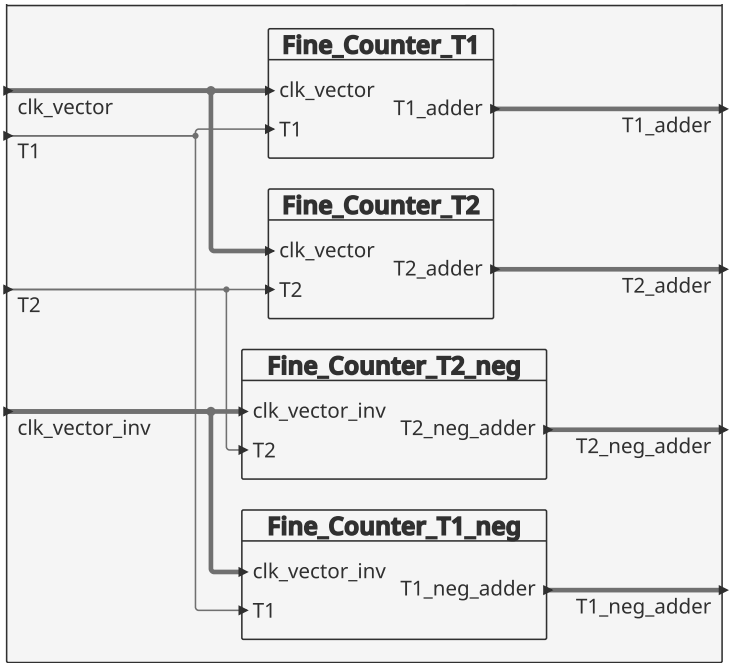


Figure 3.8: `Counter Matrix` Schematics

The `counter matrix` in Fig. 3.8 includes 4 identical components, the so called `fine counters`. They all have the exact same matrix structure; the difference is that they get different input signals. The first one, `Fine_Counter_T1`, measures the pulse T1. Because we have a DDR configuration (described in section 2.1.4.2), we need to measure the same pulse T1 again, but with inverted clocks. This is done in the component `Fine_Counter_T1_neg`. The same

holds for T2. We have two blocks that both get the signal T2 as an input, one with the 'normal' clocks and one with the inverted ones.

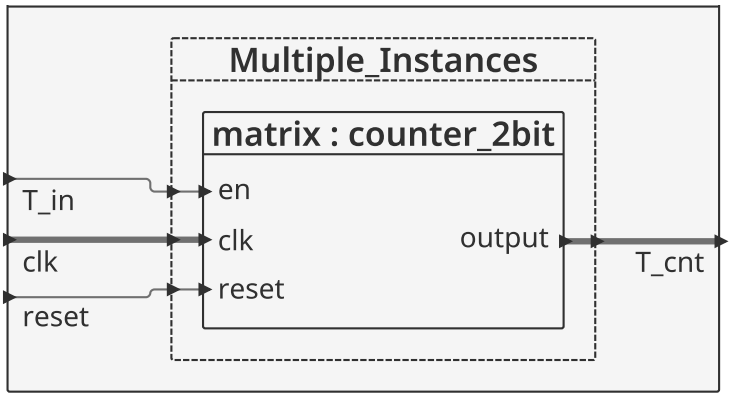


Figure 3.9: `Fine Counter` Schematics

Fine Counter The number of counters in each matrix is identical to the number of clocks in the clocking network. Since the clocking network consists of 20 clocks, each matrix consequently consists of 20 counters. The schematics of the `fine counter` is shown in Fig. 3.9. As already mentioned in section 3.2, both T1 and T2 are high for one whole period of $P_{0,0}$ plus an additional sub-period time that depends on the occurrence of T_{in}. Every clock has only one positive edge per $P_{0,0}$ period and the pulse to measure is shorter than 2 periods of $P_{0,0}$, so it can be derived that the counter has a maximum value of 2, for which a `2-bit counter` is sufficient.

After the initial measurements of the first working design, it was discovered that some of the `fine counters` did not always reach the correct counter value. Since the accuracy of the entire `counter matrix` depends on times and time differences, a `bit synchronization` was added to make the readout of the individual counters more reliable. The schematics of the improved `2-bit counter` is shown in Fig. 3.10. The `bit synchronization` consists of two FFs in a row. This prevents a metastable state for the counting FFs. In addition, a FF was placed at the output to reduce counter readout problems.

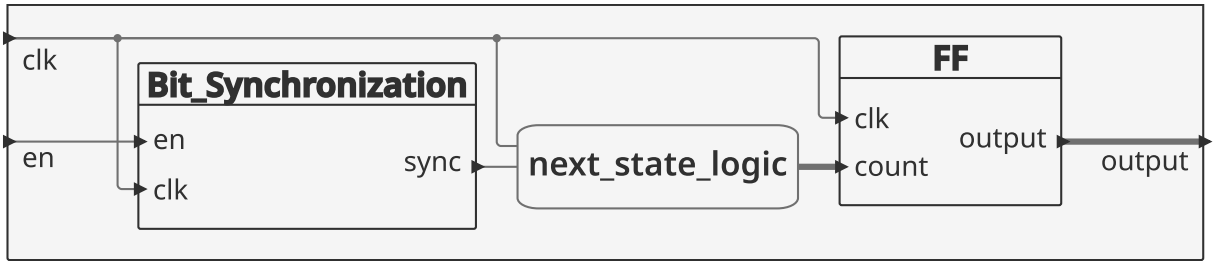


Figure 3.10: `2-bit Counter` Schematics

3.3.2.4 Adder

The `Adder` receives the current counter values of all four `counter matrices`. All of them are represented by a separate vector, of which the values of the `counter matrices` of T1 and T1_neg can be added. The same holds for T2 and T2_neg. This block is not clocked; it simply

sums up the input vectors at any time and outputs a vector for the current T1 and T2 counter values.

3.3.2.5 Synchronization

The `Synchronization` block gets the asynchronous input signals T1, T2 and Tc from the `Adder` block and synchronizes them according to the main clock. This helps to evaluate them properly in the following block. Its schematics can be seen in Fig. 3.11. Additionally, the Tc_signal, which is the Tc pulse output from the `pulse generator`, is synchronized. In the next block, it is used to detect the end of a measurement and create a reset signal for all blocks that must be reset after a measurement.

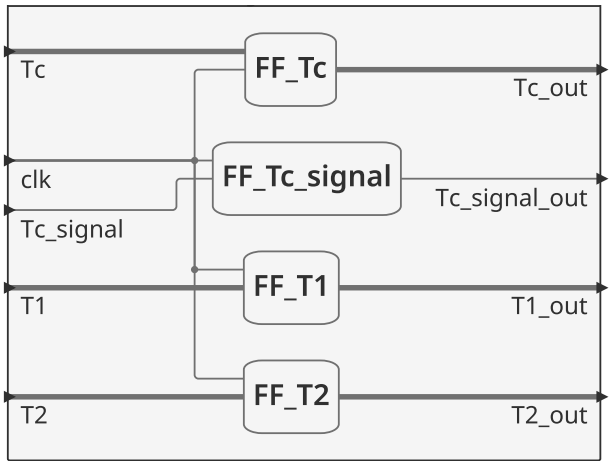


Figure 3.11: `Sync Block` Schematics

3.3.2.6 Alu

The `Alu` block is a simple state machine. By default, it is in the `st_wait_for_signal` state. It enters the next state when T1 is 'high' and additionally either T2 or Tc is 'high'.

The `st_collecting` state is used to wait for the end of the measurement and checks the Tc input, which is 'high' as long as the measurement is ongoing. It changes to 'low' simultaneously with T2 and is used as an indicator that the measurement is finished and that the state can be changed to `st_collecting_final`.

Because the processing (which includes the reading of the `counter matrices`, the adding up and the synchronization) takes some time, this state is required to make sure the signals are completely processed before they are written out. It does not do anything in particular; it only delays the next state for one period.

Now that we are in state `st_send` and all signals are ready, they can be merged into one vector, see section 3.3.3, and be output. To signal the `UART` block that the result is ready to be sent out, the `uart_enable` output is set 'high', further details to the `UART` in section 3.3.2.7.

The next state `st_reset` is reached unconditionally because it always needs to come after the `st_send` state. It does what its name suggests, it sets the reset output 'high' in order to reset the elements that are involved in processing the measurement (namely the `fine counters` in the `counter matrix` and the `pulse generator`) and need to be made ready for the next measurement, which means to set them to their defaults.

Again, the same as for the collecting state applies. To make sure the reset signal reaches every element and they get resetted properly, an extra state is added, `st_reset_again`. This extra state keeps the reset output 'high' for one further period, before the state machine changes back to the starting point in `st_wait_for_signal`.

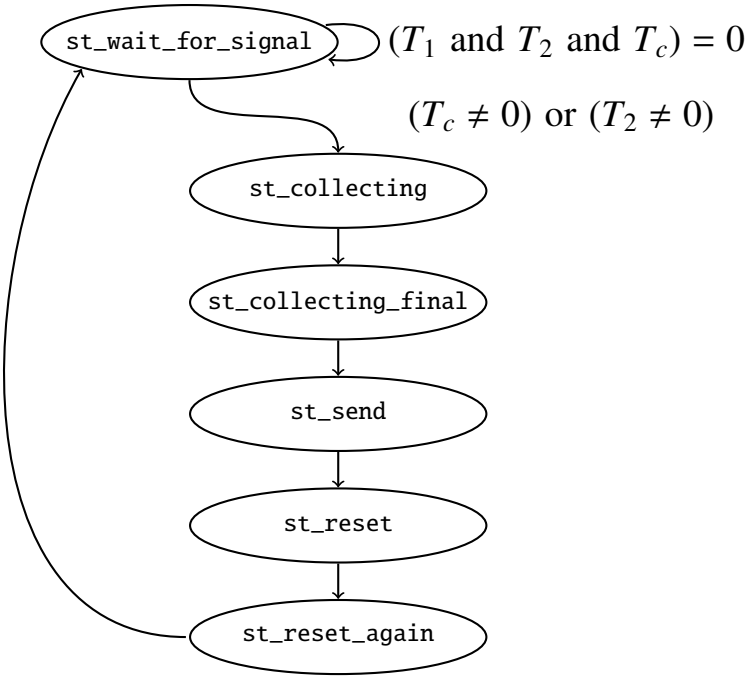


Figure 3.12: `Alu` State Diagram

3.3.2.7 UART

The UART could be taken over from the previous project [**?**]. It is a standard UART implementation based on the 'Digital Microelectronics' script [**?**]. Its receiving parts were removed because they are not used in this work. There were some bugs in the VHDL code that became apparent because certain generic parameters did not work as expected. These bugs were fixed and the parameters were adjusted according to the requirements of this system.

3.3.3 Transmission Protocol

The system transmits the times T_1, T_2 and T_c in LSBs. This includes, that the actual time has to be computed in software with the corresponding resolution. The equation Equation 3.1 can be rewritten as,

$$T_{in} = T_1 \cdot \text{LSB} + T_c \cdot t_{coarse} - T_2 \cdot \text{LSB}. \tag{3.12}$$

The LSB is defined in Equation 2.12 and t_{coarse} can be written as

$$t_{coarse} = \frac{1}{f\frac{d}{c}} = \frac{c}{f\,d}. \tag{3.13}$$

The dataframes T_1, T_2 and T_c are each 16 bits wide. With this chosen protocol and the datawidth of T_c, the theoretical measurement range is given by Equation 3.14.

$$\text{measurement range} = 2^{16}\, t_{coarse} = 2^{16}\, \frac{c}{f\, d} \tag{3.14}$$

With the described MMCM configuration in section 3.3.1 the system clock is overclocked by a factor of $\frac{37}{8}$ and the

$$\text{measurement range} = 2^{16}\, \frac{c}{f\, d} = 2^{16}\, \frac{8}{100\,\text{MHz} \cdot 37} = 141.7\,\mu\text{s} \tag{3.15}$$

can be computed. Measurements have shown that the theoretical and practical range are coherent.

3.3.4 Multi-Core Configuration

A common approach to further enhance the performance of a TDC is to parallelize the system. In such a generic structure as in this paper, a multi-core approach is easy to realize. The structure of the top-file as in Fig. 3.4 stays the same. Only a small part of the `TDC-Core` has to be parallelized, the adjusted structure is displayed in Fig. 3.13. In comparison to the single-core `TDC-Core` schematic in Fig. 3.7, the changes are rather minimal. The main difference is that the `Adder` block is now moved inside the `Counter_Matrix` (Fig. 3.14) block. The `Counter_Matrix` block is finally the block which is parallelized, which is indicated with the dashed line (`for generate`).

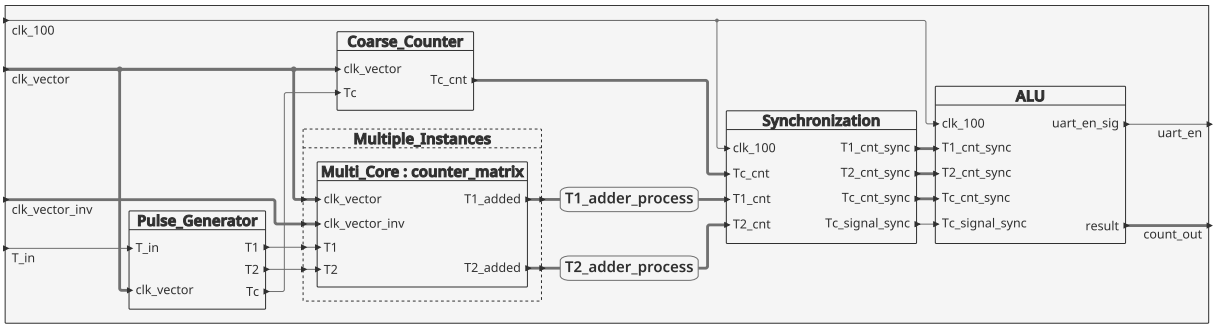


Figure 3.13: Multi-Core TDC-Core Schematics

With this multi core approach, the time pulses T_1 and T_2 are hereby measured multiple times.

The `T1_adder_process` and `T2_adder_process` blocks, are summing up the results from the four individual `Counter_Matrix` blocks. A stochastic approach is used for the gain in resolution. The summed values are divided through the number of parallel cores K. The `coarse counter` is not parallelized because it would be the same for all `fine counters` anyway.

The equation Equation 3.12 for the measured time can be rewritten as

$$T_{in} = \frac{T_1 \cdot \text{LSB}}{K} + T_c \cdot t_{coarse} - \frac{T_2 \cdot \text{LSB}}{K} \tag{3.16}$$

for the multi core approach.

For the implementation, the attribute `DONT_TOUCH` for the `Counter_Matrix` blocks must be applied to prevent logic optimization. [**?**]

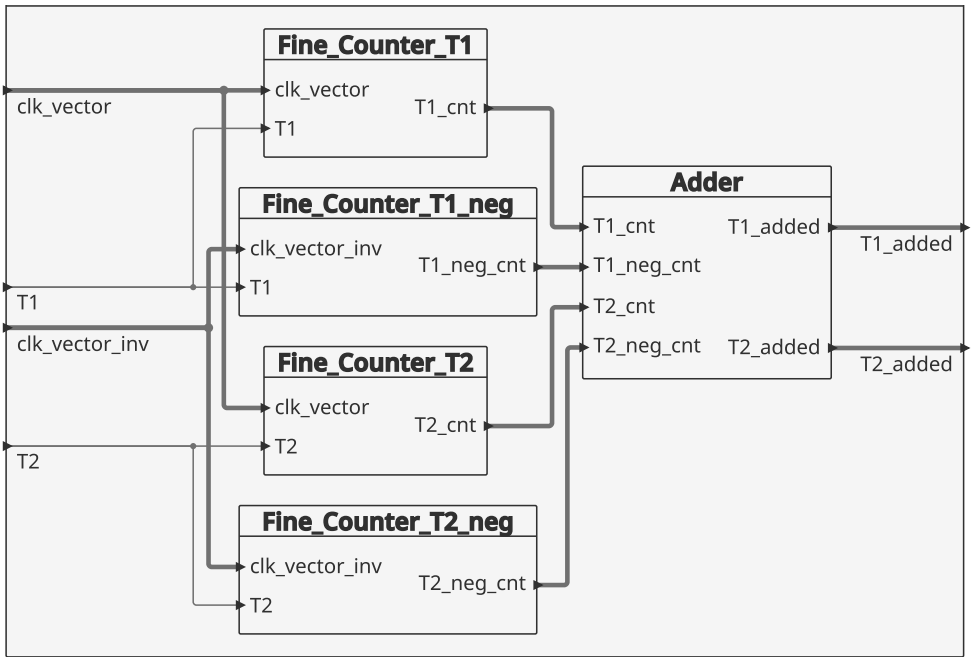


Figure 3.14: Multi-Core Counter Matrix Schematics

3.4 Variant 2

The second variant that is implemented is in large parts the same as the first variant. Its theoretical functionality is described in section 2.1.4 and its structure can be seen in Fig. 2.8. The major advantage of this method over the first one is the reduced amount of required MMCMs to achieve a certain resolution. This allows extending the application purpose of the TDC to FPGAs with a limited number of MMCMs.

An overview of the structure can be seen in Fig. 3.15. The only difference in this overview

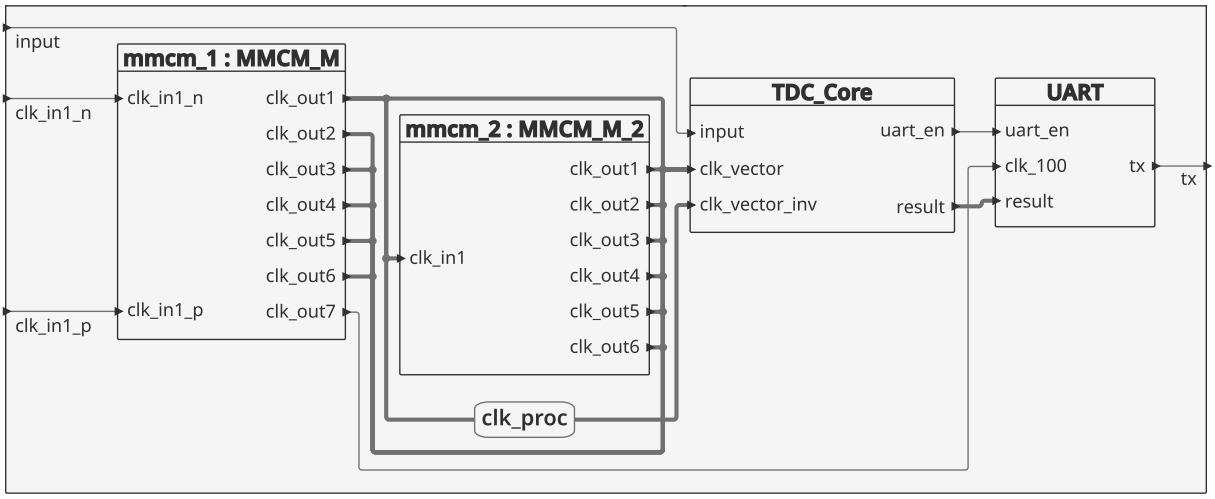


Figure 3.15: Top Schematics of Variant 2

compared to the previously described implementation is the configuration of the MMCMs.

The first variant (in section 3.3) can either be configured as a single-core or multi-core system. However, this variant can already be seen as a multi-core realization in its basic form.

Each column in Fig. 2.8 is a `counter matrix` (even though it is shown as a column, it can still be defined as a matrix). All other rows are then put in parallel to the first one with the same, but delayed input (enable) signal. Therefore, there is only one setup for this variant for which it is a matter of definition whether it should be called a single-core or multi-core structure. One could argue that it would only be a 'true' multi-core configuration, if the `delay elements` between the `counter matrices` were designed to generate delay times exactly so that all matrices got a differently shifted signal that lies between the phase shifts of the clocking network in order to reduce the phase shift step size and enhance the measuring resolution. However, achieving such exact delay times on a FPGA is not trivial. Some attempts have been made, but were not successful. They are found in section 3.5. Therefore, from here on, the delay time they add between each matrix is considered random.

3.4.1 MMCM Configuration

Unlike before, the configuration is now done on the 'Zynq Ultrascale+' FPGA, see Table 4.1 for further details. For this variant, only two MMCMs are used, both of which generate six outputs that are shifted relative to one another to generate the clocking network used for the `counter matrix`. The additional output (clk_out7) from the first MMCM is only used as a clock for the UART and not for the clocking network, so it can be left out in the discussion of the MMCM configuration. In principle, it is the same proceeding as in the first variant, only with other parameters. If something in this section is unclear, consider consulting the more thorough explanation of the procedure of finding the optimal parameters for the first variant in section 3.3.1.

Some limitations of the used FPGA are the same as before (see Equation 3.2, Equation 3.3 and Equation 3.4). However, the VCO frequency limitations have changed to

$$800\text{MHz} \leq F_{VCO} \leq 1600\text{MHz} \tag{3.17}$$

and are taken from the 'UltraScale Architecture Clocking Resources User Guide' [**?**]. The frequency boundaries at the MMCM outputs are

$$6.25\text{MHz} \leq F_{OUT} \leq 775\text{MHz}. \tag{3.18}$$

The frequency of the onboard oscillator is (by default) 300MHz and has differential outputs. The parameters of the first MMCM are:

- $M = 5$
- $D = 1$
- $O0 = O1 = O2 = O3 = O4 = O5 = 3$

and result in an output frequency of 500MHz. They are shifted in 60° steps from 0° to 300°.

The second MMCM has the first output of the first MMCM as an input and has following parameters:

- $M = 3$
- $D = 1$
- $O0 = O1 = O2 = O3 = O4 = O5 = 3$

to again generate an output frequency of 500MHz with phase shift steps of 60°. Now, they are distributed from 30° to 330°. The DDR concept is used again, so in total there is now a clocking network consisting of $2 \cdot 6 = 12 \xrightarrow{DDR} 24$ differently shifted clocks with a 15° phase shift between each of them.

By trial and error, it could be determined that twelve consecutive `counter matrices` are a good choice. The number of `delay elements` obviously depends on the number of `counter matrices` and is always one less, because the first `counter matrix` gets the input signal directly without a `delay element` at its input.

The changes from the first variant also have an impact on the resolution. For this implementation it results to

$$\text{LSB} = \underbrace{\frac{1}{300\text{MHz} \cdot 12 \cdot 12 \cdot 2 \cdot \frac{5}{3}}}_{MMCM1} = \underbrace{\frac{1}{500\text{MHz} \cdot 12 \cdot 12 \cdot 2 \cdot \frac{3}{3}}}_{MMCM2} = 6.944\text{ps}. \tag{3.19}$$

Here, however, the controversy regarding single-core and multi-core configuration that was mentioned at the beginning of section 3.4 must be kept in mind. This resolution could only be reached if the `delay elements` were not random. Because the delay time is considered random, the computation of the time pulse is similar to section 3.3.4,

$$T_{in} = \frac{T_1 \cdot \text{LSB}}{Q} + T_c \cdot t_{coarse} - \frac{T_2 \cdot \text{LSB}}{Q}. \tag{3.20}$$

Here, the parameter Q stands for the number of columns (how many delay elements), twelve in this example.

3.4.2 TDC Core

The TDC core has exactly the same structure as in the first variant and is shown in Fig. 3.7. The only difference is within the fine counters. There are still four different `fine counters` `Fine_Counter_T1`, `Fine_Counter_T1_neg`, `Fine_Counter_T2` and `Fine_Counter_T2_neg`, but now, their inside has received an addition. Each of them now includes LUTs that lie between each `counter matrix` in order to get the delay between each column as in Fig. 2.8. The new schematic is shown in Fig. 3.16.

When using the LUTs and the parallel `counter matrices`, it must be ensured that the `dont_touch` attribute is used for both, the `counter matrices` and the LUTs. Otherwise they are discarded by the logic optimizer.

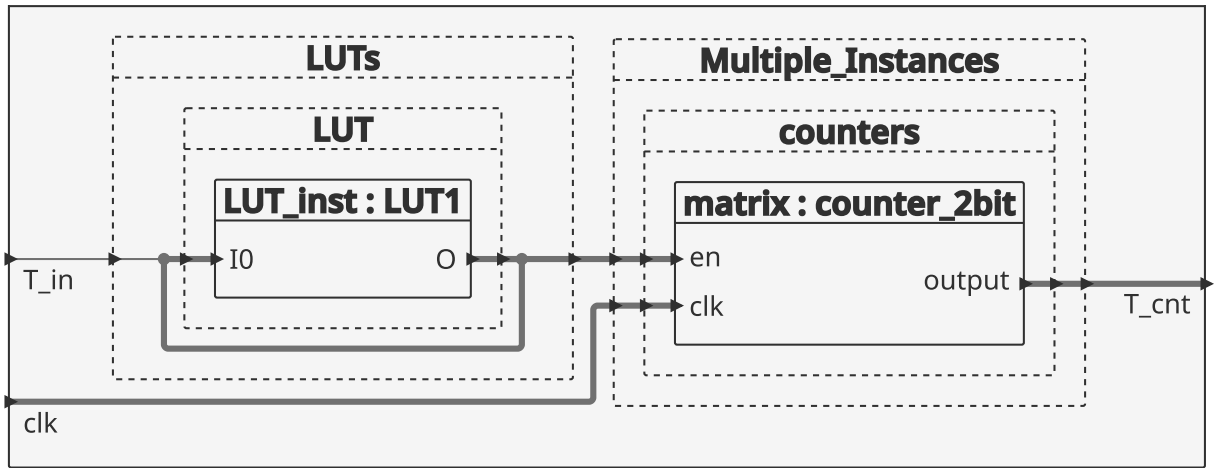


Figure 3.16: Schematics of fine counter in variant 2

3.5 Vivado Workflow

The VHDL structures of the TDCs have been described in the previous sections. The next step of an FPGA project is to implement it on a board. Due to the usage of Xilinx FPGAs, the *Vivado Design Suite* is used for the synthesis of the VHDL models.

3.5.1 RTL Simulation

The Register Transfer Level (RTL) simulation allows testing the written VHDL code. The simulation takes place on the logic level, no timing is included. Because the general structure of the VHDL design is heavily hierarchical, the simulation is a pretty straightforward process. Each component in a hierarchy levels can be tested separately and if all blocks in a hierarchy are checked, one can step up a level and test the blocks together.

In this project, almost all blocks are self-made. These blocks were first tested separately to make sure, that they behave as expected. Then, the whole system was put together piece by piece and tested accordingly until finally, the whole structure was included in one single top-file and the logical functionality of the system could be approved.

3.5.2 Synthesis and Implementation

3.5.2.1 Floorplanning

Floorplanning is an important step if timing requirements are not met. In general, Vivado handles the floorplanning by itself and does it quite well. It has been tried to improve the timing of the system by manually placing the different blocks of the VHDL structure. Unfortunately, the expectation of improvement has never been fulfilled.

Furthermore, floorpanning can be used to randomly place the different TDC-Cores in the multi core approach. This was taken into consideration, but the implementation of such a system failed. During the Vivado workflow, a problem occurred. Thanks to the consultation of the Xilinx forum, the error can be traced back to an issue in the clock network. Unfortunately, this error is a current bug in Vivado [**?**]. Due to the generally good results of the TDCs without manual floorplanning, this approach was not pursued any further.

3.5.2.2 Timing Requirements

As was already mentioned in various sections of this thesis, the underlying principle of the measuring system is to count, for how many clock edges an input signal is 'high'. Inherently, there are timing violations brought to the system solely by its structure. Violations for the Worst Negative Slack (WNS) and Total Negative Slack (TNS) are common. During the development process of such a system with intentional violations, it is important to distinguish between violations that are inherently present in the system and violations brought to the system by a poor VHDL design.

The `set_false_path` constraint helps to keep a clear view over all timing violations. It indicates that a path is not to be analyzed for a timing report [**?**]. If one is sure that a path, which is violating timing requirements, is not a problem, it can be handled with this constraint.

To get the best possible results from a multi-core setup, it is helping when the time pulses after the `Pulse_Generator` block have a constant path length to all the different counter matrices. The constraint `set_bus_skew` is an option to constrain these times [**?**]. The bus skew can be interpreted as the time difference in arrival time of a signal with the same origin at its destination throughout the chip. A value for the maximum bus skew is set with the constraint. Unfortunately, Vivado is not able to meet the given requirements and it always generates the same bus skew. Similar to the floorplanning approach, due to the generally good results of the TDCs, this approach was also not pursued any further.

3.5.2.3 High Frequencies

In one of the consulted papers [**?**], much higher frequencies than the MMCMs allow were used. More precisely, they were able to route the VCO clock directly out of the MMCMs and operate their outputs beyond their limits. Because it was not clear how they achieved this, a question was asked to the authors by mail. Their answer is shown in Appendix C. The amount of work and skills needed to get this to work would have exceeded this thesis and the focus was put back on achieving the most with the existing limitations.

3.5.2.4 Exact Timing

The second variant, section 3.4, could get much lower resolutions if the delay time between each matrix could be adjusted according to the clocks in the clocking network. As mentioned in the implementation described, different approaches were tested and are described in the following paragraphs.

An obvious option is to use the exact same delay line as was used for the TDL in the previous project [**?**]. These carry elements have the shortest possible delay time in an FPGA. They can be instantiated individually and the correct amount of delay elements can be put in a row to get the desired delay time. Although they seem to be suitable for this specific application, the uncertainty and inconsistency in these elements makes them unusable.

Buffers on the other hand should have more constant delay times. To test this out, different buffers were tested. Because of the presence of different types of buffers on the used board, mainly BUFGs and Horizontal Clock Buffers (BUFHs) were used. The problem with these buffers is that they are placed at a certain position on the chip. The routing from a logic element, e.g. a FF, to the buffer already takes a certain amount of time and then needs to go to next elements. It ends up needing a lot of time and in addition, the routing would then need to be controlled by constraints so that it stays the same throughout different implementations.

In the described implementation, LUTs were used. They have an advantage over buffers, which is that they are closer, pretty much as close as it can get, to other logic elements. Besides that, the routing process should be controlled with constraints. It was tested out with the constraint commands `set_max_delay` and `set_min_delay` but it did not work out. The Vivado router only noted that it was not possible to meet the timing requirements.

All three options could not be usefully applied to get exact delay times. This does not mean that it is impossible, but rather that it involves a very substantial effort regarding floorplanning, similar to the ones in section 3.5.2.4. The hurdles to overcome were considered too big, especially considering the authors' limited experience. The time could be better invested in other aspects of the work.

3.6 Measuring System

In order to test the implemented TDCs, a user-friendly measuring system has to be designed. As in the program for the system simulation, `Python` is the scripting language of choice. In Fig. 3.17, an overview of the measuring system is given.

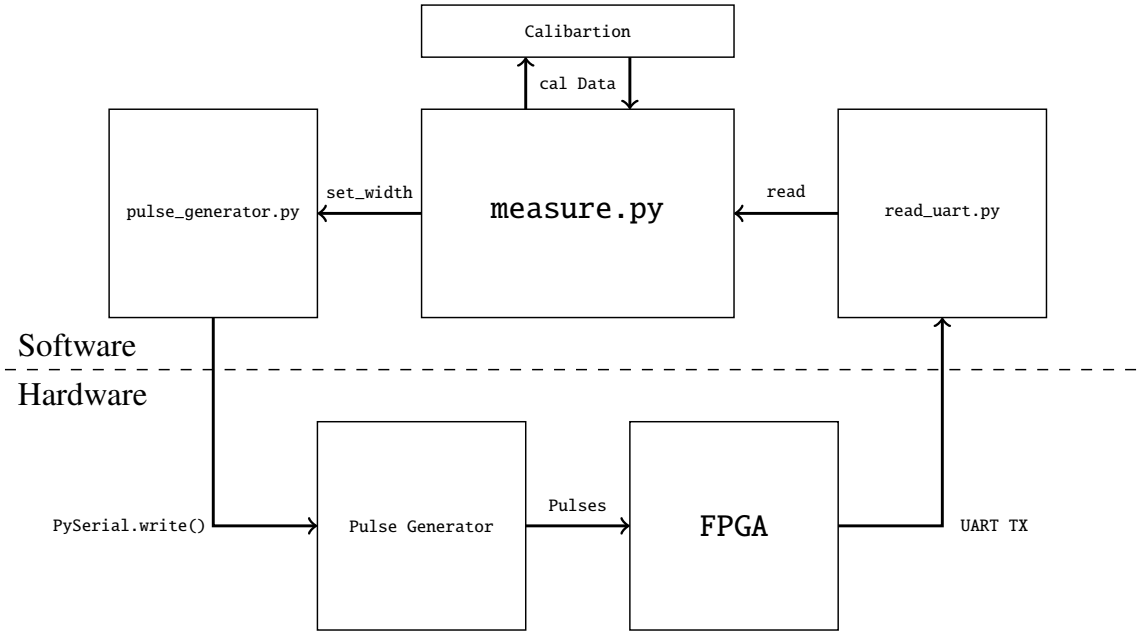


Figure 3.17: Measurement System

3.6.1 T560 Digital Delay Generator

Besides the TDC itself, an appropriate delay generator is the most crucial equipment for evaluating a new time measurement system. A digital delay generator from Highland Technology is used. The model description is given in Table 3.2.

It is capable of generating pulses with a width range from 2 ns up to 10 seconds with a 10 ps resolution [**?**]. The delay between the pulses reaches from 0 to 10 seconds, as well with a

Device	Manufacturer	P/N	S/N	Inv. No.	Cal.
MODEL T560 DIGITAL DELAY GENERATOR	HIGHLAND TECHNOLOGY	28A560-2K	2466	108731	None

Table 3.2: Delay generator device description

10 ps resolution. In a preceding semester project, the T560 was tested thoroughly [**?**]. It can be controlled by a serial RS-232 connection and the commands are given in ASCII. The Python library `PySerial` can be used for the communication.

3.6.1.1 Irregularities

Strange behavior while using the T560 has been detected. Normally, after a measurement series has been conducted, the T560 is reset by a command automatically from within the script. Then, if another measurement series is done afterwards, the delay generator is activated again with the chosen parameters. The strange thing now is, that the standard deviation of this second measurement series never goes under $\approx$ 30ps. Same applies to all further measurement series that are done after the first one. However, if the power supply is disconnected after a measurement series is finished, much better results can be achieved. The first suspicion was that the delay generator gets a worse standard deviation if its temperature rises over time, but this hypothesis was quickly dismissed. The reason must lie somewhere in the reset and startup sequence, but could not be determined specifically due to lack of time.

3.6.2 Receiving Data

As mentioned in section 3.3.3 the datawidth per dataframe is 16 bits or 2 bytes. However, all three measured results are transmitted in one UART package. Therefore, the transmitted string has be split into T_1, T_2 and T_c. Following code sequence converts the transmission into an integer representative.

```
1     data_in = serial.read(6).hex()  # 6 bytes = 3 * 2 bytes
2     t1 = int(data_in[0:4], base=16) # base=16 --> Hex
3     t2 = int(data_in[4:8], base=16)
4     tc = int(data_in[8:12], base=16)
```

3.6.3 Quality of Measurement

An important point after conducting a measurement is to ensure its quality. Several methods can show the overall performance of a TDC. Possibly the simplest method is to display the difference between the measured and true time. This is definitely a valuable property but more to identify if the system is working properly. There is normally a certain offset in such a measurement system. It is vital that such a difference stays constant throughout a measurement.

As mentioned in section 3.6.1, the pulse generator has a resolution of 10 ps. In order to get accurate measurements, more than one pulse has to be measured; normally 1000 or 10000 pulses with the same configuration are probed. Of such a large number, the standard deviation

σ can be a crucial information about the system. The standard deviation is defined as the square root from the variance, μ stands for the mean of the random variable X,

$$Var(X) = \mathbb{E}\left((X-\mu)^2\right) \tag{3.21}$$

$$\sigma = \sqrt{Var(X)} = \sqrt{\mathbb{E}\left((X-\mu)^2\right)} \tag{3.22}$$

which in this case can be rewritten for discrete samples as

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x}_i)^2}, \text{ where } \bar{x}_i = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{3.23}$$

In Fig. 3.18, one can see an example of a measurement with its corresponding histogram and

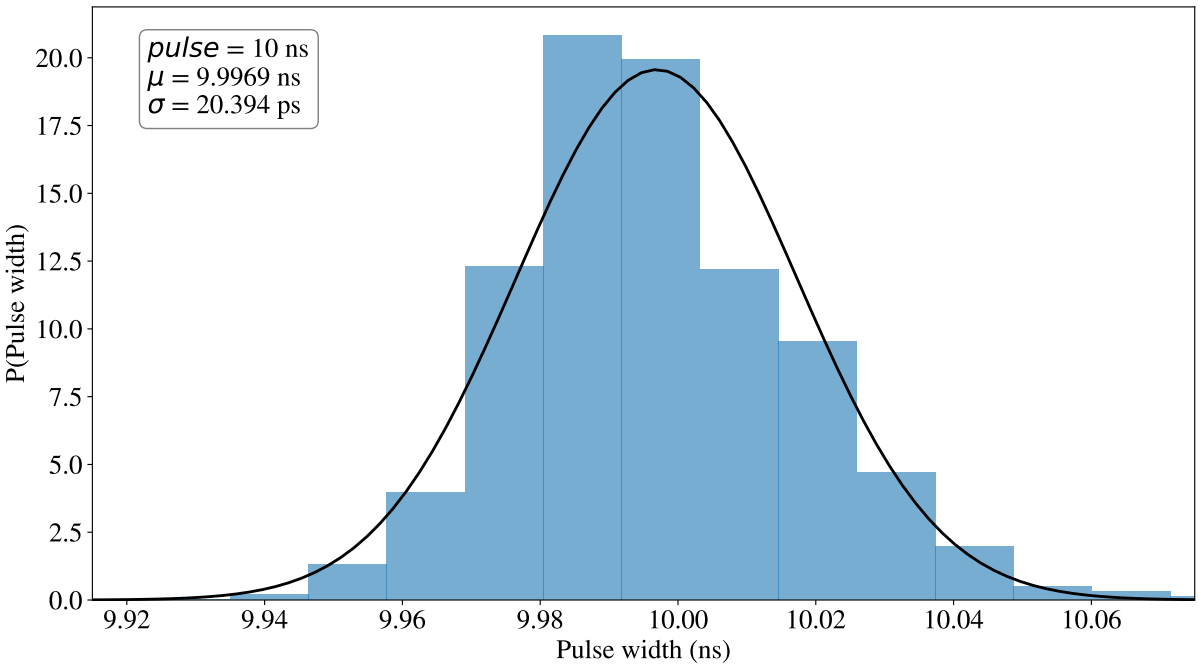


Figure 3.18: Example of a measuring series

its properties. Additionally, a normal distribution

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.24}$$

is fitted to the histogram.

3.6.3.1 Integral and Differential Nonlinearity

Other significant figures are the Integral Nonlinearity (INL) and the Differential Nonlinearity (DNL). They are commonly known in the field of Analog-to-Digital Converters (ADCs), which are normally converting a voltage to a digital number. However, the coarse principal of converting an analog number to its digital representative can be adapted to the present application. Hence the similarity, the theory of INL and DNL is as meaningful for TDCs as it is for ADCs.

The DNL describes the difference to the optimal transformation of an analog input to its digital value. The computation

$$DNL_n = \frac{t_{n+1} - t_n}{t_{LSB}} - 1 \tag{3.25}$$

is rather simple. If one measures two pulses, the DNL is the proportion of the measured step size to the ideal step size minus one. The idea is visually shown in Fig. 3.19, the ideal step size would be one LSB, whereas the actual step size is sometimes more (max DNL) or less (min DNL) than one LSB.

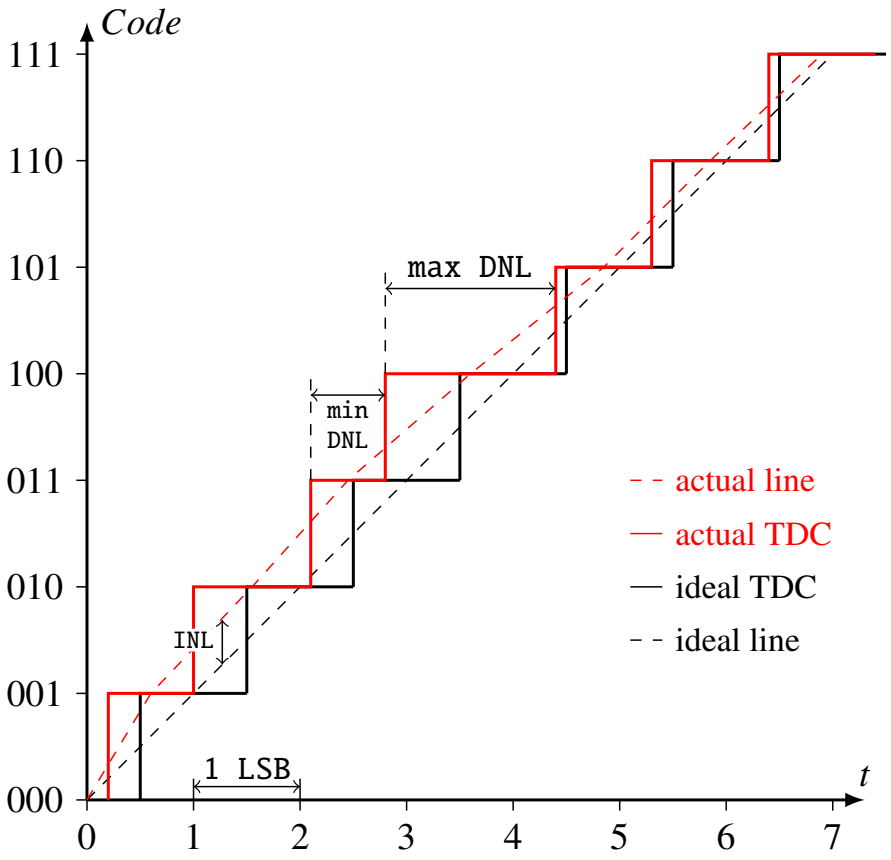


Figure 3.19: INL and DNL

Where the DNL describes the property of two neighboring points, the INL is looking at the whole system. The INL

$$INL_n = \frac{t_n - t_0}{t_{LSB}} - n \tag{3.26}$$

is the distance of a measured point to the ideal line as can be seen in Fig. 3.19. In Fig. 3.19 the DNL is more often too small than too big, therefore the INL is positive.

3.6.4 Measuring & Calibration

The class in the `measuring.py` file is responsible for setting all the parameters, doing the computation, plotting the results and carrying out the calibration. In addition, the measurement

sequence is defined in this file, thus this file is the only one the user has to interact with.

3.6.4.1 Calibration

In order to understand the process of calibration one can observe the histogram of T_1 and T_2. As an example, the same pulse with arbitrary width is measured for 10000 times. Because the pulses do not arrive aligned with the system clock on the FPGA, the width of T_1 and T_2 should be uniformly distributed over the expected range. In Fig. 3.20 the histogram of T_1 and T_2 is displayed. One can clearly see that T_1 and T_2 are not perfectly uniform distributed.

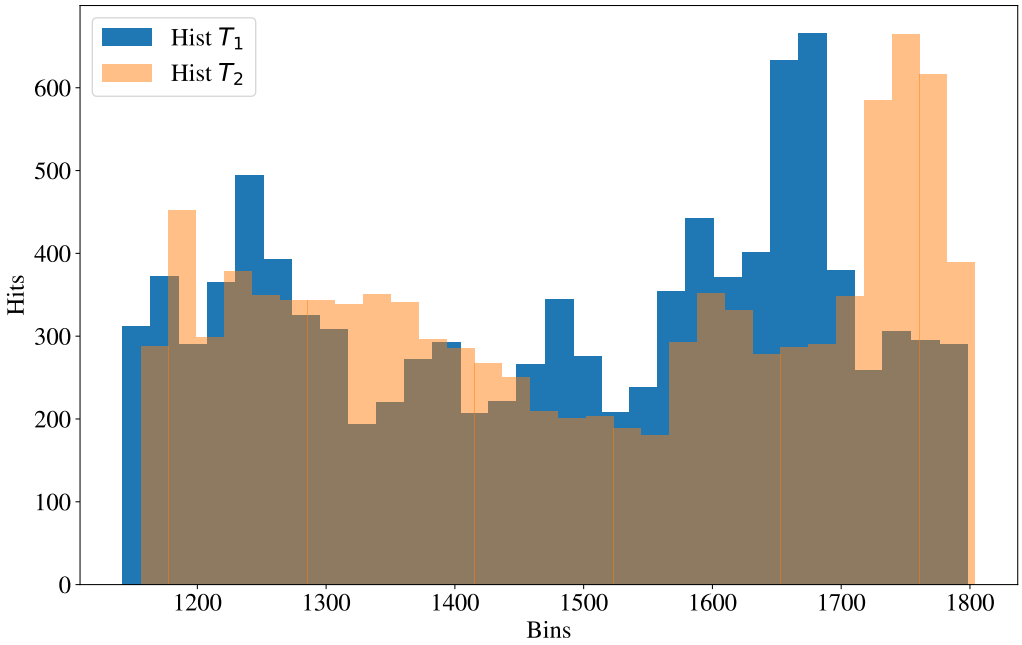


Figure 3.20: Histogram of T_1 and T_2 without calibration

Some values are generated more often than others (range 1650 → 1700 for T1), but it is known that they should be uniform distributed. For this purpose, the histogram equalization, which is normally applied in image processing, can be used to flatten the histogram [**?**]. Parameter r stands for the intensity values of an image in the range of $[0, L-1]$. The output of the transformation

$$s = M(r) \qquad 0 \leq r \leq L-1 \tag{3.27}$$

is the new value s of the input r.

It is assumed that

- $M(r)$ is a monotonic increasing function in the interval $0 \leq r \leq L-1$
- $0 \leq M(r) \leq L-1$ for $0 \leq r \leq L-1$

The transformation function can be computed with an integral over the Probability Density Function (PDF) of the input r, which is nothing but the Cumulative Distribution Function (CDF)

of the random variable r. Furthermore, the CDF is multiplied with the maximum value $L-1$.

$$s = M(r) = (L-1)\int_0^r P_r(w)\,dw \tag{3.28}$$

For the application in this project, some minor adjustments have to be made, because the distributions of T_1 and T_2 are in an expected range and not from zero to $L-1$. Firstly, the input has to be shifted to zero and secondly, the offset (minimal value of T_1 and T_2) has to be added to the transformation function $M(r)$.

$$s = M_{T_1}(T_1) = (max(T_1) - min(T_1) - 1)\int_0^r P_r(w)\,dw + min(T_1) \tag{3.29}$$

In Fig. 3.21, the calibration process and its steps can be seen.

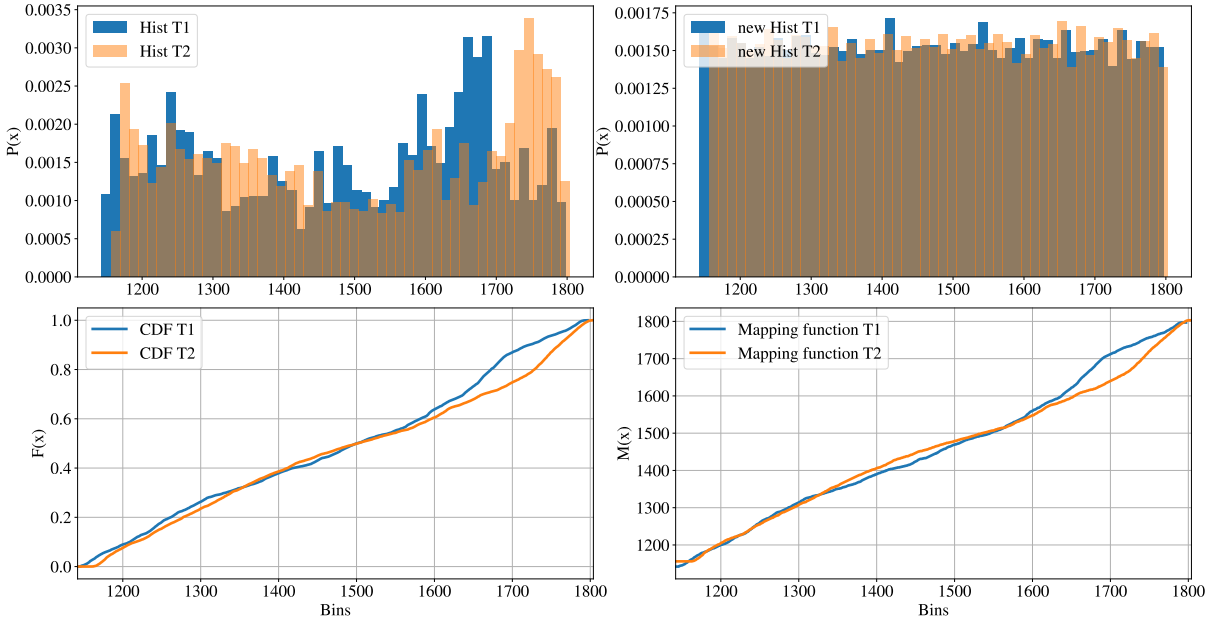


Figure 3.21: Top left: Original histogram, Top right: Original histogram transformed with the mapping function, Bottom left: CDF of T_1 and T_2 displayed in the expected range, Bottom right: Mapping function

After the calibration has been carried out, the generated mapping functions can be used for the following measurements. In Fig. 3.22 a new measurement series was taken, and the mapping function was applied. In contrast to Fig. 3.20, the histogram is now uniform.

For the implementation of the calibration process in software certain caution must be given. A common source of error is an index violation of the mapping function. As mentioned before $M(r)$ is a function in the interval $0 \leq r \leq L-1$. If afterwards in a measurement a value for T_1 or T_2 greater than $L-1$ occurs, an array index violation exception is thrown. In general should by the calibration dataset the whole range be covered, but it is possible that maybe through noise a uncovered value is measured.

To prevent this, all values in T_1 or T_2 which are greater than $L-1$ are replaced with $L-1$.

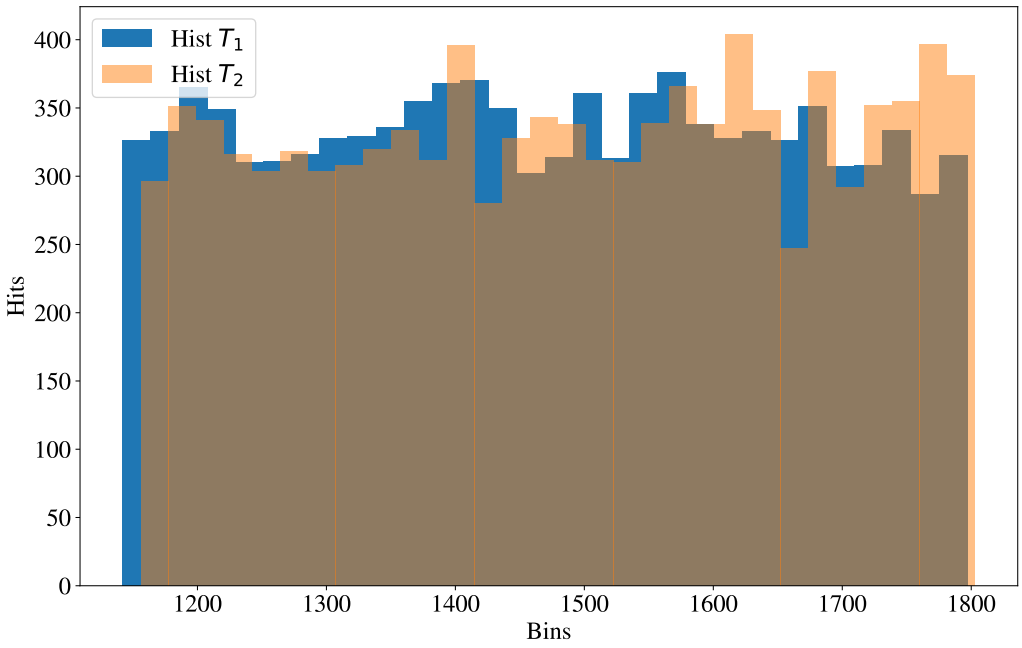


Figure 3.22: Histogram of T_1 and T_2 with calibration and a new measurement

3.6.4.2 Offset Cancellation

As mentioned in section 3.6.3, the difference from the measured time to the actual time should stay more or less constant. It is quite common that a TDC system has a certain offset. Multiple methods are possible to compensate such an error [**?**] [**?**]. Early measurements with the systems developed in this thesis have shown that the offset is following a certain curve, which is dependent on the pulse width. This offset was mostly ignored, due to the fact that it was constant. The differences to the target pulse widths for three independent measurements are shown in Fig. 3.23 . Furthermore, in Fig. 3.24 the curve is shifted so that the initial value of the curve equals to zero. This does further demonstrate the strong coherence of the different measurements. However, for the computation of the DNL and INL (described in section 3.6.3.1) the offset has to be compensated as good as possible. Because the offset calibration is not a priority in this thesis, a rather simple scheme was implemented to solve this issue.

1. Make a measurement with the same parameters as wanted for the DNL and INL.
2. Store the curve of the difference between the measurement and the ideal pulse width.
3. Make the same measurement again and subtract the stored curve from it.
4. Compute the DNL and INL.

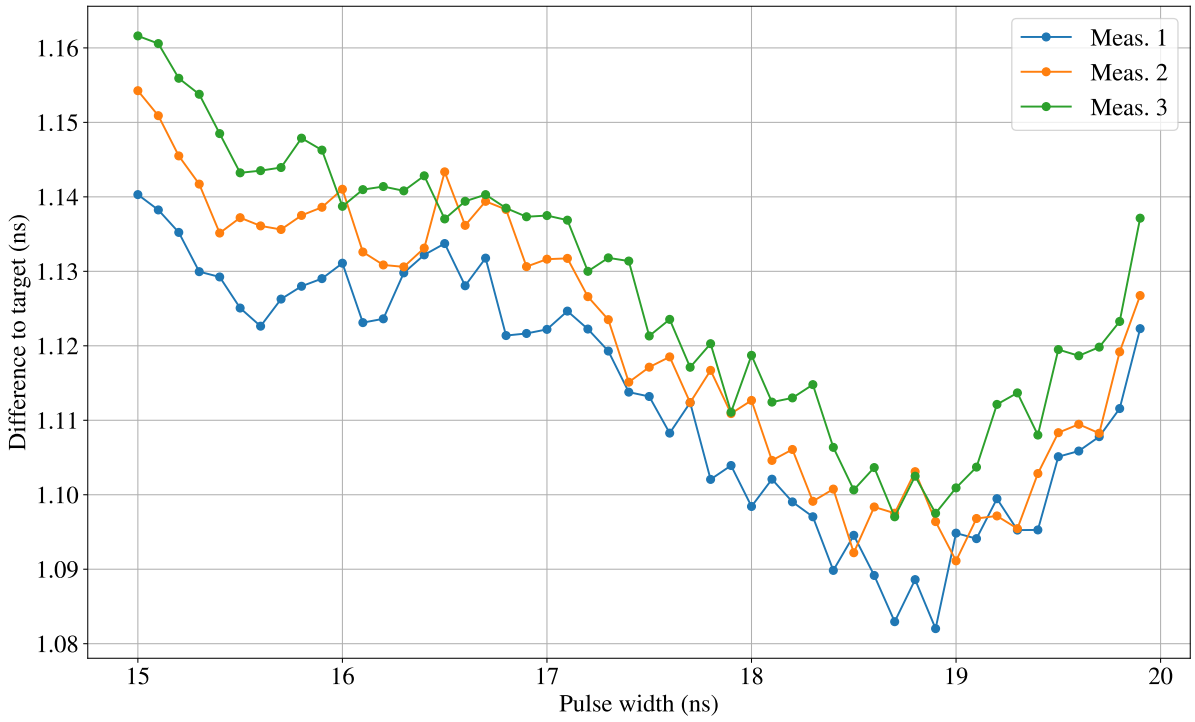


Figure 3.23: Difference to target

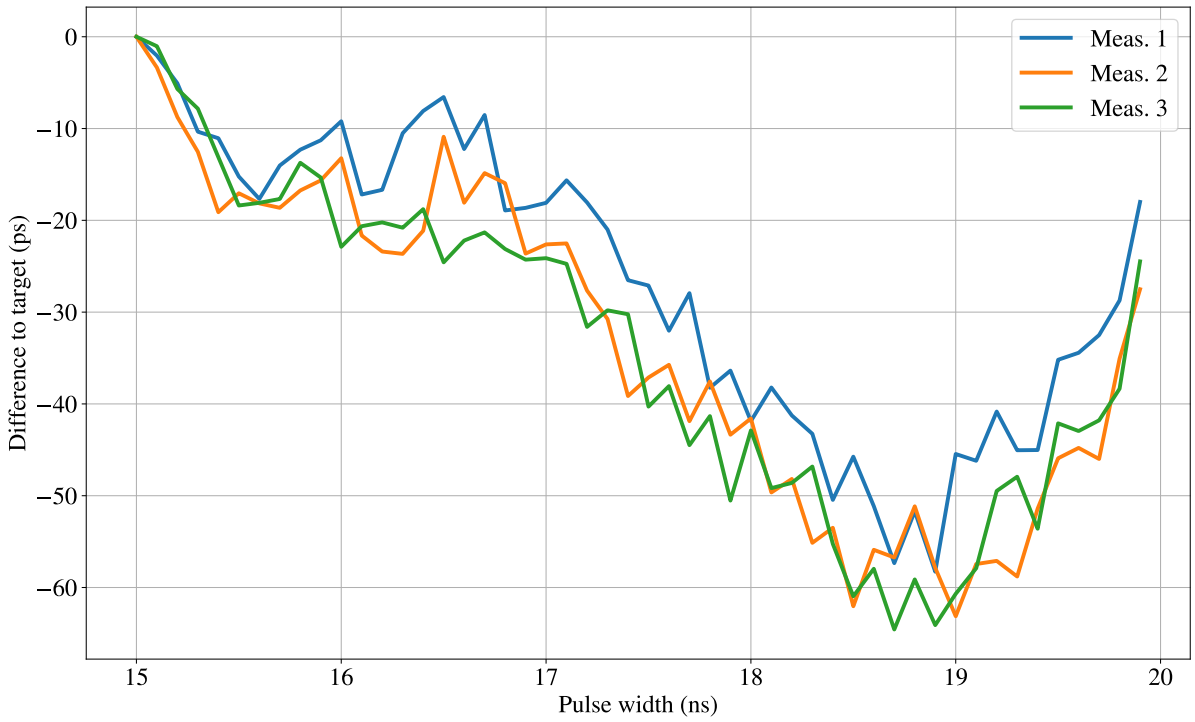


Figure 3.24: Difference to target with initial offset correction

3.6.5 Measuring Instruction

In order to conduct a complete measurement with the described system the following instruction can be followed.

- For the communication between the PC and the *Pulse Generator*, if available, an RS-232 connection should be used (a Universal Serial Bus (USB) to RS-232 adapter would probably do the job as well)
- For the communication between the PC and the FPGA, the CP210x USB to UART driver must be installed
- Connect the FPGA and load the desired *BIT-File*
- Connect the pulse generator
- For the connection between the PC and the FPGA, the activated communication port must be known and adjusted in the file.
- The `measure.py` file sets up the pulse generator, no interaction from the user is needed
- The main part of the `measure.py` file is listed below

```
if __name__ == '__main__':
    M = 5                          # number of first stage MMCMs
    N = 4                          # number of second stage MMCMs
    d = 63                         # overclocking factor numerator
    c = 8                          # overclocking factor denominator
    T = 10*10**-9                  # period of input frequency first MMCM stage in seconds
    DDR = 2                        # 1 -> without DDR, 2 -> with DDR
    initial_width = 15             # pulse width where the sweep starts in ns
    step =  0.1                    # step size for the sweep in ns
    sweep_size = 10                # how many steps per sweep
    matrix = 1                    # how many parallel cores
    n = 1000                       # how many measurements for averaging
    diff = np.load('diff.npy')     # difference for offset cancellation

    res = T/(M*N*DDR*d/c)*10**12

    m = measure(M, N, DDR, d, c, T, initial_width, matrix, COM=8)  # use the right com
        port
    diff_list, t1_list, t2_list, tc_list, t_list, width_list, mean_list, std_list =
        m.meas_sweep(n, step, sweep_size, initial_width, cal_mode=True)

    DNL, INL = m.INL_DNL((np.array(mean_list)-diff)*10**-9, step*10**-9, res)

    m.plot_histogram(t_list, width_list)
    m.plot_INL_DNL(INL, DNL, width_list)
    m.plot_T1_T2_hist(t1_list, t2_list)
    m.plot_diff(width_list, diff_list)
    m.plot_std(width_list, std_list)

    np.save('T2', np.array(t2_list).flatten())  # Data for calibration
    np.save('T1', np.array(t1_list).flatten())  # Data for calibration

    del m
```

- Set the parameters as given in the loaded program on the FPGA
- If data for the calibration is to be recorded, set `cal_mode = True`
- If one wants to activate the calibration process for the next measurement, set `cal_mode = False`

- Produce the plots as desired

For generating the mapping functions with the calibration, the script `calibration.py` in the *Calibration* folder must be run. No adjustments have to be made in this script. One can eventually toggle comments for all the plots. Make sure that the data files `T1.npy` and `T2.npy` are located in the same folder.

Chapter 4

Results

In this chapter, the results from the measurements with the developed TDC are presented. The used FPGAs are shown in Table 4.1.

Manufacturer	Device	Chip	Technology
Xilinx	NEXYS 4 DDR, Artix7	xc7a100tcsg324-1	28 nm
Xilinx	Zynq UltraScale+ ZCU102	xczu9eg-ffvb1156-2-e	16 nm
Xilinx	Kintex-7 KC705	xc7k325tffg900-2	28 nm

Table 4.1: Used FPGAs for the project

The same settings as in [**?**] were used for the measurements.

- Short range: 15 - 15.4 ns with 1 ps steps
- Mid range: 15 - 30 ns with 100 ps steps
- Long range: 15 - 1000 ns with 5000 ps steps

The measuring procedure is described in section 3.6. The tilde ~ symbol represents *range from to*.

To enhance the readability in the following sections, the results are only shown in tables. The corresponding plots can be found in Appendix D.

4.1 Kintex 7

Variant 1 in single-core and multi-core configurations.

4.1.1 Single-Core

Measurement	LSB (ps)	STD (ps)	DNL (LSB) (DNL (ps))	INL (LSB) (INL (ps))
Short Range	22.83	27.0 ~ 36.3	-0.676 ~ 0.62 (-15.4 ~ 14.2)	-0.66 ~ 0.62 (-15.1 ~ 14.2)
Mid Range	22.83	28.5 ~ 38.8	-0.726 ~ 0.715 (-16.6 ~ 16.3)	-0.792 ~ 0.403 (-18.1 ~ 9.2)
Long Range	22.83	29.0 ~ 42.9	-1.01 ~ 0.892 (-23.1 ~ 20.4)	-1.32 ~ 0.711 (-30.0 ~ 16.2)

Table 4.2: Kintex 7 Single Core Results

4.1.2 Multi-Core

Measurement	LSB (ps)	STD (ps)	DNL (LSB) (DNL (ps))	INL (LSB) (INL (ps))
Short Range	31.44	20.2 ~ 25.1	-0.42 ~ 0.393 (-13.2 ~ 12.3)	-0.522 ~ 0.337 (-16.4 ~ 10.6)
Mid Range	31.44	20.6 ~ 29.8	-0.565 ~ 0.386 (-17.8 ~ 12.1)	-0.488 ~ 0.33 (-15.3 ~ 10.4)
Long Range	31.44	21.2 ~ 37.7	-0.726 ~ 0.667 (-22.8 ~ 21.0)	-1.29 ~ 0.836 (-40.5 ~ 26.3)

Table 4.3: Kintex 7, 8-Core Results

Measurement	LSB (ps)	STD (ps)	DNL (LSB) (DNL (ps))	INL (LSB) (INL (ps))
Short Range	31.44	14.4 ~ 18.1	-0.346 ~ 0.474 (-10.9 ~ 14.9)	-0.44 ~ 0.466 (-13.8 ~ 14.7)
Mid Range	31.44	14.8 ~ 23.2	-0.452 ~ 0.447 (-14.2 ~ 14.1)	-0.325 ~ 0.646 (-10.2 ~ 20.3)
Long Range	31.44	15.0 ~ 31.3	-0.499 ~ 0.476 (-15.7 ~ 15.0)	-0.09 ~ 1.0 (-2.84 ~ 31.5)

Table 4.4: Kintex 7, 16-Core Results

4.1.3 General Note

- Single-Core
  - The achieved results are not so great, but in comparison to the single-core implementation on a Virtex-6 in [**?**] (std: (54.43 ~ 67.2 ps)) our results are promising.
  - In the paper it is mentioned that the structure on Altera FPGAs is beneficial to the structure on Xilinx FPGAs.
- Multi-Core
  - The achieved results are, as expected, better than in the single-core configuration. Note that the LSB is higher for the multi-core implementations.
  - Performance improvements from 8-core to 16-core implementations support the functionality of the parallelized design .
- Calibration
  - The calibration is for the single-core implementations not beneficial. Whereas for the multi-core, the calibration helps to improve the performance tremendously.
  - The calibration has to be newly done, if the FPGA is reset or re-programmed.

4.2 Zynq UltraScale+

Variant 2 in 'single-core' configuration.

4.2.1 Single Core

Measurement	LSB (ps)	STD (ps)	DNL (LSB) (DNL (ps))	INL (LSB) (INL (ps))
Short Range	6.94	11.8 ~ 16.4	-1.57 ~ 1.65 (-10.9 ~ 11.5)	-1.46 ~ 4.22 (-10.2 ~ 29.3)
Mid Range	6.94	13.6 ~ 17.5	-1.33 ~ 1.49 (-9.21 ~ 10.3)	-0.433 ~ 2.07 (-3.01 ~ 14.3)
Long Range	6.94	14.1 ~ 21.1	-2.41 ~ 2.37 (-16.8 ~ 16.4)	-1.07 ~ 4.64 (-7.42 ~ 32.2)

Table 4.5: Zynq UltraScale+ Results

4.2.2 General Note

- See the controversy regarding single-core and multi-core in section 3.4.
- Performance
  - The results for this implementation are very promising, even though a simpler structure as in [?] has been used, good results have been achieved.
  - An additional boost in performance can be expected with an upgrade to the same structure as in the paper.
- Calibration
  - The calibration improves the performance.
  - The calibration can be conducted once and can be used even when the FPGA has been reset or reloaded.

4.3 Nexys 4

Variant 1 in single-core and multi-core configurations.

4.3.1 Single-Core

Measurement	LSB (ps)	STD (ps)	DNL (LSB) (DNL (ps))	INL (LSB) (INL (ps))
Short Range	31.74	31.0 ~ 42.3	-0.456 ~ 0.497 (-14.5 ~ 15.8)	-1.01 ~ 0.046 (-32.2 ~ 1.46)
Mid Range	31.74	41.8 ~ 59.0	-0.595 ~ 0.542 (-18.9 ~ 17.2)	-0.972 ~ 0.301 (-30.9 ~ 9.56)
Long Range	31.74	36.2 ~ 143	-0.882 ~ 0.921 (-28.0 ~ 29.2)	-0.742 ~ 0.756 (-23.6 ~ 24.0)

Table 4.6: Nexys 4 Single-Core Results

4.3.2 Multi-Core

Measurement	LSB (ps)	STD (ps)	DNL (LSB) (DNL (ps))	INL (LSB) (INL (ps))
Short Range	31.74	16.8 ~ 19.8	-0.4 ~ 0.417 (-12.7 ~ 13.2)	-0.239 ~ 0.614 (-7.59 ~ 19.5)
Mid Range	31.74	16.2 ~ 23.9	-0.491 ~ 0.474 (-15.6 ~ 15.0)	-0.965 ~ 0.272 (-30.6 ~ 8.64)
Long Range	31.74	19.0 ~ 132	-0.69 ~ 0.89 (-21.9 ~ 28.2)	-1.21 ~ 0.895 (-38.4 ~ 28.4)

Table 4.7: Nexys 4, 16 Core Results

4.3.3 General Note

- Single-Core
  - The achieved results are not great, but compared to the LSB size it is in the expected range.
  - More problematic is that the standard deviation increases quite rapidly with longer measurements; it could not be clarified thoroughly, why the standard deviation rises so much faster than with the implementation on the Kintex.
- Multi-Core
  - Again, a large improvement to the single-core implementation
  - The problem with long measurements also exists with the multi-core implementation.
- Calibration
  - The calibration is not beneficial for the single-core implementations, whereas for the multi-core the calibration helps to improve the performance tremendously.
  - The calibration has to be newly applied if the FPGA is reset or re-programmed.

4.4 Hardware Utilization

The hardware utilization regarding the FFs and LUTs is very low. As expected, the MMCMs are the components which are utilized most. Especially FPGAs with an integrated microprocessor (Zynq UltraScale+) have, in general, a small number of MMCMs. The utilization of the previously shown implementations is listed in Table 4.8.

Device	Implementation	Variant	LUTs	FFs	MMCMs
Kintex 7	Single Core	Variant 1	742 0.36 %	925 0.23 %	7 70 %
Nexys 4	Single Core	Variant 1	659 1.04 %	829 0.65 %	6 100 %
Kintex 7	8 Core	Variant 1	4553 2.23 %	4989 1.22 %	7 70 %
Kintex 7	16 Core	Variant 1	8593 4.22 %	9597 2.35 %	7 70 %
Nexys 4	16 Core	Variant 1	7254 11.44 %	8053 6.35 %	6 100 %
Zynq UltraScale+	Singe Core	Variant 2	6702 2.45 %	7739 1.41 %	2 50 %

Table 4.8: Hardware Utilization of the different implementations

Chapter 5

Conclusion

Paper	FPGA	Structure	Dynamic Range	LSB (ps)	Standard Deviation (LSB)	DNL (LSB)	INL (LSB)
This Work	Zynq Ultrascale+	2-D Vernier	30 ns	6.94	2.21 (15.4 ps)	-1.33 ~ 1.49 (-9.21 ~ 10.3 ps)	-0.433 ~ 2.07 (-3.01 ~ 14.3 ps)
This Work	Kintex 7 serires	Stochastic PLL delay matrix	30 ns	31.44	0.62 (19.6 ps)	-0.452 ~ 0.447 (-14.2 ~ 14.1 ps)	-0.325 ~ 0.646 (-10.2 ~ 20.3 ps)
[?]	Stratix-IV	2-D Vernier semi stochastic	16.5 ns	2.5	2.69 (6.72 ps)	-0.566 ~ 0.46 (-1.4 ~ 1.15 ps)	-2.98~ 3.23 (-7.45 ~ 8.08 ps)
[?]	Stratix-IV	PLL delay matrix	1 μs	15.6	1.0 (15.6 ps)	-0.176 ~ 0.184 (-2.75 ~ 2.87 ps)	-0.157 ~ 0.137 (-2.45 ~ 2.14 ps)
[?]	Kintex UltraScale	Dual-sampling TDL	440 ns	2.3	1.7 (3.9 ps)	NA*	NA
[?]	Virtex-6	Dual-phase TDL	20 ns	10	1.28 (12.8 ps)	-1 ~ 1.88 (-10 ~ 18.8 ps)	-1.63 ~ 3.93 (-16.3 ~ 39.3 ps)
[?]	Stratix III	RO-based Vernier+ Bidirectional Operating	NA	24.5	1.14 (28 ps)	-0.20 ~ 0.25 (-4.9 ~ 6.125 ps)	0.03 ~ 0.82 (0.735 ~ 20.09 ps)
[?]	Zynq UltraScale+	Statistic TDL	NA	1.0	9.12 (9.12 ps)	-1.0 ~ 4.0 (-1.0 ~ 4.0 ps)	-1.0 ~ 3.5 (-1.0 ~ 3.5 ps)

Table 5.1: Performance comparisons among TDCs, *NA = not available

Literature Study The literature study has shown that nowadays, numerous methods to implement a high-performance TDC have been published. Especially the vernier principle has several publication with the same basic idea but a slightly different implementation approach. In Table 5.1, a comparison with prior art in the field of TDCs on FPGAs is listed. Note that the values should serve more as an overview than for a real performance comparison because it is

not always absolutely clear how the published numbers were measured. Therefore, one should not compare the numbers one by one. The values from this work are generated by a mid-range measurement sweep (15 - 30 ns), whereas for the standard deviation the average value over the measurement range is presented. The tilde ~ symbol represents *range from to*.

Performance On the first sight, the results may lure one to a false assumption. Especially the performance of the DNL and INL is, in comparison to prior art, not holding up. The main goal of this work was firstly and foremost to implement a new method. The current solution for the offset cancellation is not satisfactory. To solely rate the raw performance of TDC, the standard deviation is the right value and, regarding this, the implemented structures in this thesis are performing well.

One point that was critical in the previously used TDL TDC [**?**] is that, with the statistical calibration approach, the hardware utilization on the Ultrascale+ is almost 100 percent. This work shows that a much less resource heavy implementation (Table 4.8) can achieve quite good results. The standard deviation is not on the same level as in the above mentioned TDL TDC implementation, but there is still room for improvement in the design and, thanks to the low hardware utilization, there is still plenty of room for such improvements.

A Glimpse to Future Projects Various approaches could be taken to further improve the performance of the system. Primarily the calibration and the offset cancellation have a great potential for improvement. Additional measurements should be carried out for the calibration to validate the correct calibration range. The whole topic of calibration can be taken much further, which will lead to an improved standard deviation. Also with regard to offset cancellation, a more sophisticated method can be used to continue improving performance. In addition to these two topics, the use of enhanced configurations with higher frequencies should also lead to better results. To implement such a design successfully, the floorplanning required involves a lot of work. These improvement approaches provide good starting points for subsequent projects. Finally yet importantly, an improvement of the timing on the FPGA might also lead to an increase in performance.

Summary One of the first steps was to create a time schedule, it can be found in Appendix B. The literature study could be completed on time, which is not surprising as it was carried out right at the start of the work. As one can see, the main part, which is the implementation, was also successfully completed. It should be noted that the implementation part and the measurement part overlapped, which was not planned. The reason for this is, on the one hand, that improvements were implemented until week 15 which still had to be measured and, on the other hand, because we could not work at school due to the *COVID-19* outbreak. There is only one delay generator (Table 3.2) and we had to work from home, so the final measurements had to wait until week 15, when we were allowed to spend a day in the lab. The improvement part is only displayed with 80 percent completeness because there are still parts that can be improved. However, the main parts of the given task were completed.

Let us make a short summary of the completed project. A lot of time was invested and a lot has come of it. In the beginning, thanks to the extensive literature study, a good insight into the topic could be gained. This due to many read and studied papers. The theoretical basics for TDCs using some variation of the Vernier principle were summarized. It became clear, that there are many different possibilities to realize a TDC. Many very good results

have been achieved in various publications so far, but it is difficult to estimate the effort behind such a project. Especially, since there is very little written about the actual implementation parts, which are probably the most time consuming elements of the whole undertaking. In the end it was decided to study and reproduce the work of a Taiwanese research group ([**?**], [**?**]). The exact realization of the structure had to be designed from scratch. Most of the effort was put into the implementation, where first, the structure had to be described in VHDL. Further, the written code had to be checked by simulation. First measurements quickly showed some small problems in the implementation, which could be rapidly eliminated. With these first improvements, promising results could be achieved. Furthermore, a lot of time was spent to try different little tricks and tweaks to achieve further improvements in the results. Most of these evaluations were without a real outcome, for example the manual floorplanning and routing and the timing analysis of different elements that could be used as delay elements. The introduction of a multi-core approach for the first variant brought the best improvements. The two realized designs do not quite catch up with the work of the above mentioned Taiwanese research group, but still deliver satisfactory results.

The project was not without problems, the biggest obstacle was probably the pulse generator. As described in the paper, irregularities were discovered. Unfortunately, this happened very late in the work. Over several weeks, it was believed that the performance of the implementation could not be improved anymore because, whatever improvements were tried out, the standard deviation never got below $\approx 30\,\text{ps}$. Ultimately, it was discovered that the pulse generator was the source of problems and it was possible to get a lower standard deviation.

Fortunately, this work was not too much influenced by the *COVID-19* pandemic and could generally be carried out without major disadvantages.

Finally, a satisfying work can be presented.

Chapter 6

Declaration of Authorship

We hereby declare that we are the sole authors of this student research paper and that we have not used any sources other than those listed in the bibliography and identified as reference. We further declare that we have not submitted this thesis at any other institution in order to obtain a degree.

Noah Kälin | Michael Schmid

Rapperswil, December 3, 2025

Appendices

Appendix A

Task

BACHELORARBEIT

Für

Noah Kälin und Michael Schmid

FPGA-basierte Time to Digital Converter

1. EINFÜHRUNG

In einem Projekt mit der Universität Zürich werden mittels Fluoreszenzlebensdauer-Mikroskopie die Vorgänge im Hirn erforscht. Dabei wird die Zeit gemessen, in der ein fluoreszierendes Molekül im angeregten Zustand bleibt. Das fluoreszierende Molekül wird mittels sehr kurzem Laserimpuls (< 1 ps) angeregt, also in einen höheren Energiezustand versetzt. Durch das ankommende anregende Photon wird ein Elektron auf ein höheres Energieniveau gehoben, die Energie also im Elektron zwischengespeichert. Das Elektron bleibt für einige Nanosekunden auf dem höheren Energieniveau, bevor es wieder zurückfällt und dabei ein neues energieärmeres Photon aussendet (ein Photon mit längerer Wellenlänge). Ein Sensor (Avalanche-Photodiode) empfängt dieses Photon und wandelt es in einen elektrischen Puls.

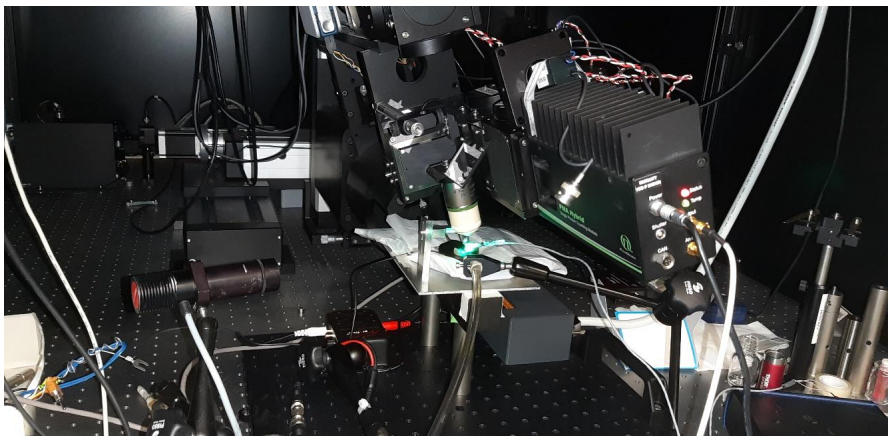


Abbildung 1: Mikroskop inklusive Photonensensor (PMA Hybrid) und Maus als Prüfling

Aus Sicht des Elektroingenieurs haben wir zwei Signale. Ein Signal mit Referenzpulsen, jeweils zum Zeitpunkt des Laserpulses und ein zweites Signal mit einem Puls pro empfangenem Photon. Eine Messung, bei der ein ganzes Bild erstellt wird, bestehend aus Einzelmessungen auf Pixelbasis. Es sind mindestens hunderte Photonen pro Pixel notwendig, um mittels statistischer Methoden die Lebensdauer der gemessenen Moleküle zu bestimmen. Für ein ganzes Bild sind somit Millionen von Einzelmessungen notwendig. Die Zeitdifferenz zwischen dem Laserpuls und Photonensensorpuls soll möglichst hoch aufgelöst (Grössenordnung einige Picosekunden) werden.

Eine Möglichkeit, die benötigte zeitliche Auflösung zu erreichen, sind sehr schnelle ADCs. Diese sind jedoch sehr teuer und benötigen eine aufwändige Datenverarbeitung, um die Samples zu lesen und zu verarbeiten. Aktuell sind ADCs mit über 100 GSPS kaum erhältlich.

Eine zweite Variante ist die direkte Zweitmessungen mit einem Time to Digital Converter (TDC). Die einfachste Form eines TDC ist ein sehr schneller Counter. Für die grobe Auflösung sind Counter sehr gut geeignet, weil diese auf einfache Art einen sehr grossen Bereich abdecken können (ein 64 Bit Counter mit einer Zählrate von 1 GHz hat einen Messbereich von 584 Jahren, aufgelöst in 1-Nanosekunden-Schritten). Mit Maximalfrequenzen im Bereich von einigen GHz sind Counter zu langsam für die Anwendung. Wird zusätzlich zum einfachen Counter eine spezielle Schaltung verwendet, die einen Puls zwischen den Clock Flanken des Counters zeitlich sehr hoch auflösen kann, entsteht eine universelle Stoppuhr mit grossem Bereich und hoher zeitlicher Auflösung, ein TDC.

2. BESTEHENDES SYSTEM MIT DELAY LINE

Der bereits existierende TDC ist auf dem Xilinx UltraScale+ FPGA (Field Programmable Gate Array) Baustein des ZCU102 Evaluation Kits implementiert. Der Kern des TDCs wird mit einem 600 MHz Clock betrieben und besteht aus zwei Komponenten, einem Counter (Grobzähler) und einer Delay Line (Feinzähler). Der Grobzähler zählt die Anzahl Clock Zyklen. Er löst somit die Zeit mit 1.666 ns auf. Der Feinzähler besteht aus einer Kette von logischen Elementen, die je ein gewisses Delay aufweisen. Hinter jedem dieser Elemente befindet sich ein Flip Flop (FF). Die Schaltung ist in Abbildung 2 gezeichnet. Der zu messende Puls propagiert durch die Delay Kette. Die steigende Clock Flanke (STOP Signal) schaltet alle FFs synchron. Anhand der FF Outputs wird nun bestimmt, wie weit der Puls durch die Delay Kette propagierte. Die gemessene Zeit ist der Wert des Grobzählers plus die Anzahl der propagierten Elemente mal das Delay eines Elements ($T = Count * T_{\mathrm{Clk}} + k * \tau$).

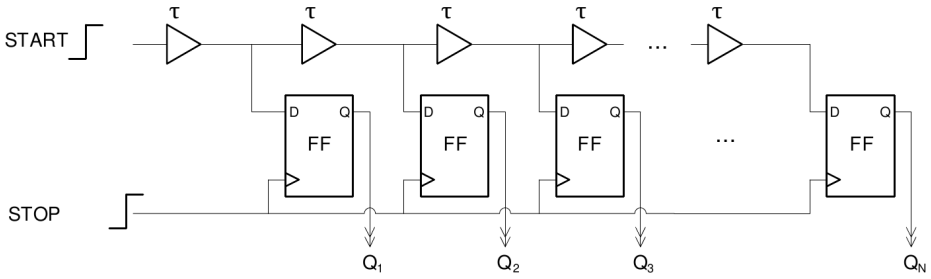


Abbildung 2: Schema einer TDC Delay Line [1]

Die Verzögerungszeit τ eines Elements bestimmt somit die zeitliche Auflösung. Ein möglichst kurzes Delay τ ist deshalb notwendig für eine hochaufgelöste Messung. Ausserdem sollten die Elemente möglichst identisch sein (auch das Routing von Element zu Element und von Element zu FF), damit sich die Delays wenig unterscheiden (Linearität). In einem FPGA sind jedoch keine für TDC Anwendungen optimierte Delaylines vervügbar.

Die wahrscheinlich am besten geeignete Delayline in einem FPGA ist die Carrychain. Diese ist eigentlich dafür optimiert, schnelle Addierer in den CLBs (Configurable Logic Block) zu realisieren. Sie ist zwischen den LUTs (LookUp Table) und FFs platziert. Weil sie auf das schnelle Weiterleiten eines Carrys optimiert ist, erfüllt sie bereits einige Voraussetzungen der Delayline. Hinter jedem Element befindet sich ein FF und das Delay eines einzelnen Elements ist kurz. Wie im Schema von Abbildung 3 zu sehen, sind jedoch zusätzliche Muxes (CYINT) in der Carrychain vorhanden, die das Delay zwischen einzelnen Elementen verlängern. Ausserdem ist die physikalische Platzierung der Elemente im FPGA (Layout) unklar.

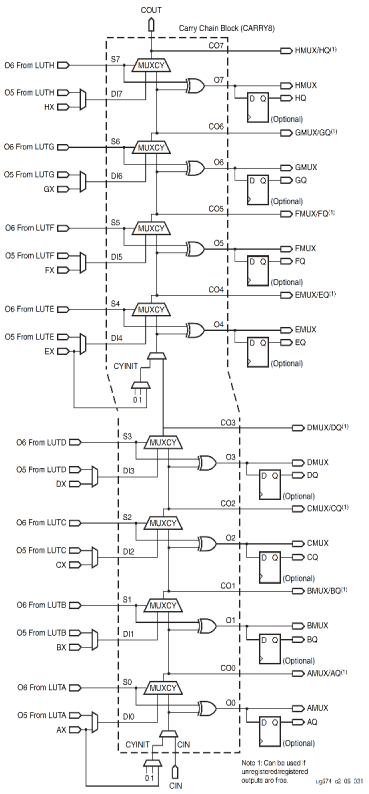


Abbildung 3: Schema einer TDC Delay Line [1]

Diese Struktur benötigt für die hohe Auflösung eine Kalibration. Eine Vorgängerarbeit hat bereits gezeigt, dass dies mit den Messdaten grundsätzlich möglich ist und auch die gewünschte Auflösung erreicht werden kann. Es gibt aber das Problem, dass der Delay Line Ansatz mit der aktuellen Kalibration sehr Hardwareaufwändig ist. Ausserdem stellt der Ansatz eine Anforderung an die Messdaten: Die Messdaten müssen unabhängig vom FPGA Clock sein, damit die Kalbration funktioniert.

3. ALTERNATIVE ANSÄTZE

Unter den Begriffen "Time-to-Digital-Converter" und "time-based circuits" finden Sie diverse Implementationen sowie käufliche Systeme von TDCs. Die Beschreibungen sind häufig spezifisch für bestimmte Anwendungen. Viele Anwendungen erwarten einen Start und einem Stopp Puls [3]. Sie finden aber auch Literatur, die sich direkt auf unsere Anwendung (FLIM) bezieht [4]. Lassen Sie sich von den Anwendungen nicht ablenken und konzentrieren Sie sich primär auf FPGA-basierte TDCs. Ein möglicher alternativer Ansatz ist die Vernier Architektur.

4. AUFGABENSTELLUNG

Das Ziel dieser Arbeit besteht darin, einen kompakten TDC auf einem FPGA zu implementieren. Dies kann mit neuen respektive alternativen Ansätzen der TDC Implementation und/oder durch Verbesserung der bestehenden Implementation realisiert werden. Die Auflösung soll durch Messungen nachgewiesen werden.

1. Arbeiten Sie sich in die Grundlagen und die Theorie ein.
2. Führen Sie eine Literaturstudie durch. Das Ziel ist dabei, den Stand der Technik bezüglich FPGA Realisierungen von TDCs aufzuzeigen.
3. Es gibt verschiedene Ansätze, wie ein TDC auf einem FPGA implementiert werden kann.
    a. Wählen Sie mindestens einen anderen Ansatz.
    b. Implementieren Sie die gewählten Ansätze auf dem FPGA Board.
    c. Spezifizieren Sie den neuen TDC. Interessant sind unter anderem Angaben zur Auflösung, maximale Messfrequenz oder dem Platzbedarf (LUT/FF/BRAM/DSP) auf dem FPGA.
4. (Optional) Verbessern Sie die interne Kalibration des bestehenden TDCs.
    a. Ist es möglich, die Kalibration ohne externe Signale durchzuführen?
    b. Kann der Ressourcenverbrauch der aktuellen Implementation gesenkt werden?
    c. Spezifizieren Sie den neuen TDC. Wie zuverlässig ist die neue Kalibration?

Zu Beginn der Arbeit ist ein Projektplan zu erstellen und mit dem Betreuer abzusprechen. Während der Arbeit sind zur Überprüfung wichtiger Arbeitsschritte verschiedene Review-Meetings einzuplanen. Von den Review-Meetings sind Protokolle zu verfassen.

Wir empfehlen Ihnen, ein Laborjournal zu führen. Dieses Journal kann vom Betreuer bei den Besprechungen eingesehen werden. Das Laborjournal ist nicht Teil des Berichtes.

5. ZIELE

- Verständnis des theoretischen Hintergrundes: Wie wird eine TDC im FPGA realisiert? Welche Schwierigkeiten können dabei auftreten?
- Funktionierende Implementationen inklusive Versuchsaufbau
- Auswertung & Interpretation der Daten sowie eine vollständige Dokumentation der Erkenntnisse

6. BERICHT

Über die Arbeit ist ein Bericht zu verfassen, in dem alle gemachten Überlegungen, Abklärungen, Berechnungen und Untersuchungen detailliert (inkl. Illustrationen) dokumentiert werden. Der Bericht soll gut leserlich geschrieben und übersichtlich gegliedert sein. Im Bericht dürfen die folgenden Teile nicht fehlen: *Inhaltsverzeichnis, Kurzfassung (Abstract) im Umfang von einer halben bis ganzen Seite, Originaltext der Aufgabenstellung, Einführung, Hauptteil mit Lösungsansatz, Lösungsbeschreibung und den Resultaten, Zusammenfassung (Summary) von 2-4 Seiten Umfang, Anhang mit allen relevanten Daten und Detail-Informationen, Literaturverzeichnis.* Der gesamte Bericht ohne Anhang soll maximal 60 Seiten umfassen. Weitere Richtlinien zum Verfassen eines wissenschaftlichen Berichtes finden Sie z.B. in [6], [7].

Der Bericht ist in zwei Druckexemplaren sowie in elektronischer Form abzugeben. Die Textverarbeitungs-Software und das Datenformat sind zu Beginn der Arbeit mit dem Betreuer und mit dem Laborassistenten abzusprechen. Alle Daten, welche für die Weiterführung der Arbeit relevant sind, sollen zusammen mit dem Bericht

in elektronischer Form im Verzeichnis \\hsr.ch\root\auw\sge\studarbeiten\MikroelSys\.....\Results abgelegt werden.

7. TERMINE

• Beginn der Arbeit und Laborbezug	Mo	17.	Feb.	2020
• Abgabe des Schlussberichtes bei P. Zbinden, Büro 8.221 bis 13:30	Fr	12.	Juni	2020
• Rückgabe von aufgeräumtem Arbeitsplatz und Schlüssel	tbd.			2020

8. ORGANISATION

- Betreuung der Arbeit: Dorian Amiet (Tel. 055 222 46 59, dorian.amiet@hsr.ch)
  Prof. Dr. Paul Zbinden (Tel. 055 222 45 84, paul.zbinden@hsr.ch)
- Besprechungen: wöchentlich, Zeitpunkt nach Absprache
- Arbeitsplatz: Zuweisung durch P. Roffler (Raum 6.005)
- Betreuung des Labors: P. Roffler (Raum 6.005, Tel. 055 222 46 36, proffler@hsr.ch) ist zuständig für Schlüssel, Arbeitsplatz, Geräte und Material

Ich wünsche Ihnen viel Erfolg

Rapperswil, 20. Februar 2020
Prof. Dr. Paul Zbinden

9. LITERATUR

[1] Erni, Werner & Keshelashvili, Irakli & Krusche, Bernd & Steinacher, M. & Heng, Y. & Liu, Zhen-An & Liu, Holiday & Shen, Xufan & Wang, Qing & Xu, Huiqiang & Aab, A. & Albrecht, Malte & Becker, Juergen & Csapó, A. & Feldbauer, Florian & Fink, M. & Friedel, P. & Heinsius, F. & Held, T. & Zmeskal, Johann. (2013). Technical design report for the PANDA (AntiProton Annihilations at Darmstadt) Straw Tube Tracker. The European Physical Journal A. 49. 10.1140/epja/i2013-13025-8.

[2] Xilinx, UltraScale Architecture CLB User Guide, UG574 (v1.5), 2017

[3] Zhang, Min & Wang, Hai & Liu, Yan. (2017). A 7.4 ps FPGA-based TDC with a 1024-unit measurement matrix. Sensors. 17. 865. 10.3390/s17040865.

[4] A. Tontini, L. Gasparini, L. Pancheri and R. Passerone, "Design and Characterization of a Low-Cost FPGA-Based TDC," in IEEE Transactions on Nuclear Science, vol. 65, no. 2, pp. 680-690, Feb. 2018. doi: 10.1109/TNS.2018.2790703

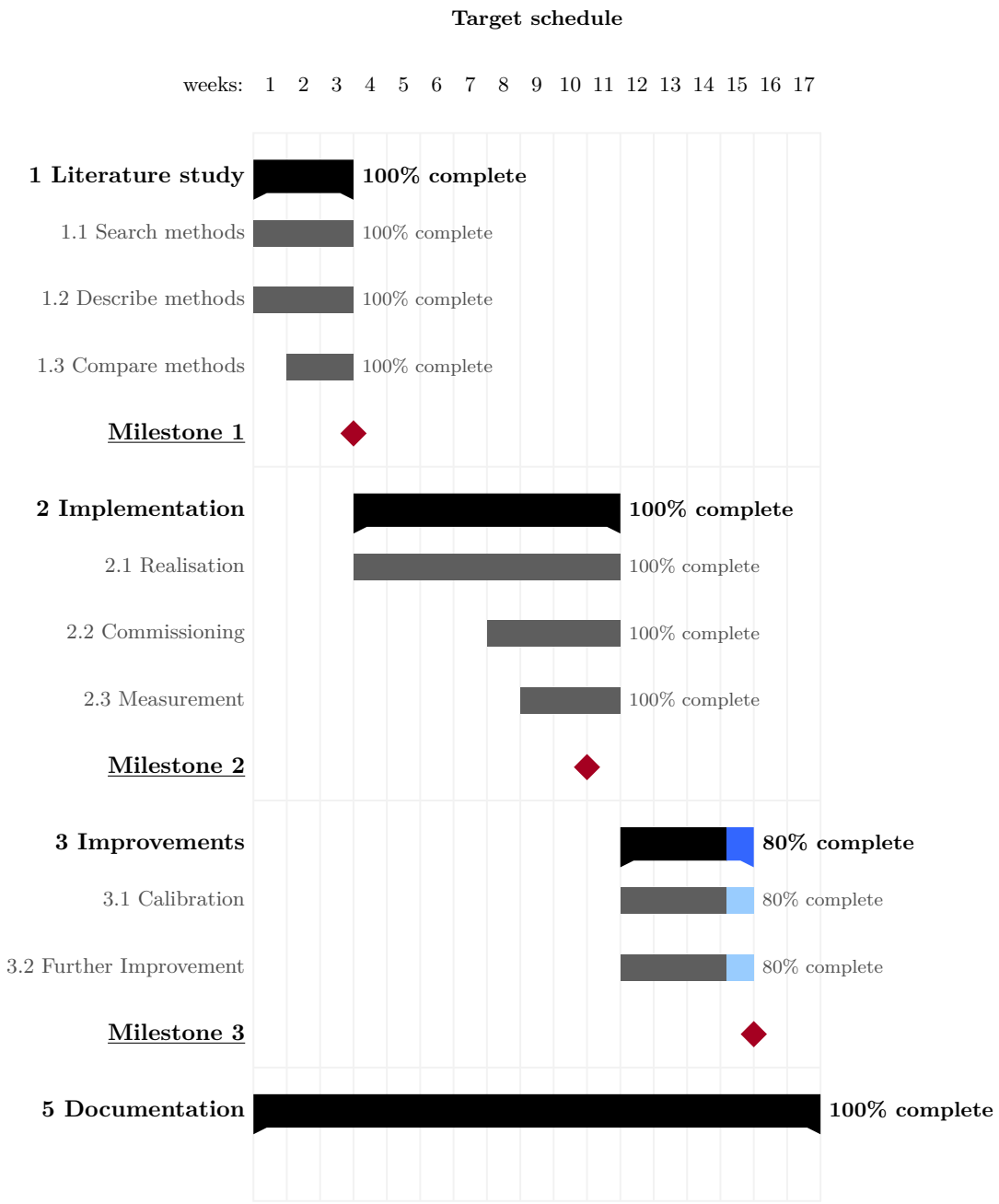
[5] S. Berrima, Y. Blaquière and Y. Savaria, "A multi-measurements RO-TDC implemented in a Xilinx field programmable gate array," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), Baltimore, MD, 2017, pp. 1-4. 10.1109/ISCAS.2017.8050436

[6] A. Verhein und A. Simeon, «Werkzeugkasten Technische Berichte,» HSR, Rapperswil, 2013.

[7] P. Mayer, 77 mal wissenschaftliches Schreiben - eine Anleitung, Basel: Advanced Study Center, Universität Basel, 2010.

Appendix B

Schedule



Appendix C

Mail Conversation

——Original message——
From:Schmid Michael<michael.schmid2@hsr.ch>
To:poki@mail.ntust.edu.tw<poki@mail.ntust.edu.tw>
Date: Thu, 07 May 2020 03:03:10
Subject: High-Precision PLL Delay Matrix With Overclocking and Double Data Rate for Accurate FPGA Time-to-Digital Converters

Dear Mr. Chen

I am very interested in the paper "High-Precision PLL Delay Matrix With Overclocking and Double Data Rate for Accurate FPGA Time-to-Digital Converters" you published last year.
I am currently rebuilding the proposed architecture on an Xilinx FPGA.
Thanks to the very informational paper it is working great so far.
There is one point where my design cannot follow the same specification as you in the paper.
On the Xilinx Virtex-6 FPGA you used a maximum frequency of 1.175 GHz. Furthermore it is written that the mixed mode clock managers (MMCMs) have been used to accomplish the overclocking and phase shifting.

In the DC and Switching Characteristics data sheet (DS152) is written that the MMCM Maximum Output Frequency is 700-800 MHz depending on the speed grade.
Then again the Maximum MMCM VCO Frequency is 1200–1600 MHz.

I am working with the clocking wizard and there I cannot exceed a frequency of 800 MHz.

My question is now, how have you been able to utilize such a high clock speed?
Or is there a possibility to get the VCO output?
I would be very grateful if you or a member of your team could help me out with this issue.

Best Regards

Michael Schmid

poki <poki@mail.ntust.edu.tw> 於2020年5月7日四下午1:48道:

Dear Ray-Ting,

Could you please help me answer Michael's question? Many thanks.

Best regards,

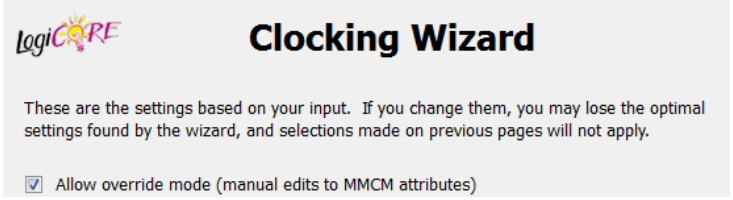
Poki

Dean, College of Applied Sciences
Professor, Dept. of Electronic and Computer Engineering
Associate Editors, IEEE Transactions on VLSI Systems / IEEE Access
National Taiwan University of Science and Technology
43,Sec.4,Keelung Rd.,Taipei,106,Taiwan
Tel: +886-2-2737-6400
Fax:+886-2-2737-6424
WWW: http://www.ece.ntust.edu.tw/et/faculty_en.php?user=poki

Dear Michael

The MMCM Maximum frequency write on the datasheet (DS152) is because the limitation on the horizontal clock and global clock Buffer which spread across entire FPGA fabric. If we drive a clocking signal beyond that limit, it may cause some excessive clock skew aexists t the fabric which is driven physically far away from the MMCM. But we can still push the frequency to its limit as we carefully constraint the floorplan according to each subcircuit's operational frequency.

The Clocking Wizard tool is build to following Xilinx's datasheet strictly, thus it will have some error by only inputting the demanded output frequency which beyond datasheet allowed. The solution for that is manual setting the parameters in the MMCM by click "Allow override mode" in the Clocking Wizard tool. See the following figure.



For example, we have a onboard oscillator working on 200MHz and we want the MMCM outputting 1175MHz. According to our paper, it is not a integer overclocking so we need to setup a fractional divider, which is not present in MMCM. But it can be solve by setting the prescaler(D in the following figure) in MMCM.

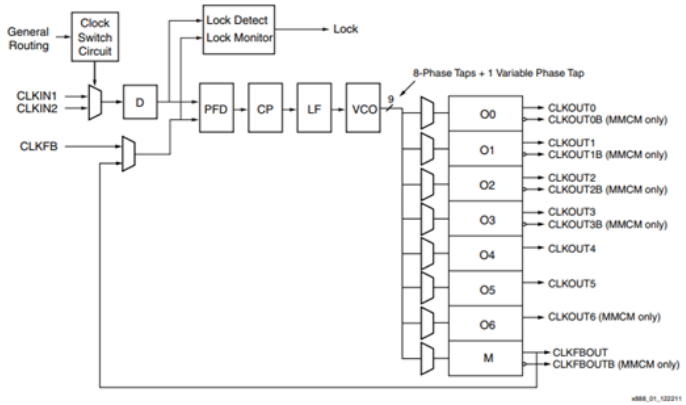


Figure 1: **MMCM and PLL Block Diagram**

Thus, by setting prescaler to 5, Feedback divider to 47 and Output0 scaler to 1, then we have 1175MHz frequency at CLKOUT0 same with our VCO frequency.

Attribute	Value
BANDWIDTH	OPTIMIZED
CLKFBOUT_MULT_F	47
CLKFBOUT_PHASE	0.000
CLKFBOUT_USE_FINE_PS	
CLKIN1_PERIOD	5.0
CLKIN2_PERIOD	10.0
CLKOUT4_CASCADE	
CLOCK_HOLD	
COMPENSATION	ZHOLD
DIVCLK_DIVIDE	8
REF_JITTER1	0.010
REF_JITTER2	0.010
STARTUP_WAIT	
CLKOUT0_DIVIDE_F	1
CLKOUT0_DUTY_CYCLE	0.500
CLKOUT0_PHASE	0.000
CLKOUT0_USE_FINE_PS	

Output Clock Summary

VCO Freq = 1175.000 MHz

Output Clock	Port Name	Output Freq (MHz)	Phase (degrees)	Duty Cycle (%)	Pk-to-Pk Jitter (ps)	Phase Error (ps)
CLK_OUT1	CLK_OUT1	1175.000	0.000	50.0	127.297	223.661

Hope this will help your implementation.

Best Wishes, Ray Wang

Appendix D

Plot Results

D.1 Kintex 7

D.1.1 Single Core

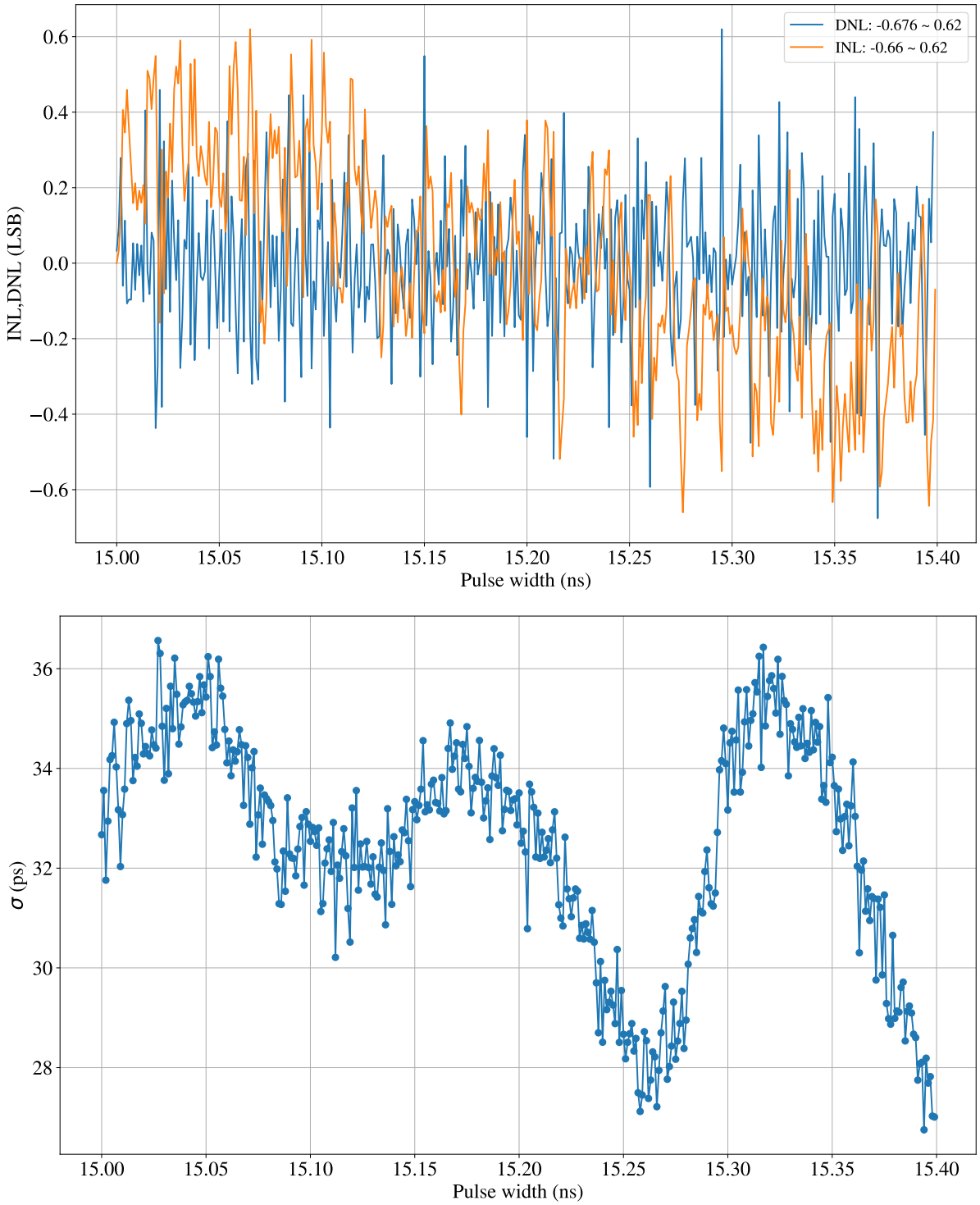


Figure D.1: Plot Kintex 7, single core, short range

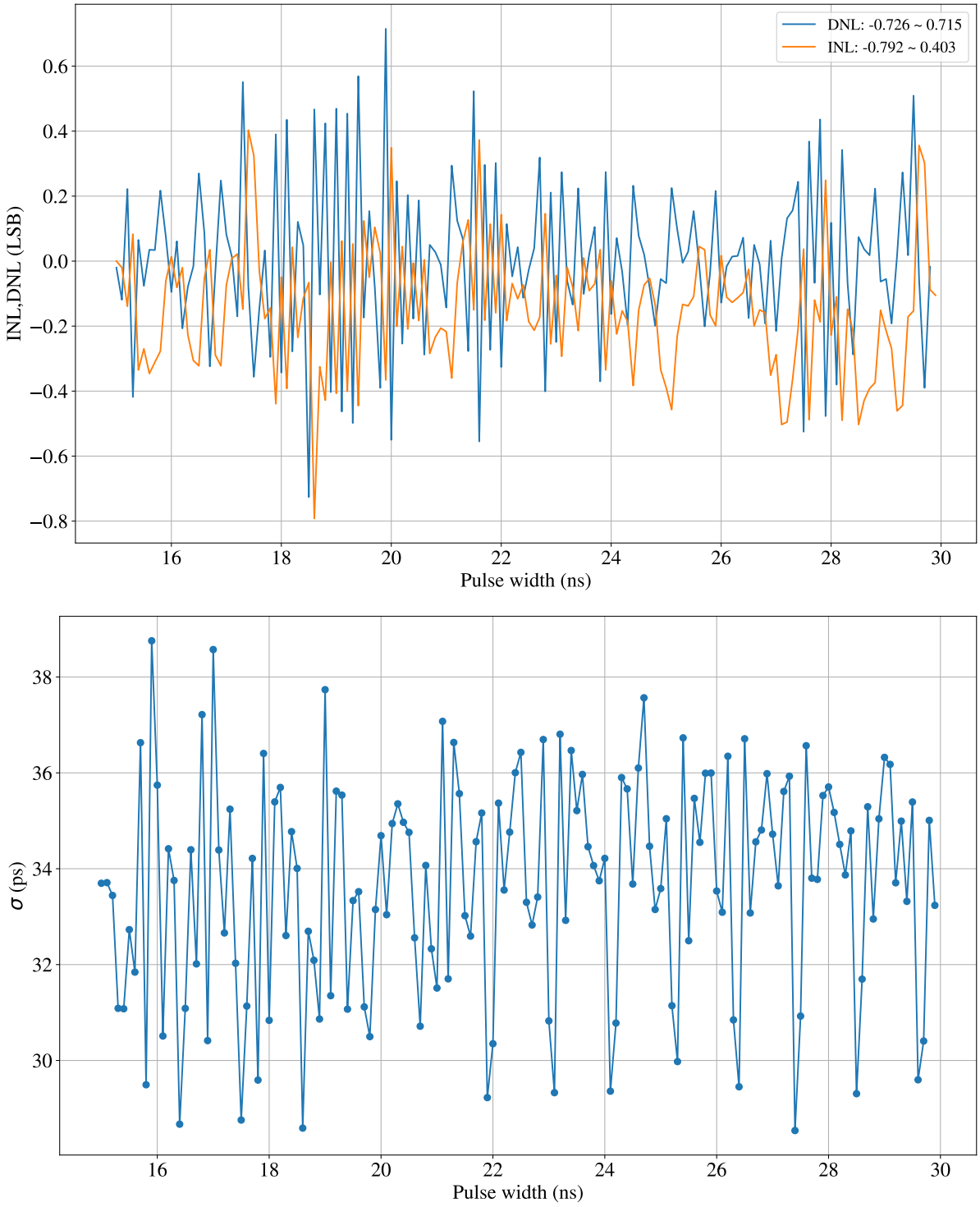


Figure D.2: Plot Kintex 7, single core, mid range

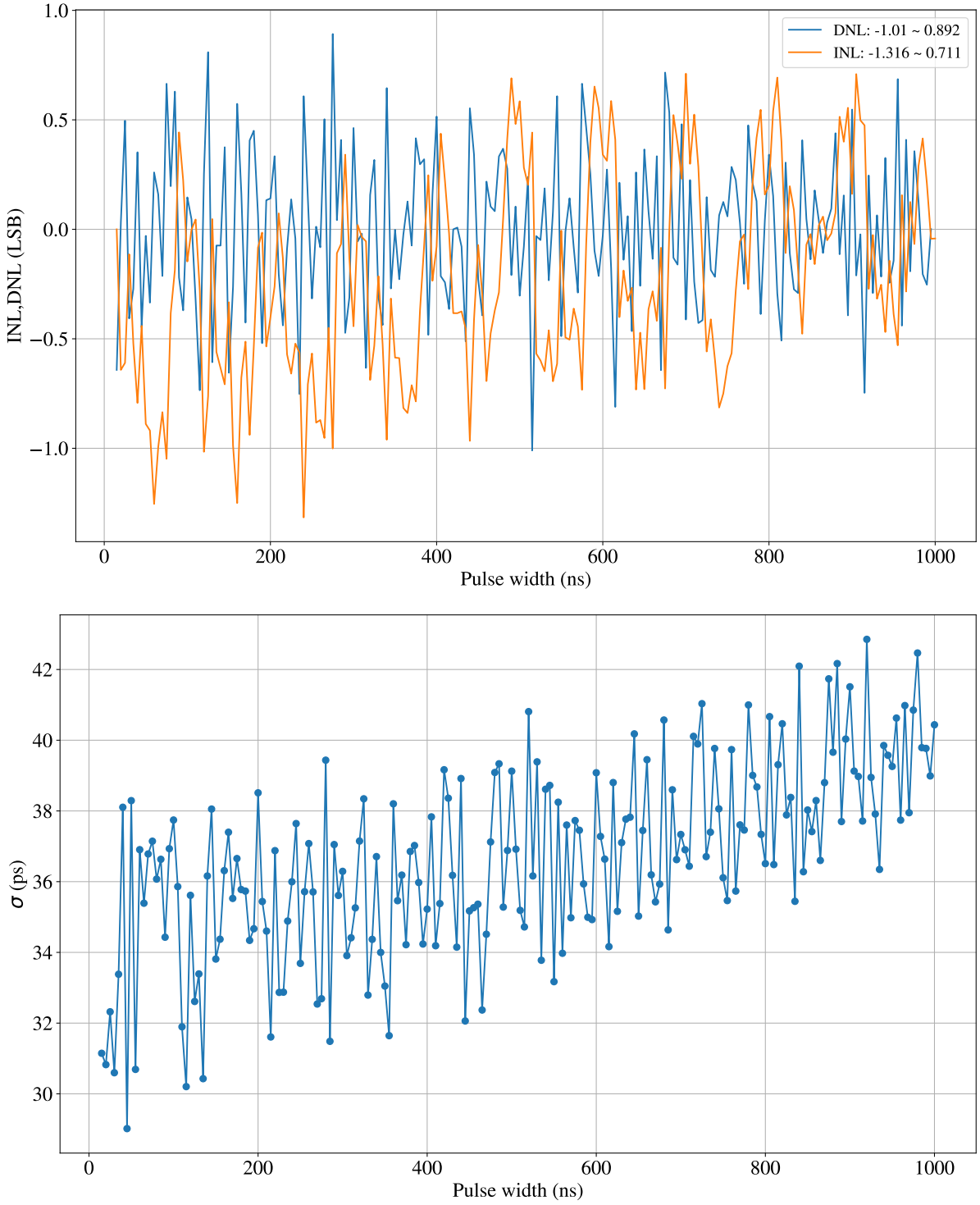


Figure D.3: Plot Kintex 7, single core, long range

D.1.2 Multi Core

D.1.2.1 8 Core

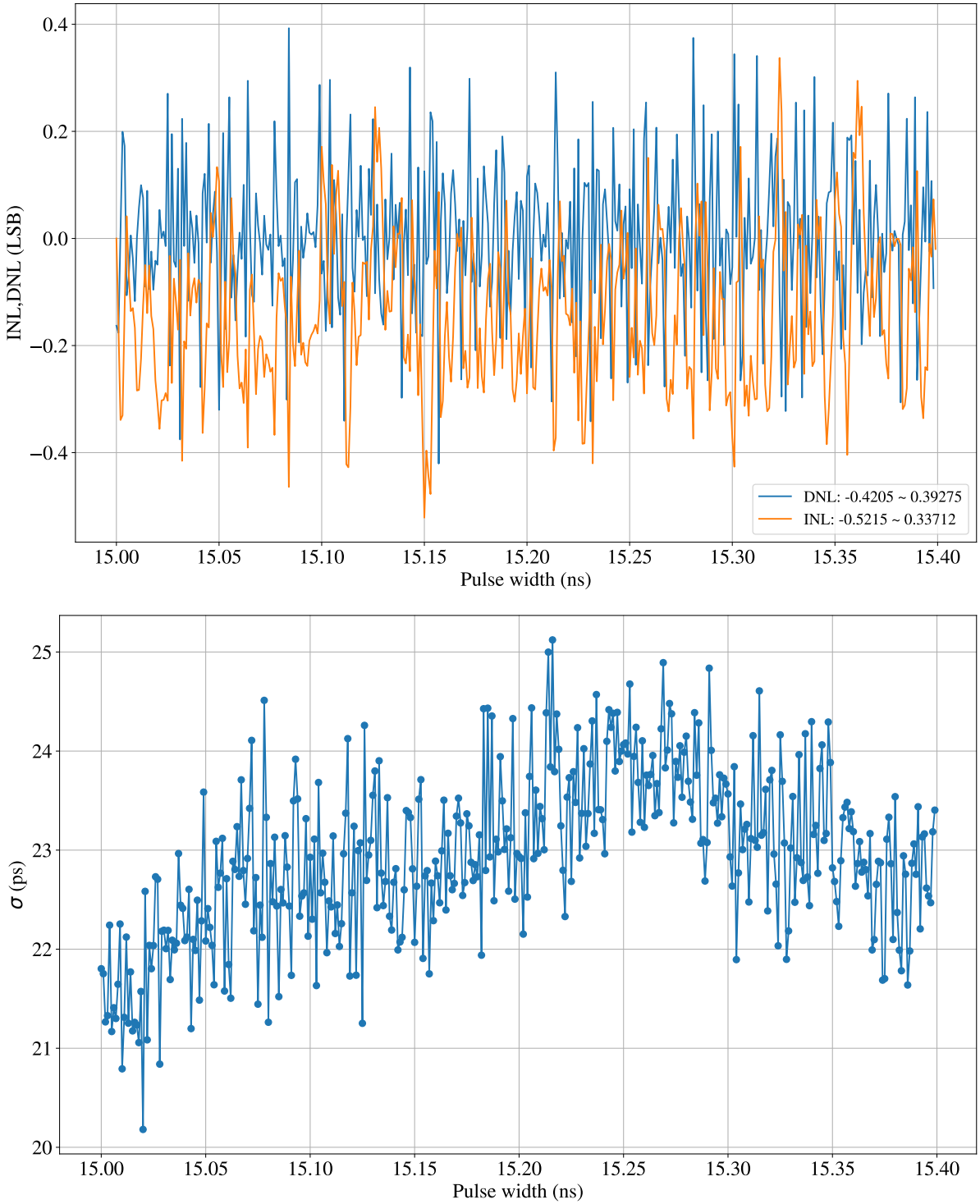


Figure D.4: Plot Kintex 7, 8 core, short range

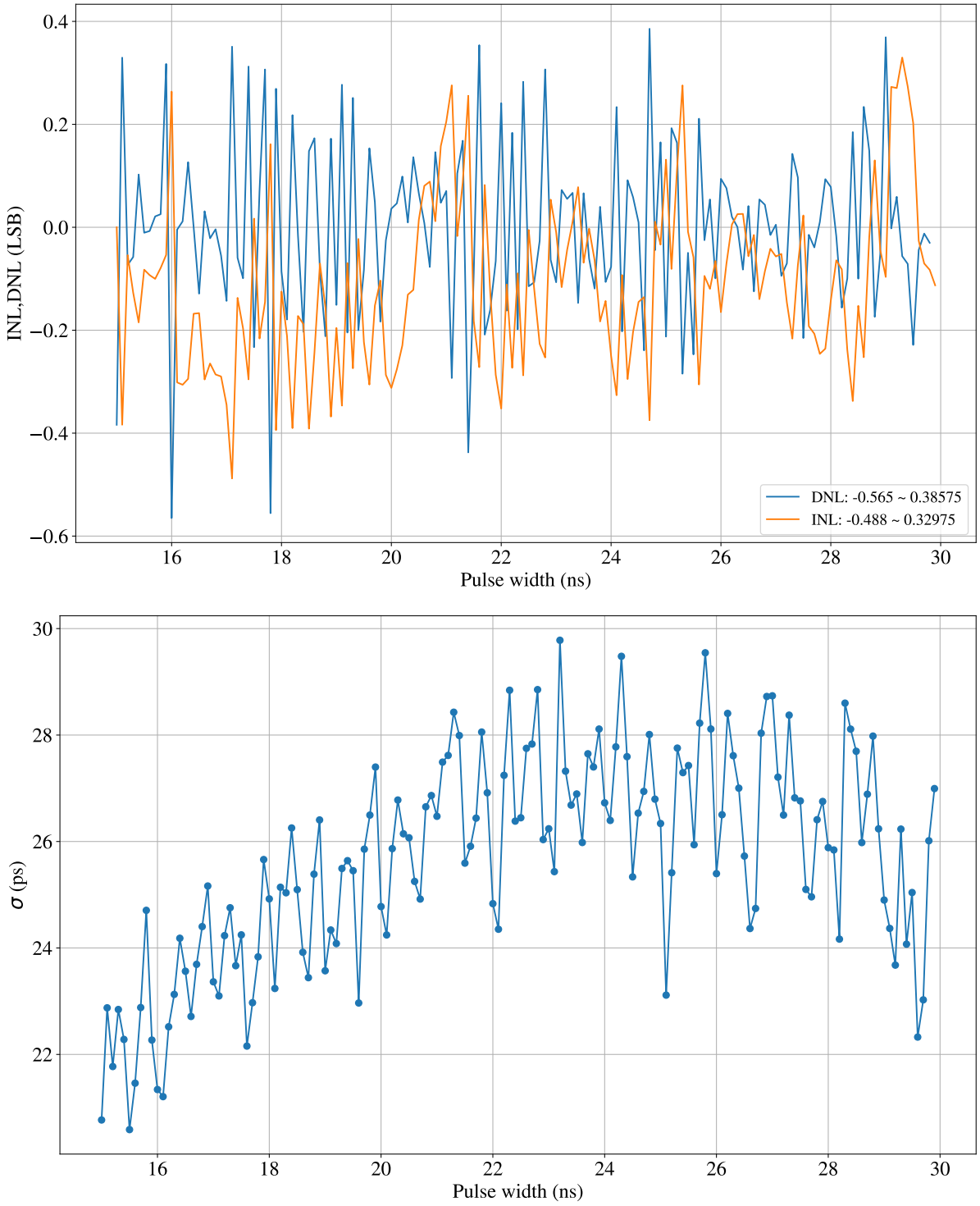


Figure D.5: Plot Kintex 7, 8 core, mid range

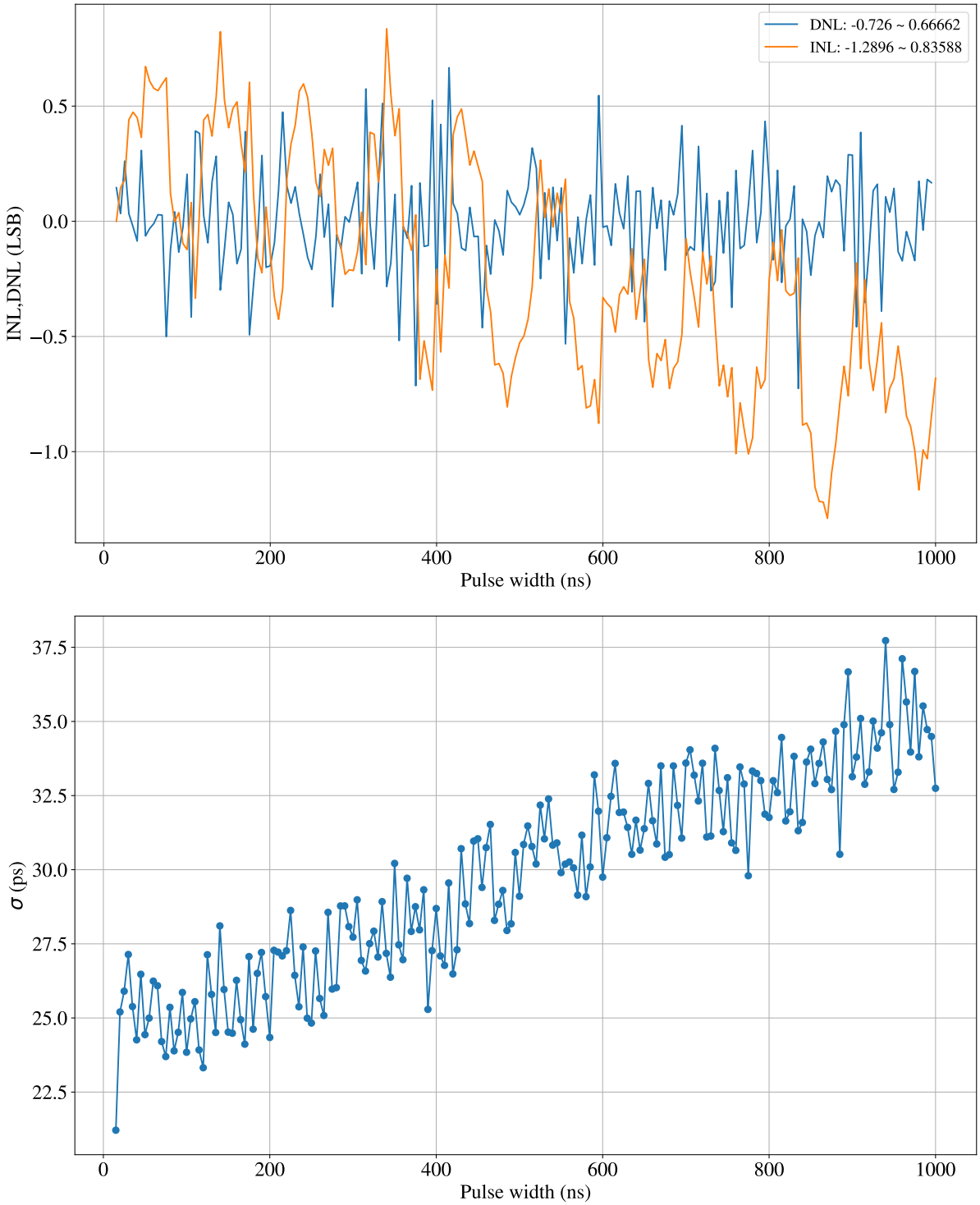


Figure D.6: Plot Kintex 7, 8 core, long range

D.1.2.2 16 Core

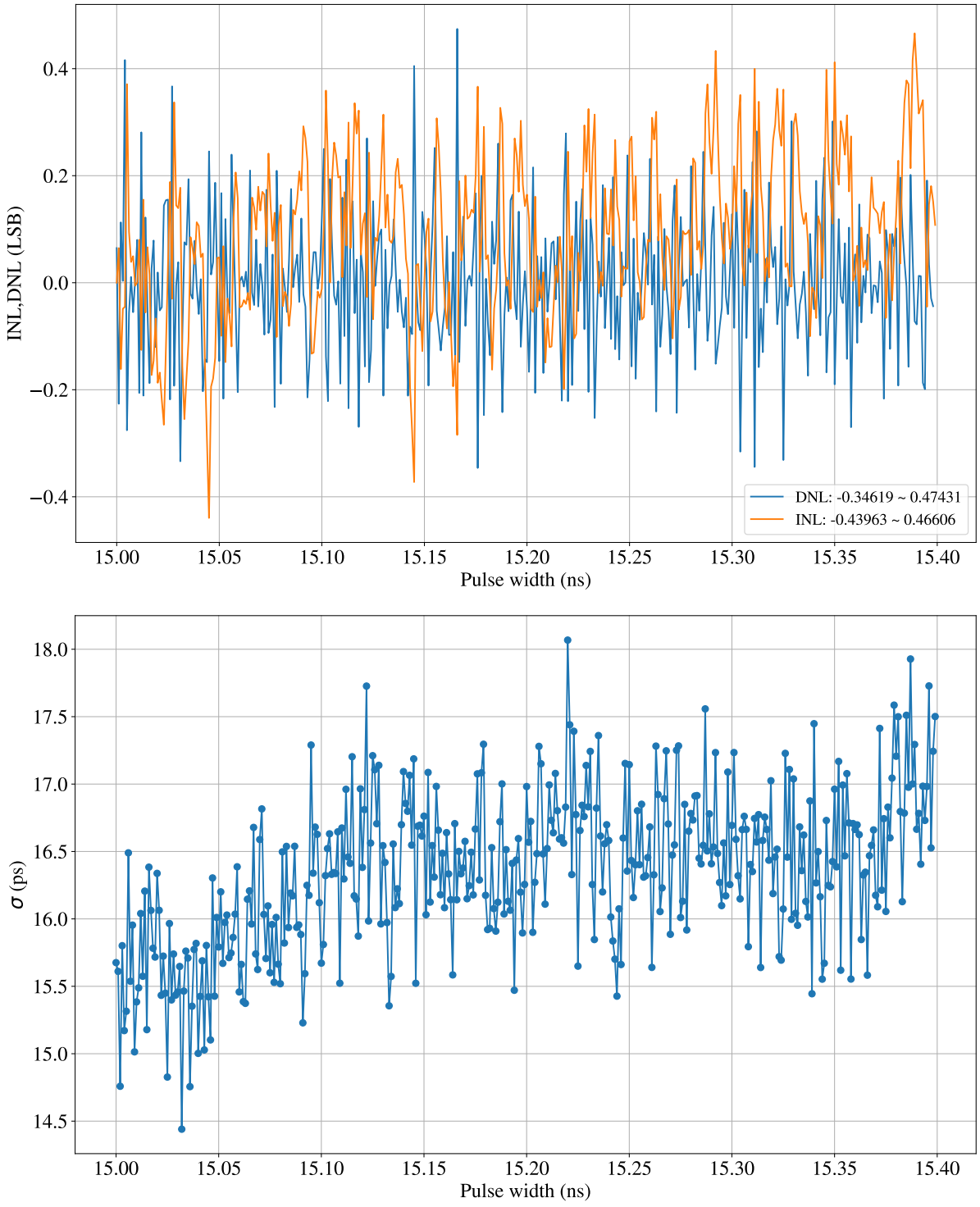


Figure D.7: Plot Kintex 7, 16 core, short range

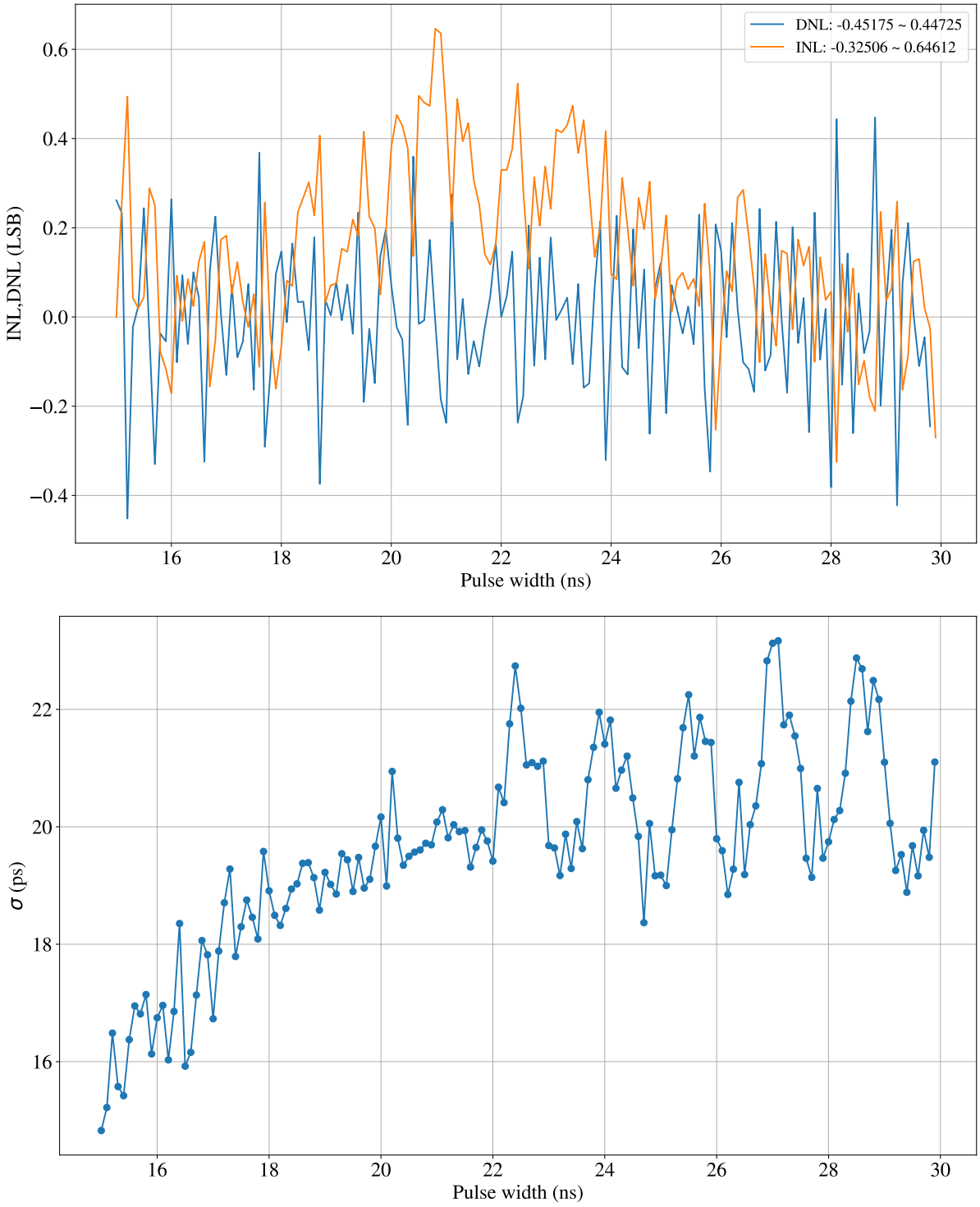


Figure D.8: Plot Kintex 7, 16 core, mid range

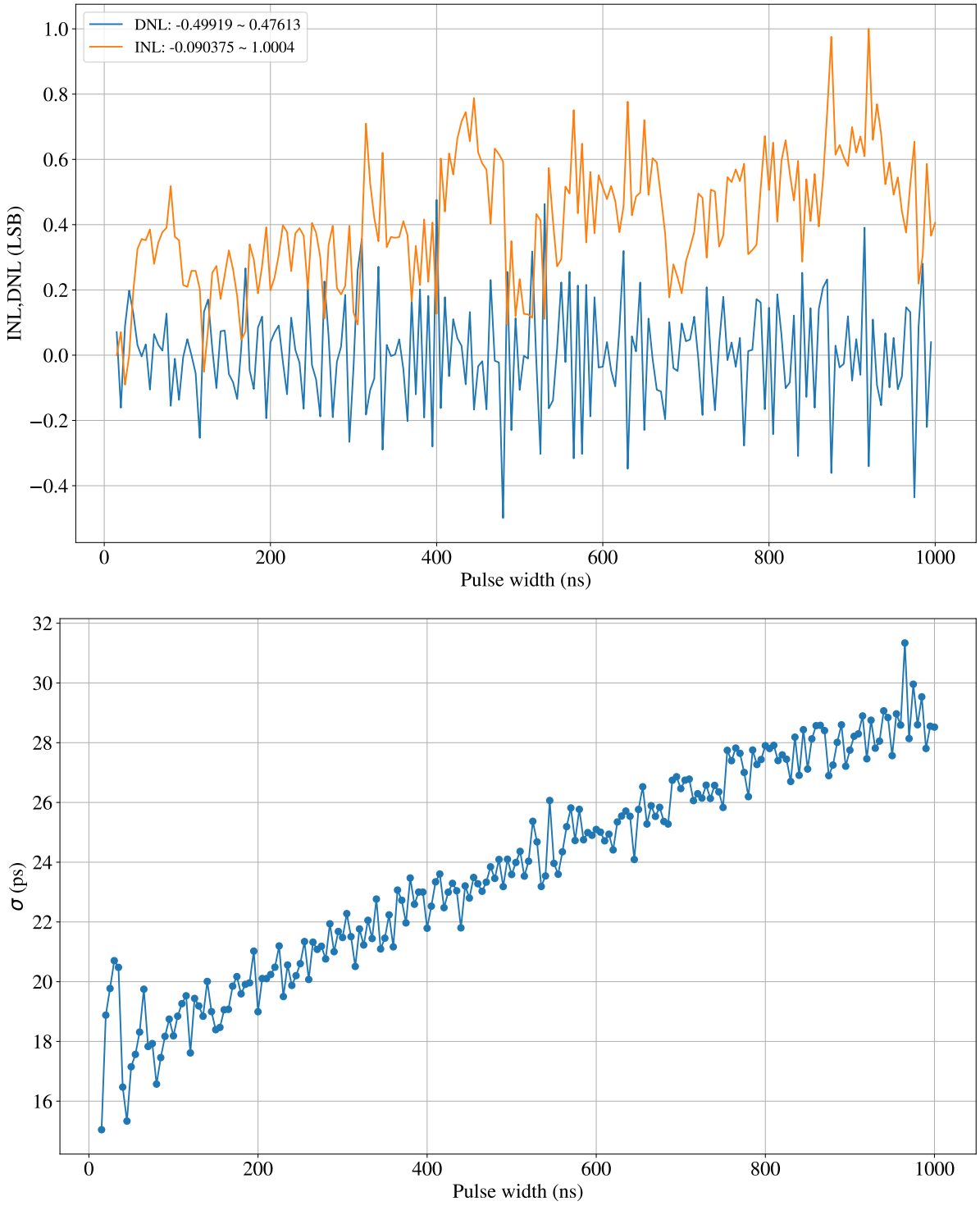


Figure D.9: Plot Kintex 7, 16 core, long range

D.2 Zynq UltraScale+

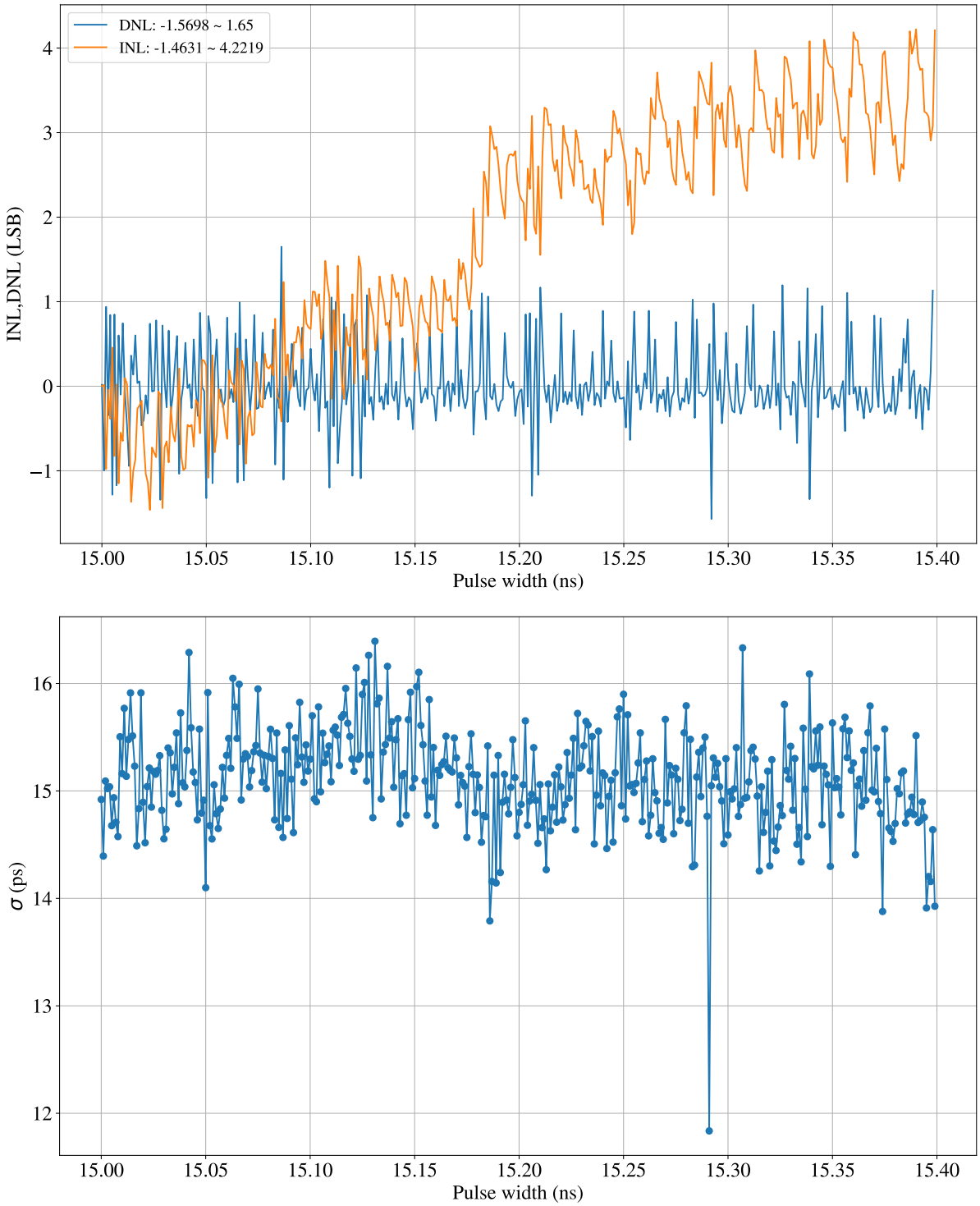


Figure D.10: Plot Ultrascale+, short range

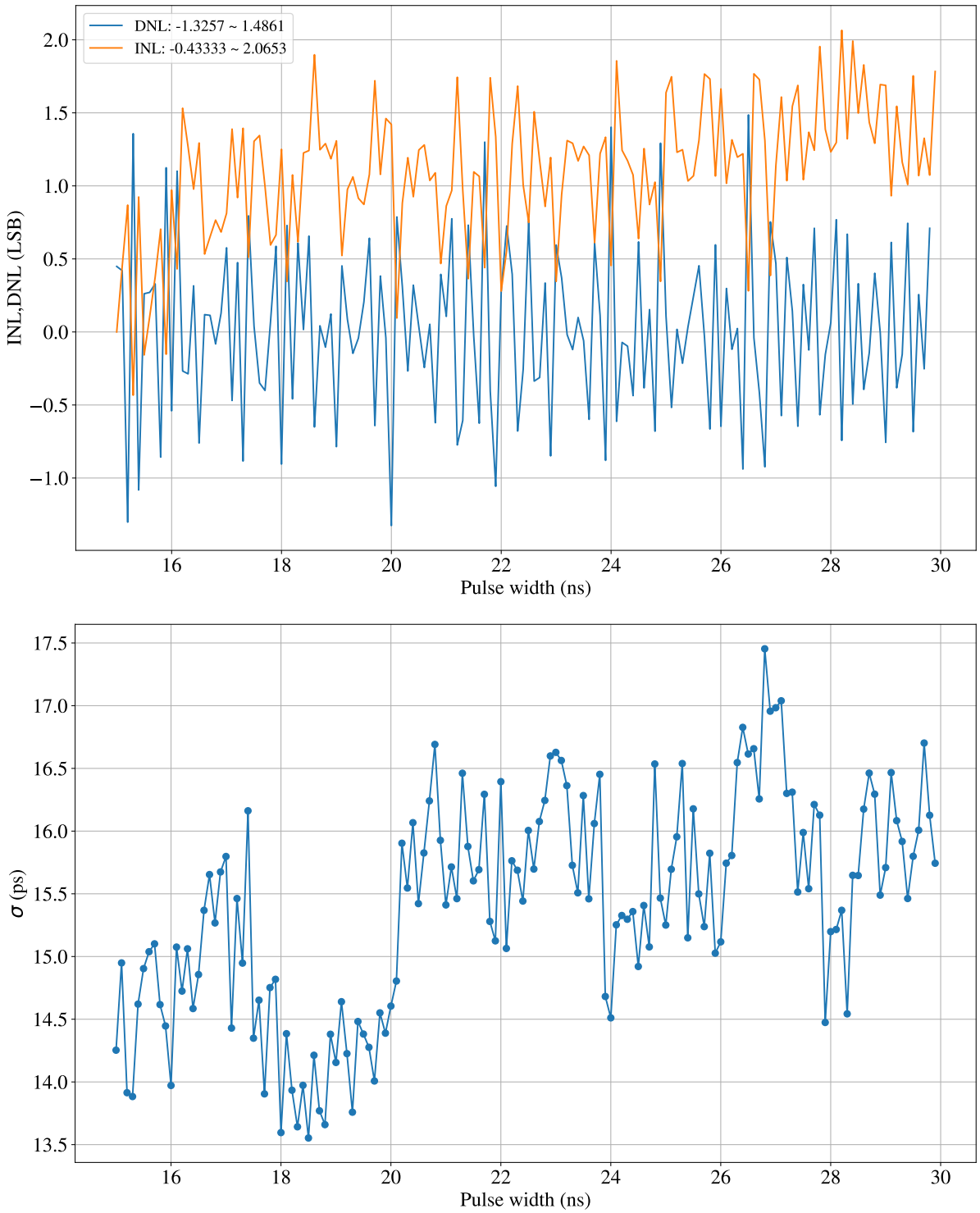


Figure D.11: Plot Ultrascale+, mid range

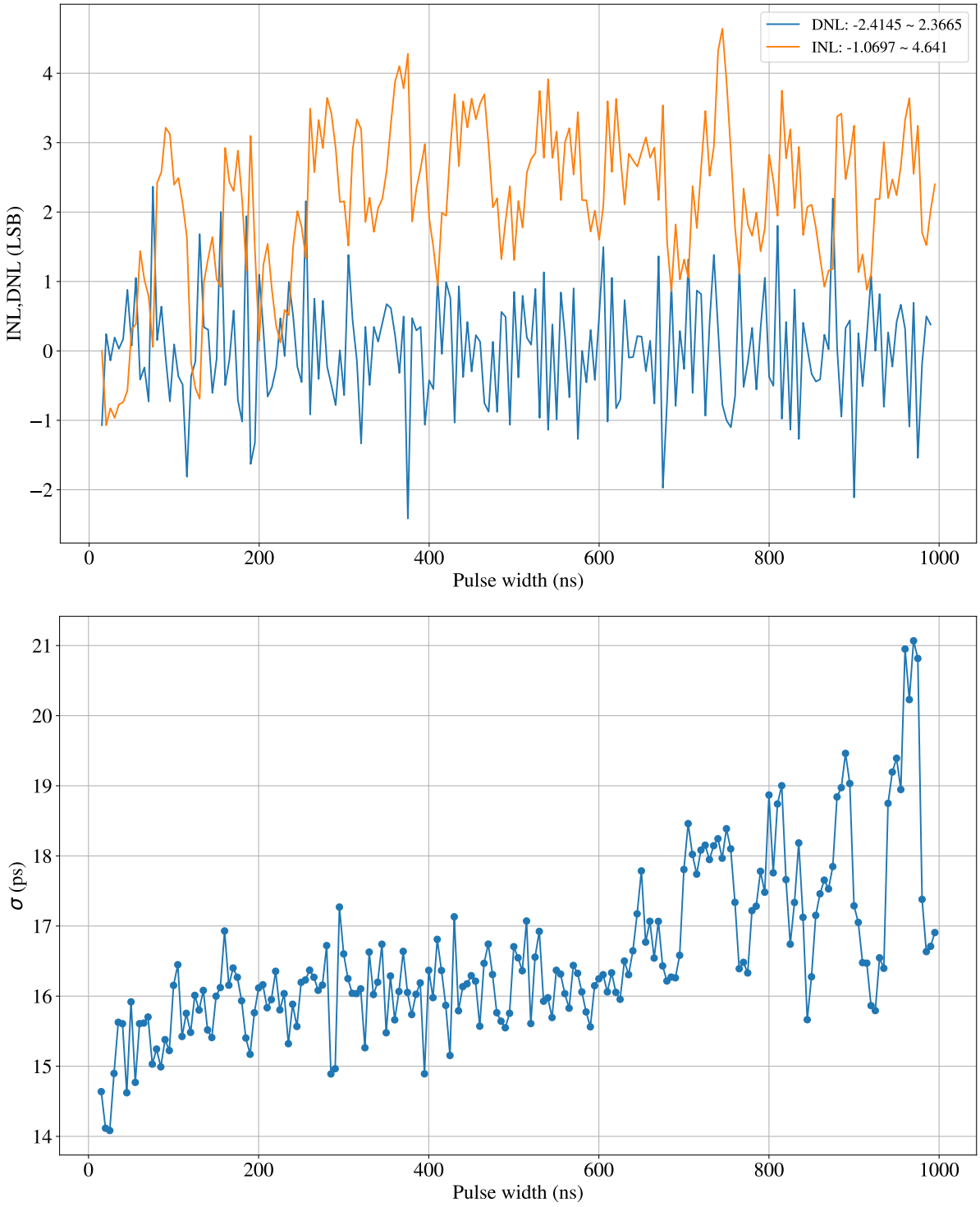


Figure D.12: Plot Ultrascale+, long range

D.3 Nexys 4

D.3.1 Single Core

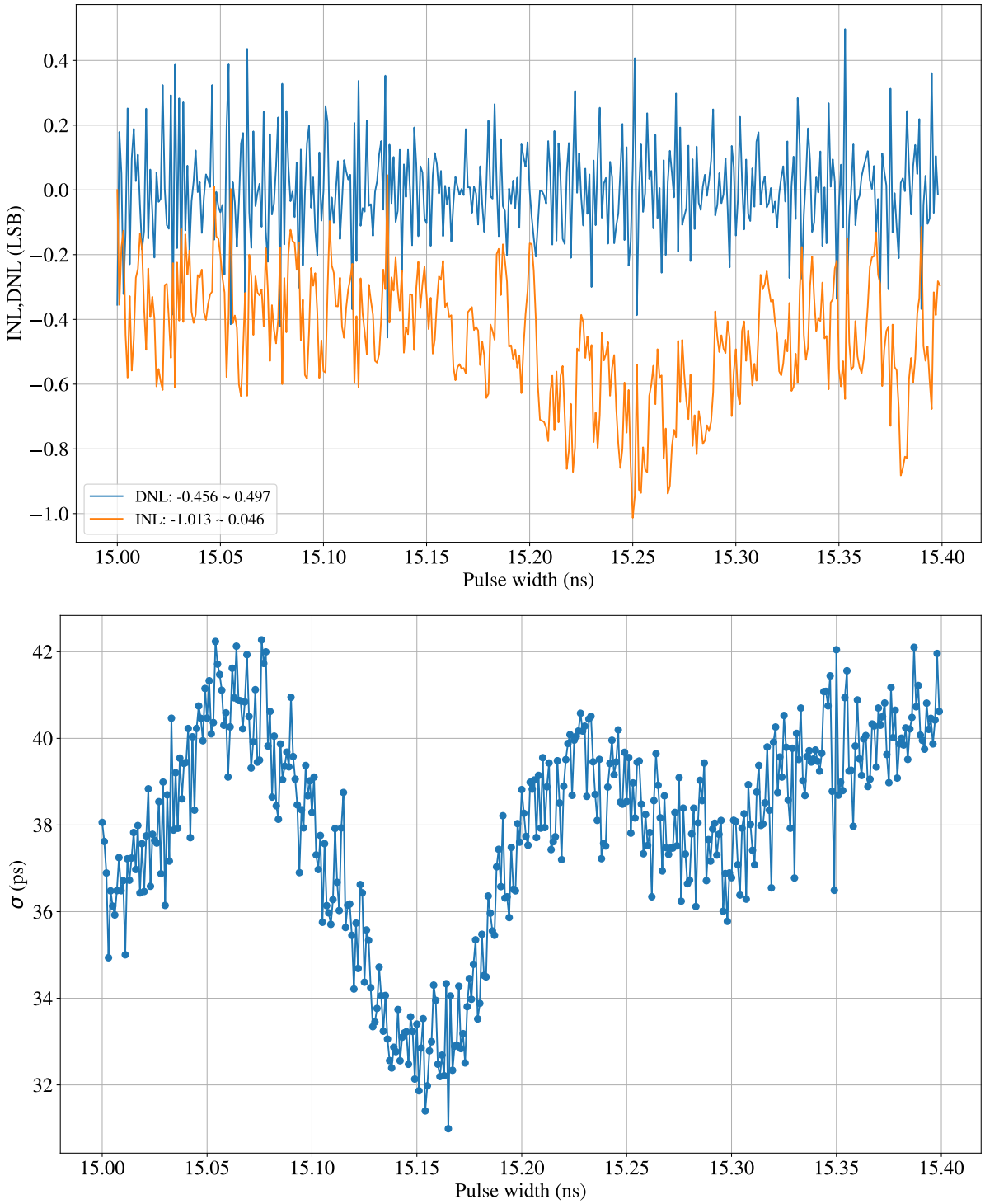


Figure D.13: Plot Nexys 4, single core, short range

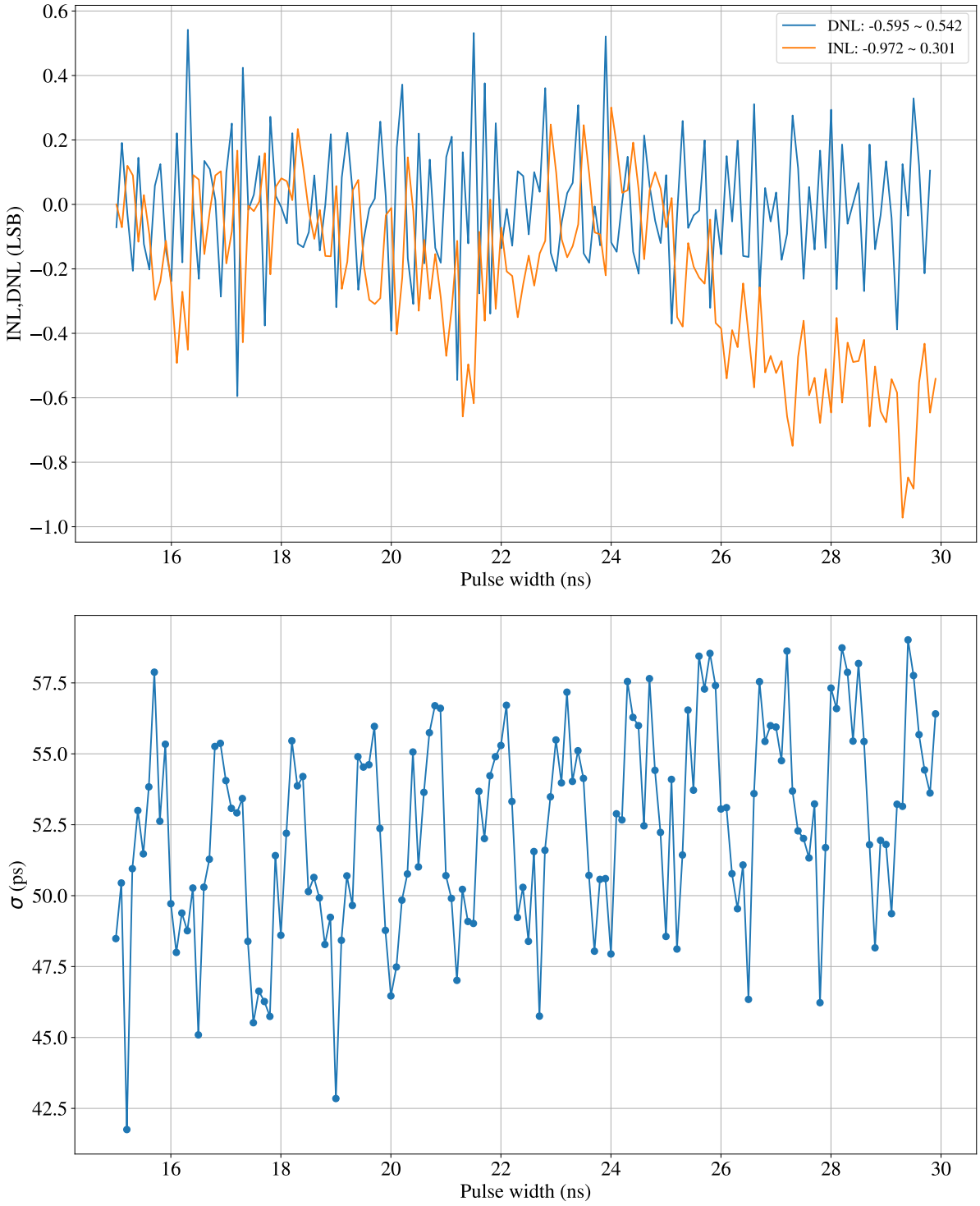


Figure D.14: Plot Nexys 4, single core, mid range

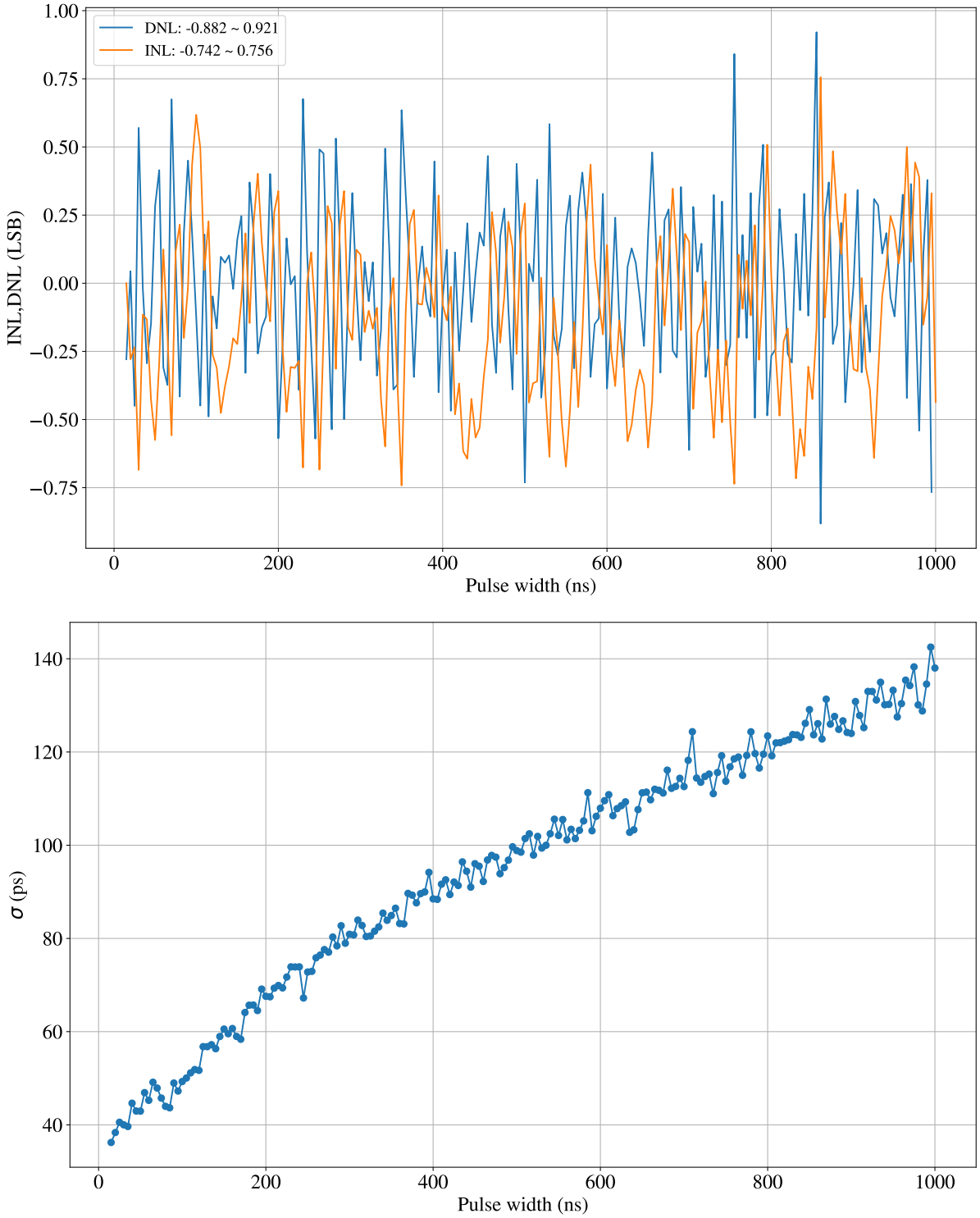


Figure D.15: Plot Nexys 4, single core, long range

D.3.2 Multi Core

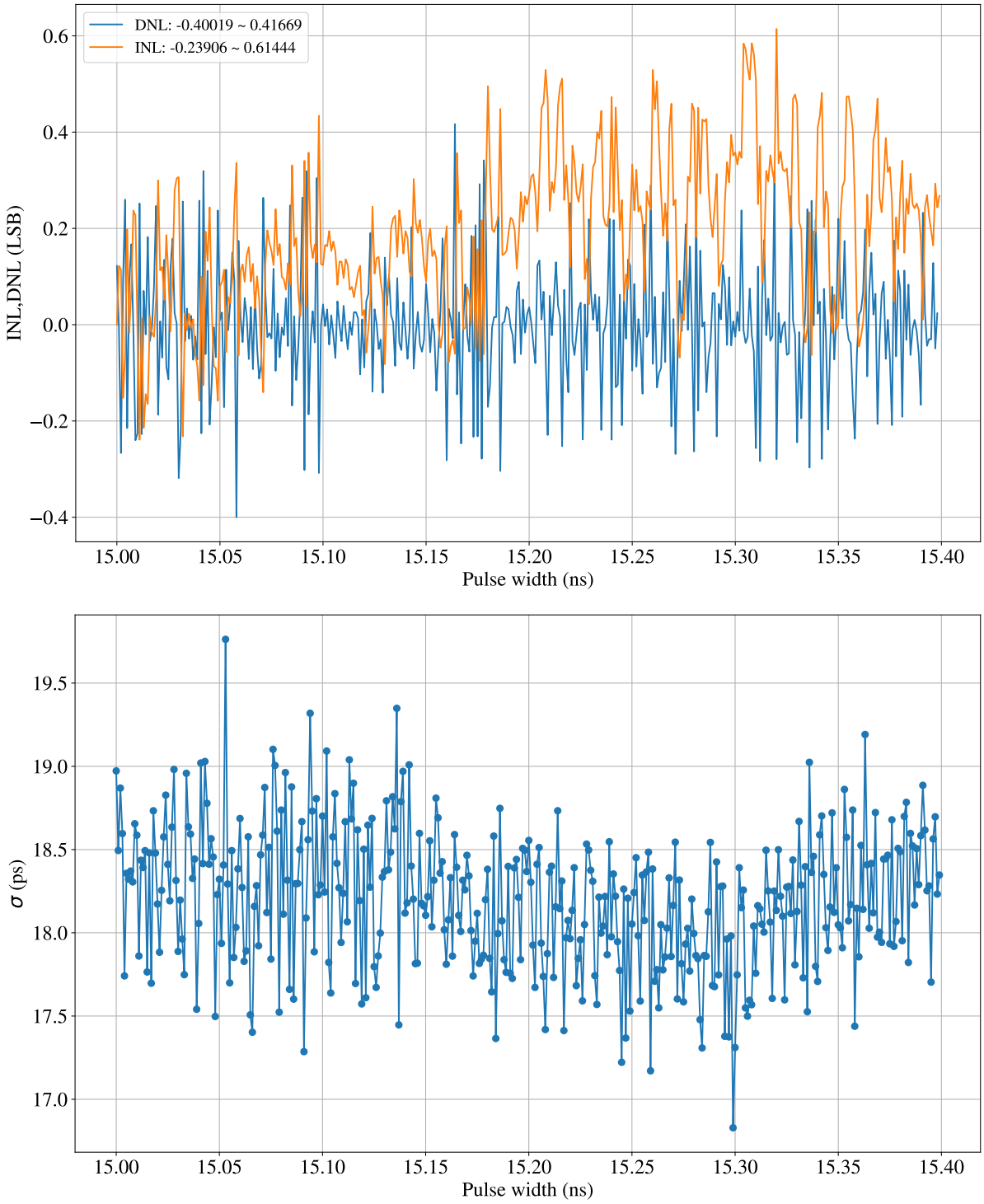


Figure D.16: Plot Nexys 4, 16 core, short range

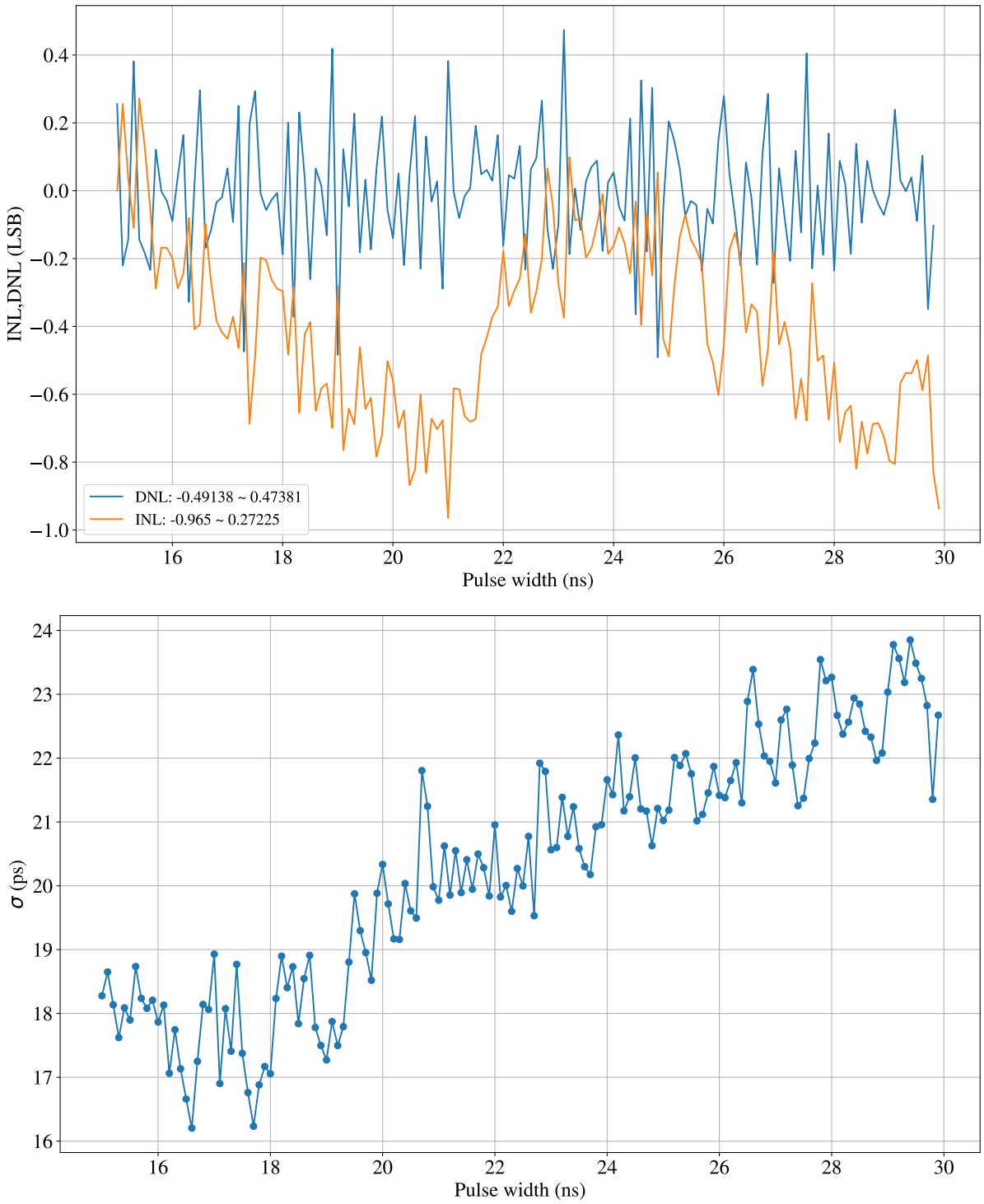


Figure D.17: Plot Nexys 4, 16 core, mid range

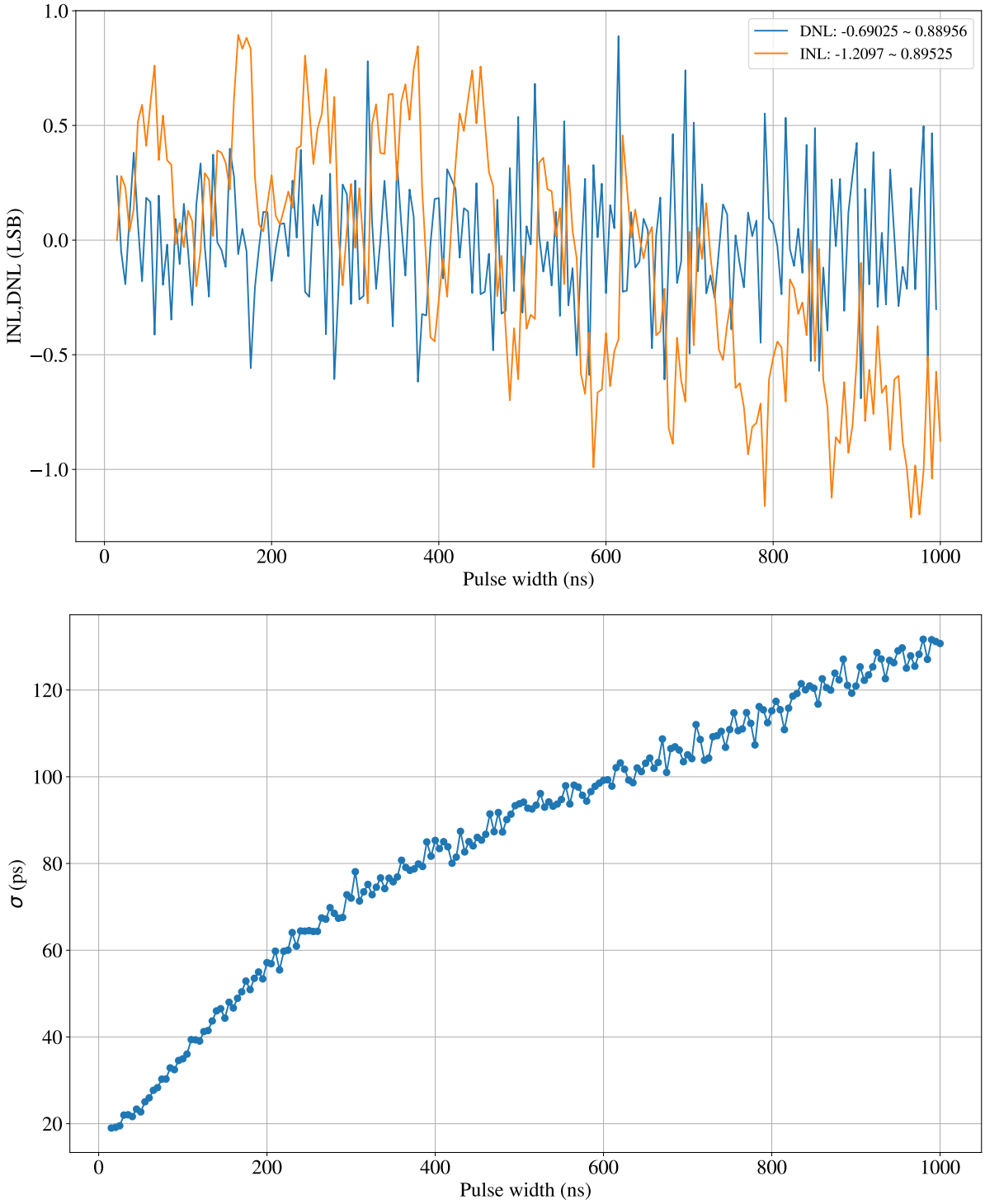


Figure D.18: Plot Nexys 4, 16 core, long range

List of Figures

List of Tables