Universidade de Aveiro
2021

ALEXANDRE
DAVID
BRANDÃO

Front-ends para LiDAR baseados em ADC e TDC

ADC and TDC-based front-ends for LiDAR

Universidade de Aveiro
2021

ALEXANDRE
DAVID
BRANDÃO

Front-ends para LiDAR baseados em ADC e TDC

ADC and TDC-based front-ends for LiDAR

"There is always a way if one does not avoid hard work."

— Demosthenes

Universidade de Aveiro Departamento de Eletrónica, Telecomunicações e Informática
2021

ALEXANDRE DAVID BRANDÃO

Front-ends para LiDAR baseados em ADC e TDC

ADC and TDC-based front-ends for LiDAR

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica do Doutor Arnaldo Silva Rodrigues de Oliveira, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Miguel Vidal Drummond, Investigador Auxiliar do Instituto de Telecomunicações .

agradecimentos / acknowledgements

Um grande obrigado,
Aos meus pais, por apoiarem-me e sempre ter confiado em mim,
Aos meus avós, por apoiarem-me e sempre ter confiado em mim,
Ao meus irmãos, por serem os meus melhores amigos e um especial agradecimento ao meu irmão por tudo e principalmente a sua disponibilidade para ajudar,
Ao Reis, Celso, Nuno, Bernardo, Josué, Pedro, Torres, Beatriz, Rios e ao Vala, por tantos anos de amizade e experiências.

Um especial agradecimento aos meus orientadores Miguel Drummond e Arnaldo Silva pela confiança, autonomia para explorar o problema, capacidade de motivar e capacidade de ensinar.

Agradeço ao Daniel Bastos pela paciência e apoio prestado durante todo o desenvolvimento do projeto.

Agradeço ao Eng.º Pedro Barbosa da Bosch por toda a ajuda prestada, e por ter gentilmente cedido uma primeira arquitetura de um TDC com base numa TDL sobre a qual a parte de TDCs desta tese foi elaborada.

Agradeço ao projeto SOFTLI, financiado pela Bosch Car Multi-media no âmbito do projeto Sensible Car (POCI010247FEDER037902). Este trabalho é financiado em parte pelo Fundo Europeu de Desenvolvimento Regional (FEDER), através do Programa Operacional Competitividade e Internacionalização, Programa Operacional Regional de Lisboa, Programa Operacional Regional do Algarve, na sua componente FEDER, e pela Fundação para a Ciência e Tecnologia, Projeto RETIOT, POCI-01-0145-FEDER-016432.

Agradeço ao Instituto de Telecomunicações por ter prestado os seus equipamentos.

Por último, agradeço à Universidade de Aveiro, por me acolher, fazer-me crescer e ajudar-me ao longos destes árduos 5 anos.

Palavras Chave

Conversores tempo para digital, Conversores analógicos para digitais, LiDAR, veículos autônomos

Resumo

Os veículos autónomos são uma tecnologia promissora para salvar mais de um milhão de vidas por ano, colhidas por acidentes rodoviários. Contudo, colocar veículos autónomos seguros no mercado requer inúmeros desenvolvimentos, a começar por sensores de visão. O LiDAR é um sensor de visão fundamental para veículos autónomos, pois permite uma visão 3D de alta resolução. Contudo, o LiDAR automotivo não é uma tecnologia madura, e portanto requer também desenvolvimento em vários aspectos.

Esta dissertação visa contribuir para a maturidade do LiDAR, com foco em arquiteturas de amostragem para front-ends de LiDAR. Foram desenvolvidas duas arquiteturas. A primeira assenta numa ADC pipelined, por sua vez implementada numa placa de teste AD-FMCDAQ2-EBZ. A ADC opera em sincronismo com o pulso emitido, e permite capturar amostras a 1 Gsample/s. A segunda arquitetura assenta num TDC implementado diretamente numa FPGA. O TDC baseia-se numa topologia tapped delay line com 45 linhas de atraso, e num descodificador à base de multiplexers, permitindo uma resolução temporal de 50 ps.

Resultados preliminares mostram que ambas as implementações operam corretamente, e são adequadas para amostrar pulsos curtos tipicamente associados a LiDAR. Em termos comparativos, a arquitectura com base numa ADC tem um consumo de potência considerável e requer uma quantidade significativa de recursos da FPGA. Contudo, esta permite amostrar a forma de onda de LiDAR sem nenhuma perda de informação, permitindo assim alcance e precisão máximos. A arquitectura com base num TDC é exatamente o oposto: tem um baixo consumo de potência e requer poucos recursos da FPGA. Contudo, permite capturar apenas uma amostra por pulso.

Keywords

Time-to-Digital Converters, Analog-to-Digital Converters, LiDAR, e autonomous vehicles.

Abstract

Autonomous vehicles are a promising technology to save over a million lives each year that are lost in road accidents. However, bringing safe autonomous vehicles to market requires massive development, starting with vision sensors. LiDAR is a fundamental vision sensor for autonomous vehicles, as it enables high resolution 3D vision. However, automotive LiDAR is not yet a mature technology, and, also requires massive development in many aspects.

This thesis aims to contribute to the maturity of LiDAR, focusing on sampling architectures for LiDAR front-ends. Two architectures were developed. The first is based on a pipelined ADC, available from an AD-FMCDAQ2-EBZ board. The ADC is synchronized with the emitted pulse and able to sample at 1 Gsample/s. The second architecture is based on a TDC that is directly implemented in an FPGA. It relies on a tapped delay line topology comprising 45 delay elements and on a mux-based decoder, resulting in a resolution of 50 ps.

Preliminary test results show that both implementations operate correctly, and are both suitable for sampling short pulses typically used by LiDARs. When comparing both architectures, we conclude that an ADC consumes a significant amount of power, and uses many FPGA resources. However, it samples the LiDAR waveform without any loss of information, therefore enabling maximum range and precision. The TDC is just the opposite: it consumes little power, and uses less FPGA resources. However, it only captures one sample per pulse.

Contents

List of Figures

List of Tables

Glossary

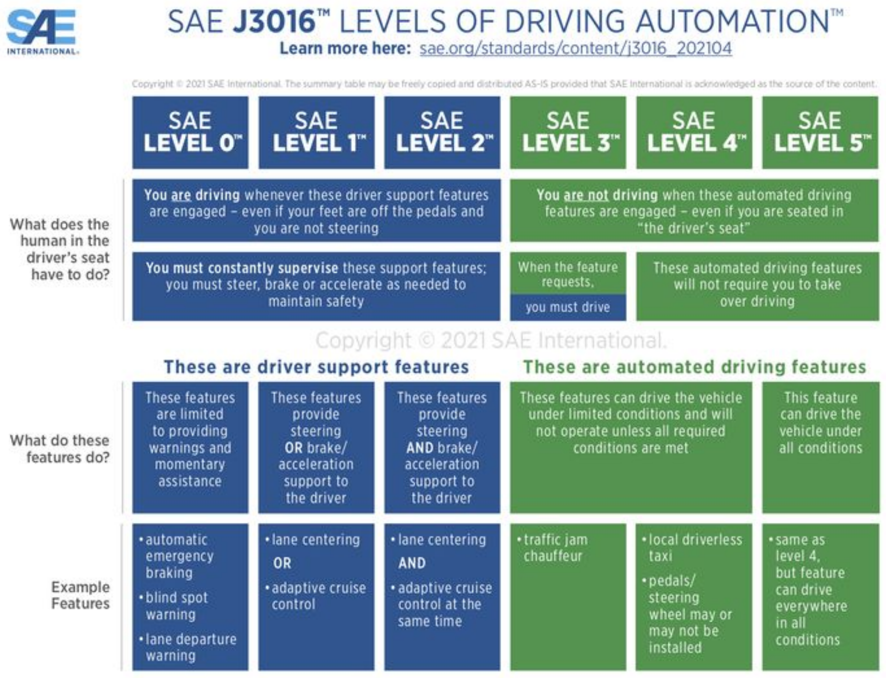
AAF	Anti-Aliasing Filtering
ADC	Analog-to-Digital Converter
AXI	Advanced eXtensible Interface
ARM	Advanced RISC Machine
AWG	Arbitrary Waveform Generator
AV	Autonomous Vehicle
BRAM	Block Random Access Memory
RCA	Ripple Carry Adder
CLB	Configurable Logic Block
CSV	Comma Separated Value
DAC	Digital-to-Analog Converter
DAQ	Data Acquisition (System)
DDR	Double Data Rate
DLL	Delay Locked Loop
DMA	Direct Memory Access
DOT	Department of Transportation
DDS	Direct Digital Synthesis
ENOB	Effective Number of Bits
ETH	Ethernet
FSM	Finite State Machine
FPGA	Field Programmable Gate Array
FIFO	First In First Out
FMC	FPGA Mezzanine Card
GRO	Gated Ring Oscillator
HDL	Hardware Descriptive Language
HPC	High Pin Count
ILA	Integrated Logic Analyzer
IP	Intellectual Property
I2C	Inter-Integrated Circuit
JTAG	Joint Test Action Group
LPC	Low Pin Count
LiDAR	Light Detection And Ranging
LSB	Least Significant Bit
LUT	Look-Up Table
MCMM	Mixed Mode Clock Manager
MSB	Most Significant Bit
OS	Operating System
PLL	Phase Locked Loop
PL	Programmable Logic
PRBS	Pseudorandom Binary Sequence
PVT	Process-Voltage-Temperature
RF	Radio Frequency
ROM	Read Only Memory
RTL	Register-Transfer Level
SAR	Successive-Approximation-Register
SONAR	Sound Navigation and Ranging
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SoC	System on Chip
SPST	Single Pole Single Throw
TDI	Test Data in
TDO	Test Data out
TDL	Tapped Delay Line
TDC	Time-to-Digital Converter
ToF	Time of Flight
TCL	Tool Command Language
UART	Universal Asynchronous Receiver Transmitter
VLSI	Very Large-Scale Integration
VDL	Vernier Delay Line
VCO	Voltage Controlled Oscillator
VHDL	Very High Speed Hardware Description Language

CHAPTER 1

Introduction

Each year approximately 1.35 million people die as a result of traffic accidents. According to the US Department of Transportation's National Highway Traffic Safety Administration, 94% of the severe traffic accidents happen because of human errors [1]. The Department of Transportation (DOT) believes that fully Autonomous Vehicles (AVs) have the potential to significantly reduce traffic fatalities by up to 90 percent, as it removes human error [2]. This means that a AVs may be able to save more than 1 million lives every year.

Although, the prospect of fully AV seems very near, it still seems paradoxically impossibly futuristic as safe AVs are extremely complex to implement. This can be easily observed by looking to the roadmap towards complete autonomy, shown in figure 1.1.


SAE J3016™ LEVELS OF DRIVING AUTOMATION™

Learn more here: sae.org/standards/content/j3016_202104

	SAE LEVEL 0™	SAE LEVEL 1™	SAE LEVEL 2™	SAE LEVEL 3™	SAE LEVEL 4™	SAE LEVEL 5™
What does the human in the driver's seat have to do?	**You are driving** whenever these driver support features are engaged – even if your feet are off the pedals and you are not steering			**You are not driving** when these automated driving features are engaged – even if you are seated in "the driver's seat"		
	You must constantly supervise these support features; you must steer, brake or accelerate as needed to maintain safety			When the feature requests, you must drive	These automated driving features will not require you to take over driving	
	These are driver support features			**These are automated driving features**		
What do these features do?	These features are limited to providing warnings and momentary assistance	These features provide steering **OR** brake/ acceleration support to the driver	These features provide steering **AND** brake/ acceleration support to the driver	These features can drive the vehicle under limited conditions and will not operate unless all required conditions are met		This feature can drive the vehicle under all conditions
Example Features	• automatic emergency braking • blind spot warning • lane departure warning	• lane centering **OR** • adaptive cruise control	• lane centering **AND** • adaptive cruise control at the same time	• traffic jam chauffeur	• local driverless taxi • pedals/ steering wheel may or may not be installed	• same as level 4, but feature can drive everywhere in all conditions

Figure 1.1: The 6 levels of a vehicles autonomy ranging from fully manual to the vehicles full autonomy reproduced from [3], [4].

1.1 Scope

Levelling up the autonomy of a vehicle requires safe decision-making algorithms, which in turn require detailed information about the vehicle surroundings. This is why vision sensors are a fundamental part of an AV. Some key vision sensors are observed in figure 1.2.

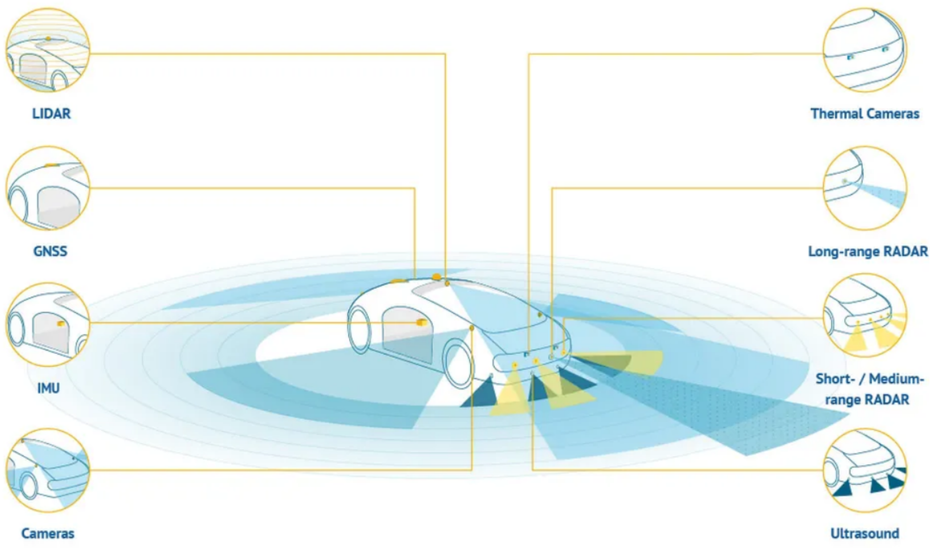
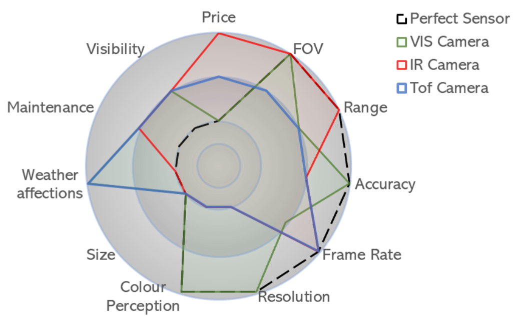


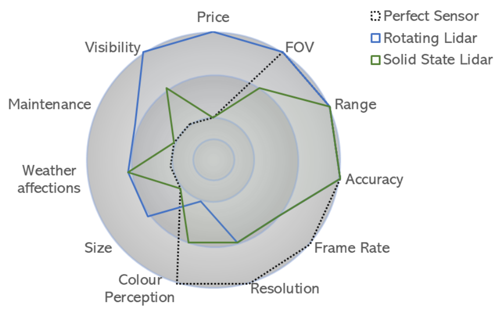
Figure 1.2: Some key sensors of an autonomous vehicle reproduced from [4].

Just like human senses, sensors must be strategically positioned in an AV to continuously feed information on the car's surroundings [5]. However, there is a limit to where sensors can be placed and there is no sensor that can fit all the requirements. Consequently, multiple sensors are required to complement each-other and whose combination is highly effective in creating a safe and automated driving experience.

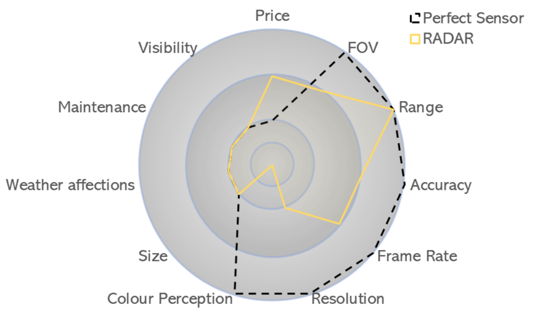
Figure 1.3 presents an overview of the various and most common sensors required for autonomous driving such as cameras, Sound Navigation and Ranging (SONAR), Light Detection And Ranging (LiDAR) and RADAR.



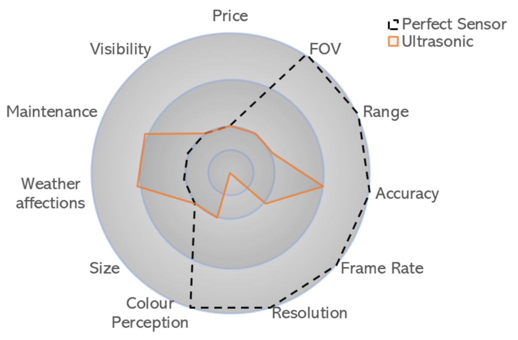
(a) Camera chart.



(b) LiDAR chart.



(c) RADAR chart.



(d) Ultrasonic chart.

Figure 1.3: Key sensors radar charts adapted from [6].

Cameras are the main sensor for 2D vision. However, cameras lack precision in providing 3D vision, a vital feature for safe driving. LiDAR has the unique capability of providing a detailed and accurate point cloud of the surroundings of the vehicle. The LiDAR sensors are now implemented in autonomous vehicles to be the eyes of the car, providing them 360 degrees view helping them change lanes, keeping a safe distance from other vehicles and pedestrian and spotting roadblocks and other obstructions [7]. However, contrary to all the others sensors, LiDAR still has a long way to go as the technology is still improving and has not yet matured. LiDAR is also far away from meeting the reasonable costs [8].

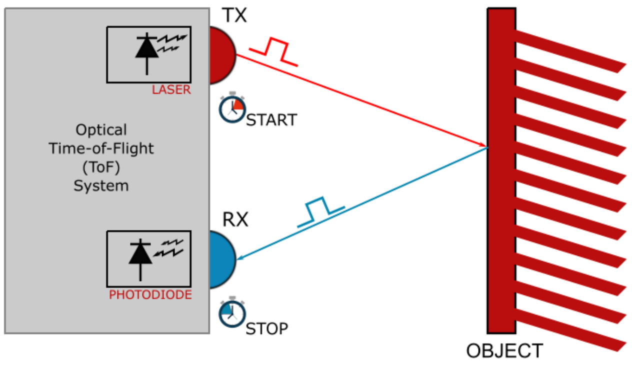


Figure 1.4: Illustration of a ToF measuring system adapted from [9].

The basic operation principe of a LiDAR, which is estimating the Time of Flight (ToF), is depicted in figure 1.4. The distance is given by (1.1).

$$d = \frac{ToF}{2c} = \frac{Stop - Start}{2c}, \tag{1.1}$$

where, d is the distance between the LiDAR and the object, c is the speed of light, *Start* is the time at which the LiDAR emits the pulse and *Stop* is the time at which the LiDAR receives the pulse.

According to (1.1), it is very important that the LiDAR is temporally precise as 1 cm takes only about 50 ps. As such, it is fundamental that the received signal is adequately sampled and digitally processed.

1.2 Motivation

The main motivation of this work is to devise suitable sampling architectures for LiDAR. Two architectures are studied, one based on an Analog-to-Digital Converter (ADC), and another based on a Time-to-Digital Converter (TDC).

On each side of the figure 1.5 is an example of an ADC and TDC based sampling methods. An ADC-based architecture is useful for a full-waveform LiDAR [10], and thus for achieving best possible range and precision. In contrast, a TDC-based architecture is not suitable for full-waveform nor optimal performance; however, it can be directly implemented in an Field Programmable Gate Array (FPGA) as an all-digital design.

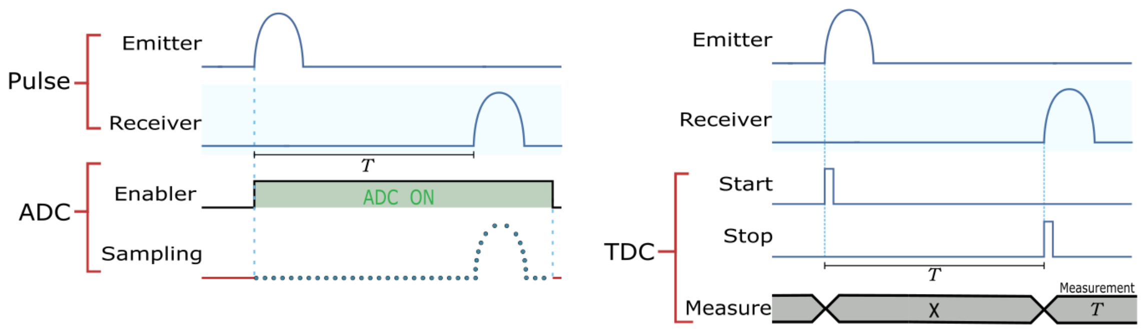


Figure 1.5: Left side: ADC-based sampling. Right side: A direct ToF estimation obtained from a TDC.

1.3 Objectives

The main objective of this dissertation is to study, implement and compare two different sampling methods suitable for LiDAR. This results in the following objetives.

1. Find out which types of ADCs are appropriate for a LiDAR.
2. Configure an ADC with a suitable architecture for operating according to the needs of a LiDAR.
3. Investigate whether a TDC can be implemented directly in an FPGA.
4. If point 3. is observed to be true, implement a TDC directly in an FPGA.
5. Compare both implementations (ADC vs. TDC) based on the obtained results.

1.4 Document Stucture

This document is divided in 7 chapters, including this introductory chapter:

Chapter 1 - *Introduction*: contextualization of the topic and scope of this document, the motivation for this research and what it attempts to clarify. Briefly describes how the document is organized and the contributions associated with this research.

Chapter 2 - *Fundamental ADC and TDC concepts*: Study and research on various state of the art implementations for ADC and TDC with the purpose of identifying the sampling architectures that best suit the application at hand.

Chapter 3 - *ADC - based front-end*: This chapter explains the steps taken to implement a sampling method based on ADC suitable for LiDAR in a workflow manner. It presents the experimental results from the work developed.

Chapter 4 - *TDC - based front-end*: This chapters describes at a conceptual level the design of the TDC architecture to implement a suitable sampling method for LiDAR.

Chapter 5 - *Implementation of a TDC*: This chapters describes the implementation design of the TDC architecture, and presents corresponding simulation.

Chapter 6 - *Experimental validation of a TDC*: This chapters presents corresponding experimental results of the implementation design of the TDC architecture.

Chapter 7 - *Conclusion*: a summary of the results and outcomes presented and discussed across the document, along with some topics for future work.

1.5 Contributions

The software and hardware developed for this dissertation can be accessed on the author's personal GitHub page, on https://github.com/alexandre-brandao/SOFTLI. Several Very High Speed Hardware Description Language (VHDL) modules were developed to meet the requirements of the dissertation, namely, ADC and TDC implementation in a LiDAR context.

- Set the ADC/DAC as fully synchronous and operating as single-shot.
- Added a python interface to operate the ADC/DAC for ADC implementation in a LiDAR context.
- Implementation and development of an all-digital TDC based on a tapped delay line and a mux-based encoder.
- Development of an all-digital TDC with features that, to the best of the author's knowledge, have not been found in the literature.[1]
- Co-author of an invention report within the scope of project SOFTLI, titled "Interference-resilient LiDAR waveform and estimation method thereof". Such an invention relies on TDCs.

[1]Please refer to section 4.10 for more detail.

CHAPTER 2

Fundamental ADC and TDC concepts

2.1 Introduction

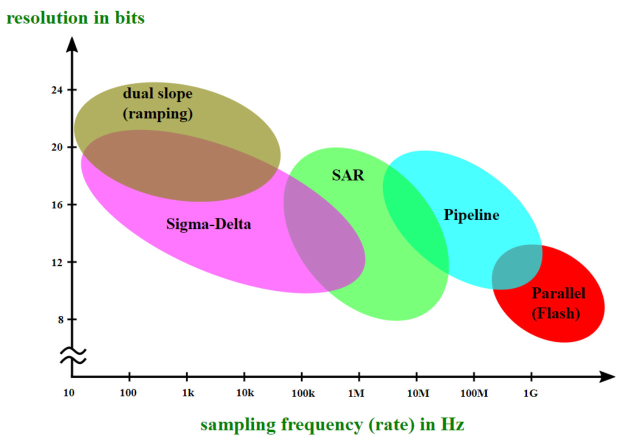
In this chapter, ADC and TDC architectures suitable for ToF measurement are examined and discussed. The chapter starts with the study of various conventional ADC architectures to find the best architecture that suits this dissertations needs based on power, speed and accuracy. Next is a study of the various TDCs architectures to investigate the possible implementation of a TDC in an FPGA. As a last study, search for a suitable thermometer encoder which is typically coupled with a TDC.

2.2 ADC Architectures

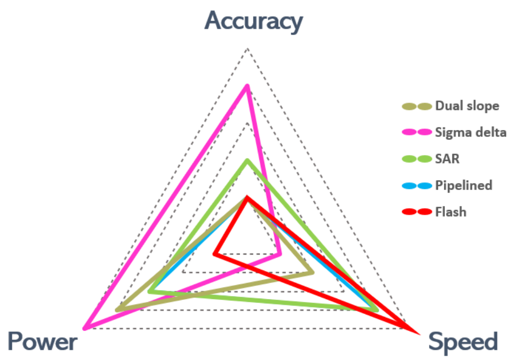
When looking for an ADC, there are three important features that should be considered:

1. Sampling rate.
2. Power consumption.
3. Resolution (Effective Number of Bits (ENOB)).

Defining such features is important as it rules out some ADC architectures, leaving others to be chosen. The reason for such a rationale is that, unfortunately, there is no "one size fits all" architecture. Figure 2.1 illustrates the tradeoffs of the conventional architectures.



(a) Conventional ADC architectures comparison diagram in terms of resolution vs speed.



(b) Some of the conventional ADC architectures trade-offs on a radar chart.

Figure 2.1: Conventional ADC architectures comparison adapted and reproduced from [11], [12].

The following subsections discuss different ADC architectures with its respective pros and cons, namely:

1. SAR.
2. Delta-Sigma A/D.
3. Dual-Slope A/D.
4. Flash A/D.
5. Pipelined A/D.

2.2.1 SAR

Successive-Approximation-Register (SAR) architecture has been used for decades. SAR ADCs are very common, stable and reliable and are also known as the "Bread and Butter" of the the Data Acquisition (System) (DAQ) world.

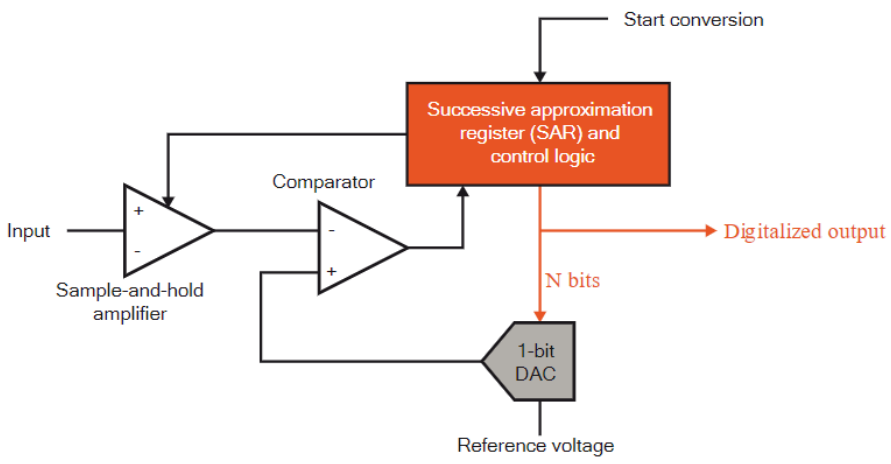


Figure 2.2: SAR Block Diagram reproduced from [13].

The `Input` signal is initially held on the sample-and-hold circuit that outputs a signal that will remain constant at the input V^- of the comparator until the circuit decides if it has successfully created a word. The output of the Digital-to-Analog Converter (DAC) is connected at the input V^+ of the comparator which creates an analog reference voltage based on the N bit digital word generated from the control logic. The N bit digital word is based on a binary search algorithm which tries to successfully divide the reference voltage until the output of the DAC matches the sampled input voltage which is being held by the sample-and-hold circuit.

The algorithm works by starting with the N-bit register, the digitalized output. This output is first set the Most Significant Bit (MSB) to `1` and the remaining bits to `0`, that is, `100....00`. This forces the DAC output half of the reference voltage, where the DAC output voltage is provided to the comparator, where it will output a high or low signal and consequently be evaluated by the control logic. At this point the MSB of the digitalized remains with the logic attribute assigned by the comparator. The SAR control logic then moves to the next bit down, forces that bit high, and does another comparison. This process continues for n successive times, with n being the bit resolution of the ADC itself, until the closest value to the actual signal is found [13], [14] as shown in the figure. 2.3.

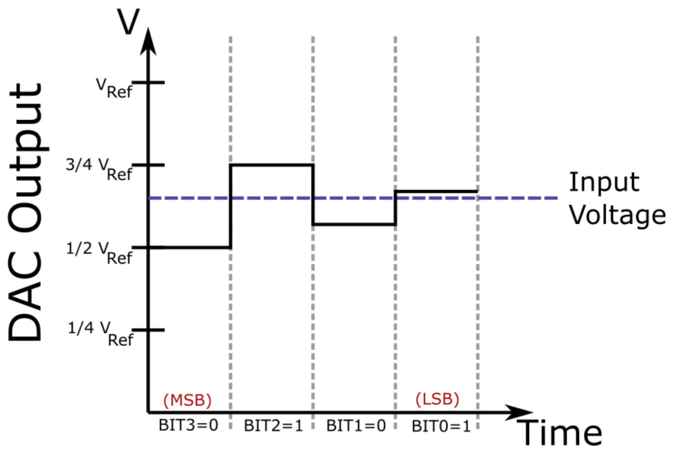


Figure 2.3: Operation of a 4-bit ADC based on SAR.

Pros and cons. This architecture is capable of handling a wide variety of signals with excellent fidelity whilst offering a good balance between speed and resolution [13] and can possess high sampling rate. The disadvantage of this architecture is that these ADCs do not possess any inherent Anti-Aliasing Filtering (AAF) to get rid of higher-frequency noise and signals, unless it is manually added.

Applications. Applications for SAR ADCs include DAQ systems, from low-end multiplexed ADC systems to higher speed ADC systems, industrial control and measurement and CMOS imaging [13].

2.2.2 Delta-Sigma A/D

Delta-Sigma architectures are known for their high bit resolution systems as they are able to strongly reduce quantization error.

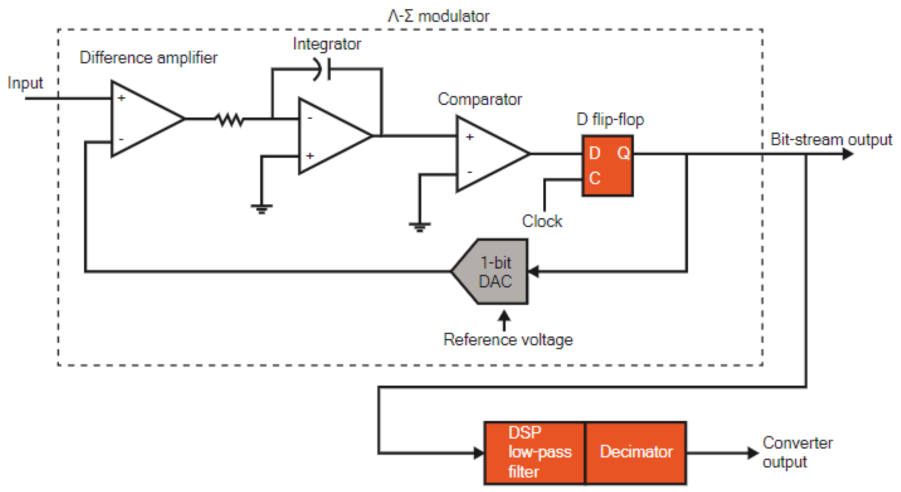


Figure 2.4: Delta-Sigma Block Diagram adapted from [13].

This architecture uses oversampling to sample far higher than the base sample rate. The Delta-Sigma ADC consists of a modulator, a filter, and a decimator described next, these ADCs are mostly digital.

$\Delta - \Sigma$ *modulator.* This module is used for noise shaping of the oversampled input signal consisting of a difference amplifier, an integrator, a comparator and a 1 bit DAC, which essentially also acts as a comparator. In the figure 2.5 is the another form the modulator representation.

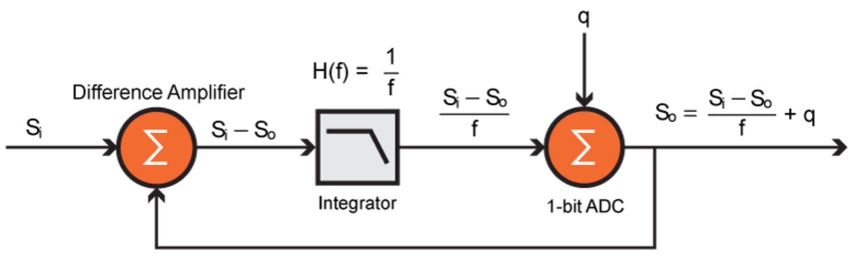


Figure 2.5: Modulator block diagram from Delta-Sigma architecture seen in the frequency domain reproduced from [15].

The integrator in this architecture acts as a low-pass filter to the input signal. Quantization noise is added to the signal output of this filter due to the 1-bit conversion process. The output of the modulator can be represented using (2.1) [15].

$$S_o(f) = \frac{S_i(f) + qf}{f+1} = \frac{S_i(f)}{f+1} + \frac{qf}{f+1}, \tag{2.1}$$

where q is the quantization noise.

The first term of the equation (2.1) is considered the signal term and the second term can be considered the noise term. As the frequency approaches zero, it can be seen that the noise term approaches zero and the output of modulator approaches S_i. As the frequency is increased, the noise term approaches q and the signal term approaches zero. Thus, the integrator acts as a high-pass filter for the quantization noise [15].

DSP Low-pass filter. The effect of the low pass filter eliminates the second term of the eq. 2.1. Resulting in a clean output signal given by (2.2).

$$S_o(f) = \frac{S_i(f)}{f+1}. \tag{2.2}$$

Decimator. Decimation is the process of discarding the unnecessary samples and is used as a mechanism to reduce the data rate to a usable value whilst maintaining the information.

Pros and Cons. This approach creates a very high-resolution data stream and has the advantage of allowing multistage AAF, making it virtually impossible to digitize false signals [13]. However, this architecture pays in speed.

Applications. Data acquisition, especially noise and vibration, industrial balancing, torsional and rotational vibration, power quality monitoring, precision industrial measurements, audio and voiceband, communications [13].

2.2.3 Dual-Slope A/D

Dual Slope ADC are accurate and possess medium to high speed in relation to other architectures.

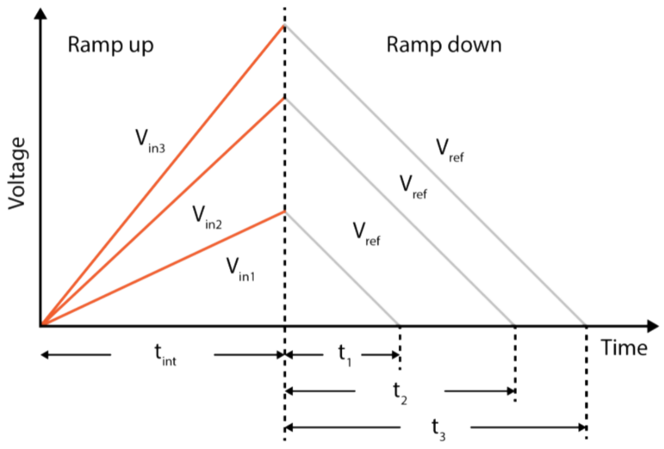


Figure 2.6: Dual-Slope ADC graphical analysis reproduced from [13].

This principle is simple, the input voltage is allowed to "run up" for a controlled period of time [13]. Once the signal has been set up for a set duration, a known voltage starts running down at a fixed speed when the polarity is changed. When the known voltage hits the value 0, the systems compares the time stamps and calculates the input voltage based on the time stamps relation.

Pros and Cons. This architecture is simple, very precise and accurate, but as the graph indirectly shows, the setup lacks the speed.

Applications. Handheld and benchtop multimeters[13].

2.2.4 Flash A/D

Flash ADCs are very well known ADCs. This is due to to their conversion speed and is also very simple to understand. Their conversion speed is the fastest out of the architectures and operate with negligible latency.

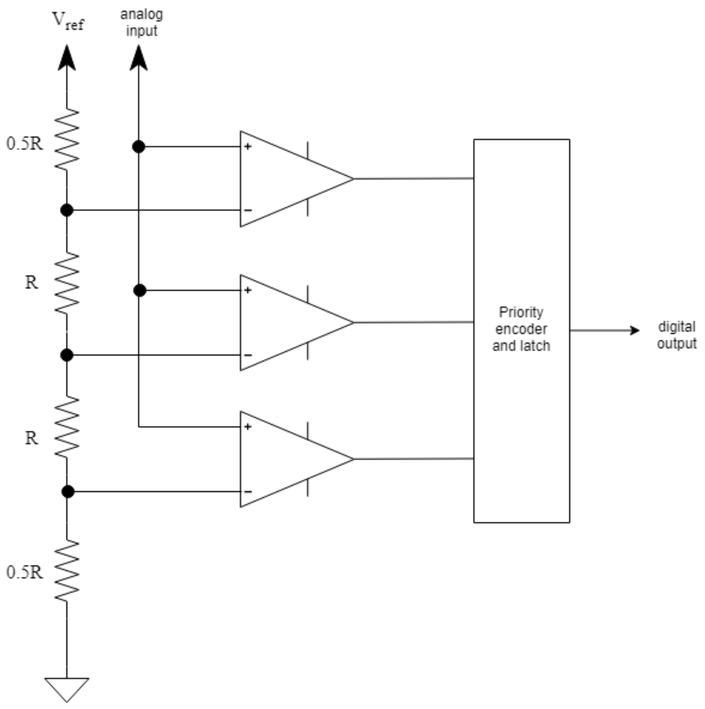


Figure 2.7: Flash ADC diagram reproduced from [16].

The flash ADC resolution is limited by the number of resistors, this means that if we want an N bit flash ADC, there is a need for 2^N resistors and $2^N - 1$ comparators.

Pros and Cons. This circuit is ideal for the need of high sampling rates, however there's a big price to pay with power vs resolution.

Applications. Fastest digital oscilloscopes, microwave measurements, fiber optics, RADAR detection, and wideband radio [13].

2.2.5 Pipelined A/D

The Pipelined ADC stands on the middle ground in terms of speed. It is faster than the SAR and delta-sigma architectures but slower than flash ADC.

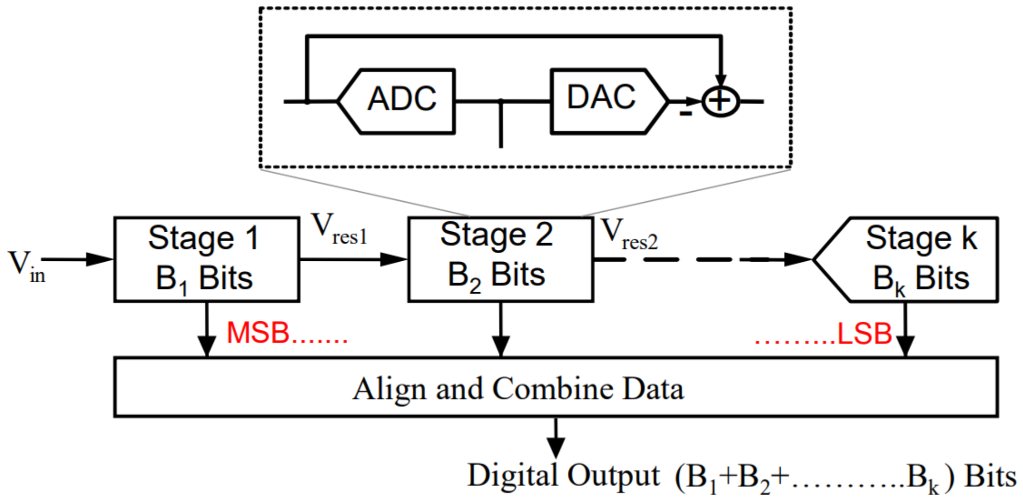


Figure 2.8: Pipelined ADC diagram reproduced from [17].

This architecture cascades low resolution stages to obtain high overall resolution, can also be seen as the pipelined version of the flash ADC. The flash ADC latches all of its comparators to one reference voltage, however, in a pipelined ADC, the analog signal is not latched by all comparators at the same time, thus spreading out the energy required to convert the analog to a digital value as seen in the figure 2.8. Each stage performs coarse A/D conversion and computes its quantization error. In this type of ADC, the conversion takes two steps. During the first step, the most significant bits of the digital output are determined by the first stage flash ADC. Then a DAC converts this digital result back to an analog signal to be subtracted from the input signal. This residue is amplified by the inter-stage gain amplifier and then sent to the second stage flash ADC. The second stage flash determines the least significant bits of the digital output, as shown in the figure 2.9.

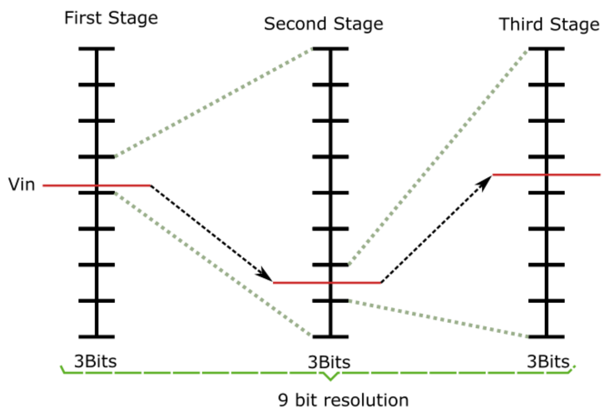


Figure 2.9: Pipelined operation for a 9 bit ADC [18].

Pros and Cons. This architecture allows high resolution without using huge amounts of energy. On the other hand, this architecture is slower than the Flash ADC.

Applications. Digital oscilloscopes, RADAR, software radios, spectrum analyzers, HD video, ultrasonic imaging, digital receivers, cable modems, and Ethernet [13].

2.2.6 Summary

The table 2.1 summarizes the features of the presented architectures.

ADC Type	Slope	Sigma-Delta	SAR	Pipelined	Flash
Accuracy	6-20 bits	16-32 bits	6-16 bits	8-16 bits	$<$12 bits
Speed	Slowest	Slow	Moderate	High	Highest
Bandwidth	Slowest	Slow	Moderate	High	Highest
Latency	Moderate	Highest	Low	High	Lowest
Power	Low	Moderate	Low	Low	High
Area	Small	Small	Small	Small	High
Pros	Accurate, inexpensive	High stability and dynamic performance, inherent anti-aliasing protection	Good speed/resolution ratio	Very fast	Fastest
Cons	Low speed	Hysteresis on unnatural signals	No inherent anti-aliasing protection	Limited resolution	Low bit resolution

Table 2.1: ADC state of the art architectures performance comparison [13], [19]–[21].

Having analyzed the different ADC architectures, it is now time to choose a suitable architecture for a LiDAR receiver. The waveform to be sampled is a short pulse with a pulse width between 10 ns and 100 ns, with a repetition rate lower than 10 kHz. A suitable ADC for sampling such a pulse with high fidelity should have:

1. A high sampling frequency between 200 Msamples/s to 1 Gsample/s.
2. Power consumption lower than 1 W.
3. An ENOB of at least 8.

Table 2.1, aids in choosing the ADC based on the features above. Such a high sampling frequency, excludes Slope, Sigma-Delta and SAR architectures. Out of the two fastest architectures, there is a need for low power consumption. As such, the chosen architecture is the pipelined architecture which has the best match in terms of the accuracy, power and speed.

Based on the chosen architecture, we chose to purchase an ADC based on pipelined architecture of the AD9680 ADC, which in turn was found integrated in the AD-FMCDAQ2-EBZ kit. The AD-FMCDAQ2-EBZ Board from Analog Devices provides an ADC that has an on-chip buffer and sample-and-hold circuit designed for low power, small size, and ease of use, which is very practical. The ADC also has an ENOB of 10.4bits when operating at 10 MHz which is greater than the required 8 bits and is able to sample wide bandwidth analog signals of up to 2 Gsamples/s which the required was a maximum of 1 GHz. To add to that it uses a low jitter clock generator which can run up to 3 Gsamples/s and a reference design compatible with our FPGA, a ZC706 evaluation board.

2.3 TDC Architectures

TDCs precisely measure the time intervals between two events, popularly known as start and stop, and have applications in a large number of time measurement systems and subsequently have a variety of industrial and research applications. They are widely used in digital storage oscilloscopes, logic analyzers and high-energy particle physics experiments [22]. The simplest form of a TDC is a digital counter. However, to achieve a high resolution TDC, one needs to use a very high frequency counter and this is not energy efficient. The resolution of counter-based TDCs can be improved significantly by resolving the counter residual error with a high resolution fine TDC based on gate delay [23].

This section provides an overview of some important existing architectures of TDCs, focusing on its principle of operation and emphasizing suitability for FPGA implementation. The following architectures with its respective pros and cons are discussed:

1. Time-to-Amplitude Converter.
2. Pulse Shrinking.
3. Tapped Delay Line.
4. Vernier Delay Line.
5. Phased Clocks.
6. Gated Ring Oscillator.

2.3.1 Time-to-Amplitude Converter (TAC)

The first approach at measuring ToF was quantizing a signal into a digital word. This can be seen in the figure 2.10.

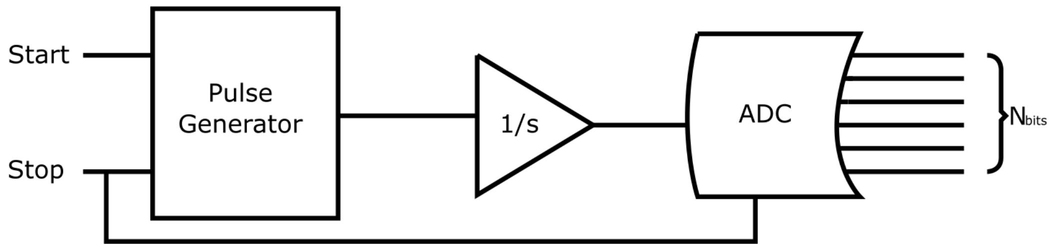


Figure 2.10: Single-Slope Analog-To-Time-Interpolation converter reproduced from [24].

As with many TDCs, it has a start and stop command, in this case, triggering the commands, start and stop, respectively, define the width of the pulse of the Pulse Generator, in this architecture the start command triggers a pulse to stay at a logical **1**, or a constant and positive voltage that is then introduced into a integrator.

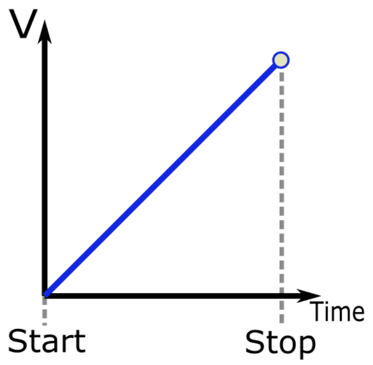


Figure 2.11: Single-Slope Analog-To-Time-Interpolation converter graphical illustration.

The integrator behaviour can be seen in the figure 2.11. Once the `stop` signal is triggered, the value at the output of the integrator, is an analog signal, which is introduced into an ADC and is then converted into a word which can be mapped into a value in time[24]. The number of bits at the output of the ADC will dictate the maximum value of time that can be measured.

$$DR = 2^N \cdot T_{LSB} \tag{2.3}$$

Where DR is the Dynamic Range of the ADC converter and T_{LSB} is the time it takes the Least-Significant-Bit to transition from a logical `0` to `1`.

2.3.1.1 Pros and Cons

This architecture is easy to understand and very intuitive, however, the reason why it is so rarely used nowadays is because the final system resolution is defined by the ADC resolution and the conventional usage of the ADC. All blocks should have a linear response and the integrator itself is very dependent on its implementation. Finally, the presence of the ADC makes this architecture unsuitable for the FPGA implementation.

2.3.2 Pulse Shrinking

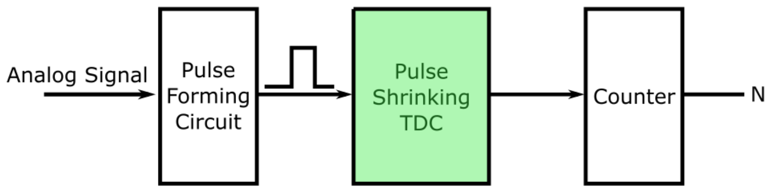


Figure 2.12: TDC common architectural block diagram for Pulse Shrinking adapted from [25].

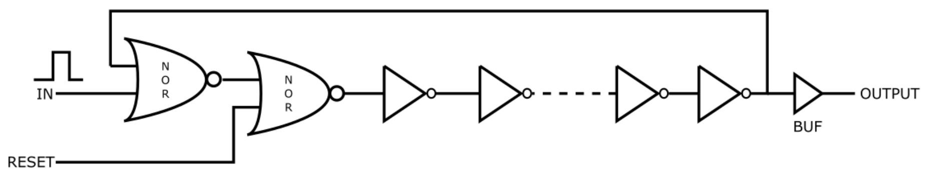


Figure 2.13: Pulse Shrinking TDC reference architecture adapted from [25].

Pulse shrinking architectures are very straightforward, as the name implies, it consist in, given an input signal, repeatedly shrink the input pulse until it ceases to exist.

To shrink the pulse, the input signal must propagate through a delay line which is set by a consecutive number of inverters, ideally with the same width and an even number of NOT gates as seen in figure 2.13. If all inverters maintain the same size, the pulse would maintain its duration and because it has a closed loop, these architectures exhibit a ring oscillator behavior where the number of NOT gates determine the oscillation frequency. For this circular behavior, the signal is fed back into a NOR Gate as seen in figure 2.13. Knowing that for each loop that occurs, the pulse is shrunk by a set amount and that the counter is incremented in each loop, set by `OUTPUT` signal, figure 2.13, to the counter block [25]. The relation between the value of the counter, in other words, the number of loops that occur times the shrinking factor tell us the length of the input pulse.

2.3.2.1 Pros and Cons

This architecture offers great resolutions and low power consumption. The disadvantage of this architecture is that as it is not possible to create custom cells and hard control the shrinking time in an FPGA. Nevertheless, the pulse shrinking delay line can be implemented in FPGA subjected to limitations [25], [26].

2.3.3 Tapped Delay Line

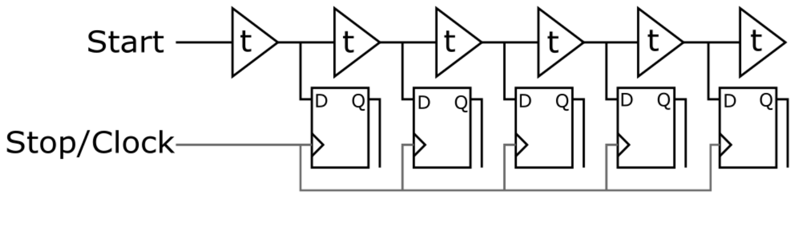


Figure 2.14: Tapped Delay Line.

Tapped Delay Line (TDL) architectures are very popular due to their simplicity in implementation, the idea behind the TDLs is to use the intrinsic propagation delay of a basic element, also known as a digital cell to be able to make fine measurements [27], [28]. As shown in figure 2.14, we can see that when a `start` signal is set, it starts to propagate through the delay line, the idea is to delay the signal through these taps. In which each tap delays a signal by the delay introduced from its implementation. The `stop` or the `clock` signal is set so that at each rising edge of the clock or when the `stop` signal is triggered to store a "photo" of the state of the delay line, making it possible to infer at which position the signal stopped.

It is important to known that this architecture is easily affected by Process-Voltage-Temperature (PVT) conditions, this means that the bigger the delay line, the less linear it is. This can be seen in the figure 2.15.

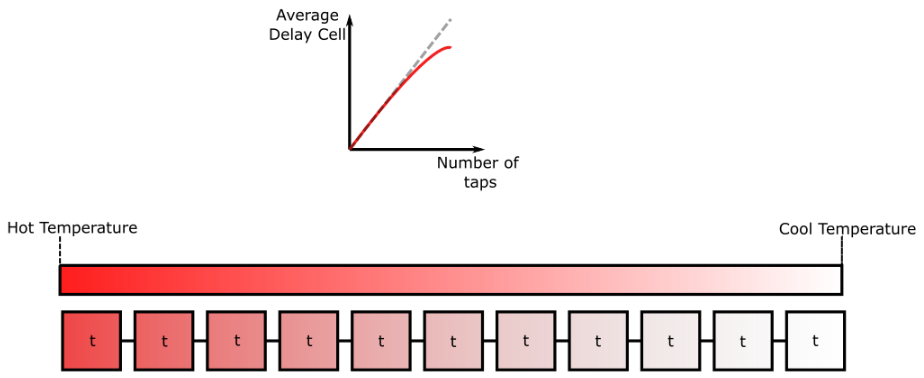


Figure 2.15: Tapped Delay Line temperature effect of average delay per digital cell.

The idea behind figure 2.15 is that if every delay cell suffered the same variation, that would mean that the average delay cell would increase or decrease linearly, however, because temperature in many cases does not affect the delay line in a uniform manner, the TDL tends to deteriorate in linearity, thus leading to a transfer curve that differs from the desired linear behaviour. To minimize effects which are propagated through the delay chain, it is desirable to have a delay chain as short as possible.

Like the other mentioned architectures, all of them have a problem with dynamic range and in this particular case, there is a limit to how many digital cells can be present in a limited area, and second, due to PVT conditions, it is not recommend to have a large number of digital cells. The TDL is typically seen coupled with a counter, known as a coarse counter which deals with big jumps in time and the TDL deals with the fine measurements. This type of schemes can be seen in the figure 2.16.

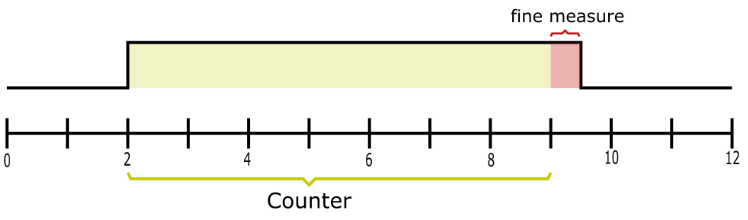


Figure 2.16: Coarse counter presence for increasing the dynamic range of the TDL.

2.3.3.1 Pros and Cons

The advantage of this architecture is that it is possible to develop a fully digital TDC at the cost of the FPGA resources in which digital cell chains are always accompanied by their following register chains in a modern FPGA fabric, which is convenient for the software compiler to automatically generate the TDL. The algorithm is relatively simple and it has low power consumption [29]. However, it is prone to glitches, easily affected by bubbles[1] and has metastability problems as the latching input might transition exactly when being captured, violating setup/hold constraints [30]. Another disadvantage of this architecture is the fact that it is limited by the technology itself and the most significant limitation of these architectures is the difficulty to predict the placement and routing delays as well as the time delay of the logic gates itself. The consequence of this inevitable hardware restriction is a non-stable resolution of the designed TDC [31].

2.3.4 Vernier Delay Line

This architecture was proposed to remove the limitation of the cell delay being limited by technology in the TDL.

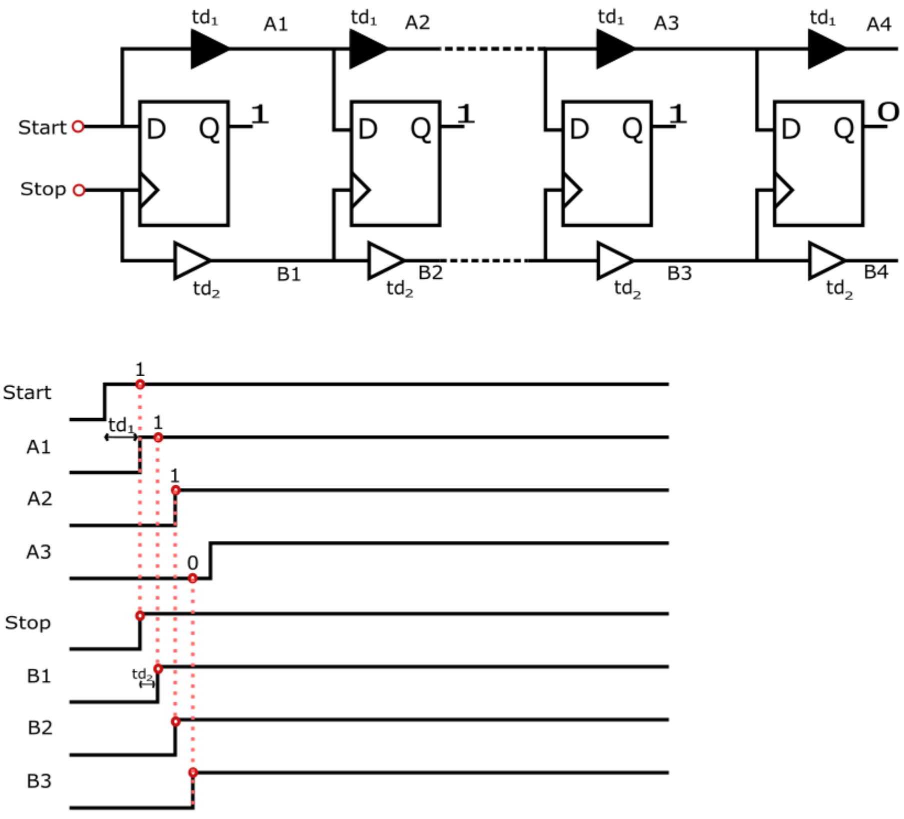


Figure 2.17: Vernier Delay Line concept reproduced from [24].

It's basic setup and principle of operation can be seen in the figure 2.17. The proposed architecture consist of 2 parallel delay lines, in which each delay element that belongs to the start line must be close to and higher than the delay of the elements of the stop line. This means that when the start sets off and starts propagating through its delay line and after some time, the `stop` signal is triggered. When the stop signal is triggered, it must be able to catch up to `start` signal, in other words, the `stop` signal must propagate faster than the start signal.

[1]For more information about bubbles, please see section 2.3.

The measurement is done based of the difference between the `start` and `stop` signal, this means that there is direct relationship between how far into the delay line it took the `stop` signal to catch up to the start signal.

2.3.4.1 Pros and Cons

High resolution can be achieved and the delay mismatch is low due to same delay elements used as delay cells. Regarding FPGA implementations, the Vernier Delay Line (VDL) architecture is usually implemented using a dual ring oscillator schemes, paired with a phased detector [32]. However, implementation on FPGAs is complex due to creating the delay elements in an FPGA making it much more complex than the TDL.

2.3.5 Phased Clocks

If the resolution demands are not very high, then the phased clocked architecture is typically used to reduce resource usage. The architecture can be seen in the figure 2.18.

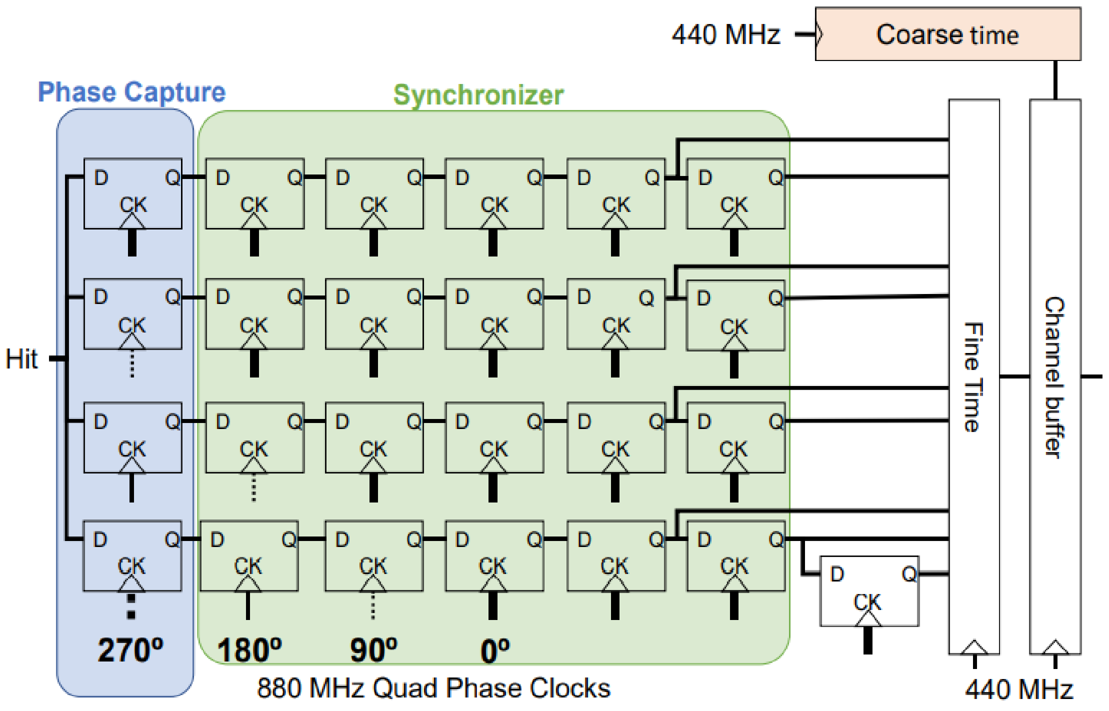


Figure 2.18: Phased Clocks architecture reproduced from [33].

The basic structure of a phased clock architecture is that uses multiple phases as bins to sample the `hit` signal which can be generated via pulse generator as shown in figure 2.10 with a start and stop command. The input signal is divided in four parts and detected by the D-type flip-flops based on four different clock phases. The locations of the four D-type flip-flops are constrained to control the difference between the divided signal paths. The vertical double lines, thick lines, double dashed lines, and dashed thick line correspond to the clock phases 0, 90, 180, and 270, respectively. The outputs from the four D-type flip-flops are aligned step by step by the chains of additional D-type flip-flops [34]. The fine time counter extracts a three-bit time count from the pattern of the outputs from the chains. The fine time count is combined with the data from a coarse time counter to output a reasonable measurement. Unlike the TDL, the delay is not generated by the intrinsic cell propagation time but instead is given by the phase difference between the clocks used. The higher the number of clock phases, the higher will the resolution be [33].

2.3.5.1 Pros and Cons

This architecture poses lots of problems with high frequencies and routing skew. On the other hand, an advantage of this architecture is its simplicity. Another advantage of this architecture is that for FPGAs this is a great implementation as there are multiple ways of generating phased clocks such as using Phase Locked Loop (PLL), Delay Locked Loop (DLL) or even delay lines for phase shifting. This architecture is also PVT insensitive in comparison to other architectures [35].

2.3.6 Gated Ring Oscillator

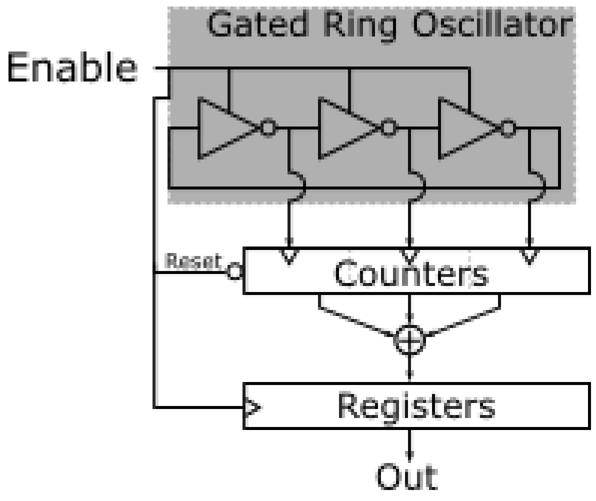


Figure 2.19: GRO architecture reproduced from [36].

This architecture operates only when the `Enable` signal is high level and stops when this signal is at low level. The outputs of the Gated Ring Oscillator (GRO) are used as the clocks which drives the counter [37]. One important note, the GRO topology is different from a traditional ring oscillator, such as the Pulse Shrinking architecture by its property to freeze between two consecutive time measurements. If instead of freezing, the system is reset between measurements, there is no possibility to use the previous state to improve the resolution by analyzing the quantization error.

2.3.6.1 Pros and Cons

The benefit from the gating technique, especially with successive measurements, is the increased effective time resolution which cannot be achieved for a simple measurement [38]. However, the main issue is to initialize and maintain a well-defined oscillation through gating operation, since the ring freezes and starts from the previous state without reset [39]. Implementation in an FPGA would be very complex.

2.3.7 Summary

Type	Advantages	Disadvantages	FPGA Implementation
TDL	Clock is not required. High resolution can be achieved and has a simple design. Low resource usage. Simple to calibrate.	Needs very high frequencies for lower number of delay cells for better performance. Needs calibration.	Simple
VDL	Delay mismatch is low due to same delay elements used as delay cells. High Resolution can be achieved. Clock is not necessarily required. These lines have higher complexity but are able to offer performance levels close to the ones achieved by tapped delay lines	Component variant delay due to many buffers in one delay line. Higher complexity when compared to the TDL. Requires a minimum of 2 TDLs on FPGA. Hard to calibrate.	Challenging
GRO	Simple and delivers high resolution.	High area usage. When propagating a pulse along a chain of buffers configured as a ring, due to mismatch on the cells' rise and fall times, a pulse shrinking/stretching effect can occur leading to the cease of the signal.	Complex
Pulse Shrinking	Offers a decent resolution although higher resolutions can be achieved	The shrinking factor cannot be fully controlled in an FPGA. Resolution still falls flat against the TDL when implemented in FPGA and the extra complexity involved to compete with the TDL is not justified.	Complex
TAC	Simplicity.	Requires the presence of an ADC. Weak linearity.	No
Phased clocks	Best linearity and very low hardware resource utilization. Good for low resolution applications and PVT insensitivity. No calibration.	Sensitive to routing skew and very unstable at high frequencies.	Simple

Table 2.2: TDC State of the Art architectures summary [24], [40].

Table 2.2 summarizes each one of the architectures advantages and disadvantages. The TDL-based TDC is the simplest to understand and possibly the least complex to implement in an FPGA, turning it into an attractive option. Another reason to consider the TDL is the significant number of publications directly related to the TDL, allowing for a more mature architecture.

2.4 Encoding Logic for Thermometer Code

TDC architectures typically offer their measurement as a word known as a thermometer code which must be converted into a binary code. These thermometer codes were mainly seen in flash type ADC where the output was digital and seen as a sequence of `1`s and `0`s. This sequence known as a thermometer code, is an entropy coding of a sequence of `1`s followed by a `0` or vice-versa which depending on the state of the chain can indicate position of the pulse.

Table. 2.3 is representation of the conversion of the thermometer codes to binary codes to figure out the number of ones that are present.

Decimal Number	**Thermometer Code**	**Binary Code for ones**	**Binary Code for zeros**
0	0000000	000	111
1	0000001	001	110
2	0000011	010	101
3	0000111	011	100
4	0001111	100	011
5	0011111	101	010
6	0111111	110	001
7	1111111	111	000

Table 2.3: Thermometer encoding.

An appropriate definition for bubble errors is when there is one or more invalid bits in a thermometer code then it is termed as bubble error and the thermometer codes depending on the FPGA design suffer from bubble errors. For example the binary word `111110000` may turn into a `110010000`. We can see this effect in the figure 2.20.

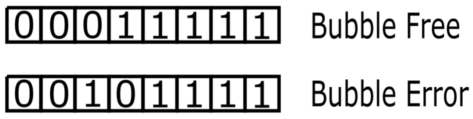


Figure 2.20: Bubble effect.

The main cause are due to uneven propagation delays in the design or uncertainty introduced in the sampling moment due to meta-stability. Now, choosing the proper design for the encoder is crucial to minimize the resulting error with the minimum delay possible, this means the encoder must be "bubble proof" so the encoder should output a reasonable number.

As such, the following thermometer encoders are presented along with its pros and cons:

1. ROM based thermometer encoder.
2. Wallace Tree based encoder.
3. Fat Tree based encoder.
4. Multiplexer based encoder.

2.4.1 ROM based Thermometer-to-Binary converter

The Read Only Memory (ROM) based technique is one of the first techniques implemented in thermometer encoding, also very popular to this day and was initially adopted for flash ADC classified as Binary-ROM encoder using a parallel layout [41], [42]. The circuitry and scheme layout of this encoder can be seen in the figure 2.21 is used to count the number of ones in a binary representation.

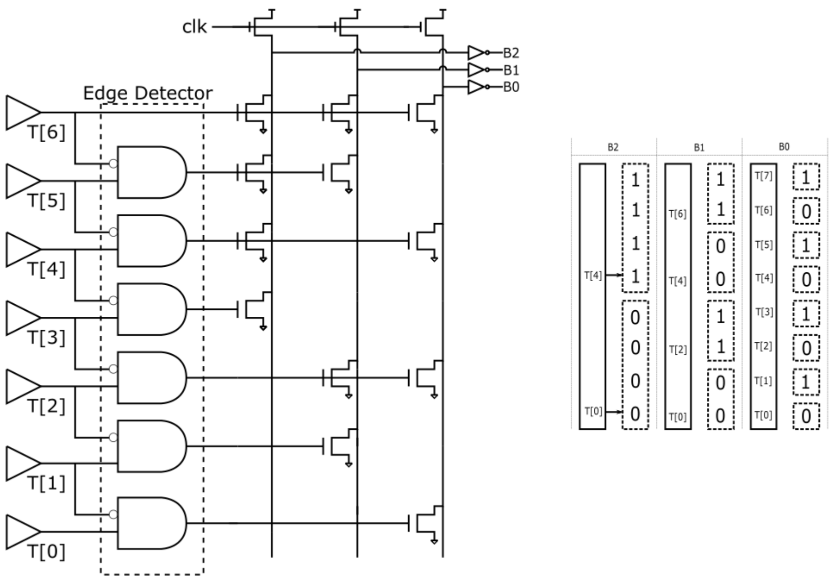


Figure 2.21: ROM based encoder circuitry and scheme adapted from [41].

The layout of figure 2.21 at first looks relatively complex but the reality of the situation is that it is not. To start, the schematics show us a 7 bit thermometer code input and a 3 bit binary code output. Between the input and the output there is an Edge Detector, or what the original authors call the 1 of N block or One-Hot-Encoder[41]. Essentially, the edge detectors detects transitions of a logical `1` to logical `0` of the thermometer code, in which it is always necessary to assume that the code is clean and so there are no bubble errors. The edge detector is composed of multiple AND gates with 1 input non-inverted and the other input inverted. Next is the layout of the transistors, visible on the right side of figure 2.21. Finally, the output has inverters at it is output and their purpose is just to invert whatever value is at the input of the inverter.

Next, the example in the figure 2.22.

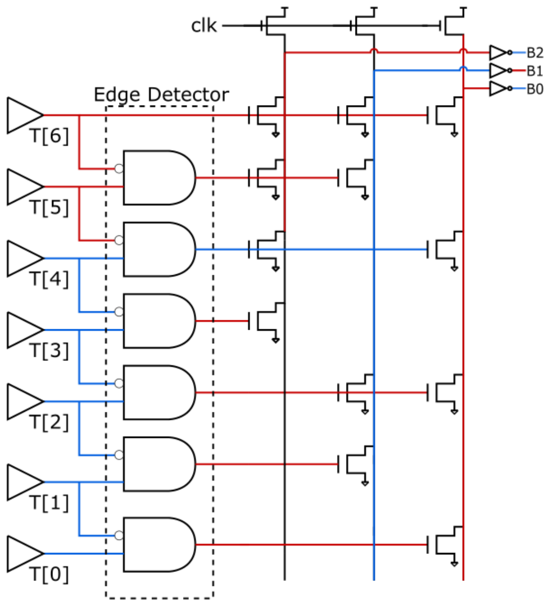


Figure 2.22: ROM based encoder example.

The blue lines in the figure 2.22 represent the logical value `1` and the red lines represent the logical value `0`, the color of the lines also represent the state of the lines. So, with an input of `1111100`, the expected output should be the binary word `101`. Looking closely at the figure, B2 and B0 are blue and B1 is red which represent the binary word `101`, so as expected, the circuit works and counted correctly the number of `1`s. Basically, it stores combination of input variables and for each combination it will generate the output.

Next, looking at the same circuit one last time in the figure 2.23.

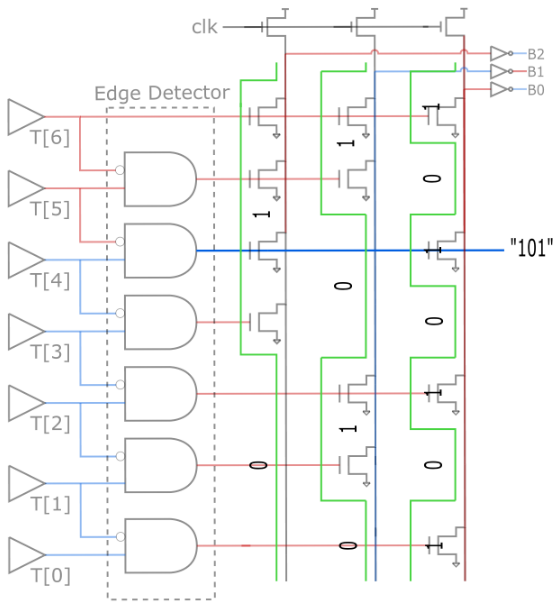


Figure 2.23: ROM based encoder example analysis.

We can see that the scheme on the right side of the figure 2.21 and the figure 2.23, makes the pattern of the transistors very visible, highlighted in green and the blue line from the edge detector passes through the green lines when it is logical values are `1`, `0`, `1`, respectively. However, the output is inverted, so it doesn't make sense for it to be actually, `1`, `0`, `1`, respectively, in reality, looking again at the same figure, these values are acting on the transistor gate inverting the expected logical value, so it is possible to infer that it is actually the inverted values of the expected output and that's why the output is inverted.

This method is able to correct at least one bubble error.

To note that it is possible to implement a fully digital Binary-ROM based encoder as mentioned by [41] and seen in the figure 2.24.

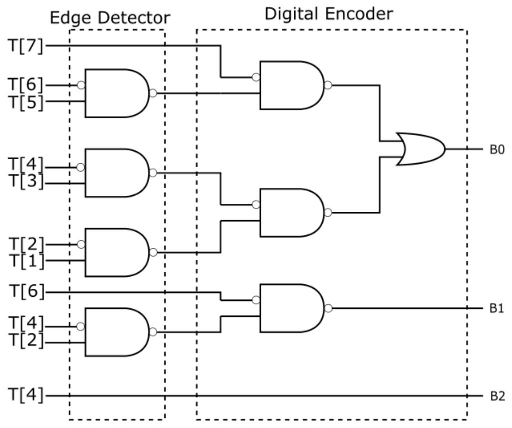


Figure 2.24: Full digital ROM based encoder reproduced from [41].

2.4.1.1 Pros and Cons

The advantage of this circuit is its simplicity. However, relation to the other architectures it is slow, power consumption is high in general and grows with increased thermometer codes and increased data speed might turn this technique into a victim of bubble error making it unsuitable for very high frequency applications, to add to this more circuitry is required to fix these bubble errors.

For an FPGA, that positions its logic cells in very specific locations, make it hard for these kind of techniques to be accepted as the probability of uneven propagation delay occurring is much worse than implementation via Very Large-Scale Integration (VLSI). In other words, for high speed operations, this technique will only degrade in performance and implementation on the FPGA is hardly feasible.

2.4.2 Wallace Tree based Thermometer-to-Binary converter

The Wallace Tree encoder is, like the ROM based Encoder, a technique to calculate the number of ones, but instead uses Full Adders as illustrated in the figure 2.25

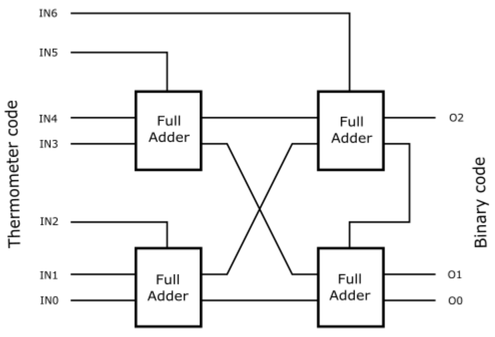


Figure 2.25: 3 Bit Wallace Tree Encoder.

In the figure 2.26 we can see that its basic cells, the Full Adders are arranged in a specific way to count the number of logical `1`s at their input for its final binary output. In this particular example, the blue lines represent the logical value 1 and the red lines represent the logical value 0. For an input `111111111000000` the output is `1001` which represents the decimal value of 9.

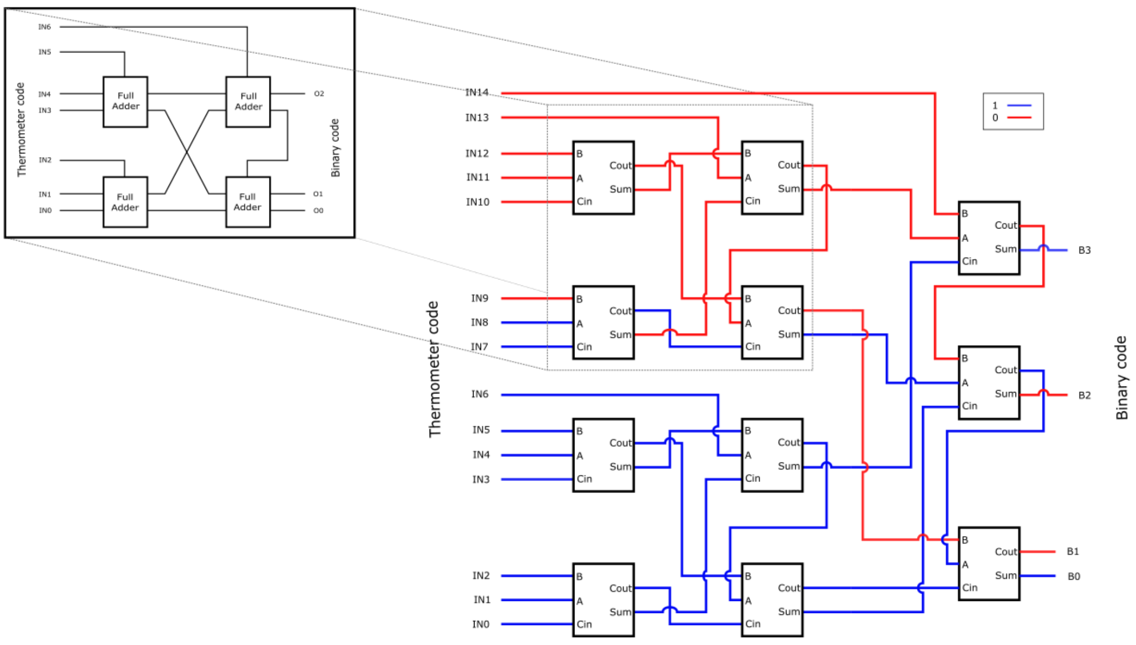


Figure 2.26: 4 Bit Wallace Tree encoder example adapted from [43].

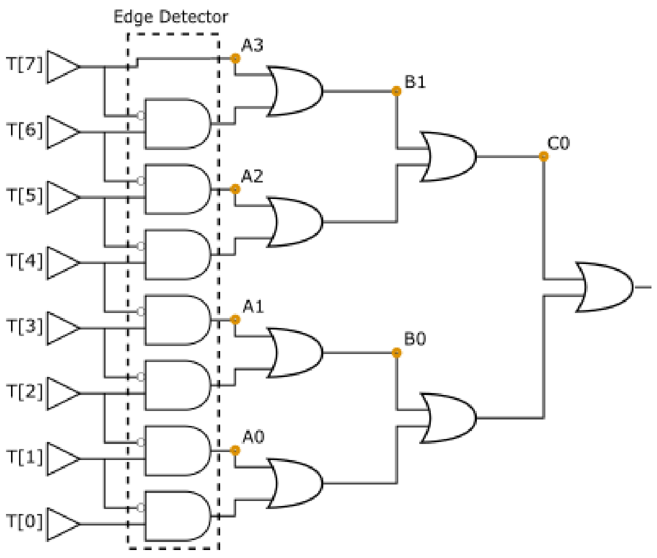
The operations of the encoder can be easily pipelined, allowing the thermometer codes to be converted into fine timestamps without introducing dead time. Using this encoder severely reduces the probability of having missed TDC codes since it does not aim at locating the transition point in the thermometer code [42], [44].

2.4.2.1 Pros and Cons

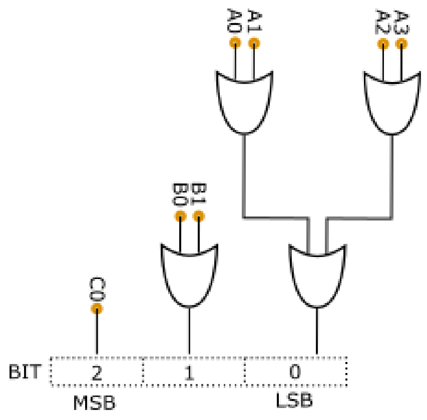
The advantage of this circuit is its flexibility and the fact that its topology can be selected according to speed and power. The disadvantage is the fact that relative to the other architectures it has a huge area, high power consumption, it is relatively complex to implement on an FPGA with scalability and has a long critical path [45].

2.4.3 Fat-Tree based Thermometer-to-Binary converter

The fat tree shown in figure 2.27 is relatively simple technique where OR Gates are main component. This simplicity allows for direct conversion from the input to the output calculating the number of ones.



(a) Free Tree logic using OR gates.



(b) Fat Tree output logic.

Figure 2.27: Fat Tree encoder adapted from [46].

It's behavior is very similar to ROM based encoder where instead of using a representation via transistors, this architecture uses some combinatorial logic to produce an output.

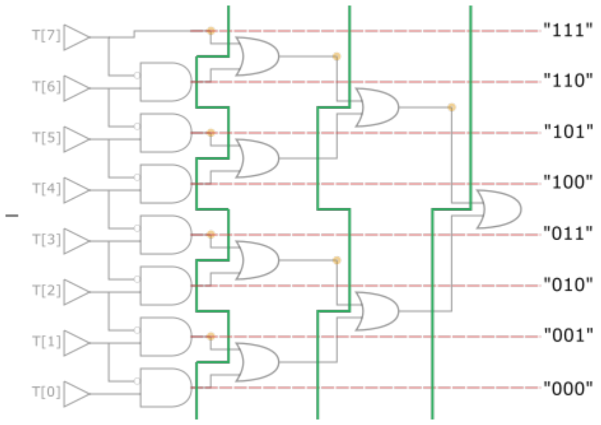


Figure 2.28: Fat Tree logic analysis.

On the right side of the figure 2.27, the bit C0 decides if it is a logical `1` or a `0` of the MSB. Next is the OR gate between B0 and B1 which will once again decide if it is a logical `1` or a `0` of the second MSB, which is half of the the MSB. Finally, Least Significant Bit (LSB) chooses whether the least significant bit is a logical `1` or `0`. Once again, depending on the thermometer code, one of the AND gates will be active outputting a given binary code.

2.4.3.1 Pros and Cons

The advantages of this technique is that its implementation on FPGA is not complex, it only requires the usage of Look-Up Tables for implementation. However, implementation with a thermometer code size which can easily altered seems to be relatively complex. Aside from this, it also has significant power consumption [47].

2.4.4 Multiplexer Based Thermometer-to-Binary Converter

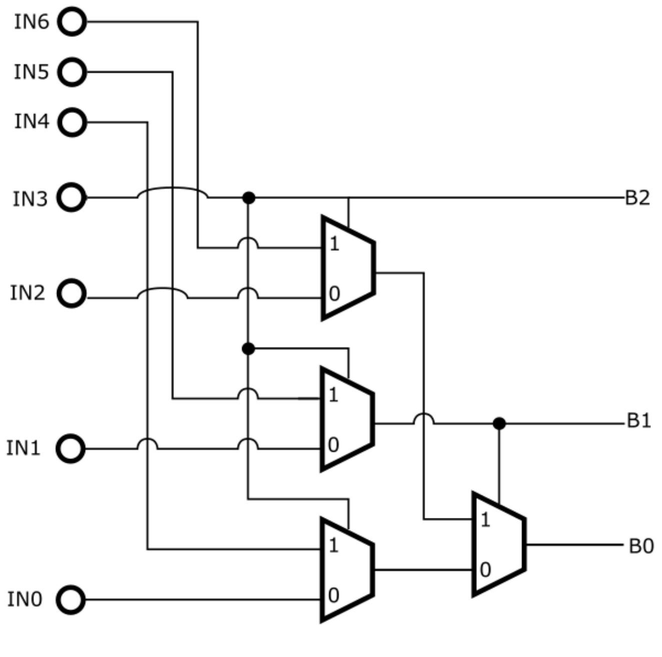


Figure 2.29: Standard mux based encoder.

The standard muxed based encoder topology is a relatively simple algorithm and works on binary search algorithm. It is mainly composed to 2:1 multiplexers. This algorithm was originally adopted for Flash ADC Thermometer to Binary conversion but is now used on TDCs [42], [48].

To briefly explain the algorithm, the idea behind this is the following. Let's say there's 7 bit input thermometer code as seen in figure 2.29, and a 3 bit output. The number of output bits always have to be able to fit the value of the number of input bits in binary. So, in this case, the decimal value of 7 can be represented in 3 bits as `111`.

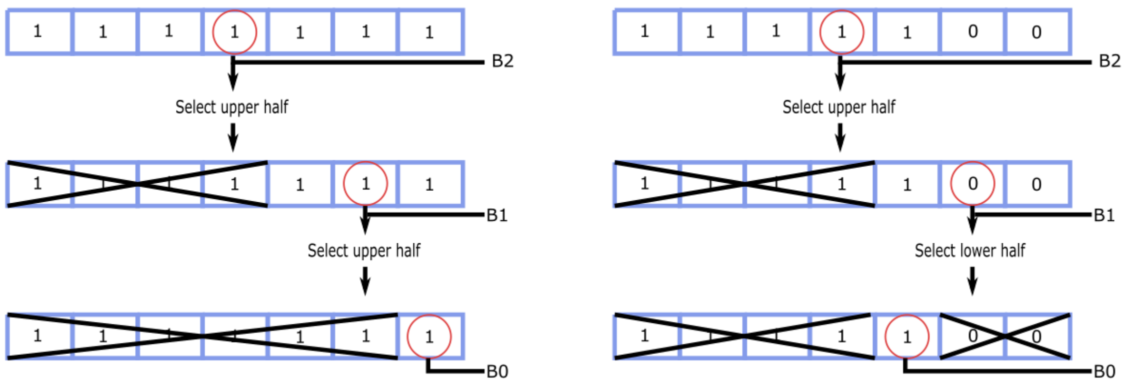


Figure 2.30: Mux based encoder algorithm.

The figure above is the explanation of algorithm, on the left side the input thermometer code is all logical `1`, now the object on this algorithm, is like the previous architectures, to count the number of ones. To do this, first the whole input is selected, where the bit at the center of the bus will select the upper or lower half of the input. On the right and left side both bits at the same location are set `1` and as such will select the upper half of the input bus and discard all the other values and B2 will be equal to logical `1`. Now the process repeats itself, the bit at the center of the available input will select the upper or lower half, on the left side of the figure the upper half is selected and on the right side, the lower half is selected, discarding the bit itself from the input and the bits on the opposite side of the bus. The finally the remaining bit is selected. Thus, for the case of the left side of the image, the input is `1111111` and the output is supposed to be `111` and the algorithm delivered `111` which is correct. On the right side of the image, the thermometer input is `1111100` and the output is supposed to be "101" and the algorithm delivered `101` which is once again correct.

On the next figure 2.31 is another example of the state of figure 2.29 when the input is `1111000`.

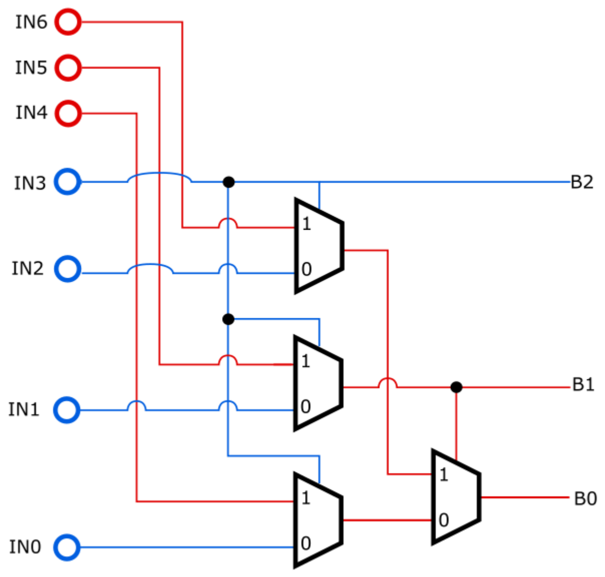


Figure 2.31: Mux Based encoder example.

This algorithm for bubble correction has at least 1 degree of bubble suppression.

Pros and cons. The algorithm itself is very intuitive and easy to implement using 2:1 multiplexers, which means using FPGA Look-Up Tables (LUTs). It requires less hardware and has a short critical path which is very ideal to implement in an FPGA operating at very high frequencies. Scalability is not issue as this algorithm can be easily expanded to higher resolution of bits [49] and uses low power [45], [47]. However, this structure has a disadvantage of large fan-out which increases the overall power consumption [50].

2.4.5 Summary

Encoder Type	**Advantages**	**Disadvantages**
Standard Mux Based Encoder	Simple, scalability is not an issue and has short critical path. Bubble error correction up to the third order. Low power and less latency in comparison to the other architectures. No clock signal required.	Large fan-out with increased thermometer code.
Wallace Tree Encoder	Extremely simple and scalability is not an issue. Precisely matched with best approximated value of the output. Capable of self bubble error correction. No clock signal required.	Has large delay, occupies large areas and is not suitable for the high speed operations.
ROM Based Encoder	Extremely simple and scalability is not an issue. No clock signal required. The binary encoder that can remove the bubble error up to the fourth order.	Power hungry.
Fat Tree Encoder	Has lower latency and smaller area when compared to ROM based encoder and wallace tree encoder. No clock signal required.	Large delay and power hungry.

Table 2.4: Thermometer state of the art encoders summary [51], [52].

Table 2.4 shows that all architectures offer bubble suppression at least to 1 degree. However, the most efficient method in terms of speed, latency and simplicity seems to be the multiplexer-based method. As a result, the TDC to be implemented is based on a TDL architecture and on a standard mux-based thermometer encoder.

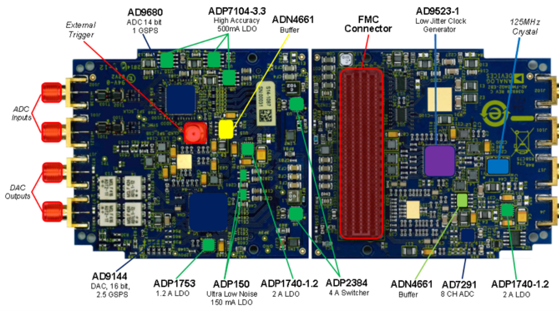
CHAPTER 3

ADC-based front-end

3.1 Introduction

This chapter is organized in a workflow manner and describes the problem and solution for the measurement of the ToF using an ADC on an FPGA, namely the ZC706 Evaluation Kit Board [53] combined with the AD-FMCDAQ2-EBZ Board [54] from Analog devices. Both the FPGA board and the AD-FMCDAQ2-EBZ Evaluation board are shown in the figure 3.1.

(a) ZC706 evaluation board.



(b) AD-FMCDAQ2-EBZ evaluation board.

Figure 3.1: Boards used for DAC/ADC front-end Implementation.

This chapter is divided in multiple parts:

1. Brief introduction to the hardware involved and Hardware Descriptive Language (HDL) designs used to get started.
2. An overall discussion of the AD-FMCDAQ2-EBZ kit, making its principle of operation and limitations crystal clear.
3. Breakdown of the steps taken for a DAC to emit single-shot pulses.
4. Breakdown of the steps taken for the ADC to achieve synchronization between DAC/ADC.
5. Python user commands to control the FPGA.
6. ADC experimental validation.

3.2 Initial setup

To operate both boards, Analog Devices provides a reference design[1] for the typical user to get started with the AD-FMCDAQ2-EBZ Board. Both a C project and a Vivado reference design[2] are provided in their github to setup for Vivado 2018.3. These files are accessable on **https://github.com/analogdevicesinc/hdl** for Vivado's reference design and **https://github.com/analogdevicesinc/no-OS** for the C project code.

By default, the AD-FMCDAQ2-EBZ is already able to transmit and receive test signals. However, they are not perfectly synchronized and thus this chapter addresses the problem of synchronizing both modules.

Starting by understanding the overview of the AD-FMCDAQ2-EBZ block diagram of the design and explaining the design changes step by step aids in breaking down the complex design as was given(see appendix B) to reach this chapters objective.

3.3 AD-FMCDAQ2-EBZ Evaluation Board overview

The AD-FMCDAQ2-EBZ Board provides 2 ADC and 2 DAC channels with full synchronization capabilities [55]. Component wise, this module has three important components.

- AD9680[3] dual, 14-bit, 1.0 GSPS, JESD204B ADC.
- AD9144[4] quad, 16-bit, 2.8 GSPS, JESD204B DAC.
- AD9523-1[5] provides a 14-output, clock distribution function with low jitter performance.

The AD-FMCDAQ2-EBZ board is clocked by an internally generated carrier platform via the FPGA Mezzanine Card (FMC) connector.

3.3.1 Analog Devices Reference Design overview

Figure 3.2 shows the block diagram for the reference design available from Analog Devices.

[1]Please see appendix A for initial setup.
[2]Please see appendix B for reference design.
[3]https://www.analog.com/media/en/technical-documentation/data-sheets/AD9680.pdf
[4]https://www.analog.com/media/en/technical-documentation/data-sheets/AD9144.pdf
[5]https://www.analog.com/media/en/technical-documentation/data-sheets/AD9523-1.pdf

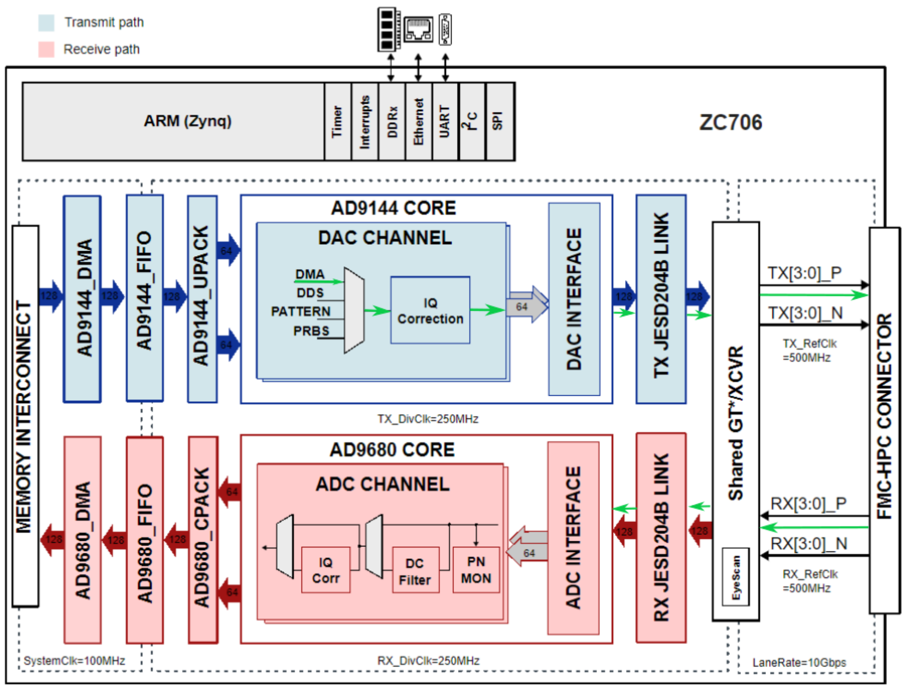


Figure 3.2: Analog Devices Setup for a Xilinx Block Diagram on the ZC706 adapted from [56]

The figure 3.2 shows how the modules interact with each other and the datapath. The reference designs illustrate that it can be run on a ZYNQ processor and that the following modules and communications protocols are available.

- Timers.
- Interrupts.
- Double Data Rate (DDR).
- Ethernet (ETH).
- Universal Asynchronous Receiver Transmitter (UART).
- Inter-Integrated Circuit (I2C).
- Serial Peripheral Interface (SPI).

For communication with external modules, only the UART and ETH is available. The diagram informs us that it uses the memory DDR as either a workspace or for storing information.

In terms of datapath, the diagram also tells us that for operating the DAC, 128 bits of information must be deposited into a First In First Out (FIFO), whose interface is a data streaming interface. The next step is to pass the data into the AD9144_upack which also has a streaming interface. The upack utility core from Analog Devices uses a technique known as interleaving by “shuffling” over the bus of 128 bits of data.

Next up is the AD9144 core which must be correctly configured to warn the core where the information is coming from. The core needs to know if it is from an internal data generator such as Direct Digital Synthesis (DDS), pattern or Pseudorandom Binary Sequence (PRBS) or from the external DDR via Direct Memory Access (DMA). Finally, there is the transceiver, also known as Shared GT or XCVR which instantiates the transceiver and correctly sets it up.

For operating the ADC, it is a matter of following the data flow in the opposite direction, where the cpack utility core unpacks the shuffled data from the DAC and the ADC core requires no specific configurations unlike the DAC. The ADC sampled data is sent to the DDR via DMA.

To note that the DAC interface is channel based and the ADC has an internal DC filter present, leading to the removal of the DC component of any incoming signal.

3.3.2 AD-FMCDAQ2-EBZ Clocking

Understanding how the clocks were distributed and generated on the board was a crucial step to understanding how to manage and adjust the project to our desire.

The AD-FMCDAQ2-EBZ platform uses a PLL from the AD9523-1 with the reference for the PLL being sourced from an external 125 MHz crystal. The AD9523-1 PLL2 Voltage Controlled Oscillator (VCO) has a locking range of 2.940 GHz to 3.100 GHz, outputting a nominal frequency of 3 GHz [57].

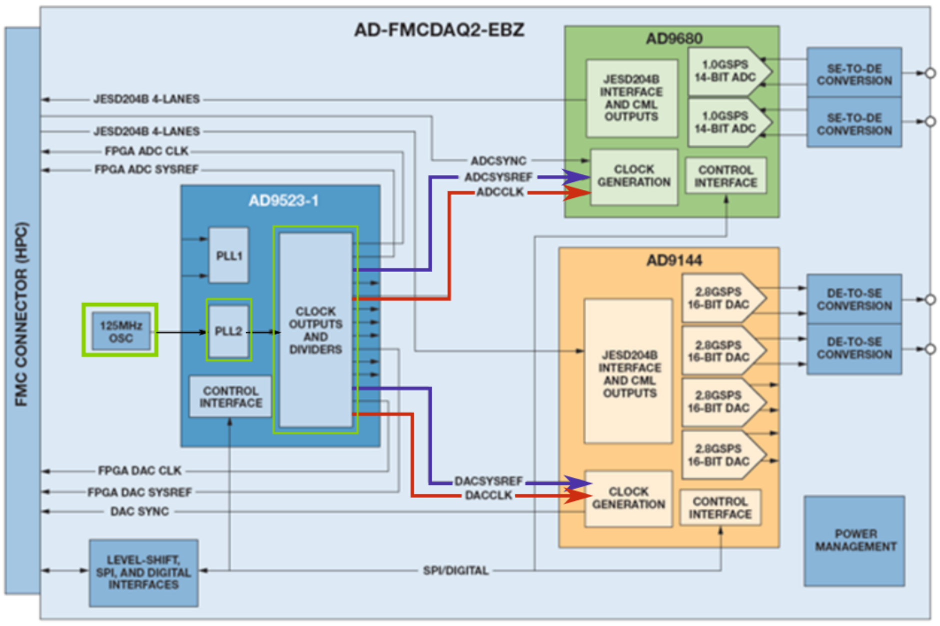


Figure 3.3: AD-FMCDAQ2-EBZ Block Diagram[6].

Looking at the figure 3.3, there's a 125 MHz crystal which drives the PLL2, generating a nominal frequency of 3 GHz which goes through a combinatorial block which feeds the ADC and DAC of the system.

From the description supplied by Analog Devices in relation to clocking of the ADC and DAC, namely, the `ADCCLK` and `DACCLK` reference in the red arrow of the figure. These clocks determine the sampling rate of the converters and if a harmonic relationship is required, then the combinatorial block which acts as a divider for PLL2 can be freely configured [57]. As a result of this, so long as both of them are configured with the same divider then at least those two points are guaranteed to be synchronized.

[6]https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcdaq2-ebz.html

3.3.3 Bare Metal AD-FMCDAQ2-EBZ Software

Analog Devices also provides a C project complementing their reference design in Vivado. To test their design the DACs were directly connected to the ADC, in order to conduct loopback tests as seen in figure 3.4.

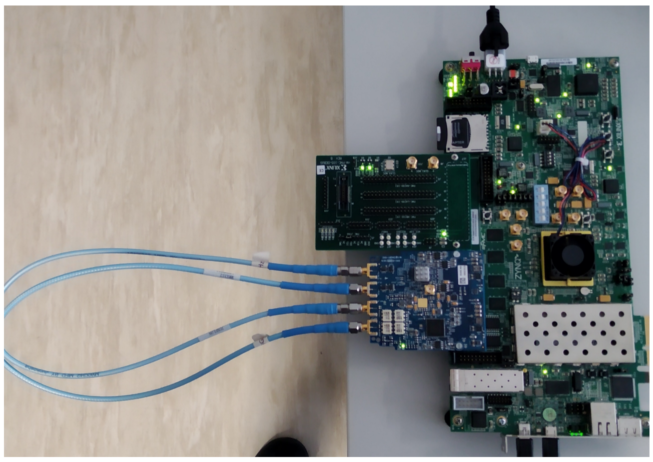


Figure 3.4: Closed loop AD-FMCDAQ2-EBZ.

Using their software, the results were stored in a Comma Separated Value (CSV) file, where the outputted values were already in two's complement. Thus, resulting in the figure 3.5.

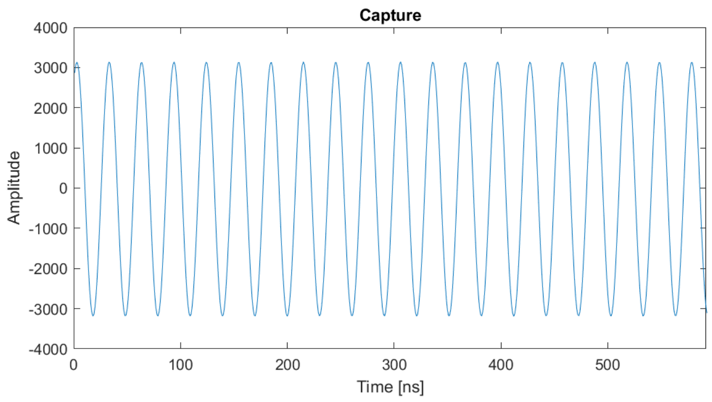


Figure 3.5: Sine wave being repeatedly outputted cyclically by the DAC.

Their software had already been set up with a sine wave test waveform.

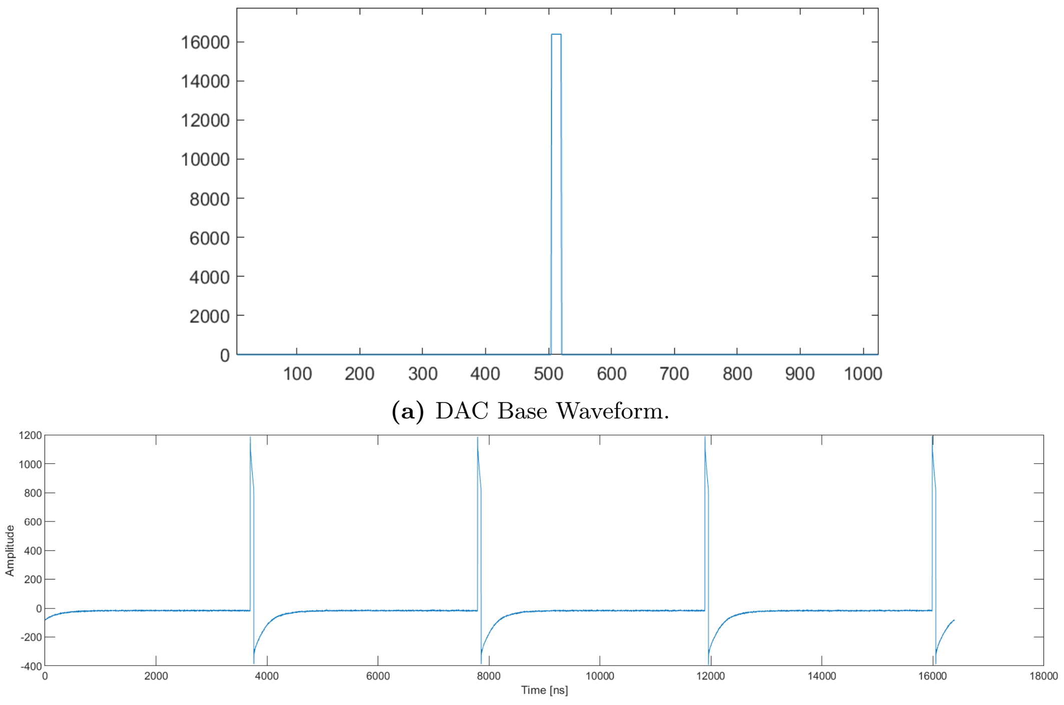
3.4 DAC/ADC Front End implementation

This section discusses how the ADC and DAC full synchronization is achieved.

3.4.1 Operating the DAC

The first objective is to get the DAC to produce a desirable waveform. The first step was to load a predefined signal into a buffer of 1024 positions and set up the DMA, where the buffer would then be loaded into a FIFO via DMA which would then output the same waveform cyclically. The cyclic behavior is shown in the bottom waveform of the figure 3.6.

Multiple waveforms were tested and the final decision was to set the pulse with a very small duty cycle to bypass the DC Blocker within the ADC. Shown in the top waveform of the figure 3.6.



(a) DAC Base Waveform.

(b) Captured Waveform for a duration of 16μs.

Figure 3.6: Capturing the waveforms when the DAC was still operating cyclically.

The results shown in the figure 3.6 looked promising as the waveform could now be selected and captured with some degree of distortion, however the waveform was still being reproduced cyclically and the software wasn't able to stop the cyclic behaviour. So the decision was to use same idea of the buffer in software and instead implement a buffer that would carry the same narrow pulse and repeatedly output the pulse in hardware, written in VHDL for the reference design and as a consequence lower the number resources used by FPGA and lowering the complexity of the blocks.

By setting up the waveform and converting these waveform values into hexadecimals representations of the 2's complement representation values, allowed for easy passage of a waveform of 1024 positions to an RTL memory block. The figure 3.7 is a simple illustration of the steps taken to pass the values to the RTL block.

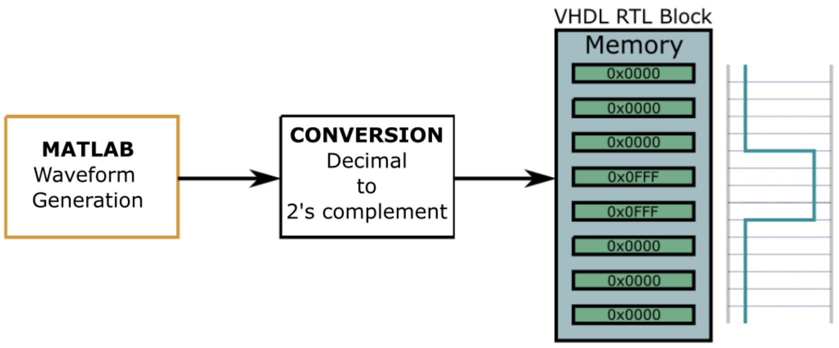


Figure 3.7: From waveform generation to RTL memory block.

The new block interface inserted is shown in figure 3.8.

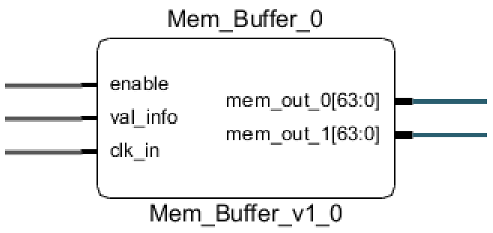


Figure 3.8: First Version of the Memory Buffer operating in cyclic mode.

The behavior of the new RTL block is shown in the figure 3.9.

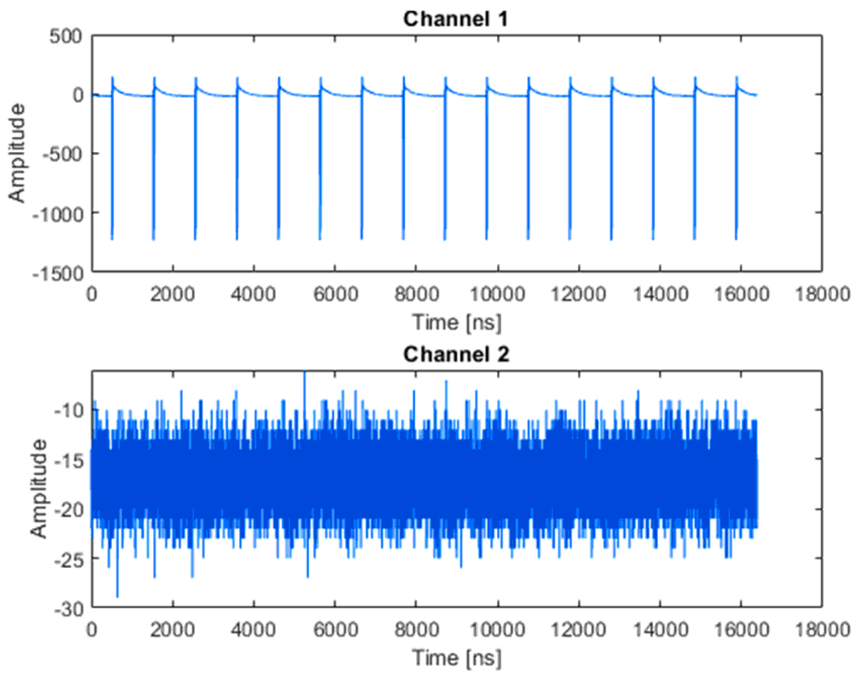


Figure 3.9: Short pulses produced by the DAC. The bottom plot is the noise of the effect of disconnecting the channel 2.

At this point, the DAC is generating a short pulse every $1\,\mu$s. After proving that this block showed the same results and used less resources, the next step was to remove the cyclic behaviour by adding a single-shot mode that would be triggered by the user at the touch of key. Such an implementation required the addition of a new axi-slave custom Intellectual Property (IP) block to communicate with the Zynq and the current block generating the signal would have to be modified to only output a given signal at the command of the single-shot.

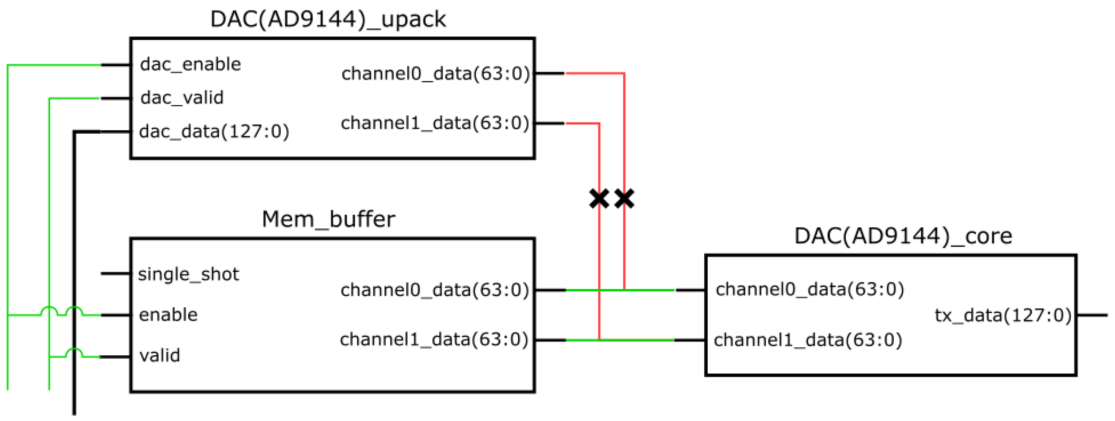


Figure 3.10: Simplified diagram of the DAC upack block substitution for the new RTL mem_buffer block.

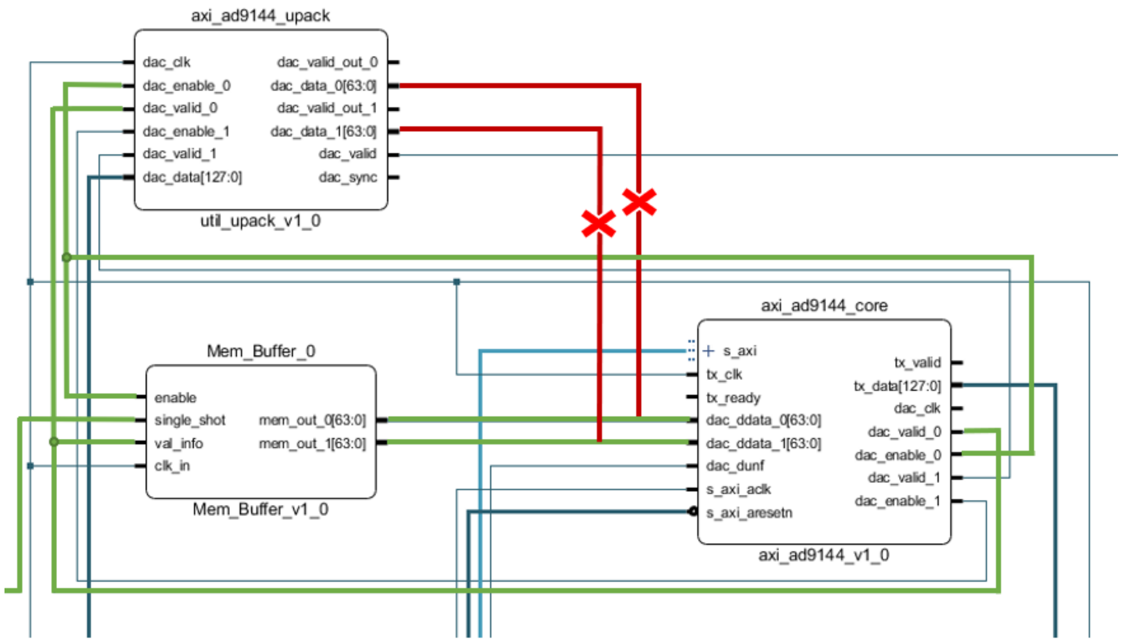


Figure 3.11: Adding the new updated RTL block to the Vivado Reference Design operating in single-shot.

The figure 3.10 is a simplified diagram of the actual connections which occurred in Vivado shown in the figure 3.11.

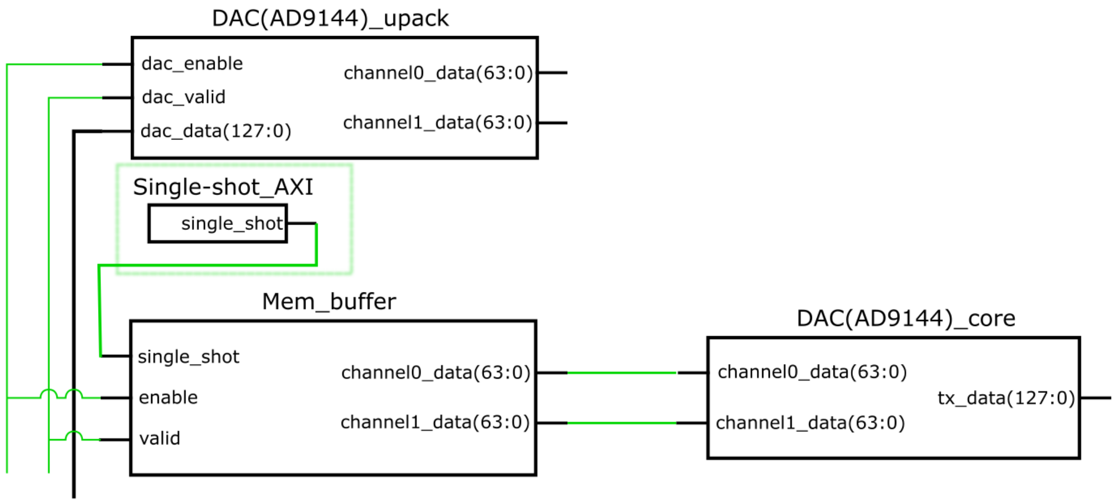


Figure 3.12: Block diagram of the insertion of single-shot axi block.

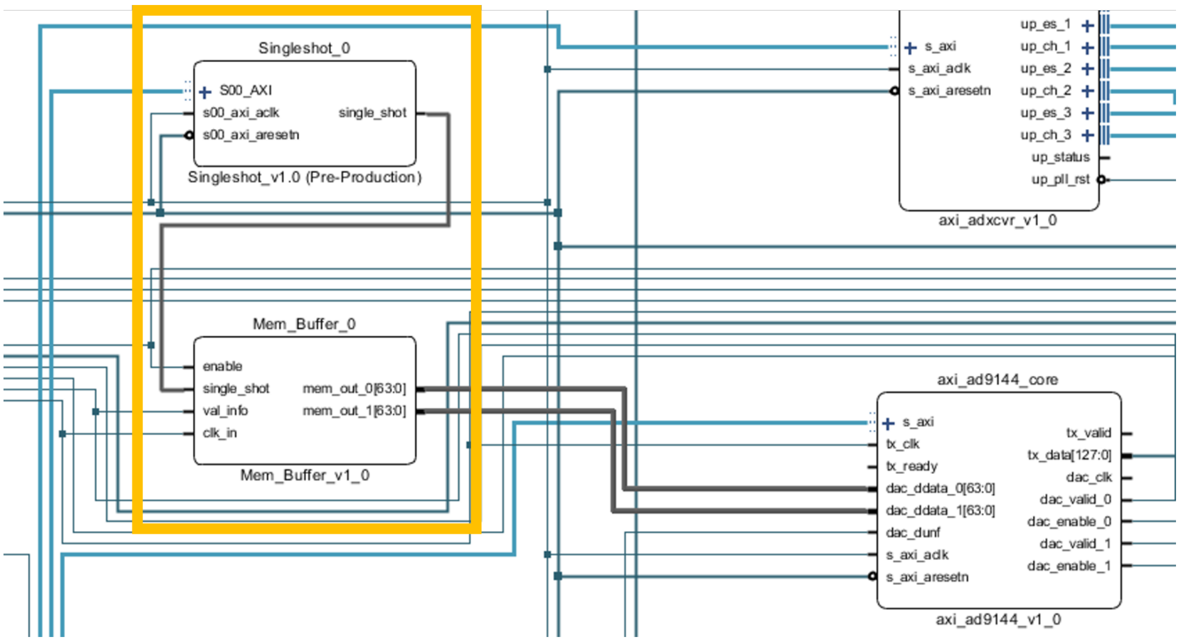


Figure 3.13: Connecting the new RTL block with the single shot RTL block.

With the newly updated RTL block and single-shot block added, shown in the figure 3.12, the system only outputted one waveform manually triggered shown in the results section, figure 3.22.

3.4.2 Operating the ADC

Operating the ADC was not as complex as the DAC was already configured, needless to say that complexity of the blocks combined that would make up the ADC were much more complex. As shown in figure 3.15.

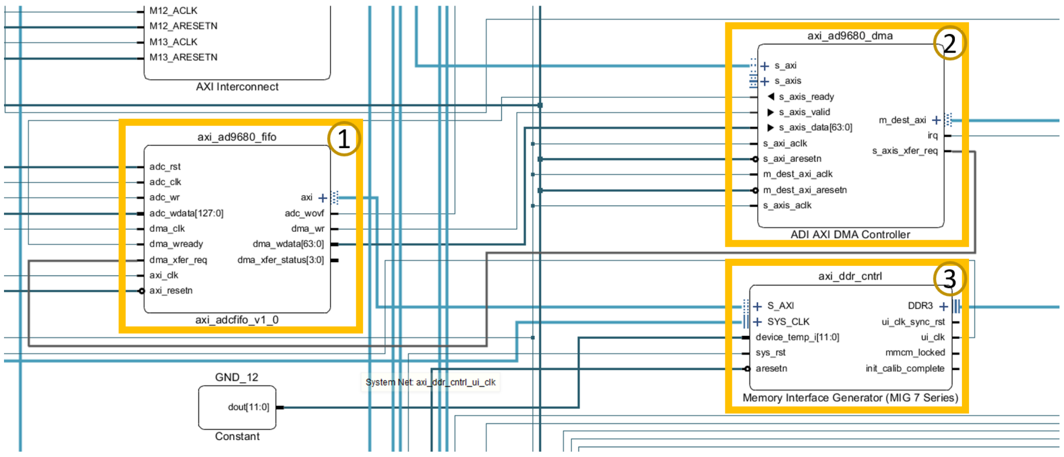


Figure 3.14: Identified the block involved with the ADC for capturing and storing.

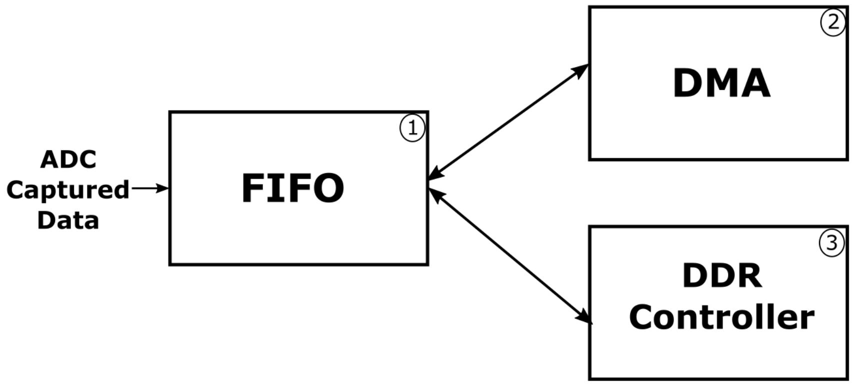


Figure 3.15: Simplified block diagram of the block design for capturing and storing data.

As shown in figure 3.16, the data is packed 4 samples per channel and each sample is worth 16bits of data, this means a total of 64 bits per channel by which the two channels together make up a total of 128 bits of data.

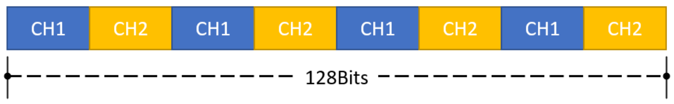


Figure 3.16: Sample distribution.

For a sampling rate of 1 GHz and loading 4 samples in parallel, this means that data is streaming at 128 bits@250MHz, referenced as 1 on the figure 3.15 as the FIFO, but the DMA, referenced as 2 on the figure 3.15, reads the data at a much lower rate of 128bits@100MHz, Analog Devices solution to this problem is by offloading the incoming data into the Programmable Logic (PL) DDR3 Memory up to a maximum of 1 Gigabyte of data referenced in 3 on the figure 3.15 as the Memory Interface Generator [58]. However, evaluating precisely how complex these blocks, created many variables. It looked far to complex for our purpose. As such, one solution is by applying simpler blocks where the objective was to use the signal from the single shot module and apply this signal to state machine that would consequently trigger the capture and deposit a set number of samples into a memory block.

The starting point was developing a state machine that would control this process, and so the following Finite State Machine (FSM) was created as shown in the figure 3.17.

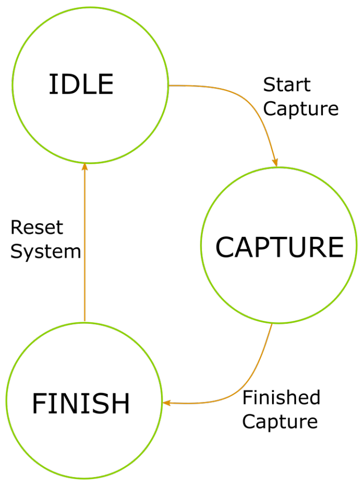


Figure 3.17: Finite state machine diagram for the ADC.

The FSM consists of 3 simple states, `IDLE` state is the starting state and is also where the systems waits for some trigger condition, `CAPTURE` state where a set number of samples should be deposited and a `FINISH` state to stop the capture and reset the system. The next step is by constructing and signalling the correct modules that would make up a fully functional ADC capture module. The next modules are required.

1. A memory block to store the samples from the ADC.
2. A counter to track the samples that are being stored and the memories position.

The first versions of the implementation used a memory written in VHDL for test purposes, the memory has a standard interface shown in figure 3.18.

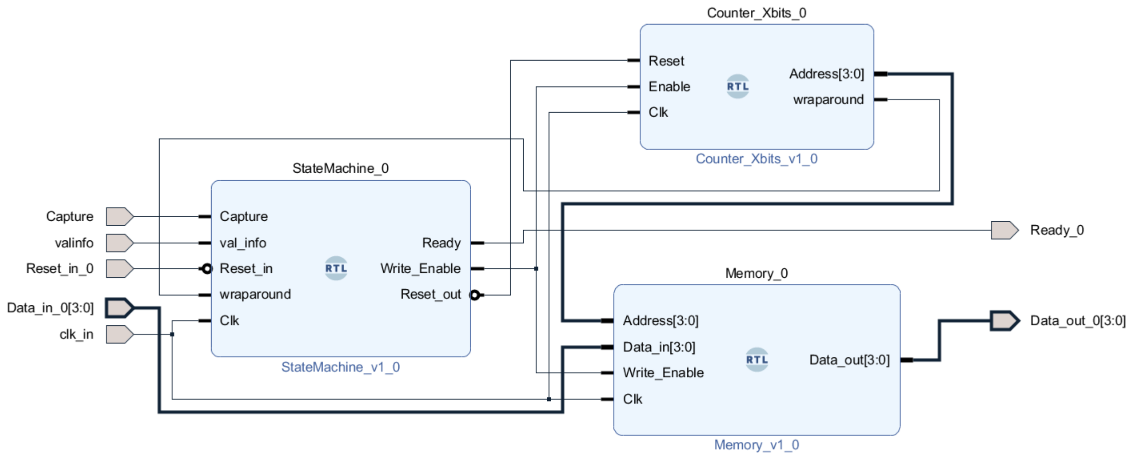


Figure 3.18: Memory controller

The table 3.1 is a brief description of the effects of the FSM on the counter and the memory block.

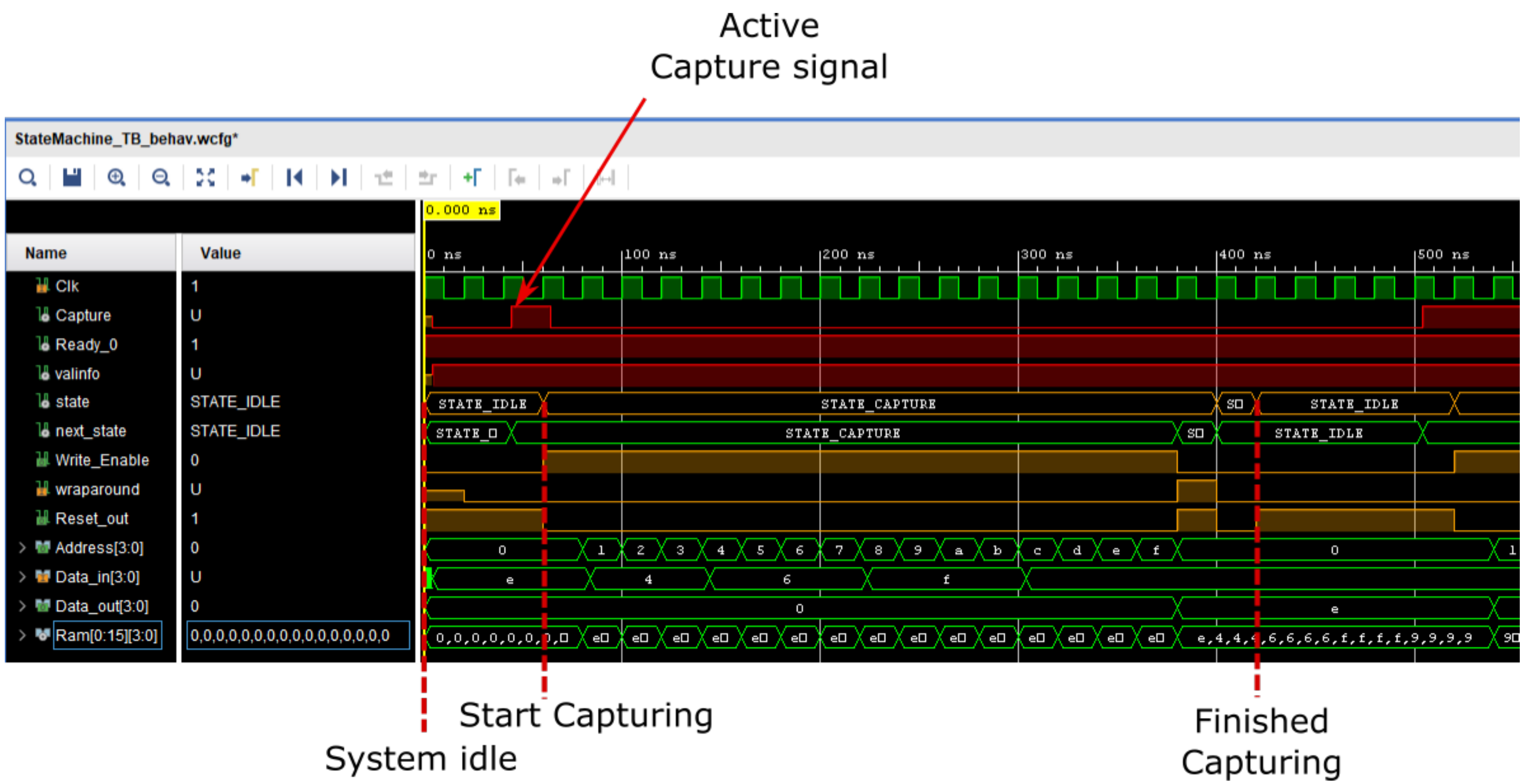
Current State	Next State	Memory	Counter	Transition condition
IDLE	**CAPTURE**	No action	Reset Counter	Capture[7] signal detected
CAPTURE	**FINISH**	Enable Memory Write data	Enable Counter Increment Counter	Wraparound[8] signal detected
FINISH	**IDLE**	Disable Memory	Reset Counter	Automatic

Table 3.1: Memory controller actions.

The results are shown in figure 3.19 of the testbench from Vivado.

[8]In this context, this is the same as the single-shot signal.

[8]This signal is set when the counter hits the maximum value attainable.



Name	Value
Clk	1
Capture	U
Ready_0	1
valinfo	U
state	STATE_IDLE
next_state	STATE_IDLE
Write_Enable	0
wraparound	U
Reset_out	1
Address[3:0]	0
Data_in[3:0]	U
Data_out[3:0]	0
Ram[0:15][3:0]	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Figure 3.19: Testbench - memory controller in a simulated environment.

The current memory could not be used as the user had no way to access this memory. The trick was substitute the current memory for a Xilinx native IP block known as Block Random Access Memory (BRAM), which also had a standard interface, and a BRAM controller that would allow the BRAM to be connected to the Zynq via Advanced eXtensible Interface (AXI)-Interconnects. The connections shown in the figure 3.20 were set to allow the user to access the BRAM.

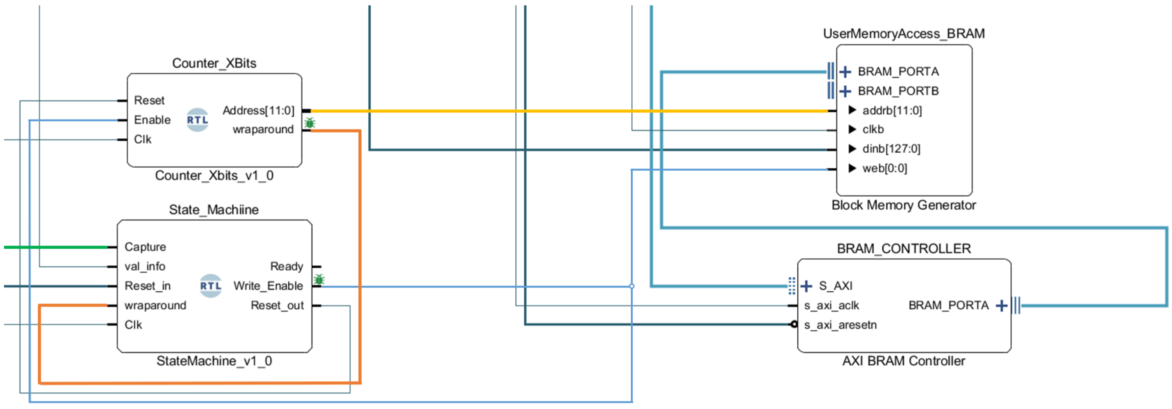


Figure 3.20: Memory controller connected to the BRAM.

Highlighted in green is the single-shot signal actings a a trigger mechanism.

3.4.3 Python user interface

One of the requested tasks was to have a user interface that would use Python. Knowing Vivado uses Tool Command Language (TCL) allows for interacting with design tools to generate the bitstream and load into the FPGA. The trick was to write TCL script that would take the generated bitstream and load onto the FPGA by using Python to execute the TCL script. However, the data required for it to be read and displayed graphically. The Python script was further developed to connect and read from the serial line in which the FPGA would output the current captured values in memory when a single shot would be triggered, when would then store these values values in a buffer on the computer which would consequently be displayed graphically.

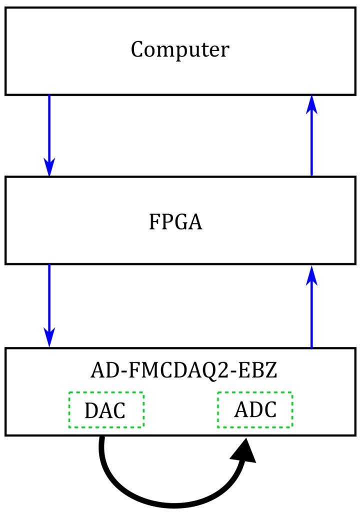
The table 3.2 shows the various commands at which the user can use to manage the interface and communicate with the DAC and ADC.

Command	Description
i	Shows an introductory message.
s	Single shot trigger command.
r	Read samples stored in BRAM.
HG	Display graph of the read samples.
x	Terminate program.

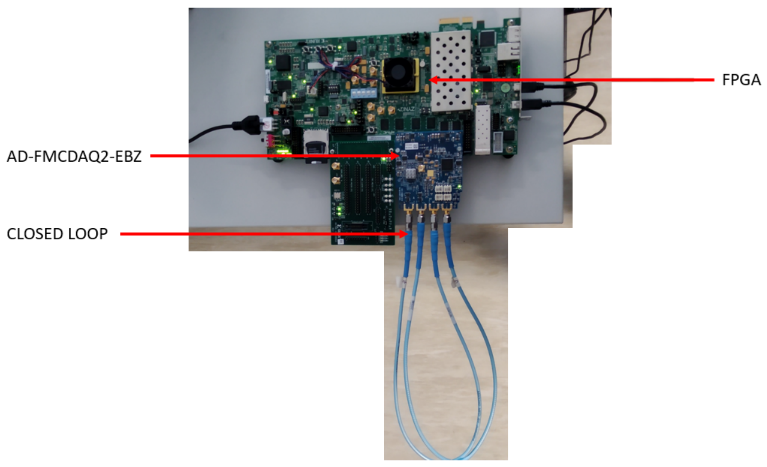
Table 3.2: User interface operations.

3.5.1 Setup

The experimental setup for capturing and displaying the generated waveform is shown in figure 3.21.



(a) Setup diagram.



(b) Labeled setup photo.

Figure 3.21: Experimental setup.

Figure 3.21 shows the AD-FMCDAQ2-EBZ in a closed loop configuration where each channel of the ADC and DAC are directly connected to each other via coaxial cables.

3.5.2 Results

Results are shown in figure 3.22.

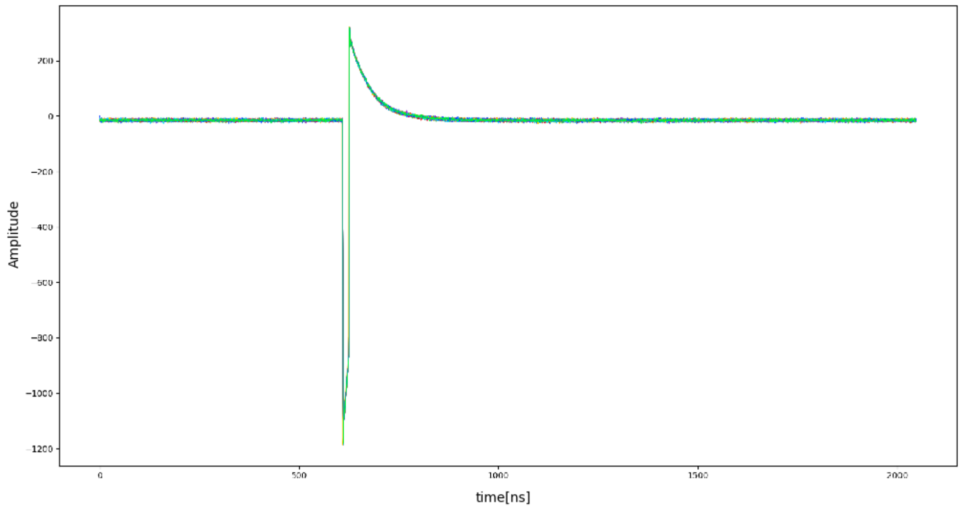


Figure 3.22: 50 waveforms generated by the DAC and captured by the ADC, with both operating in single-shot.

Waveforms are perfectly overlapped for the rising and falling edges, which means that jitter is lower than the original sampling period of the ADC, of 1 ns. The fact that multiple captured waveforms, each comprising a single pulse, are overlapped means that single-shot and synchronous operation was successfully achieved.

Resource	Utilization	Available	Utilization(%)
LUT	27234	218600	12.46
LUTRAM	527	70400	0.75
FF	31349	437200	7.17
BRAM	10.50	545	1.93
DSP	16	900	1.78
IO	40	362	11.5
GT	4	16	25.00
BUFG	3	32	9.38

Table 3.3: ADC - FPGA resource usage on the ZC706.

The table 3.3 shows a great resource usage and a total on chip power usage of 4.871 W derived from the power analysis of post-implemented design.

CHAPTER 4

TDC-based front-end

TDC techniques and thermometer encoders were reviewed in chapter 2. From all techniques, we chose to implement a TDC based on a TDL and mux-based thermometer decoder.

As such, this chapter describes the system architecture at a conceptual level. Key constructs are identified, including significant architectural elements such as components and relationships among them, as well as architectural mechanisms.

4.1 System overview

Figure 4.1 is an overview of the proposed TDC architecture using the Tapped Delay Line topology at the center of the systems architecture.

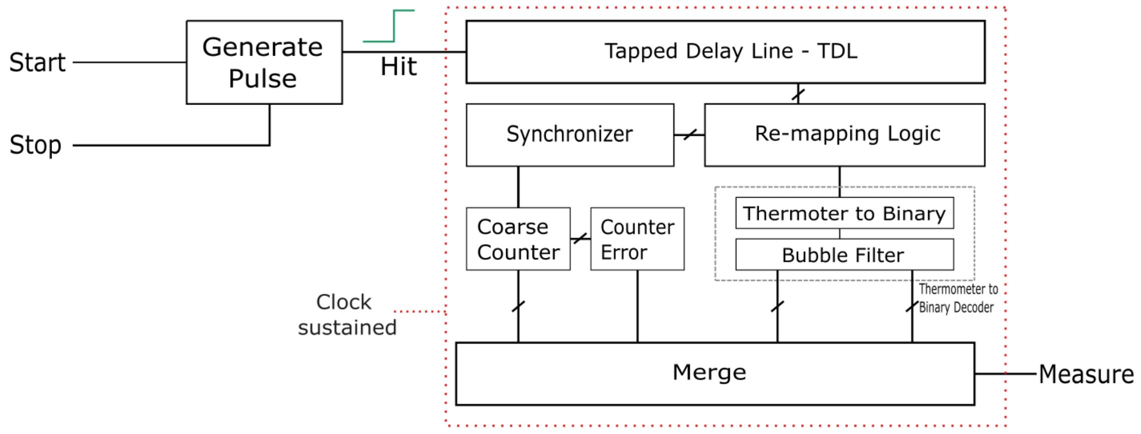


Figure 4.1: System architecture overview.

The system is composed of:

- **Generate Pulse:** This is the systems "kick-starter" whose objective is to deliver a constant pulse, known as the `hit` signal, which is set to a logical `1` the moment the `start` signal is set and is set to a logical `0` the moment the `stop` signal is set.

- **Tapped Delay Line:** The "heart" of the system, the whole system is based on the TDL architecture and this block is responsible for evaluating the delay line by periodically updating the systems current delay-line state.

- **Re-mapping Logic:** This block stores 2 specific instances of the delay chain, where one is the time the delay line is filling and the other is when the delay line is emptying.

- **Synchronizer:** The "brain" of the system, this block deals with signalling the Remapping logic and the Coarse Counter to operate at certain instances according to the state of the delay line.

- **Coarse Counter:** The coarse counter is a very popular module in implementations of the TDL and is completely responsible for counting the number of clock cycles that the TDL is full.

- **Counter error:** This is a simple block that is responsible for limiting the maximum measurement range of the Time-to-Digital Converter.

- **Thermometer-to-Binary Decoder:** This block is responsible for acquiring the thermometer codes stored in the Re-mapping Logic and converting the thermometer codes into a binary word.

- **Merge:** This block is the last block of the system that delivers a measurement in picoseconds. The role of this block is to calculate ToF based on the value of the coarse counter and the two binary codes of the decoded thermometer codes stored in the Re-mapping logic.

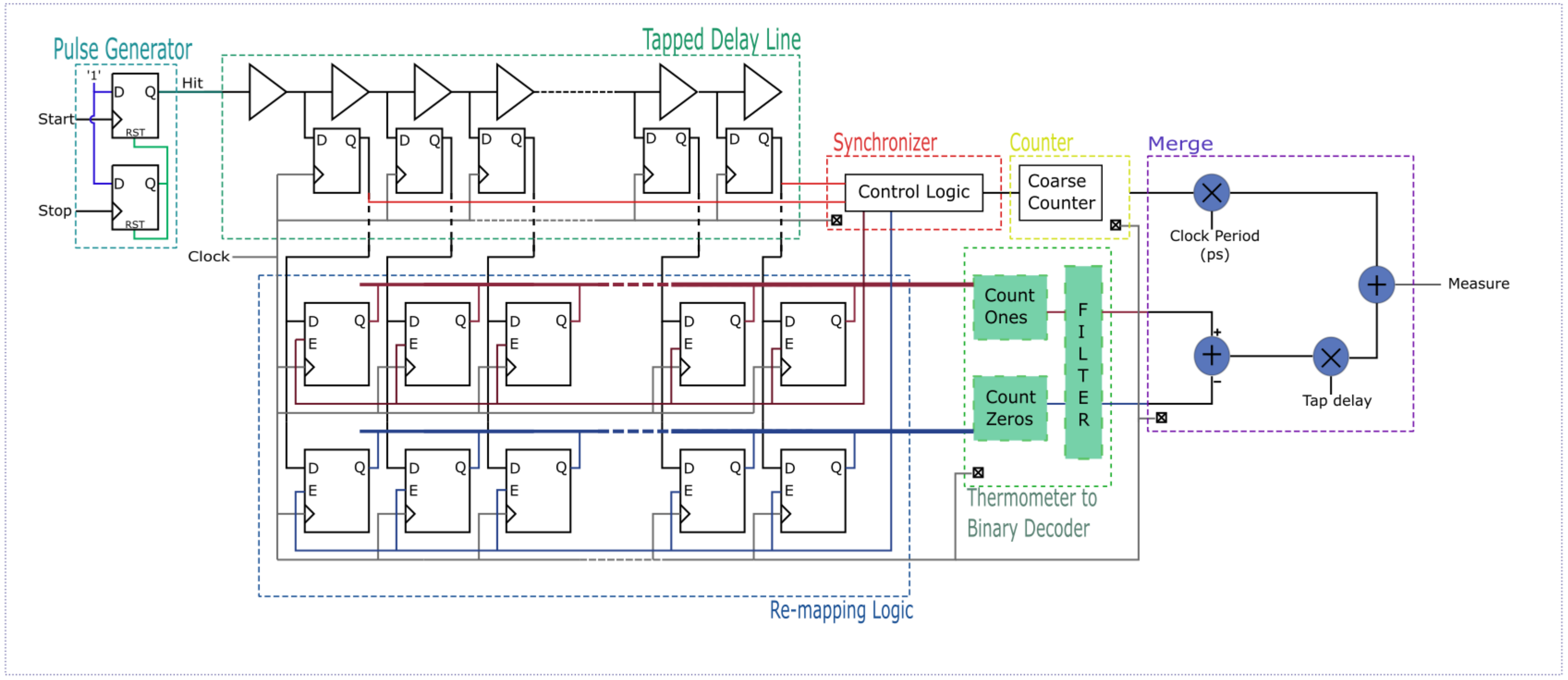


Figure 4.2: Simplified RTL block diagram of the system architecture.

4.2 Pulse generator

This section is focused on the Pulse Generator.

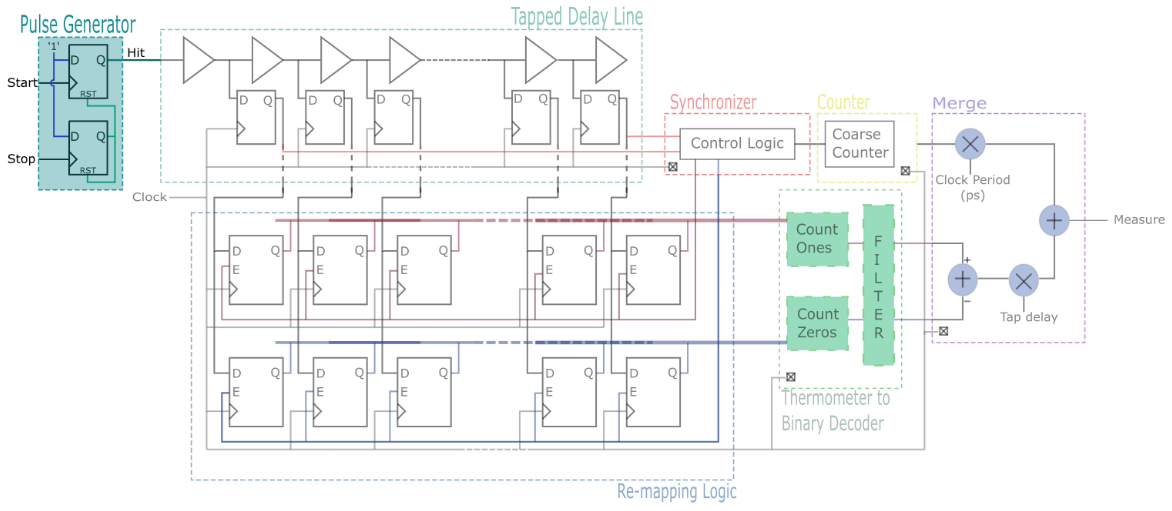


Figure 4.3: Pulse Generator block.

The system must be able to measure the width of the `hit` signal which is generated via the Pulse Generator block.

The behaviour of the Pulse Generator block can be seen in the figure 4.4.

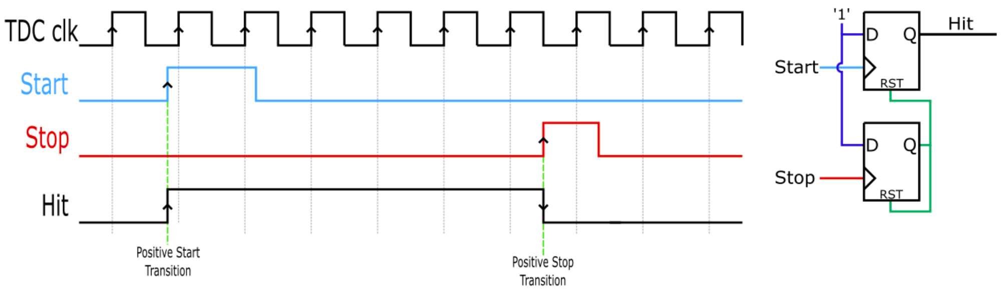


Figure 4.4: Pulse Generator timing diagram

From the figure 4.4, on the rising edge of the `Start` signal, referenced in light blue, the `hit` signal is set to a logical `1` and the rising edge of the `Stop` signal referenced in red sets the `hit` signal to a logical `0`.

In technical terms, at the rising edge of the `Start` signal, the D Flip-Flop fixed with a logical input `1`, sets the output of the D Flip-Flop equal to its input and retains this value until it is reset. The reset is controlled by another D Flip-Flop fixed with a logical `1` at its input and at the rising edge of the stop signal, its output will equal its input resetting both D Flip-Flops and as a result the `hit` signal will be set with a logical `0`. Note that it possesses an asynchronous behaviour as it is not dependent on any clock. The `TDC clk` is there to show that this block is uncorrelated to the clock signal. To note, that the `Stop` signal could have been used to trigger the asynchronous reset however, the system is based off the edge detection of both input signals and the reset mechanism is more complicated than what is illustrated in the figure 4.4 as the `hit` signal can be reset under certain circumstances which is covered in the implementation.

4.3 Tapped delay line

This section is focused on the Tapped Delay Line.

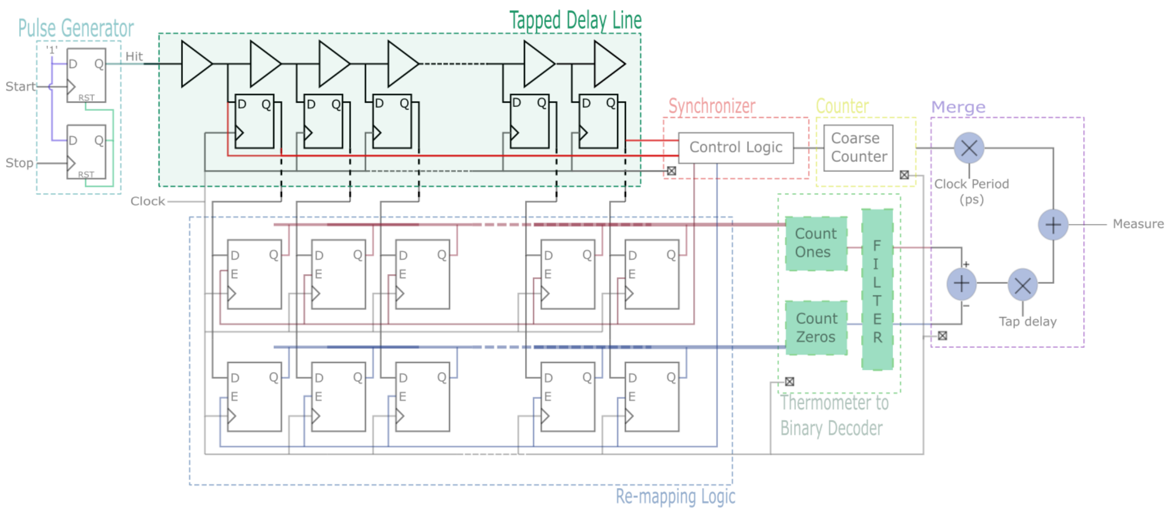


Figure 4.5: TDL block.

For this architecture, the time it takes to traverse the whole delay line, must be close to and bigger than the clock period, as the tap delay line should act to represent the sub-resolution of the system. The figure 4.6 can be seen as ruler that measures time, in which the 2 ns jumps shown represents the systems major resolution, represented by the clocks period, and the residual measure, known as a fine measure or minor resolution is measured by knowing how many digital cells were crossed over by a signal.

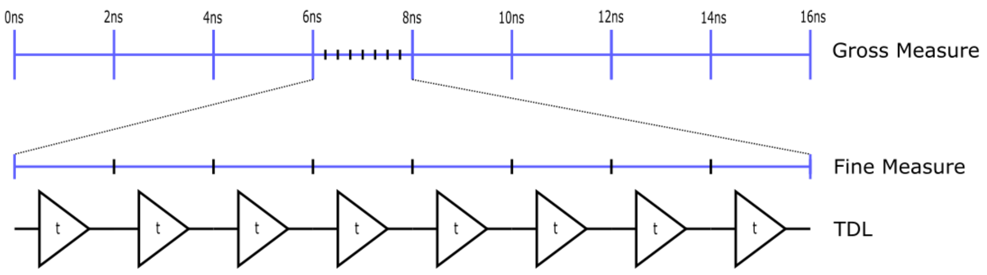


Figure 4.6: Timing resolution expansion.

4.3.1 Digital cell

This architecture uses 4 cascaded digital adders, also known as Ripple Carry Adders (RCAs) blocks as shown in figure 4.7.

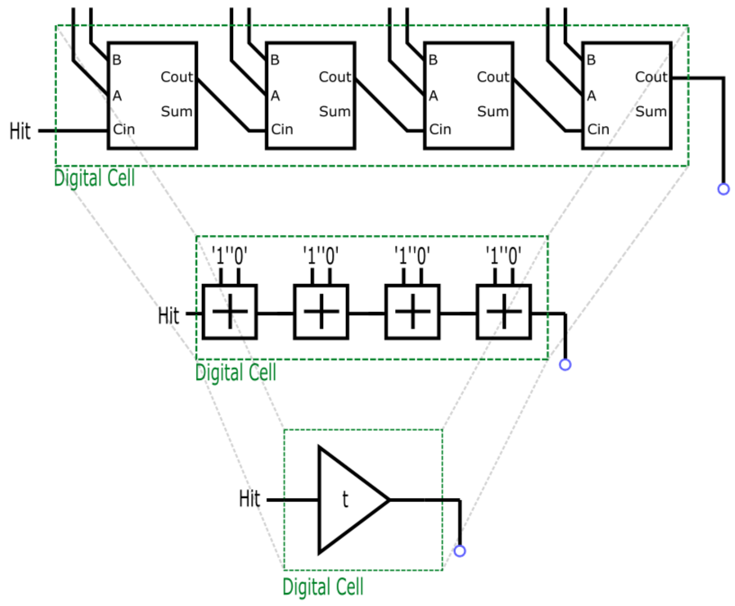


Figure 4.7: Digital cell.

In figure 4.7 are the cascaded blocks of adders. A question that follows the previous statement is, how do those combinatorial blocks propagate the input signal? The answer is by looking at the table 4.1, the fixed input A and B as `0` and `1` respectively, creates a buffer between the `Cin` and the `Cout`, the `Cout` always inherits the value of the `Cin`. In conclusion, when the `Hit` signal is a logical `1` then four taps later the output is a logical `1` and when the `Hit` signal is a logical `0` then four taps later the output is a logical `0`, each 4 taps is a digital cell of the delay line.

A	B	Cin	Cout
0	1	1	1
0	1	0	0

Table 4.1: Carry Lookahead Adder truth table.

This concludes this Digital Cell section.

4.3.2 Tapped delay line topology

Understanding the digital cells is a crucial step to understanding how the delay line is constructed and its principle of operation. A fully functional delay line is created by cascading a set number of digital cells. For this topology, it is very important that the total delay imposed by the delay line is smaller than and close to the sampling period as the idea is to have the delay line slicing a clock period into a set number of digital cells.

For simplicity, in figure 4.5 painted in green on the section Tapped Delay Line there are 5 small boxes which contain the current value of a digital cell and when cascaded represent the delay line. The boxes act as the delay line samplers storing a “photo” of the state of the delay line at each positive clock edge presented in figure 4.8. For simplicity and example, the resolution is dictated by only 8 digital cells and thus only 8 containers are required.

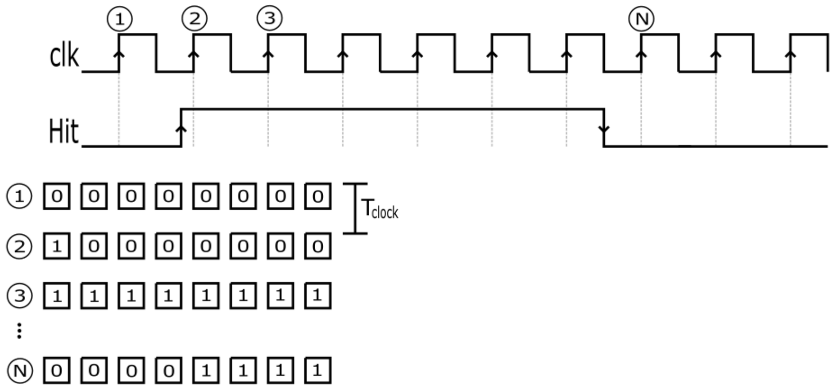


Figure 4.8: Ideal representation of the system sampling the hit signal through the delay line.

From the figure 4.8, on the first sample of the `hit` signal, the delay line state is all 0, on the second sample of the `hit` signal there is one container with logical `1`, on the third sample of the `hit` signal, the delay line is full and continue full until the Nth sample, where the delay line is half empty and on the (N+1)th sample, delay line state would be the same as the first sample which is empty. This explains how the behaviour of the delay line adapts to the `hit` signal.

4.4 Synchronizer

This section is focused on the Synchronizer.

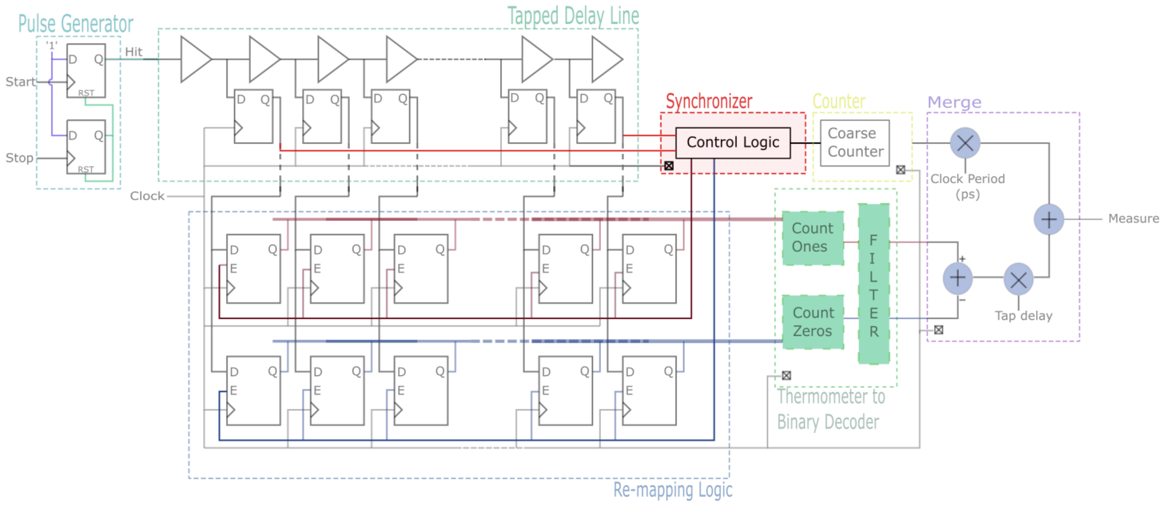


Figure 4.9: Synchronizer block.

The Synchronizer is the head of the TDC architecture, deals with signalling the enabler of the counter and the enablers of the Remapping Logic at very specific instances.

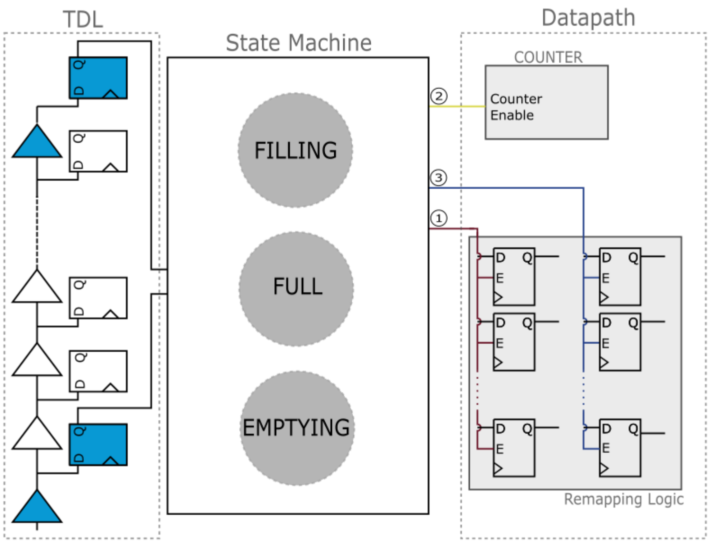


Figure 4.10: Synchronizer datapath.

In the figure 4.10, the Synchronizer can be seen as a state machine in which its states are dependent on the first and last elements of the delay line and the current state.

When evaluating the TDL, there's actually 4 possible states:

- Filling
- Emptying
- Full

- Empty

As shown in the figure 4.11, the first time the system looks at the hit signal, it is seen as filling, then it is full and then it is emptying, finally, after that, it is empty and the cycle repeats itself when the `hit` signal appears. The empty cycle can be represented as a idle state as no event is is occurring and as such is omitted in the figure 4.10.

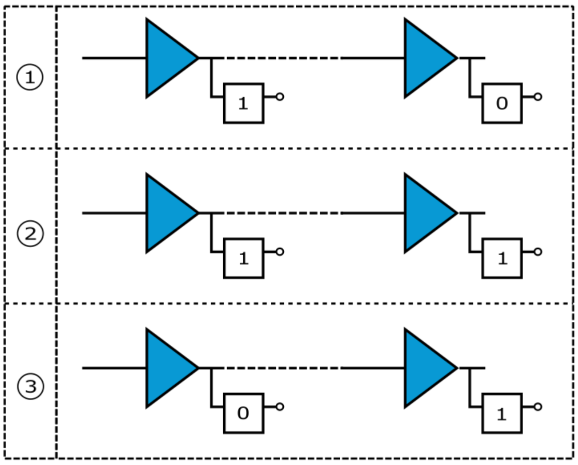


Figure 4.11: Tapped delay line states.

The figure 4.11 is an illustration of the first 3 states mentioned, namely, one is filling, two is full and three is emptying. When evaluating the delay, the only states that are important are when it is filling, case 1, that means that the first and last element are `1` and `0`, respectively. Case 2 states that the TDL is full when the first and last element are `1`. Finally for case 3, which is when the TDL is emptying so its first and last element are `0` and `1`, respectively, thus only one of those 3 states can be active at once.

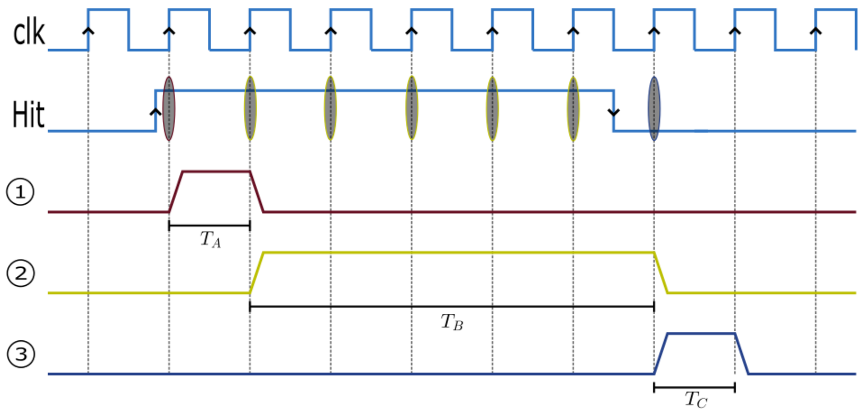


Figure 4.12: Synchronizer timing diagram.

On the figure 4.12, this particular hit signal behaviour can now be evaluated. The zone circled in red represent case 1 of the delay line state when it is filling and will last for a duration of T$_A$, where T$_A$ and T$_C$ is the duration of one clock period. In terms of clock periods, it is quite intuitive to evaluate part of the `Hit` signal which is at least 5 clock periods long, that means that the counter enable, seen as the second case in the timing diagram should stay enabled for at least 5 clock periods, which is T_B long, finally, storing the last part of the `Hit` signal as shown in blue in the timing diagram will sample the final state where it is emptying and sets its enable for a duration of T_C.

4.5 Re-mapping logic

This section will be focused on the Remapping Logic.

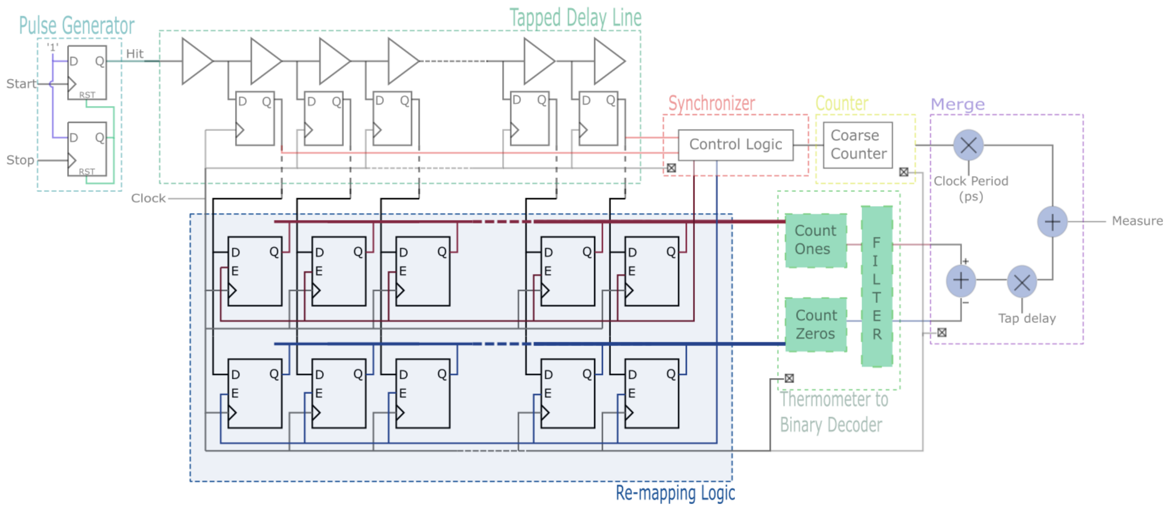


Figure 4.13: Remapping Logic block.

The remapping logic serves to save specific states of the delay line, from the figure 4.8, they would be the Nth and 1st state which can also be a representation of sampling the first and last pieces of the `Hit` Signal, highlighted in red and blue on the figure 4.14

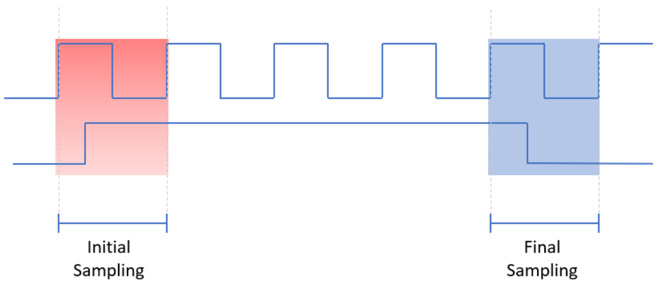


Figure 4.14: First and last sampling stages of the hit signal.

From the figure 4.14 these are instances of which the synchronizer must be able to identify and use its control signals to correctly identify and end of the signal must be stored.

4.6 Counter

This section is focused on the Counter.

The counter is another module who receives one of the Synchronizer's control signal which must be able to identify the time at which and for how long this module should be stay enabled.

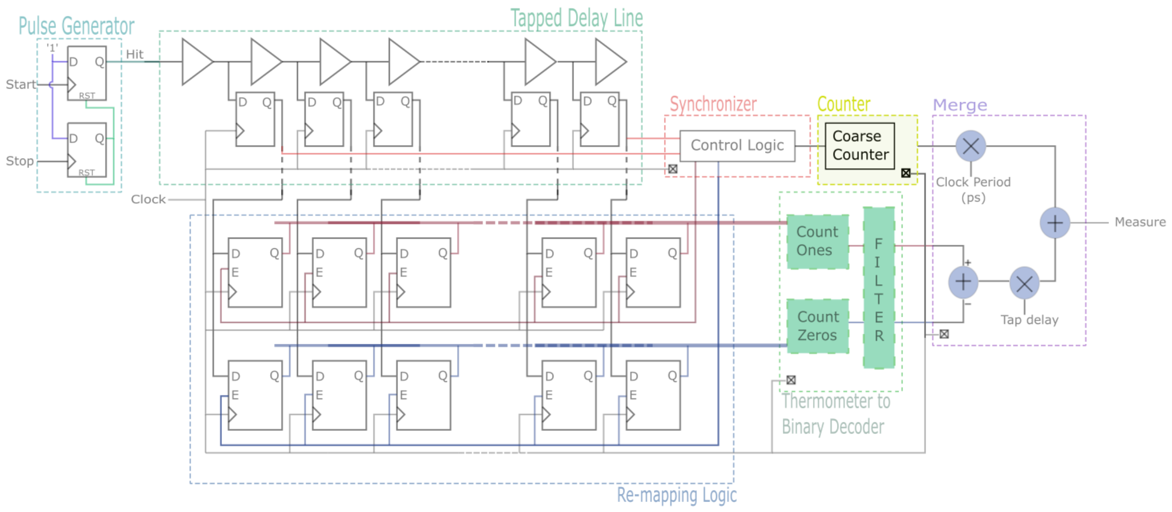


Figure 4.15: Counter block.

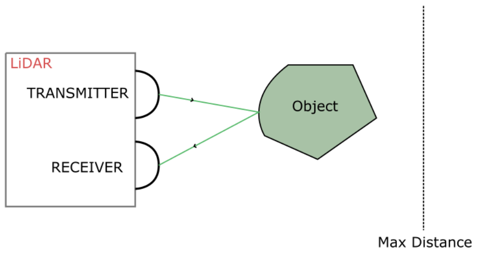
Counters can be made asynchronous or synchronous, in each of these types, is an up counter or a down counter. This architecture uses a synchronous up counter as the objective is to count the number of clock cycles that the tapped delay line is full.

4.7 Counter error

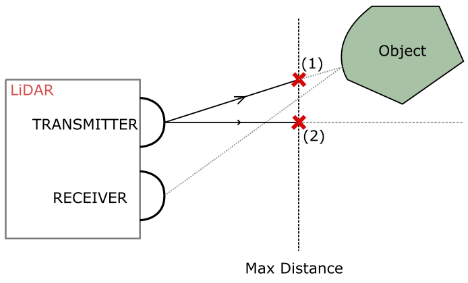
This section is focused on the Counter Error.

Unlike the previously module, this module only appears on the overview architecture. This module's function is the simplest out of all the modules.

Like many sensors similar to the LiDAR, which have a receiver and sender, the device must be able to emit a signal and receive the signal. What if after emitting the signal there is no notification of the reflected signal? In the figure 4.16, where the first situation is when the device receives the signal and the second situation, which can actually be interpreted into 2 possible scenarios, one in which the signal was never reflected to the receiver and the second case scenario, where the signal was reflected but is already above the maximum distance, as such, the solution is to discard the signal.



(a) Signal conditions met.



(b) Signal conditions not met.

Figure 4.16: Signal travelling conditions.

To discard the emitted signal seen on the right side of the figure 4.16 when the signal conditions are not met is by analysing the direct relationship between the maximum distance, the speed of light, in others words, the propagation velocity and the rate at which the counter is updated then it is possible to infer how much the counter has to count until it signals when the distance is above a certain threshold by using a simple formula.

$$T_{\text{trip}} = \frac{d}{c}, \tag{4.1}$$

where d is the maximum travelling distance of the signal in meters, c is the speed of light in $\mathrm{m\,s^{-1}}$ and finally T is the trip time to reach the distance, d, in seconds.

As T_{trip} is related to the counter by the relationship between the T_{trip} and T_{clock} which can be described in the following formula:

$$N_{\text{counts}} = \frac{T_{\text{trip}}}{T_{\text{clock}}}, \tag{4.2}$$

for the case of this section, N_{counts} is the number of counts required to reach the maximum signal trip time allowed and T_{clock} is the systems clock period.

4.8 Thermometer-to-Binary decoder

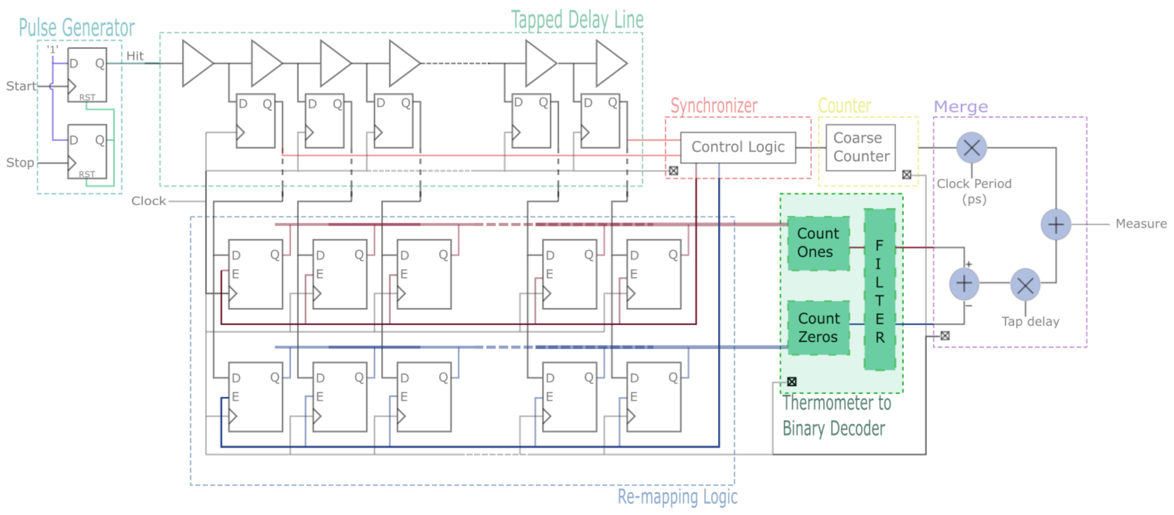


Figure 4.17: Thermometer to Binary Decoder block.

The Thermometer-to-Binary converter uses the standard Mux Based Encoder. However it is a pipelined version of the muxed based encoder where at the end there is a one counter to get around the bubbles. Pipelining increases stability and allows for automatic scaling without worrying about timing constraints which greatly aids the synchronous system.

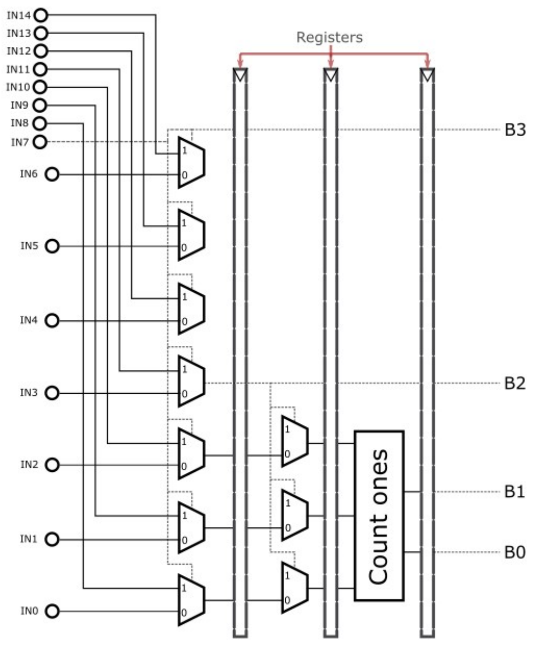


Figure 4.18: 14-bit pipelined mux based encoder with a counter in the final stages [59].

This module is used for a ones counter. The zero counter also uses the same module but instead subtracts the number ones from the total number of bits.

4.9 Merge

This section is focused on the Merge block.

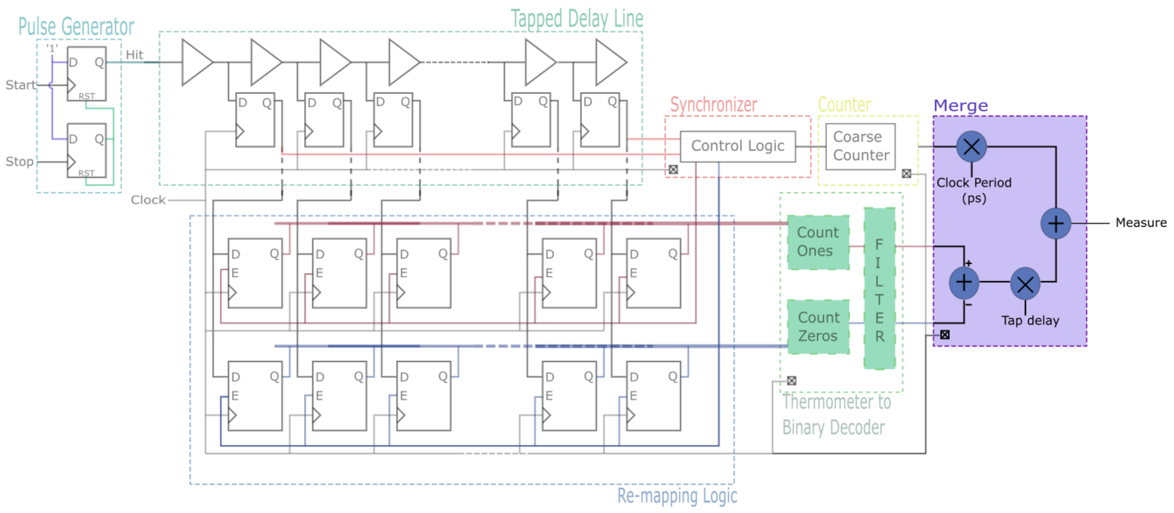


Figure 4.19: Merge block.

The Merge is where all the important information converge at. The merge takes in the value of the counter and the values at the binary word of the stored thermometer codes and is be able make a precise measurement of the ToF. The following steps will breakdown the formula that makes the precise measurements.

$$T_{\text{measure}} = T_{\text{clock}} \cdot N + (Fine_{\text{start}} - Fine_{\text{stop}}) \cdot T_{\text{delay}}, \tag{4.3}$$

where, T$_{measure}$ is the output in picoseconds, N is the counters values is given by $N = N_{\text{full}} + 1$, N_{full} is the number of times the hit signal is seen as full, T$_{\text{clock}}$ is the systems clock period, Fine$_{\text{start}}$ is the number of ones calculated from first stored thermometer codes, Fine$_{\text{stop}}$ is the number of zeros from the second stored thermometer codes and T$_{\text{delay}}$ is the time it takes to traverse one digital cell.

To review some important concepts so far:

- The system is composed of 3 important states shown in figure 4.11 when evaluating the hit signal.
- The counter provides the amount of times the hit signal was seen as full.
- The remapping logic stores the states in which the hit signal is seen as filling and emptying as shown in figure 4.20.
- The systems operates at a given frequency and the size of the delay line should be bigger than and close to the systems clock period.
- The Thermometer-to-Binary converter counts the number of ones in the filling state and the number of zeros for the emptying state.

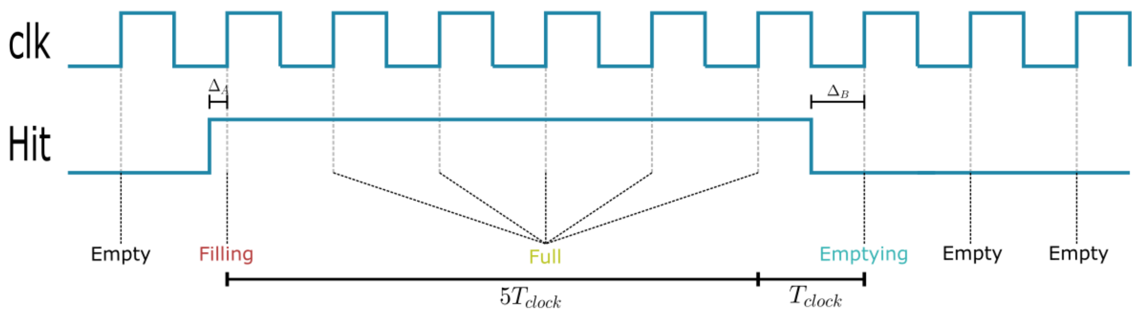


Figure 4.20: Observation of System States

From (4.3), on the figure 4.20, the 5+1 will the be the counters value and T_{clock} is the clocks period. The $(5{+}1)T_{\text{clock}}$ part of the equation gives us the gross estimate of the measurement. The mentioned value 1 in the section of the counter is directly related to the calculations ahead.

In the figure 4.21, the first case takes the ends of the sampled hit signal for the fine measurements calculations.

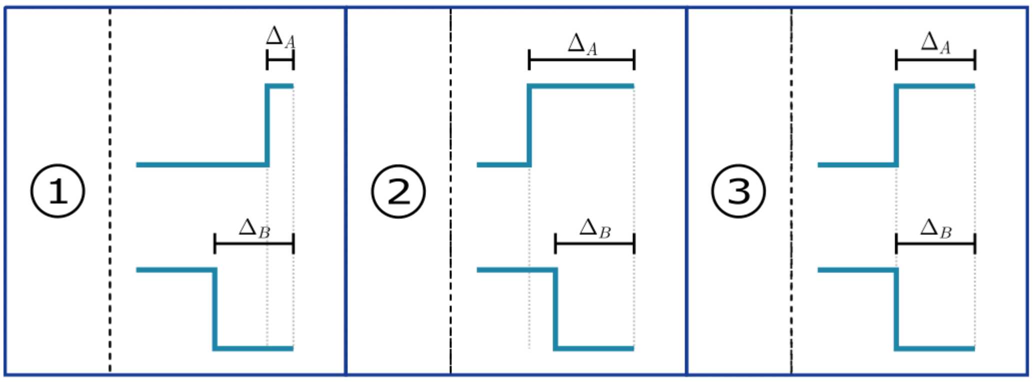


Figure 4.21: Possible Case Scenarios of the First and Last Piece of the TDL

In the first case of the figure 4.21, $\Delta_{\text{A}} < \Delta_{\text{B}}$. In this particular case where there's Δ_{B} space available from T_{clock} space, then the remaining space of when Δ_{A} is added, can be given by:

$$\Delta_{\text{rem}} = \Delta_{\text{B}} - \Delta_{\text{A}}, \tag{4.4}$$

Where delta Δ_{rem} is the remaining free slot of time available, shown in the figure 4.22.

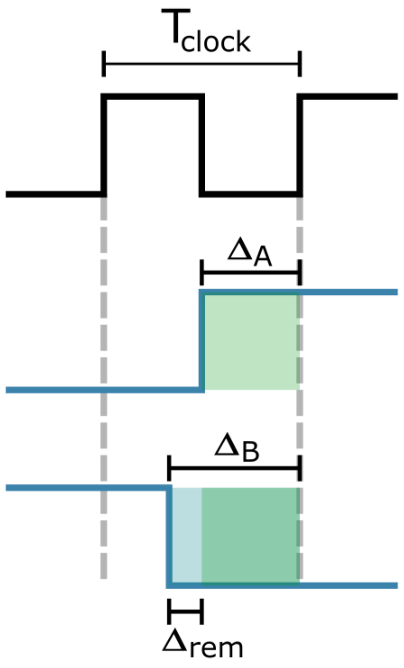


Figure 4.22: Fine Measurement Analysis

Relating Δ_{rem} to T_{clock} for the fine measurements can be given by:

$$T_{\text{fine\_measure}} = T_{\text{clock}} - \Delta_{\text{rem}} = T_{\text{clock}} + \Delta_{\text{A}} - \Delta_{\text{B}}, \tag{4.5}$$

Where $T_{\text{fine\_measure}}$ is the slot of time occupied by the time of the first and last piece of the hit signal when it is on. This last equation is the fine measurement of the equation.

Next step is to formulate and condense the original equation of the gross estimate and the fine estimate.

$$T_{\text{measure}} = T_{\text{clock}} \cdot N_{\text{full}} + T_{\text{clock}} + \Delta_{\text{A}} - \Delta_{\text{B}}. \tag{4.6}$$

The (4.6) originates the value 1 which occurs in the counter and previously mentioned, where T_{clock} is the substitution for the added one. A compressed version of the previous formula which highlights the presence of the number 1:

$$T_{\text{measure}} = (N_{\text{full}} + 1) \cdot T_{\text{clock}} + \Delta_{\text{A}} - \Delta_{\text{B}}. \tag{4.7}$$

The (4.7) is divided into a gross and fine estimate. For the gross estimate:

$$T_{\text{gross\_measure}} = (N_{\text{full}} + 1) \cdot T_{\text{clock}}. \tag{4.8}$$

For the fine estimate:

$$T_{\text{fine\_measure}} = \Delta_{\text{A}} - \Delta_{\text{B}}. \tag{4.9}$$

Resulting in the final equation.

$$T_{\text{measure}} = T_{\text{gross\_measure}} + T_{\text{fine\_measure}} = T_{\text{clock}} \cdot N + \Delta_{\text{A}} - \Delta_{\text{B}}. \tag{4.10}$$

Finally, Δ_{A} is the number of ones of the first piece of the hit signal, also known as $Fine_{\mathrm{start}}$ times the delay of one digital cell and Δ_{B} is the number of zeros of the last piece of the hit signal of the hit signal, also known as $Fine_{\mathrm{stop}}$, times the delay of one digital cell.

To validate the (4.10), the analysis of the figure 4.21 presents 2 extras cases. The third case, $\Delta_{\mathrm{A}} = \Delta_{\mathrm{B}}$, where $N_{\mathrm{full}} = 5$ as shown in the figure 4.20, then $N = 6$ for the gross measure. If both Δ_{A} and Δ_{B} are equal, then Δ_{A} fits in Δ_{B} free time slot and according to the (4.10), where $T_{\mathrm{measure}} = 6$ clock periods, proving the equation is correct. For the second case where $\Delta_{\mathrm{A}} > \Delta_{\mathrm{B}}$, as shown in figure 4.21, case 2, then once again according to the equation (4.10) the result will lead to $\Delta_{\mathrm{A}} - \Delta_{\mathrm{B}} > 0$ and as a consequence $T_{\mathrm{measure}} > 6$ clocks periods which means that the sign is correct. Once again proving the correctness of the equation.

4.10 Summary

The overall system behaviour can be summarized in the following timing diagram shown in the figure 4.23.

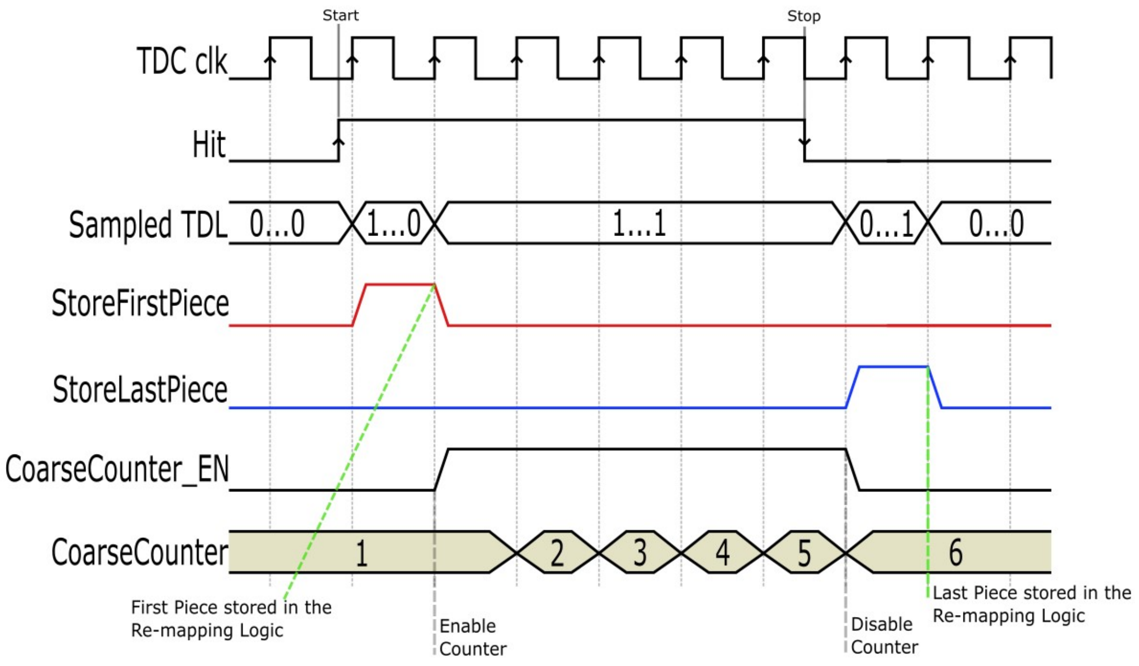


Figure 4.23: System architecture timing diagram.

This architecture features simple control logic with an intuitive state machine and a TDL with a mux based decoder for thermometer decoding. To the best of the author's knowledge, this particular architecture could not be found in the literature. Although there appears to be novelty in the proposed architecture, a comparison with other architectures in terms of performance cannot be given at the moment, as further tests beyond those shown in the following chapter are needed. The main advantages of this architecture are its simplicity and repeatability.

CHAPTER 5

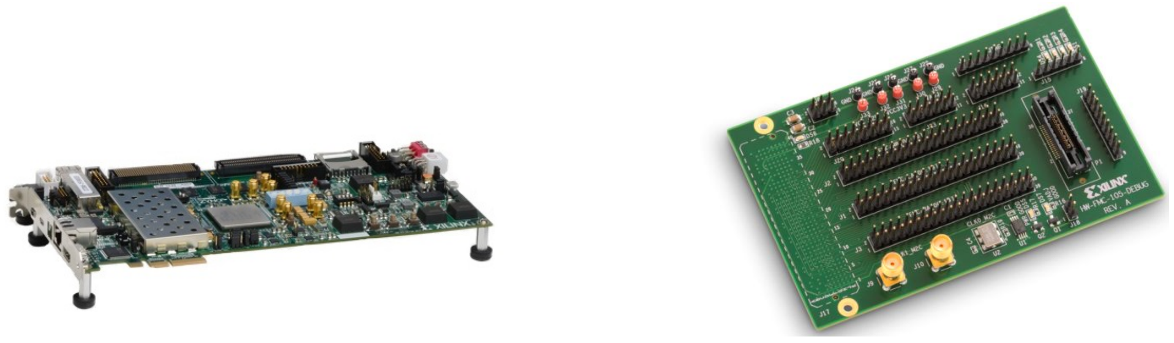
Implementation of a TDC

5.1 Introduction

The chapter presents the implementation of the TDC architecture explained in chapter 4, in which the chapter will start by a brief introduction of the hardware involved along with the Vivado software used to get the project started. Next, the chapter will analyze and explain each module and expose the simulation results.

5.2 Implementation and simulation results

This project was set to operate at 500 MHz and implemented in a ZC706 Evaluation Board together with the HW-FMC-XM105-G debugging board for testing along with the Xilinx Vivado Software Design and its Software Development Kit (SDK).

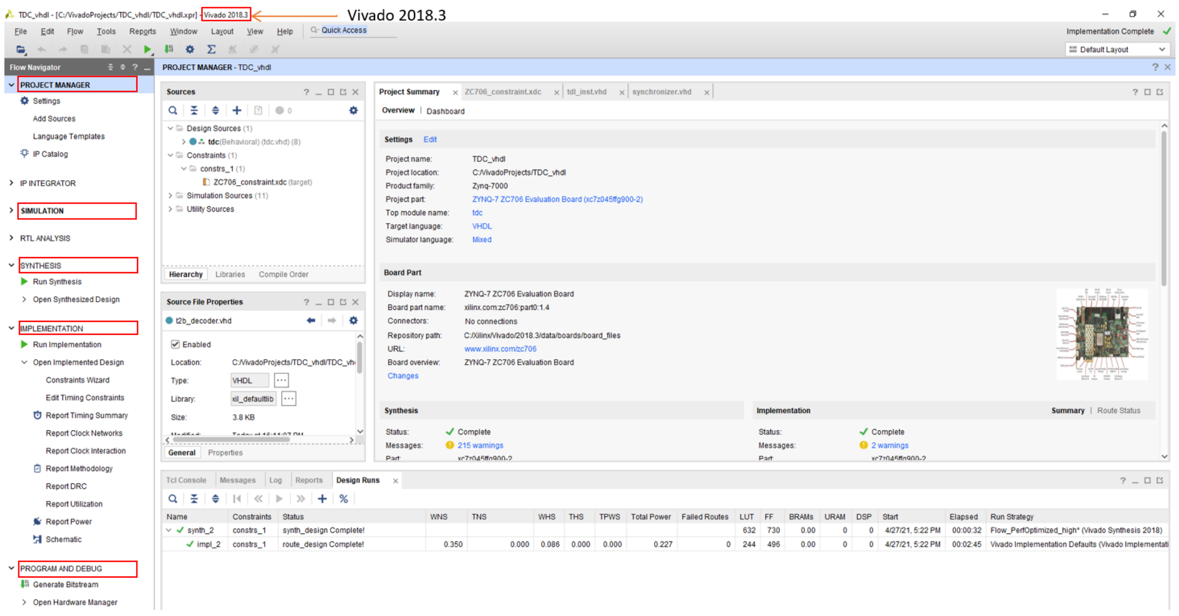


(a) ZC706 evaluation board. **(b)** FMC XM105 Debug Card.

Figure 5.1: Boards used for TDC testing.

5.2.1 Vivado Design Suite

The work here presented was developed with the Vivado Design Suite 2018.3 as seen in figure 5.2 to project the described hardware's behavior onto an FPGA. The Vivado Design Suite integrates 5 important phases when creating and implementing a project. First, it is the creation of the project and specifying the target FPGA from a huge catalogue, in this case, it was the ZC706 Evaluation Board. The second phase includes the projects construction and running the simulation to verify its functionality, third is running the synthesis, fourth is running the implementation and check the reported data such as, temporal characteristics, total resource usage and design rule checking. With all the steps above accomplished, the final step is generating the bitstream and loading a particular file, describing the hardware for one particular FPGA.



Settings

Project name:	TDC_vhdl
Project location:	C:/VivadoProjects/TDC_vhdl
Product family:	Zynq-7000
Project part:	ZYNQ-7 ZC706 Evaluation Board (xc7z045ffg900-2)
Top module name:	tdc
Target language:	VHDL
Simulator language:	Mixed

Board Part

Display name:	ZYNQ-7 ZC706 Evaluation Board
Board part name:	xilinx.com:zc706:part0:1.4
Connectors:	No connections
Repository path:	C:/Xilinx/Vivado/2018.3/data/boards/board_files
URL:	www.xilinx.com/zc706
Board overview:	ZYNQ-7 ZC706 Evaluation Board

Synthesis — Status: ✓ Complete; Messages: 215 warnings

Implementation — Status: ✓ Complete; Messages: 2 warnings

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy
✓ synth_2	constrs_1	synth_design Complete!								632	730	0.00	0	0	4/27/21, 5:22 PM	00:00:32	Flow_PerfOptimized_high* (Vivado Synthesis 2018)
✓ impl_2	constrs_1	route_design Complete!	0.350	0.000	0.086	0.000	0.000	0.227	0	244	496	0.00	0	0	4/27/21, 5:22 PM	00:02:45	Vivado Implementation Defaults (Vivado Implementati

Figure 5.2: Vivado 2018.3 Interface with focus on "Flow Navigator".

5.2.2 Module Implementation in VHDL

All of the following hardware was written in VHDL and the technical structural architecture division is seen in the figure 5.3.

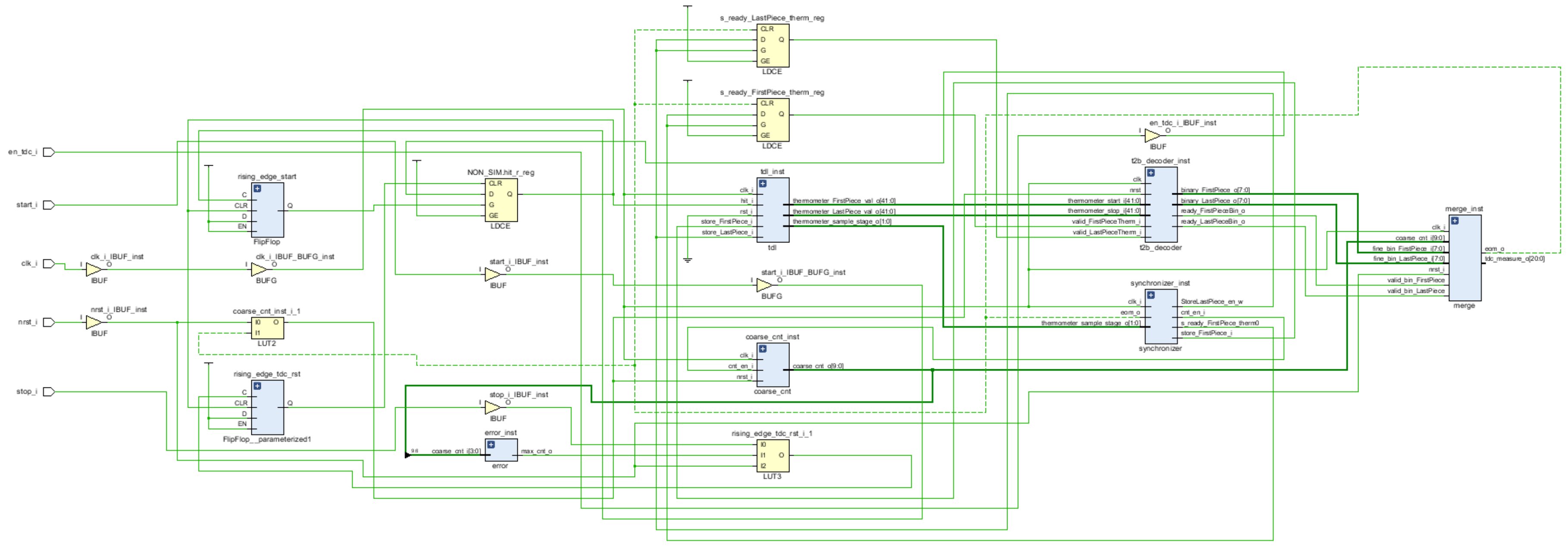


Figure 5.3: System implementation overview.

On the figure 5.3 consists of the combination of the various module, where the `tdl_inst` integrates the implementation of the Tapped Delay Line together with the Re-mapping Logic. The `T2B_decoder_inst` integrates the Thermometer to binary encoder. The `synchronizer_inst` integrates the synchronizer module, also known as the control logic of the architecture, the `coarse_cnt_inst` implements the counter, the `Start_edge_detect` and `Stop_edge_detect` are two external modules to detect activity in the start and stop input of the systems input, very similar to the pulse generator, the `merge_inst` represents the merge block and finally the `error_inst` describes the counter error module.

5.2.2.1 TDL combined with the Re-mapping Logic

The implementation starts by implementing the TDL alone. The module is presented in the figure 5.4.

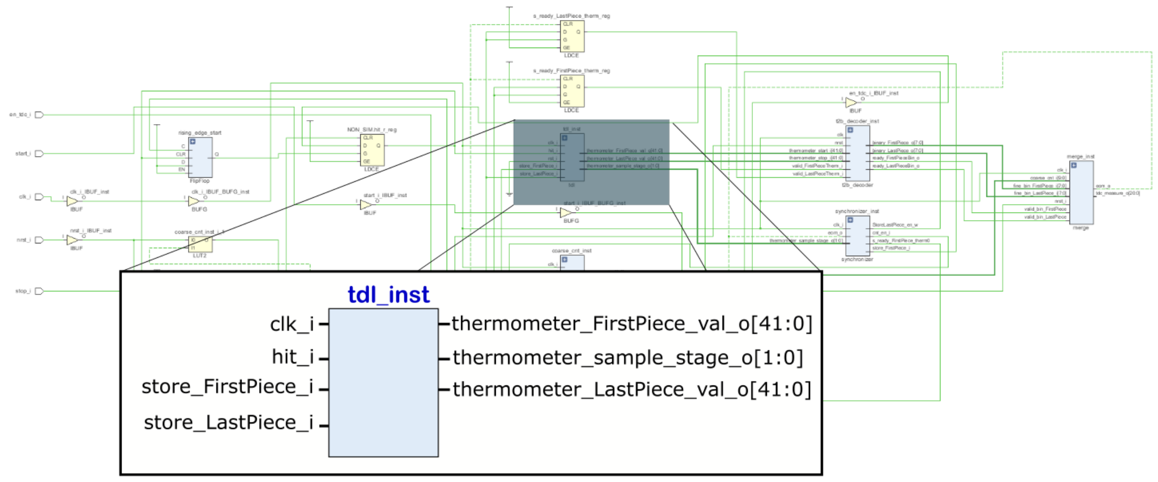

tdl_inst
clk_i
hit_i
store_FirstPiece_i
store_LastPiece_i
thermometer_FirstPiece_val_o[41:0]
thermometer_sample_stage_o[1:0]
thermometer_LastPiece_val_o[41:0]

Figure 5.4: TDL instantiation module.

According to the proposed TDC architecture mentioned in section 4.3, digital cells are based on ripple carry adders. The ZC706 Evaluation board has a primitive that implements 4 bit ripple carry adders, known as CARRY4 Primitives which is shown in the figure 5.5.

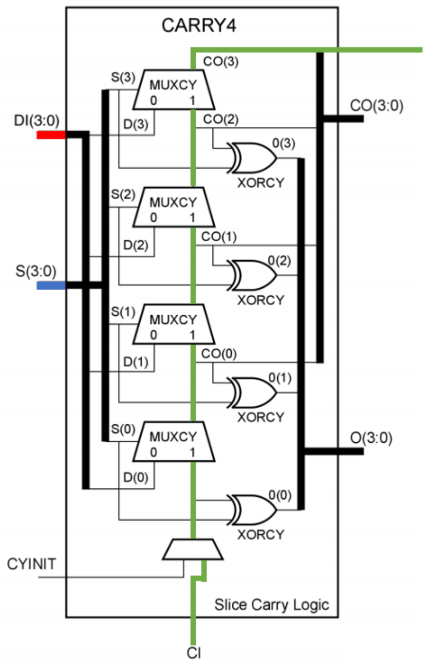
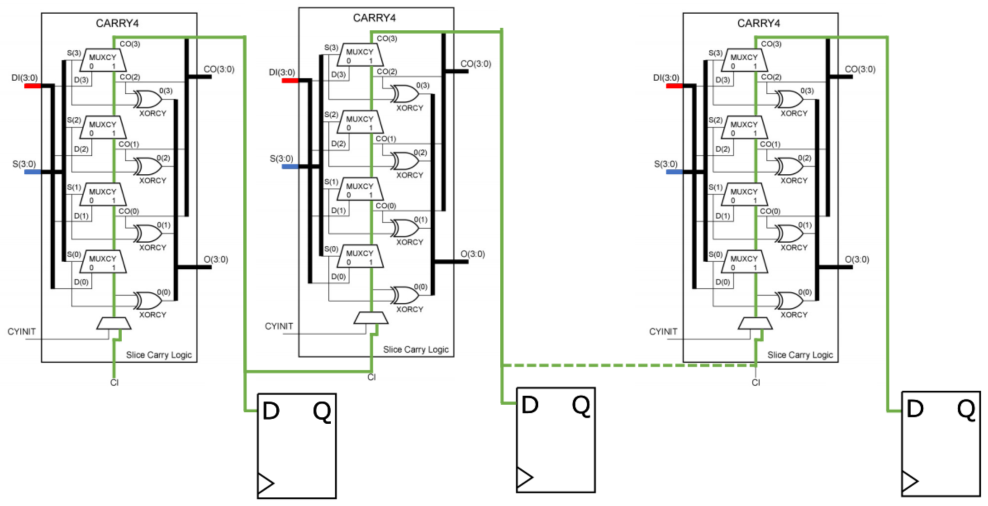


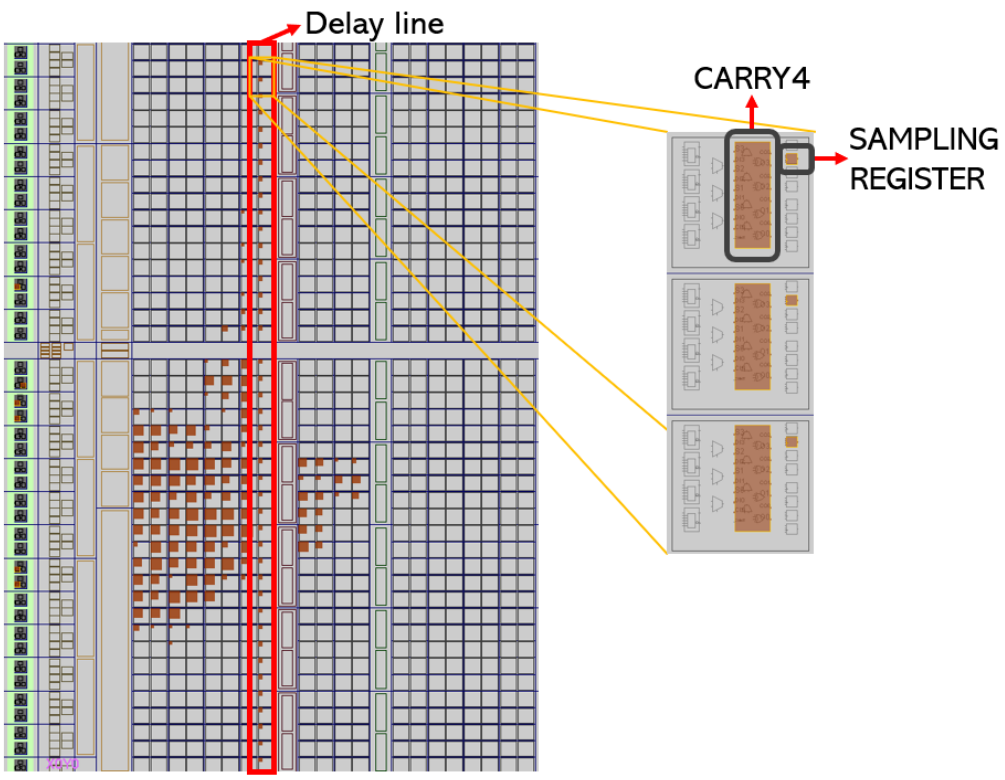
Figure 5.5: CARRY4 Primitive from Zynq 7000 libraries [60].

The CARRY4 Primitive is a digital cell, the red marker means that the input is set to a set of logical `0`s, the blue marker means that they are set to logical values of `1`s. The blue marker represents the multiplexer selector input and is set to `1` to set the green path. In summary, the four cascaded multiplexers represent a digital cell.

Instantiating one of these in VHDL is a rather trivial process as that is available in the Zynq7000 HDL library documentation. Cascading these CARRY4 primitives results in the construction of the TDL.



(a) Carry4 Delay line instantiation.



(b) Delay line FPGA layout.

Figure 5.6: Tapped Delay line representing both conceptual and actual implementation.

The VHDL code for the Tapped Delay Line can be described as the following:

```
carry_delay_line: FOR i IN 0 TO NUM_CARRY4_STAGES-1 GENERATE

        FirstCarry4: IF i = 0 and SIM_MODE = FALSE  GENERATE
             ATTRIBUTE LOC OF delayblock :
                 LABEL IS "SLICE_X"&INTEGER'image(Xoff)&"Y"&INTEGER'image(Yoff+i);

               attribute dont_touch  of delayblock: label is "TRUE";
            BEGIN
                delayblock:
                CARRY4
                    PORT MAP(
                        CO     => tdl_val_w(3 downto 0),  -- 4-bit carry out
                        O      => open,                   -- 4-bit carry chain XOR data out
                        CI     => '0',                    -- 1-bit carry cascade input
                        CYINIT => hit_i,                  -- 1-bit carry initialization
                        DI     => "0000",                 -- 4-bit carry-MUX data in
                        S      => "1111"                  -- 4-bit carry-MUX select input
                        );
         END GENERATE;

        NEXT_CARRY4: IF i > 0 and SIM_MODE = FALSE GENERATE

                  attribute dont_touch  of delayblock: label is "TRUE";
            BEGIN

                delayblock:
                CARRY4
                    PORT MAP(
                        CO     => tdl_val_w((4*(i+1)-1) downto i*4),
                        O      => open,
                        CI     => tdl_val_w((4*i)-1),
                        CYINIT => '0',
                        DI     => "0000",
                        S      => "1111"
                        );
            END GENERATE;

END GENERATE;
```

Code snippet 1: Implementation of the VHDL code for the Tapped Delay Line.

```
carry_delay_line: FOR i IN 0 TO NUM_CARRY4_STAGES-1 GENERATE

        SIM_first_carry4: IF i = 0 and SIM_MODE = TRUE GENERATE
        begin
            tdl_val_w(3 downto 0) <=
                    ((hit_i & hit_i & hit_i & hit_i) and "1111") after TAP_DELAY_TIME;
        end GENERATE;

        SIM_next_carry4 : IF i > 0 and SIM_MODE = TRUE GENERATE

        BEGIN
            tdl_val_w(4*(i+1)-1 downto 4*i) <=
                    (tdl_val_w(4*i -1 downto 4*(i-1)) and "1111") after TAP_DELAY_TIME ;
        END GENERATE;
END GENERATE;
```

Code snippet 2: Simulation code for Tapped Delay Line.

On the left image of the figure 5.3 and on the right side of the figure 5.6 are both the representation of the delay line. The bottom image of the figure 5.6 is the actual implementation of the tapped delay, which uses cascaded CARRY4 Primitives.

An important note is the attribute LOC of the delay line on the code 1 is to position the delay line in very specific location on the FPGA, however, when allocating the delay chain, the trick is not to allocate all of the CARRY4 elements per slice but only allocate the first CARRY4 block.

When the chain is synthesized, connections are made from the carry out of one CARRY4 to the carry in of another CARRY4. On the die, however, there is only one possible connection the carry in of a CARRY4 which is from the carry out of the CARRY4 immediately below it creating a dedicated route which does not use the fabric routing[1]. To backup the following statement, in the documentation of the 7 series Configurable Logic Block (CLB) there are carry chain primitives [61] which are made up of two or more CARRY4 primitives that use a dedicated route when instantiated and the CARRY4 are placed in a vertical alignment.

In summary, the synthesis tools infer a carry chain logic from arithmetic HDL code and automatically does this placement, so only the first element needs to be manually placed.

As for the placement of the sampling register right next to CARRY4 primitives, the LOC attribute is not required as in most cases as the synthesizer by default already optimizes this route as seen in figure 5.6.

The set is still incomplete as the `tdl_inst` implementation is still left to be combined with the Re-mapping Logic. Adding the Remapping logic results in the figure 5.7 .

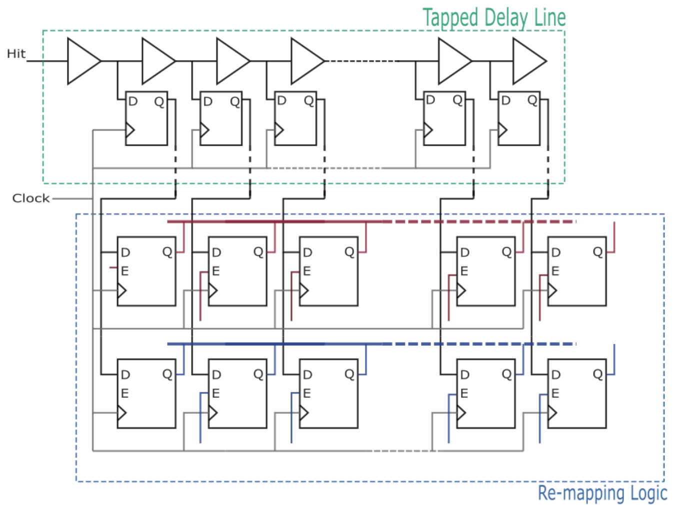


Figure 5.7: TDL and Re-mapping Logic combined.

The difference between the Re-mapping Logic registers and the registers connected directly to Tapped Delay Line is the existence of an enabler. The enabler is set by the control logic

[1]Further information on https://forums.xilinx.com/t5/Implementation/Problem-trying-to-LOC-a-CARRY4/td-p/802074

and therefore this module has a `store_FirstPiece_i` and `store_LastPiece_i` which are these registers enablers.

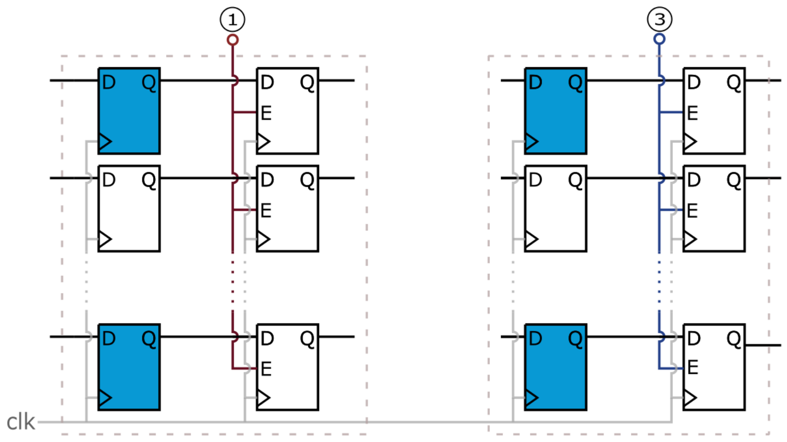


Figure 5.8: Re-mapping Logic register datapath structure.

The figure 5.8 is another point of view of the registers position relative to the registers of the Tapped Delay Line. The `store_FirstPiece_i` is the enabler of the set of registers on the left side of the figure 5.8 and `store_LastPiece_i` is the enabler of the set of registers on the right side of the figure 5.8.

These first line of registers have two purposes.

1. Sampling the TDL.
2. Solve meta-stability issues which frequently occurs in these architectures.

The second set of registers are used to capture to stabilize the value at the input of the store registers before accepting these values.

The implementation on figure 5.8 is shown in code 3.

```
FIRST_STAGE_SAMPLING: FOR j IN 0 TO NUM_STAGES-1 GENERATE
     -- 1st line registers - Sampling Stage
     TDC_VAL_REG : FDCE
         generic map(INIT => '0')
         port map(
             D  => tdl_val_w(j),
             CE => '1',
             C  => clk_i,
             CLR=> rst_i,
             Q  => tdl_val_r(j)
             );

     END GENERATE;

 SECOND_STAGE_FIRST_PIECE_SIGNAL_REG: FOR k IN 0 TO NUM_CARRY4_STAGES-1 GENERATE
     attribute dont_touch  of tdc_thermometer_firstpiece_val_reg: label is "TRUE";
     BEGIN
         tdc_thermometer_firstpiece_val_reg : FDCE
         generic map(INIT => '0')
         port map(
             Q  => tdl_thermometer_FirstPiece_val_r(k),
             D  => tdl_val_r((4*(k+1))-1),
             CE => store_FirstPiece_i,
             C  => clk_i,
             CLR=> rst_i
             );
 END GENERATE;

 SECOND_STAGE_LAST_PIECE_SIGNAL_REG: FOR l IN 0 TO NUM_CARRY4_STAGES-1 GENERATE
   attribute dont_touch  of tdc_thermometer_lastpiece_val_reg: label is "TRUE";
 begin
      tdc_thermometer_lastpiece_val_reg : FDCE
         generic map(INIT => '0')
         port map(
             Q  => tdl_thermometer_LastPiece_val_r(l),
             D  => tdl_val_r((4*(l+1))-1),
             CE => store_LastPiece_i,
             C  => clk_i,
             CLR=> rst_i
             );
 END GENERATE;
```

Code snippet 3: Re-mapping Logic VHDL code.

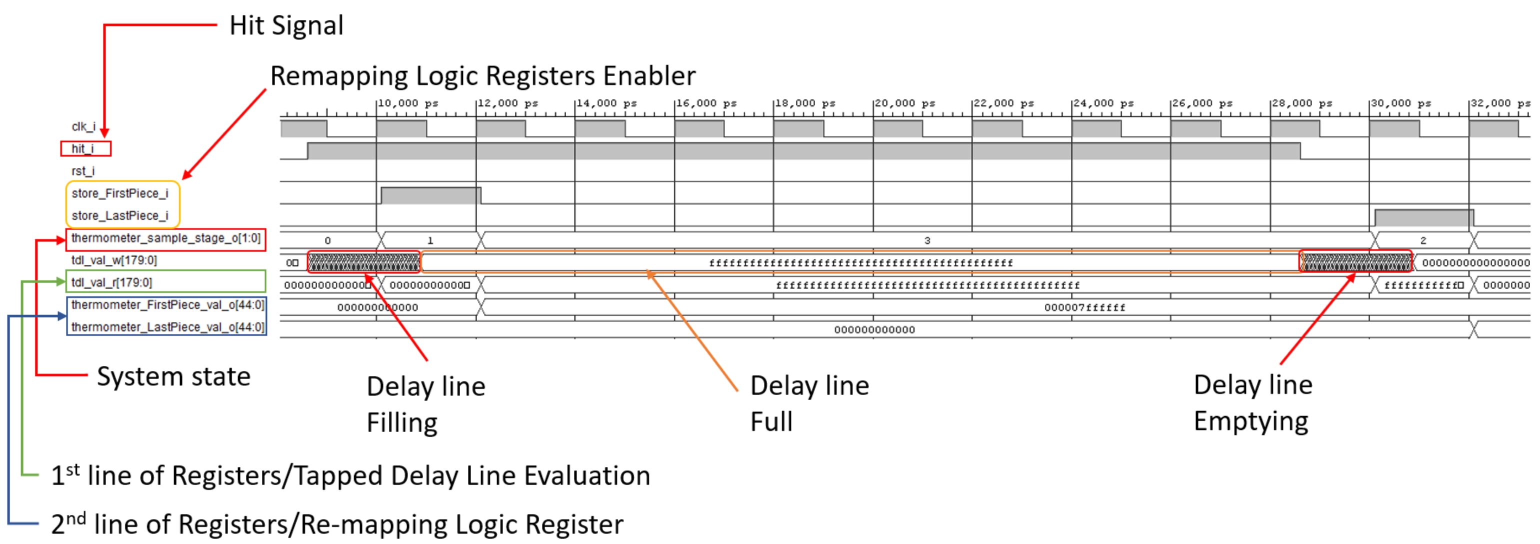


Figure 5.9: TDL instantiation simulation results.

The simulation is under the assumption which the size of the tapped delay line is set to 45 digital cells and the delay considered for each digital cell is 50 ps From the simulation results in the figure 5.9 and the figure 5.4, the module has multiple i/o signals such as the clock seen as `clk_i`, the popular hit signal seen as `hit_i`, the reset signal seen is `rst_i`, the Re-mapping Logic Registers Enablers and the first and last element of the first line of registers, also known as the first and last elements of the sampled delay line. When one of the Re-mapping Logic registers enablers are active, then at the end of the clock cycle they store the value of the first line of registers, so long as the systems state is seen as filling. The same behaviour applies when the system is seen as emptying.

5.2.2.2 Counter VHDL

Out of all the sections, the counter is the easiest to describe. As previously stated on the architecture, it is a simple up counter where its starting value is set to 1.

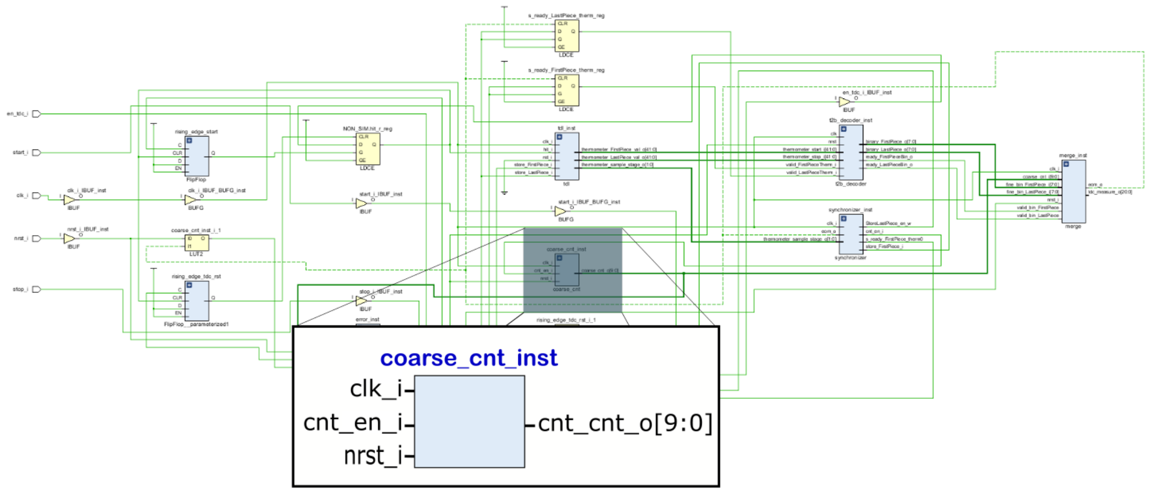


Figure 5.10: Counter module.

```vhdl
entity coarse_cnt is
     Port (  clk_i       : in  STD_LOGIC;
             nrst_i      : in  STD_LOGIC;
             cnt_en_i    : in  STD_LOGIC;
             coarse_cnt_o: out STD_LOGIC_VECTOR(9 DOWNTO 0)
     );
end coarse_cnt;

architecture Behavioral of coarse_cnt is
    signal s_cnt_r : unsigned(9 DOWNTO 0) := (OTHERS => '0');
begin
    process(clk_i)
    begin
        if(rising_edge(clk_i)) then
           if( nrst_i = '0') then
                 s_cnt_r <= to_unsigned(1, 10);
           elsif( cnt_en_i = '1' ) then
                s_cnt_r <= s_cnt_r + 1;
           end if;
        end if;
    end process;

    --OUTPUT CURRENT VALUE
    coarse_cnt_o <= std_logic_vector(s_cnt_r);

end Behavioral;
```

Code snippet 4: Counter VHDL code.

The counter is set to 10 bits on a device running at 500 MHz as it is registered for a 200m distance. Using the eq. 4.1 the resulting ToF for 400m, which is double due to the reflection of the signal, results in 1.4 µs. Next, using the eq. 4.2 results in 704 counts.

$$\lceil \log_2 N_{counts} \rceil = 10, \tag{5.1}$$

This last equation proves that the minimum number of bits required to store the decimal number 704 is 10 bits.

The simulation results are seen in the figure 5.11.

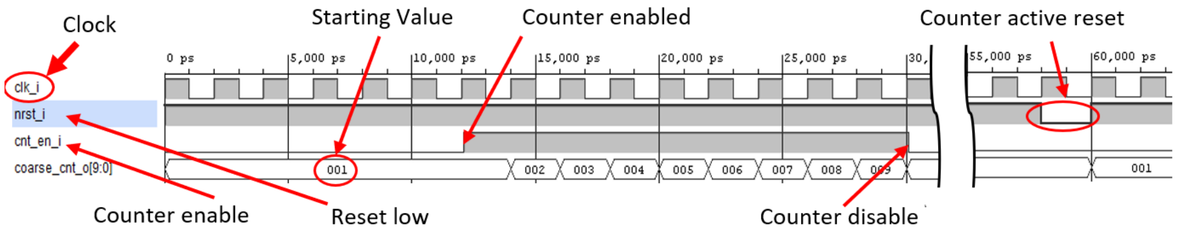


Figure 5.11: Counter simulation results.

5.2.2.3 Counter error VHDL

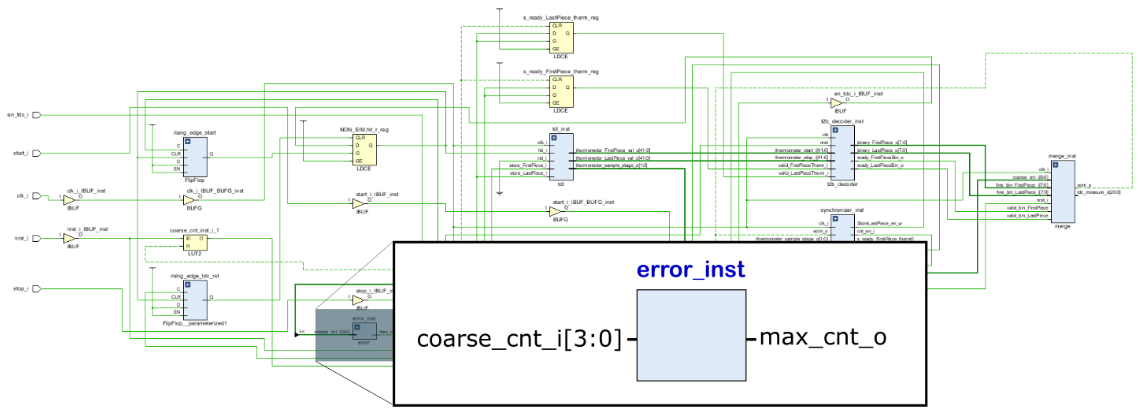


Figure 5.12: Counter error module.

Using the eq. 4.2 for a system running at 500 MHz and a T_{trip} of 1.4 μs results in nearly 700 counts, as seen in the counters implementation section 5.2.2.2, whose binary representation of the decimal value 700 is `10 1011 1100`. The maximum value was limited to 704 whose binary representation is `10 1100 0000`.

When the four most significant bits of the counter match the binary sequence `1011` then `max_cnt_o` signal is set off, forcefully bringing down the hit signal for further measurements ending in the maximum possible value attainable by the system.

5.2.2.4 Synchronizer

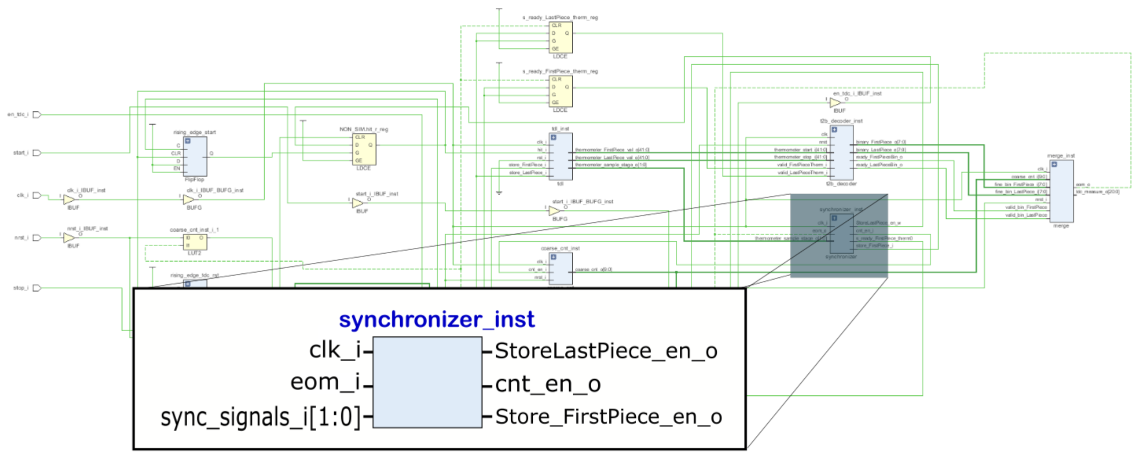


Figure 5.13: Synchronizer module.

The synchronizer is a simple module as seen in the fig 4.10 represented by the FSM.

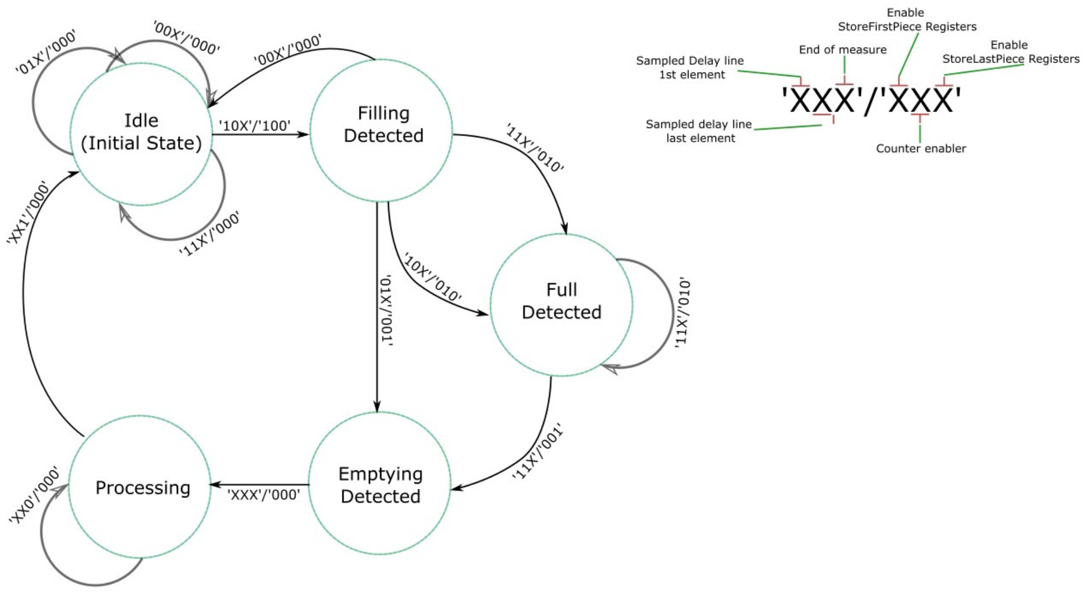


Figure 5.14: Synchronizer - Finite state machine.

The FSM represented in the figure 5.14 is a mealy FSM. The default state is represented by “idle” state which represents inactivity, however, when the first and last element are `10`, respectively, then the output enables the `storeFirstPiece` registers and having detected the delay line as filling, passes onto the next state, “Filling Detected”. At this point, this state should be able to cope with a special situations that occurs when the delay line is larger than a clock period as depicted in the figure 5.15.

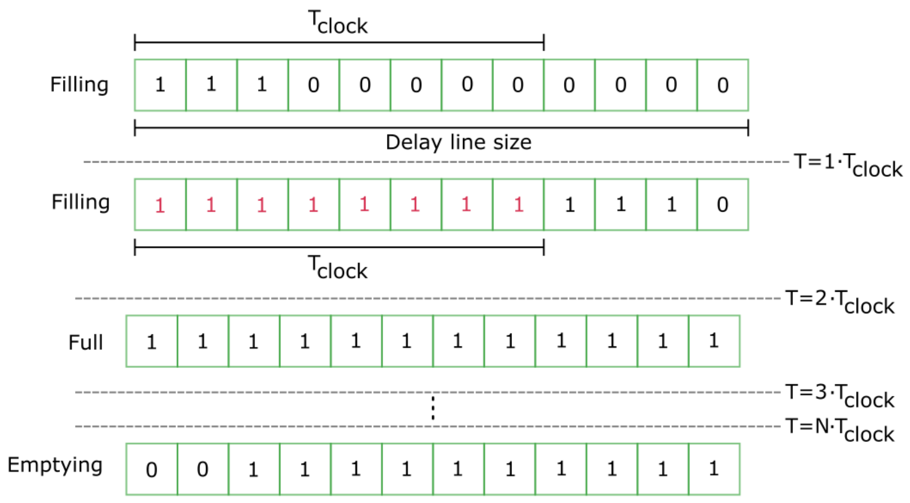


Figure 5.15: Delay line temporal analysis of anomaly detected.

The situation mentioned on the figure 5.15 references the case where line is seen as filling twice instead of passing to a full state or an emptying state. To cope with such a situation, at the state "Filling detected" if it detects first and last elements of the sampling registers as `10` once again or `11`, then the next state is seen as "Full detected" and enables the counter. However, there is a case where the system may go from filling to emptying and as such, we may skip immediately to the "Emptying detected" state as the system recognizes it is emptying, enabling the `storeLastpiece` registers.

Onto the next state, "Emptying detected", is skipped as the necessary signals have been successfully triggered, skipping to next block, which is the processing blocks. During this stage, the system is tasked with processing the information until the end of measurement is triggered.

The simulation results are seen in the figure 5.16.

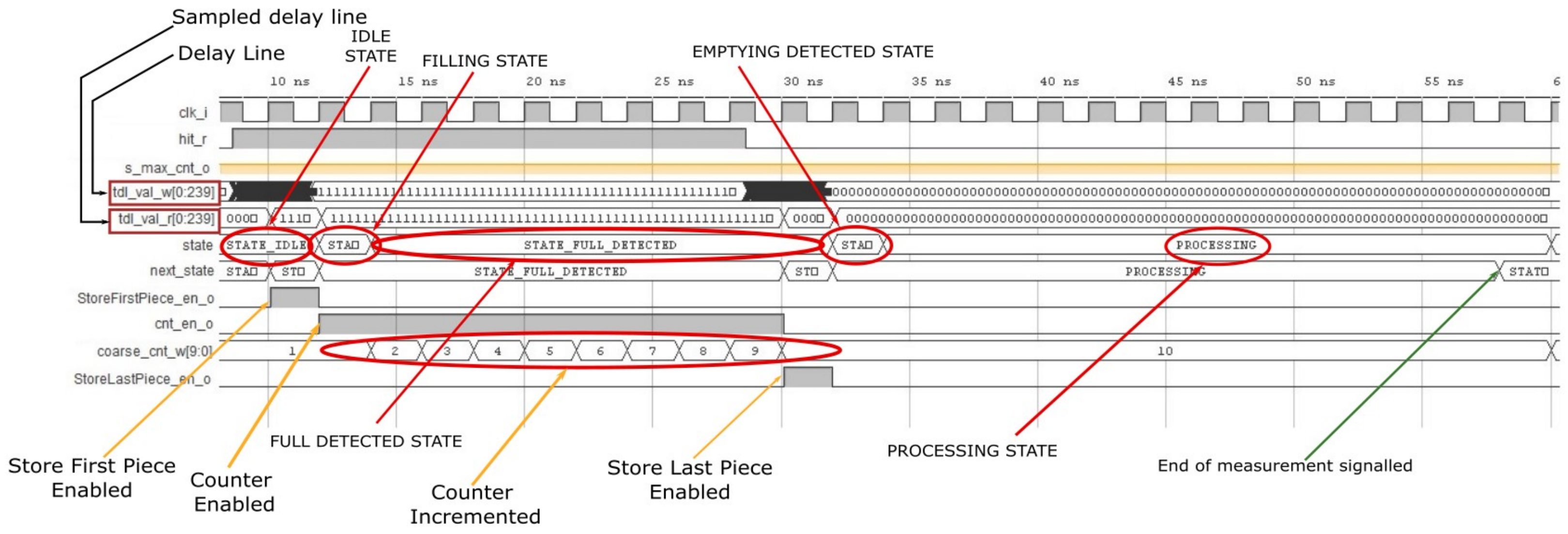


Figure 5.16: Synchronizer simulations and FSM behavior.

From the simulations results, first the `storeFirstPiece_en` was active as it is made to store the first end of the hit signal and then the counter is enabled as it sees the next state as full detected and finally and `storeLastPiece_en` is enabled to store the last end of the hit signal.

Although the system is able to skip from the filling state to the emptying state, it shouldn't be used to measure distances directly related to the ToF smaller than 2 ns, as the system is only able to perceive the signal so long as there are 2 consecutive captures as illustrated in the figure 5.17.

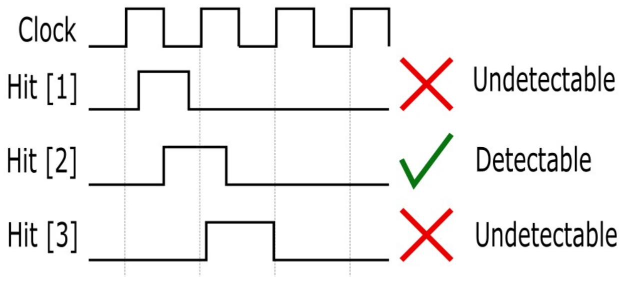


Figure 5.17: System limitation representing case scenarios where the system may fail to detect.

5.2.2.5 Thermometer to Binary encoder

The figure 5.18 presents the thermometer to Binary encoder interface.

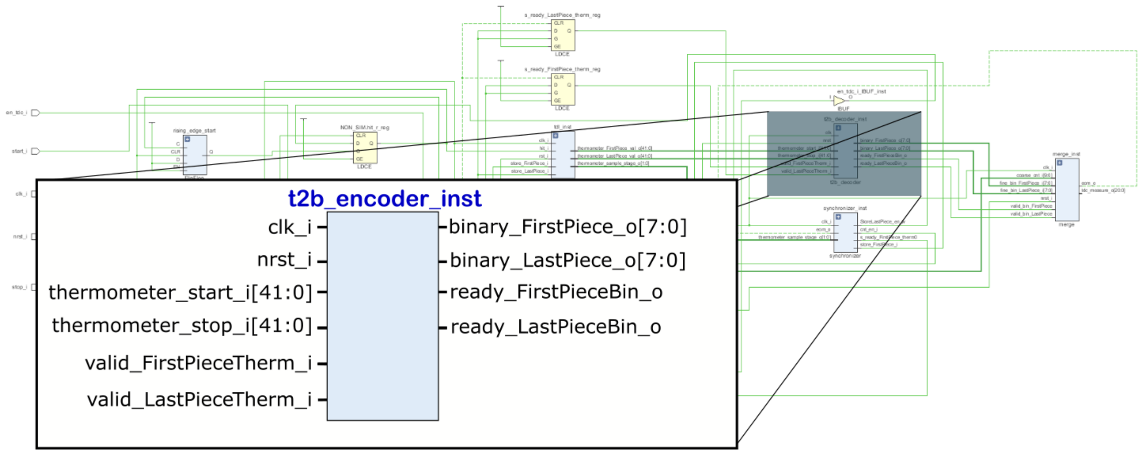


Figure 5.18: Thermometer to Binary module.

This section converts the thermometer code to a binary code and provides signals for streaming with further modules.

The thermometer was implemented by making use of the code available from a free open source website[2] to count the number of ones using a pipelined mux based encoder. However, as seen in the merge section of the system architecture, it's important to count the number of ones and zeros and therefore, it was crucial to adapt their free open source code into something the could count the number of ones and zeros.

[2]More information on https://cas.tudelft.nl/fpga_tdc/TDC_basic.html

The RTL module of their code can be seen in the figure 5.19.

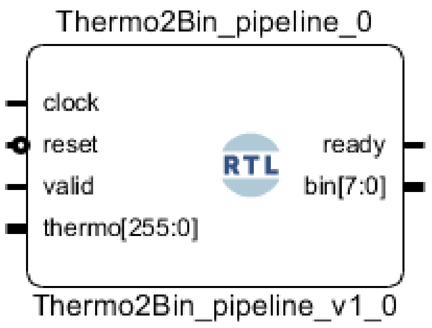


Figure 5.19: Pipelined Thermometer to Bin encoder blackbox.

From the figure 5.19, the block is able to output number of ones of the thermometer code using the pipelined mux based encoder. This block required for it to be alerted of when the data is ready through its valid port and after a few clock cycles the output will produce the number ones and assert the ready signal, signalling that the output is ready.

The block diagram on the figure 5.20 shows the explicit changes.

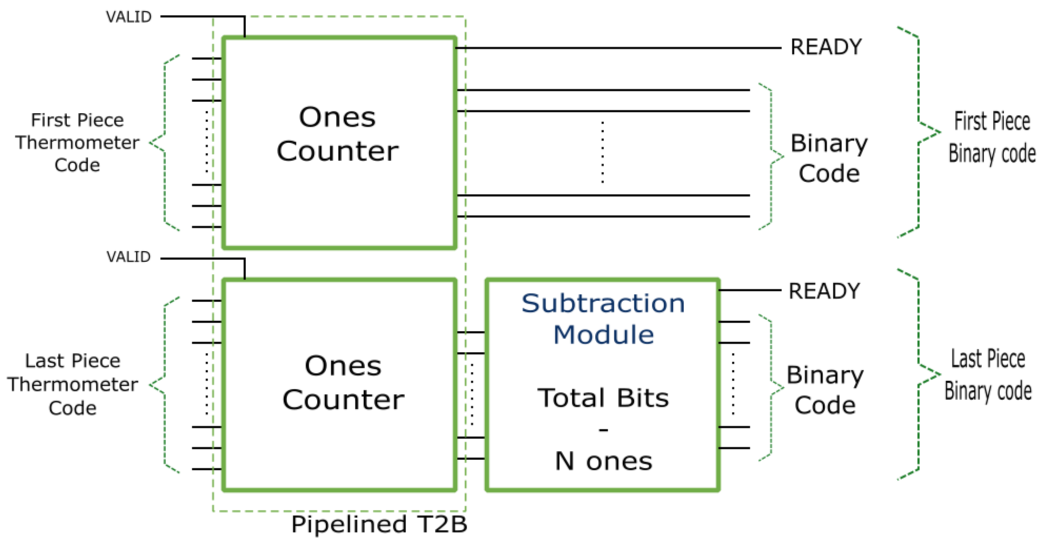


Figure 5.20: T2B block diagram.

In figure 5.18 and figure 5.20, there are a lot of similarities as the only difference is the lack of the clock source and the reset signal. The block diagram show us the open source code, namely the Pipelined T2B to count the number ones instantiated twice, one for each thermometer code. As the Last Piece thermometer code is required to count the number of zeros, an extra block as added to subtract the number of ones from the total from the total number of bits on thermometer code.

As each one of these blocks finish processing the data, there are two `ready` signals, one for each thermometer code, signalling that the data is ready to be used. This behaviour provides a streaming mechanism for the block that will stream this data to the following Merge block.

Although the architecture allows for flexible resizing of the thermometer code, the output of this block is always an 8 bit output, therefore, there can only be a maximum of 255 CARRY4 instantiations.

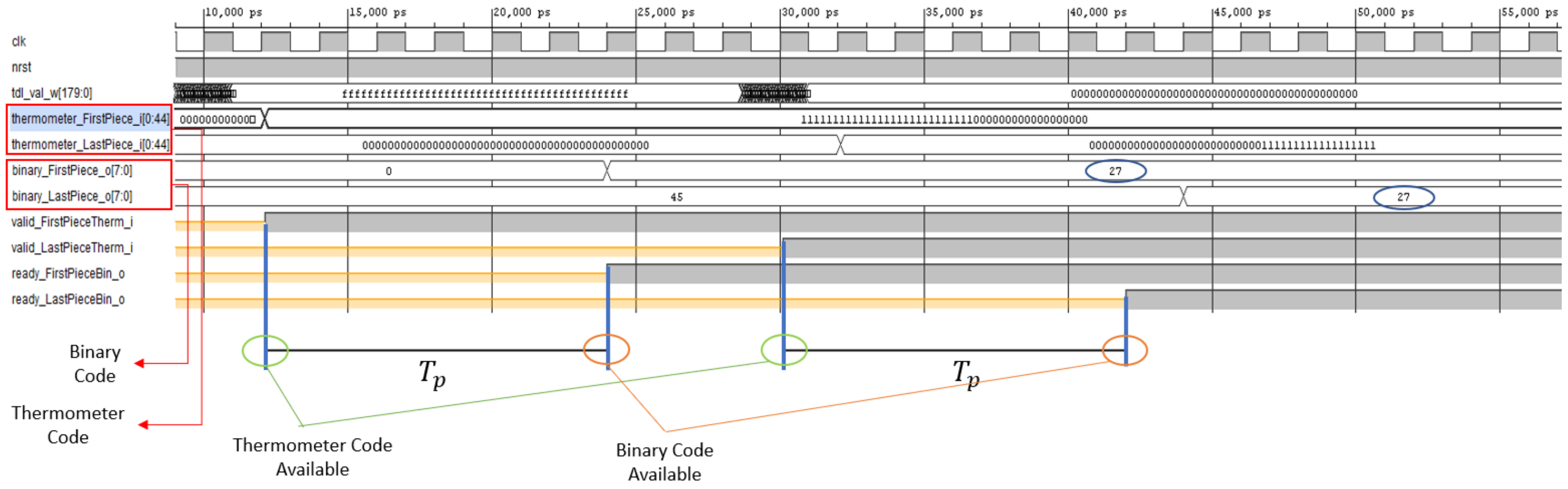


Figure 5.21: T2B simulation results.

The number of digital cells under test for simulation is 45 and therefore from the simulation, the binary codes start with 0 for the FirstPiece thermometer codes and 45 for the LastPiece thermometer code as the system is idle the number of zeros is the number of digital cells and the number of ones is the number is zero.

There are two valid signals and ready signals, in which once the thermometer codes are available, the system takes T_p nanoseconds to process the information, where T_p is a multiple of a number of clock cycles which depends on the size of the thermometer code.

5.2.2.6 Merge

The merge block module can be seen in the figure 5.22.

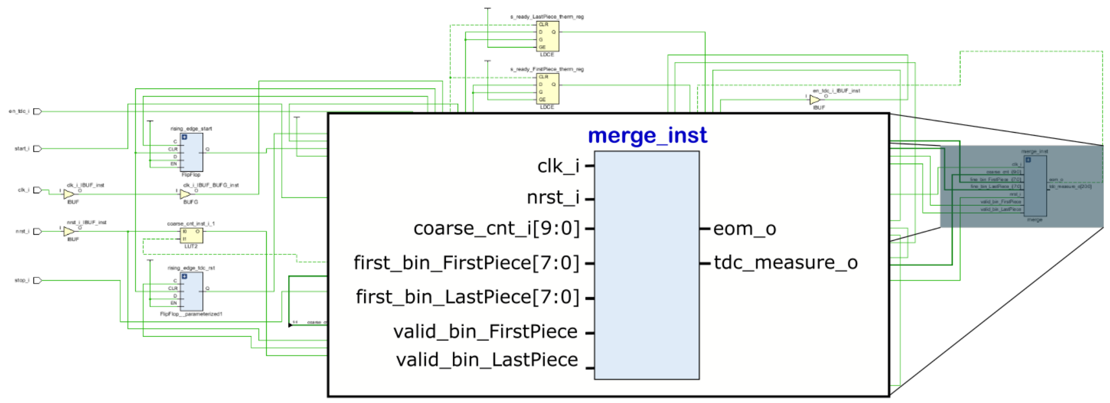


Figure 5.22: Merge module.

The `fine_bin_FirstPiece_i` and `fine_bin_LastPiece_i` signals represent number of ones and zeros of the thermometer codes, the `coarse_cnt_i` represent the gross measure of the system.

In its essence, it is an arithmetic block, that takes in the all previous signals and applies the eq. 4.3 and outputs a given measure and setw a end of measurement, seen as `eom_o` signal in the figure 5.22. The measurement is only initialized when both the ready signals from the thermometer to binary encoder are set. In other words, the block only processes the information once the hit signal is analyzed. However, when running at very high frequencies, namely at 500 MHz these operations tend to take longer than one clock period, thus posing a problem. The solution was to add counter to counter the problem as seen in the figure 5.23.

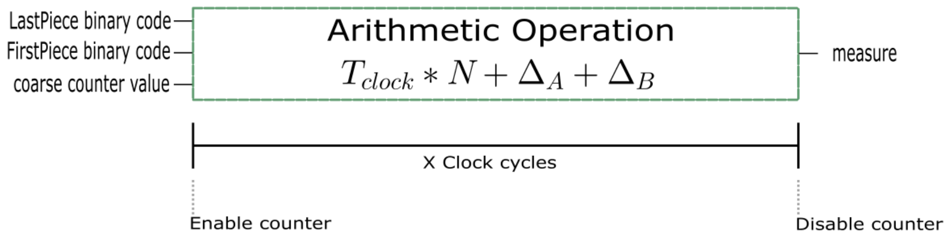
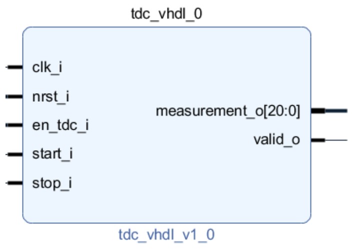


Figure 5.23: Included counter for timing correction.

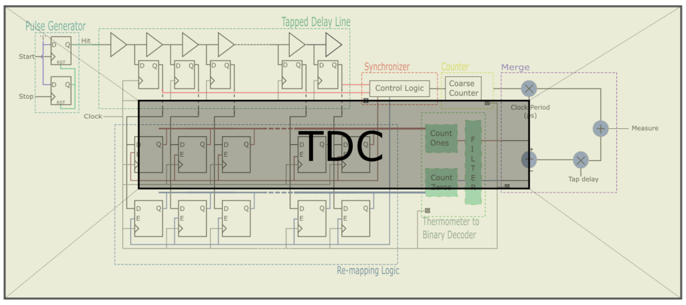
When the `LastPiece` and `FirstPiece` binary codes are available, the counters enabler is set to count for a set number of clock cycles, in this case it was set to 6. The moment the counter is enabled, the arithmetic block simultaneously starts processing the data ensuring a total of 6 clock cycles to process the data and therefore also disabling the counter.

5.2.2.7 TDC

This module's interface can be seen in the left side of the figure 5.24 combining the effort of all the previous modules implementations seen on the right side of the figure 5.24. Important to remember, the purpose is to accurately define the time interval marked by the `start` and `stop` signal.



(a) TDC interface.



(b) TDC aggregating all the previous module in one location working together.

Figure 5.24: TDC simplified RTL view and interface.

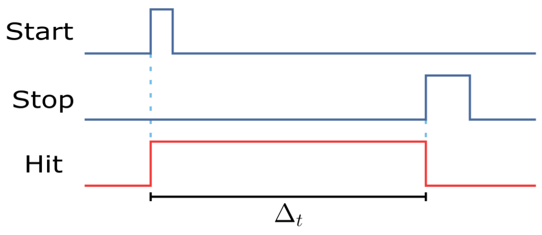


Figure 5.25: Hit signal illustration.

Where Δ_t is the time interval in which the TDC must define.

The systems interface also has an enabler, that acts as the pause button of the system, a reset to the system back to its initial state and a clock which dictates the systems operating frequency of 500MHz.

In the figure 5.26 are the simulation results of the system. Also,important to remember that the delay of each digital cell is set to 50 ps, the number of digital cells of the delay line is 45.

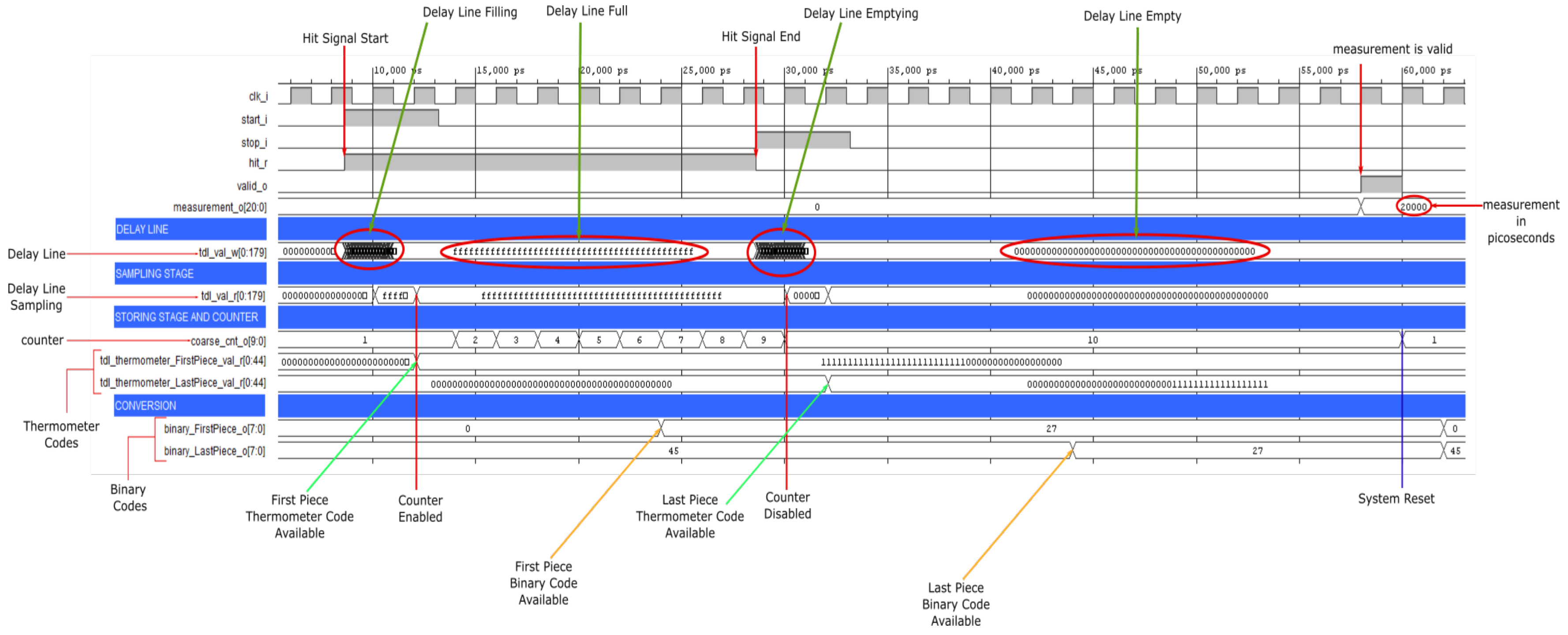
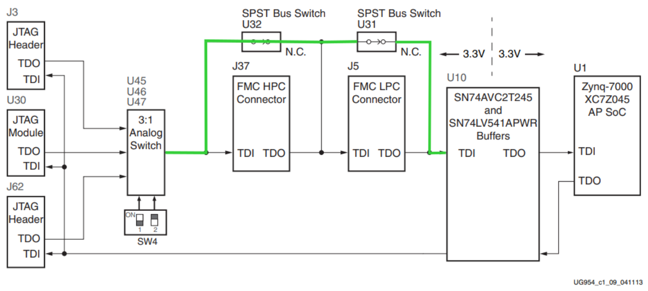


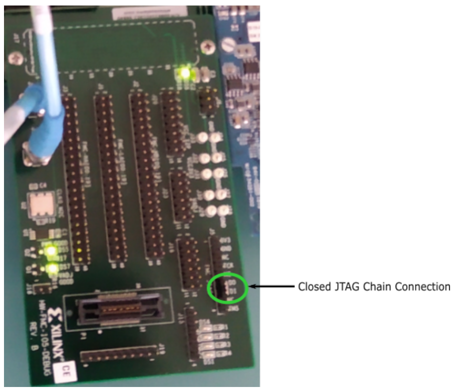
Figure 5.26: TDC simulation results.

5.2.2.8 *Hardware compatibility*

Before starting with the Experimental validation it is important to note that the HW-XM105-G Debug Board developed problems when trying to program the bitstream to the FPGA with the board connected to the High Pin Count (HPC) or Low Pin Count (LPC). The problems which surged were directly related to the FPGA and its Joint Test Action Group (JTAG) connectivity due to the fact that the JTAG connectivity on the ZC706 allows a host computer to download bitstreams to the System on Chip (SoC) and allows for the usage of the debug tools seen by, for example, Vivado's Integrated Logic Analyzer (ILA). In the figure 5.27 are the ZC706 JTAG connections related to the FMC HPC and LPC.



(a) ZC706 JTAG Chain Block Diagram [62].



(b) Labeled location for closing the JTAG Chain U31 Bus Switch using a bypass jumper.

Figure 5.27: Closing the ZC706 FMC LPC JTAG Chain.

In the ZC706, when a mezzanine card is attached to the LPC or HPC connectors. The cards are automatically added to the JTAG chain, this itself implies new connections are setup. The Single Pole Single Throw (SPST) switches which are normally closed, transition to an open state when an FMC is attached and therefore, this loop must be closed by using a bypass jumper [62] unless the card is already correctly configured and closes the JTAG chain. In this particular case, as shown in the figure 5.27, the loop is closed by a bypass jumper connecting the input of FMC LPC Test Data in (TDI) to the output Test Data out (TDO), or in other words, closing the SPST Bus Switch `U31` connection seen on the left side of the figure 5.27.

CHAPTER 6

Experimental validation of a TDC

6.1 Introduction

This chapter provides the results and the methods used to validate the implemented architecture shown in the last chapter. The following results are discussed:

1. Digital cell delay estimation.
2. Linearity estimation.
3. Repeatability estimation.

6.2 Digital cell delay estimation

First and foremost objective was to measure the average delay per digital cell.

6.2.1 Experimental Setup

The setup for doing so is illustrated on the diagram of the figure 6.1.

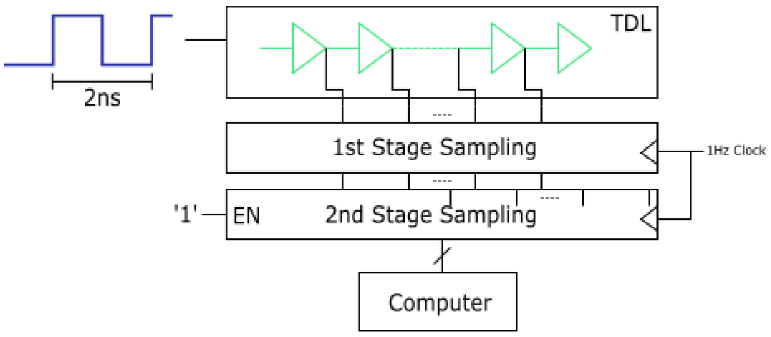
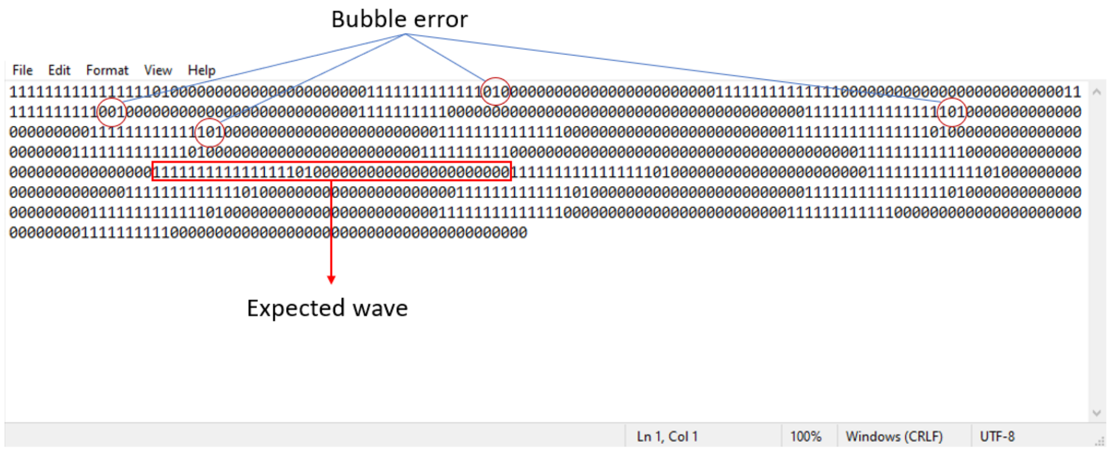


Figure 6.1: Setup for the number of digital cells estimation necessary to fit one clock cycle.

As shown in figure 6.1, by injecting the clock signal into a sufficiently big tapped delay line, initially set as 200 digital cells and sampling rate of 1 Hz, enabled free running sampling and time enough to evaluate the thermometer code.

The output of the second sampling stage is the thermometer code from one of the re-mapping logic storage units where its enable was forcefully set to be always on. Due to this, it was possible to constantly sample from the delay line at a very low frequency and as a consequence, it was possible to see the number of clock cycles that would fit in the delay line.

6.2.2 Results



```
1111111111111111010000000000000000000000111111111111101000000000000000000000000001111111111111100000000000000000000000000011
1111111111001000000000000000000000000011111111110000000000000000000000000000000000000000011111111111111110100000000000000
000000000111111111111101000000000000000000000000011111111111111000000000000000000000000001111111111111111010000000000000000
0000000111111111111101000000000000000000000000001111111111000000000000000000000000000000000000000011111111111100000000000000
0000000000000000111111111111111110100000000000000000000000111111111111111101000000000000000000000000111111111111101000000000
0000000000000111111111111101000000000000000000000000111111111111101000000000000000000000000001111111111111111010000000000000
000000000111111111111101000000000000000000000000011111111111111000000000000000000000000000111111111111000000000000000000000
0000000011111111110000000000000000000000000000000000000000
```

Figure 6.2: Visual results of the thermometer code sampled twice.

The results of the figure 6.2 allowed for the adjustment of the size of the delay line to fit one clock cycle which resulted in the values seen by the simulation, averaging to around 40 digital cells. Using the following formula which supposes the tapped is slightly greater than one clock period.

$$T_{clock} \leq N_{Digital\ cells} \cdot \tau_{delay}, \tag{6.1}$$

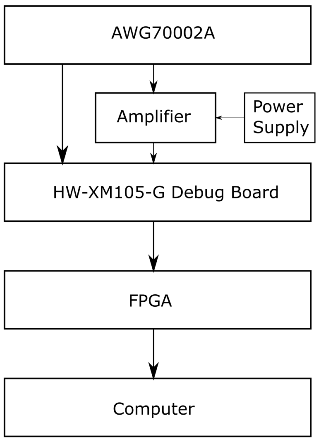
results in the substitution of T_{clock} for the clocks period equivalent to 2 ns and $N_{Digital\ cells}$ of 40 resulting in estimating the average digital cell delay of around 50 ps. However, it had to be slightly greater than the current clock period due to external factors that may provoke sufficient variation for the condition above to fail and as consequence the TDL was setup with 45 digital cells.

Having an estimate of the delay per digital cell was crucial for testing the TDC as the calculations in the merge block requires a value for the expected delay to proceed with the practical tests.

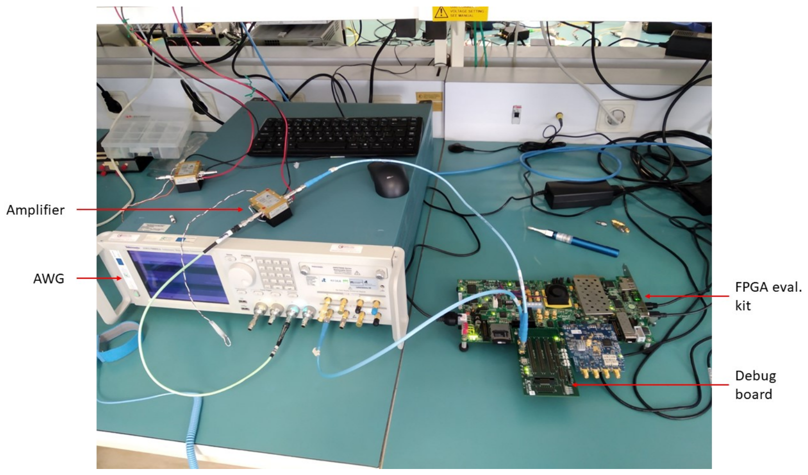
6.3 Estimation Linearity

6.3.1 Experimental Setup

In order to observe the linearity of the estimations, the following experimental setup shown in figure 6.3 was considered.



(a) Setup diagram.



(b) Labeled setup photo.

Figure 6.3: Experimental setup.

An Arbitrary Waveform Generator (AWG), model AWG70002A, operating at 15 Gsample/s was used to generate two short pulses delayed by a given number of samples. Marker 2 of the AWG was used as a start signal, whereas the output of channel 2 was used as a `stop` signal. The signal outputted by channel 2 had to be amplified such in order to rise to the same level as the other channels in order to apply to the FPGA. For that a broadband amplifier SHF 810 was used.

In the FPGA, the pins which were used for setting the `start` and `stop` signal on the debug board HW-XM105-G were the `P_CLK1_M2C` and `N_CLK1_M2C` pins which would translate to the `AD28` and `AC28` pin on the ZC706 Evaluation Board [62], [63].

The figure 6.4 shown is the Vivado practical TDC setup for testing linearity.

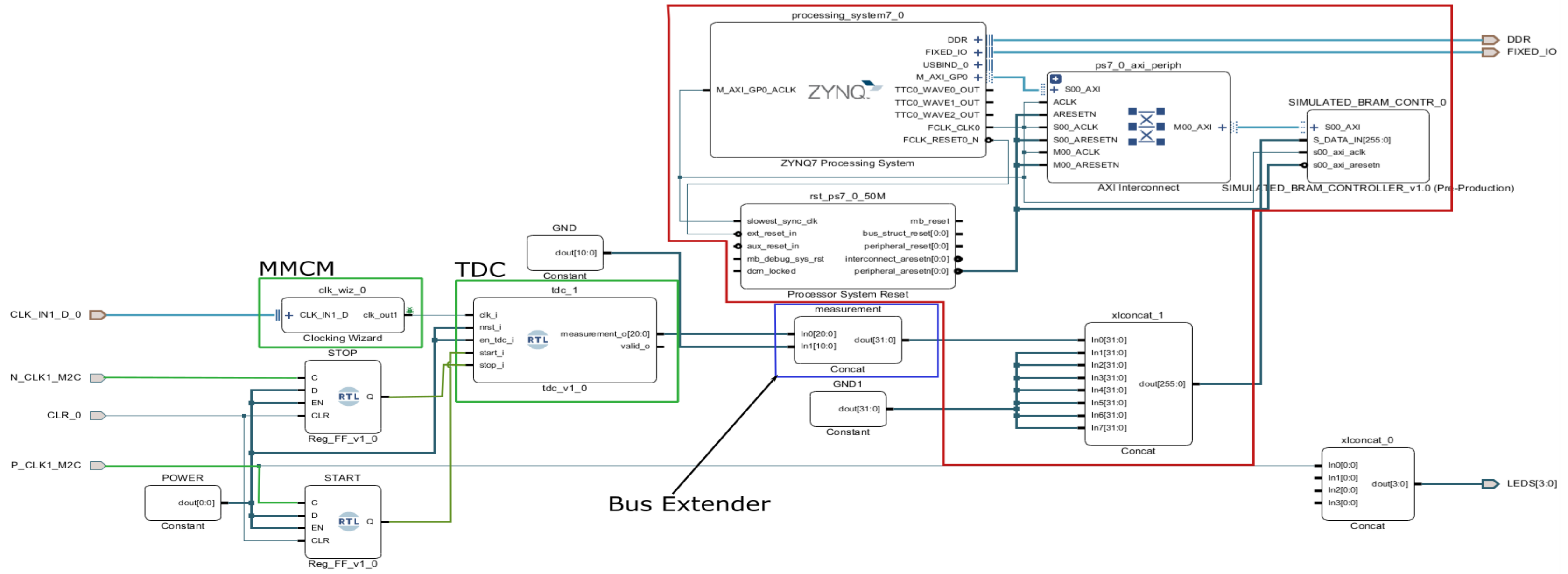


Figure 6.4: Vivado - TDC Testing block diagram setup.

In figure 6.4, the zone delimitted in red allowed to user to use the terminal to access the values of the TDC, the zone delimitted in blue is used to extend the 21-bit bus outputted from the TDC to a 32-bit bus for compatibility purposes with the following block. The Mixed Mode Clock Manager (MCMM) is used for generating a clock of 500 MHz for the TDC.

6.3.2 Results

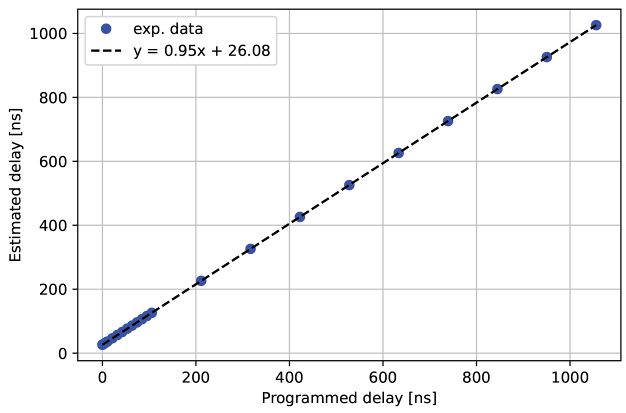
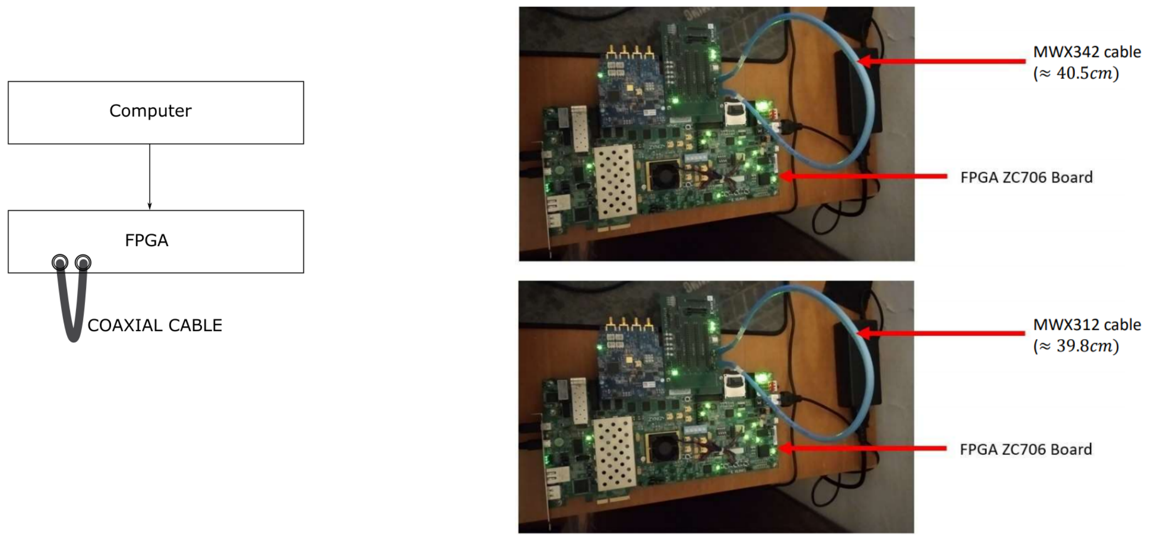


Figure 6.5: Estimated delay as a function of the programmed delay.

Figure 6.5 shows that the estimation results produced by the TDC increases linearly with the programmed delay. The TDC having a linear relationship between the programmed delay and the estimated delay is a very desirable feature and was effectively achieved.

6.4.1 Experimental setup

In order to observe the estimation repeatability, the following experimental setup shown in figure 6.6 was considered.



(a) Setup diagram. **(b)** Labeled setup photo.

Figure 6.6: Experimental setup.

The FPGA uses the `P_CLK1_M2C` and `N_CLK1_M2C` pins from the HW-XM105-G debug board to create a loop as shown on the figure 6.6 on the FPGA.

The Vivado variance test setup can be shown on the figure 6.7.

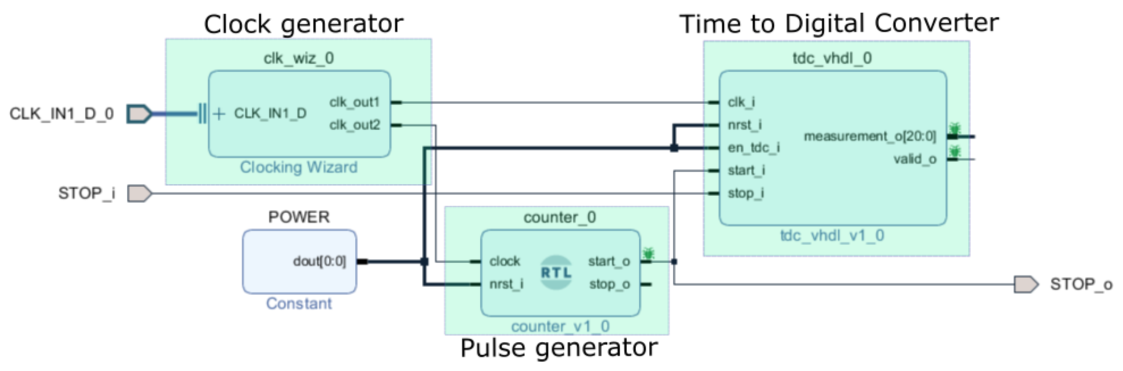
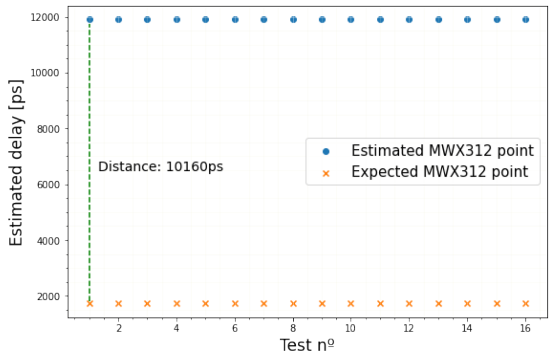


Figure 6.7: Vivado design setup for variance test.

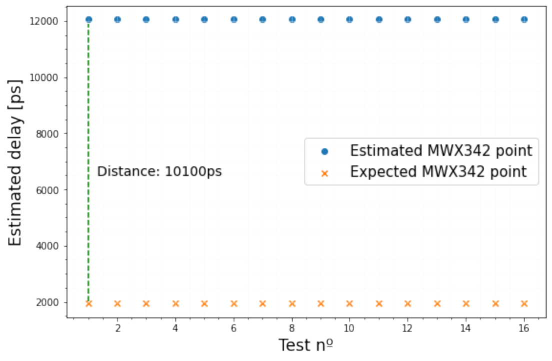
In the design show in figure 6.7 there is a 500 MHz clock supplying the TDC and the pulse generator. The pulse generator emits a pulse whose bifurcation occurs for both TDC inputs `start_i` and `stop_i`. One path of the pulse generator goes directly to the input `start_i` of the TDC, the other path is a longer path which imposes a delay, controlled externally by inserting a cable between `STOP_i` external input and `STOP_o` external output which are connected to the `P_CLK1_M2C` and `N_CLK1_M2C` pins, respectively. The external input `STOP_i` is connected to the input `stop_i` of the TDC.

Two different coaxial cables were used, with different lengths, namely a Radio Frequency (RF) cable from the MWX312 series with the length of approximately 39.8 cm±1 mm that has a propagation delay of 4.9 ns m^{-1} and a cable from the MWX342 series with the length of approximately 40.5 cm±1 mm that has a propagation delay of 4.3 ns m^{-1} [64]. Under these conditions, the total delay imposed by the MWX342 and the MWX312 cable is 1.9502 ns and 1.7415 ns, respectively, whose expected difference in delay is 0.2087 ns.

6.4.2 Results



(a) MWX312 cable test.



(b) MWX342 cable test.

Figure 6.8: Repeatability results.

From the figure. 6.8 the tests resulted in perfect repeatability. The MWX312 expected delay was 1740 ps whereas the TDC estimated 11 900 ps for 16 samples, whereas the MWX342 expected delay was 1950 ps whereas the TDC estimated 12 050 ps for 16 samples. The expected difference in delay estimated is therefore 150 ps whereas the expected difference in delay is 210 ps. These results are very positive, as only a slight deviation in the difference between the expected delay which occurred between both cables which maintained a steady difference of 150 ps.

6.5 Resource usage

Resource	Utilization	Available	Utilization(%)
LUT	246	218600	0.112
FF	501	437200	0.114
IO	27	362	7.458
BUFG	2	32	6.25

Table 6.1: Base TDC FPGA resource usage on the ZC706.

Table 6.1 shows minimal resource usage in relation to the total resource usage. A total on chip power usage of 227 mW derived from the power analysis of post-implemented design, out of which, 200 mW are from device static and 27 mW from on chip-power, whereas a breakdown on the on-chip power, the TDL and the thermometer to binary encoder consumed the highest power, namely, 7 mW and 8 mW, respectively.

6.6 Summary

All the results are very promising, as the TDC exhibited a linear behavior and perfect repeatability and the table 6.1 showed low resource usage and a reported power consumption of 227 mW.

CHAPTER 7

Conclusion

LiDAR is a key sensor that enables an AV to observe its surroundings in 3D with high resolution. However, in contrast to complementary sensors such as cameras and RADAR, LiDAR is not yet a fully mature sensor. One of the aspects that requires development is sampling architectures for a LiDAR receiver. Such is the focus of this dissertation.

This dissertation started by studying different sampling architectures based on ADC and TDC, with the goal of choosing suitable architectures for LiDAR. We have then successfully implemented a first sampling architecture based on a pipelined architecture ADC, and a second sampling architecture based on a TDC, in turn consisting of a TDL topology and a mux-based encoder.

The ADC-based front-end complied with all target parameters, namely sampling frequency and ENOB, with the exception of power consumption. Its power consumption of 1.65 W per channel at 1 Gsample/s is too high for a LiDAR receiver, given that a LiDAR comprises tens of channels, which requires tens of ADCs.

The TDC-based front-end also complied with all target parameters, namely a temporal resolution of 50 ps which allows for a spacial resolution of 1 cm. The implemented TDC takes few resources, and only consumes 227 mW. The expected sampling frequency of the TDC due to an associated processing time of 26 ns and for a window of 1 µs is around 38 Msample/s.

Even though implementation and tests of both architectures were successful, these must be considered preliminary. Nonetheless, we should compare both architectures. The implemented ADC is appropriate for a full-waveform LiDAR receiver that requires a sampling rate up to 1 GHz. The TDC is appropriate for processing pulses as short as 5 ns, as the detection only occurs on the rising edge of the pulse. Consequently, if the waveform boils down to a short pulse, the TDC should be employed for a matter of simplicity, cost (no external chip required) and power consumption.

7.1 Future work

In the near future there is room for further optimization of the TDC. More extensive tests should then be performed, and calibration must be addressed.

In the medium to long term future we should consider investigating multi-channel TDCs, such that a full-waveform LiDAR can be implemented with TDCs. This would provide the advantages of both TDCs and ADCs, however without the disadvantages of ADC, namely, cost and power consumption.

Bibliography

[1] C. Filiz, *An autonomous vehicles prevent traffic accidents?* Last seen on 30/05/2021, 2020. [Online]. Available: https://www.intechopen.com/books/accident-analysis-and-prevention/can-autonomous-vehicles-prevent-traffic-accidents-.

[2] A. Lafrance, *Self-driving cars could save 300,000 lives per decade in america*, Last seen on 30/05/2021, 2015. [Online]. Available: https://www.theatlantic.com/technology/archive/2015/09/self-driving-cars-could-save-300000-lives-per-decade-in-america/407956/.

[3] S. Blanco, *SAE updates, refines official names for 'autonomous driving' levels*, Last seen on 30/05/2021, 2021. [Online]. Available: https://www.caranddriver.com/news/a36364986/sae-updates-refines-autonomous-driving-levels-chart/.

[4] *2020 autonomous vehicle technology report*, https://www.wevolver.com/article/2020.autonomous.vehicle.technology.report, Last accessed: 15/05/2021, 2020.

[5] A. Mutschler, *Sensor Fusion Challenges In Cars*, https://semiengineering.com/sensor-fusion-challenges-in-cars/, Last accessed: 15/06/2021, 2020.

[6] F. Rosique, P. J. Navarro, C. Fernández, and A. Padilla, "A systematic review of perception system and simulators for autonomous vehicles research," *Sensors*, vol. 19, no. 3, 2019, ISSN: 1424-8220. DOI: 10.3390/s19030648. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/648.

[7] M. Choudhary, "Why LiDAR is important for autonomous vehicle?" 2020. [Online]. Available: https://www.geospatialworld.net/blogs/why-lidar-is-important-for-autonomous-vehicle/.

[8] *The 25-year-old billionaire building the future of self-driving cars*, https://www.theverge.com/22298001/luminar-austin-russel-ceo-interview-self-driving-cars, Last accessed: 08/06/2021, 2021.

[9] A. Vaughan, *Time of flight & lidar: Optical analog front end design*, Last seen on 30/05/2021. [Online]. Available: https://www.ti.com/lit/an/sboa337/sboa337.pdf?ts=1622376881229&ref_url=https%5C%253A%5C%252F%5C%252Fwww.ti.com.cn%5C%252Fsitesearch%5C%252Fcn%5C%252Fdocs%5C%252Funiversalsearch.tsp%5C%253FlangPref%5C%253Dzh-CN%5C%2526searchTerm%5C%253D.

[10] D. Bastos, P. P. Monteiro, A. S. R. Oliveira, and M. V. Drummond, "an overview of lidar requirements and techniques for autonomous driving," in *2021 Telecoms Conference (ConfTELE)*.

[11] *Tutorials: Microcontroller systems (micsy)*, Last seen on 1/06/2021, 2021. [Online]. Available: http://www.weigu.lu/tutorials/microcontroller/08_ad_da_converter/index.html.

[12] L. Danial, N. Wainstein, S. Kraus, and S. Kvatinsky, "Breaking through the speed-power-accuracy tradeoff in ADCs using a memristive neuromorphic architecture," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 5, pp. 396–409, 2018. DOI: 10.1109/TETCI.2018.2849109.

[13] *Types of A/D converters - the ultimate guide*, released on 22. April 2021 and last seen on 23. May 2021, DEWESOFT, 2021. [Online]. Available: https://dewesoft.com/daq/types-of-adc-converters#adc-features.

[14] *Tutorials: Microcontroller systems (micsy)*, Last seen on 1/06/2021, 2021. [Online]. Available: https://www.maximintegrated.com/en/design/technical-documents/tutorials/1/1080.html.

[15] *Understanding the Delta-Sigma ADC*, Last seen on 1/06/2021, 2016. [Online]. Available: `https://www.allaboutcircuits.com/technical-articles/understanding-the-delta-sigma-analog-to-digital-converter/`.

[16] *Flash ADC*, Last seen on 30/05/2021, 2021. [Online]. Available: `https://www.mathworks.com/help/msblks/ref/flashadc.html`.

[17] *Pipelined ADCs and more*, Last seen on 30/05/2021, 2010. [Online]. Available: `https://inst.eecs.berkeley.edu/~ee247/fa10/files07/lectures/L22_2_f10.pdf`.

[18] *Pipeline ADCs*, Last seen on 30/05/2021, 2000. [Online]. Available: `http://people.ece.umn.edu/~harjani/courses/8331/ADC-pipeline_lecture.PDF`.

[19] *SAR ADCs vs. Delta-Sigma ADCs: Different architectures for different application needs*, Video published by Training Ti, 2015. [Online]. Available: `https://training.ti.com/adcwebinar#:~:text=The%5C%20SAR%5C%20converter%5C%20takes%5C%20a,time%5C%20and%5C%20performs%5C%20a%5C%20conversion.&text=Delta%5C%2DSigma%5C%20converters%5C%20provide%5C%20best,signal%5C%20once%5C%20for%5C%20each%5C%20conversion.`.

[20] E. Kandilakis, "Ultra-low energy time-mode ADC with background calibration for biomedical sensing applications," M.S. thesis, Delft University of Technology, 2021.

[21] *Analog-to-digital converter architectures and choices for system design*, Writing by Brian Black amd last seen on 23. May 2021, ANALOG, 1999. [Online]. Available: `https://www.analog.com/en/analog-dialogue/articles/analog-to-digital-converter-architectures-and-choices.html`.

[22] B. Swann, B. Blalock, L. Clonts, D. Binkley, J. Rochelle, J. Breeding, and K. Baldwin, "A 100-ps time-resolution cmos time-to-digital converter for positron emission tomography imaging applications," *Solid-State Circuits, IEEE Journal of*, vol. 39, pp. 1839–1852, Dec. 2004. DOI: `10.1109/JSSC.2004.835832`.

[23] S. Naraghi, "Time-based analog to digital converters," M.S. thesis, University of Michigan, 2009.

[24] R. F. F. Granja, "11.7b time-to-digital converter with 0.82ps resolution in 130nm cmos technology," M.S. thesis, Técnico Lisboa, 2018.

[25] D. Dinkar Toraskar, M. P. Mattada, and H. Guhilot, "Time domain ADC using pulse shrinking tdc," in *2016 International Conference on Circuits, Controls, Communications and Computing (I4C)*, 2016, pp. 1–4. DOI: `10.1109/CIMCA.2016.8053264`.

[26] *Comparative Study of Delay Line Based Time to Digital Converter Using FPGA*, Last seen on 1/06/2021, 2017. [Online]. Available: `https://www.irjet.net/archives/V4/i9/IRJET-V4I9208.pdf`.

[27] J. Wu, "Several key issues on implementing delay line based TDCs using FPGAs," *IEEE Transactions on Nuclear Science*, vol. 57, no. 3, pp. 1543–1548, 2010. DOI: `10.1109/TNS.2010.2045901`.

[28] J. Wang, S. Liu, Q. Shen, H. Li, and Q. An, "A fully fledged TDC implemented in field-programmable gate arrays," *IEEE Transactions on Nuclear Science*, vol. 57, no. 2, pp. 446–450, 2010. DOI: `10.1109/TNS.2009.2037958`.

[29] K. Cui and X. Li, "A high-linearity Vernier time-to-digital converter on FPGAs with improved resolution using bidirectional-operating vernier delay lines," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 8, pp. 5941–5949, 2020. DOI: `10.1109/TIM.2019.2959423`.

[30] M. Függer, A. Kinali, C. Lenzen, and T. Polzer, "Metastability-aware memory-efficient time-to-digital converters," in *2017 23rd IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, 2017, pp. 49–56. DOI: `10.1109/ASYNC.2017.12`.

[31] F. Dadouche, T. Turko, W. Uhring, I. Malass, N. Dumas, and J.-P. Le, "New design-methodology of high-performance TDC on a low cost FPGA targets," *Sensors and Transducers*, vol. 193, pp. 123–134, Oct. 2015.

[32] R. Machado, J. Cabral, and F. S. alves, "All-digital time-to-digital converter design methodology based on structured data paths," vol. 7, IEEE, 2019.

[33] R. Machado, C. Jorge, and F. S. Alves, “Recent developments and challenges in fpga-based time-to-digital converters,” *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 11, pp. 4205–4221, 2019. DOI: 10.1109/TIM.2019.2938436.

[34] Y. Sano, Y. Horii, M. Ikeno, O. Sasaki, M. Tomoto, and T. Uchida, “Subnanosecond time-to-digital converter implemented in a kintex-7 fpga,” *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 874, pp. 50–56, 2017, ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2017.08.038. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900217309245.

[35] M. Mattada and H. Guhilot, “62.5â ps lsb resolution multiphase clock time to digital converter (tdc) implemented on fpga,” *Journal of King Saud University - Engineering Sciences*, 2021, ISSN: 1018-3639. DOI: https://doi.org/10.1016/j.jksues.2021.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1018363921000143.

[36] Z. Soni, D. K. Panda, and A. B. Sarbadhikari, “Comparative study of delay line based time to digital converter using fpga,” International Research Journal of Engineering and Technology (IRJET), Tech. Rep., Sep. 2017.

[37] W. Gao, D. Gao, C. Hu-Guo, and Y. H, *Integrated High-Resolution Multi-Channel Time-to-Digital Converters (TDCs) for PET Imaging*, Northwestern Polytechnical University and Institut Pluridisciplinaire Hubert Curien, 2011. [Online]. Available: https://www.intechopen.com/books/biomedical-engineering-trends-in-electronics-communications-and-software/integrated-high-resolution-multi-channel-time-to-digital-converters-tdcs-for-pet-imaging.

[38] H. S., “Time-to-digital converter basics. in: Time-to-digital converters,” *Springer Series in Advanced Microelectronics*, vol. 29, Oct. 2010. DOI: 10.1007/978-90-481-8628-0_2.

[39] A. A. Muntean, “Design of a fully digital analog sipm with sub-50ps time conversion,” M.S. thesis, Delft University of Technology, 2017.

[40] C. Priyanka and P. Latha, “Design and implementation of time to digital converters,” in *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 2015, pp. 1–4. DOI: 10.1109/ICIIECS.2015.7193116.

[41] Y.-J. Chuang, H.-H. Ou, and B.-D. Liu, “A novel bubble tolerant thermometer-to-binary encoder for flash a/d converter,” in *2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT).*, 2005, pp. 315–318. DOI: 10.1109/VDAT.2005.1500084.

[42] A. Chunn and R. K. Sarin, “Comparison of thermometer to binary encoders for flash adcs,” in *2013 Annual IEEE India Conference (INDICON)*, 2013, pp. 1–4. DOI: 10.1109/INDCON.2013.6726138.

[43] T. Pardhu, S. Manusha, and K. Sirisha, “A low power flash adc with wallace tree encoder,” in *2014 Eleventh International Conference on Wireless and Optical Communications Networks (WOCN)*, 2014, pp. 1–4. DOI: 10.1109/WOCN.2014.6923067.

[44] P. Carra, M. Bertazzoni, M. G. Bisogni, J. M. Cela Ruiz, A. Del Guerra, D. Gascon, S. Gomez, M. Morrocchi, G. Pazzi, D. Sanchez, I. Sarasola Martin, G. Sportelli, and N. Belcari, “Auto-calibrating tdc for an soc-fpga data acquisition system,” *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 3, no. 5, pp. 549–556, 2019. DOI: 10.1109/TRPMS.2018.2882709.

[45] R. Jogdand, P. Dakhole, and P. Palsodkar, “Low power flash adc using multiplexer based encoder,” Mar. 2017, pp. 1–5. DOI: 10.1109/ICIIECS.2017.8276157.

[46] D. Lee, J. Yoo, K. Choi, and J. Ghaznavi, “Fat tree encoder design for ultra-high speed flash a/d converters,” The Pennsylvania State University, Department of Computer Science & Engineering, Report.

[47] S. Hussain, R. Kumar, and G. Trivedi, “Methodology and comparative design of an efficient 4-bit encoder with bubble error corrector for 1-gsps flash type adc,” *IET Circuits, Devices & Systems*, vol. 14, no. 5, pp. 629–639, 2020. DOI: https://doi.org/10.1049/iet-cds.2019.0499. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cds.2019.0499. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cds.2019.0499.

[48] Bui, V. Hieu, S. Beak, S. Choi, J. Seon, Jeong, and T. Ted., "Thermometer-to-binary encoder with bubble error correction (bec) circuit for flash analog-to-digital converter (fadc)," in *International Conference on Communications and Electronics 2010*, 2010, pp. 102–106. DOI: `10.1109/ICCE.2010.5670690`.

[49] I. P.Latha Dr. R. Sivakumar, "Implementation of mux based encoder for time to digital converters architecture," vol. 4, 2018.

[50] M. J. Azmi and S. A. Imam, "Power and area efficient ADC with suitable encoders and comparators: A review," *International Research Journal of Engineering and Technology (IRJET)*, 2015, ISSN: 2395-0072. DOI: `https://doi.org/10.1016/j.jksues.2021.01.007`. [Online]. Available: `https://www.irjet.net/archives/V2/i1/IRJET-V2I186.pdf`.

[51] M. Mrinal, J. Jaijee, P. Bhulania, A. Mehra, H. Rana, and S. Khanna, "Study and designing of fourth order bec circuit for flash analog to digital converter using mux based encoder," *Indian Journal of Science and Technology*, vol. 10, pp. 1–7, May 2017. DOI: `10.17485/ijst/2017/v10i18/106370`.

[52] S. M. Mayur, "Design of novel multiplexer based thermometer to binary code encoder for 4 bit flash adc," in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, 2017, pp. 1006–1009. DOI: `10.1109/RTEICT.2017.8256750`.

[53] *ZC706 evaluation kit*, Last seen on 12/06/2021. [Online]. Available: `https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html`.

[54] *AD-FMCDAQ2-EBZ board*, Last seen on 12/06/2021. [Online]. Available: `https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcdaq2-ebz.html`.

[55] *AD-FMCDAQ2-EBZ introduction*, Last revision on 03 Jan 2021 21:49, was approved by Robin Getz., ANALOG DEVICES, 2021. [Online]. Available: `https://wiki.analog.com/resources/eval/user-guides/ad-fmcdaq2-ebz/introduction`.

[56] *AD-FMCDAQ2-EBZ HDL reference design*, 14 Dec 2020 15:23 was approved by Stanca-Florina Pop, ANALOG DEVICES, 2020. [Online]. Available: `https://wiki.analog.com/resources/eval/user-guides/ad-fmcdaq2-ebz/reference_hdl#xilinx_block_diagram`.

[57] *AD-FMCDAQ2-EBZ clocking*, Last revision 19 Jan 2018 10:14 was approved by Alexandru Ardelean, ANALOG DEVICES, 2018. [Online]. Available: `https://wiki.analog.com/resources/eval/user-guides/ad-fmcdaq2-ebz/clocking#overview`.

[58] *AD-FMCDAQ2-EBZ HDL Design DMA*, Last seen on 12/06/2021, 2018.

[59] *Basic FPGA TDC design*, `https://cas.tudelft.nl/fpga_tdc/TDC_basic.html`, Last accessed: 06/06/2021, 2015.

[60] *Xilinx 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide for HDL Design*, UG768, Rev.14.7, XILINX, 2013. [Online]. Available: `https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf`.

[61] *7 series FPGAs configurable logic block*, UG474, Rev.1.8, XILINX, 2016. [Online]. Available: `https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf`.

[62] *ZC706 evaluation board for the Zynq-7000 XC7Z045 SoC*, UG954, Rev.1.8, XILINX, 2019. [Online]. Available: `https://www.xilinx.com/support/documentation/boards_and_kits/zc706/ug954-zc706-eval-board-xc7z045-ap-soc.pdf`.

[63] *FMC XM105 debug card user guide*, G537, Rev.1.3, XILINX, 2011. [Online]. Available: `https://www.xilinx.com/support/documentation/boards_and_kits/ug537.pdf`.

[64] *MWX cable series datasheet*, `https://www.aspen-electronics.com/junkosha-datasheets.html`, Last accessed: 06/06/2021, 2019.

[65] F. Fallah, "Introduction to Zynq™ Architecture." [Online]. Available: `https://www.aldec.com/en/company/blog/144--introduction-to-zynq-architecture`.

[66] *AD-FMCDAQ2-EBZ introduction*, Rev.0, ANALOG DEVICES. [Online]. Available: `https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf`.

Appendix A

QUICK START GUIDE FOR AD-FMCDAQ2-EBZ ON XILINX ZC706

All the tools here are done in Windows 10 and assumes all the modules of the ZC706 is working.

If not, then it is highly recommended to follow XILINX Documentation in the following link before proceeding

https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html#documentation (Search for XTP242 – ZC706 BIST Tutorial). This will require **Vivado** and **SDK 2015.4**

Xilinx Windows 10 Setup

Vivado 2018.3

1. Create or if already created, login into the **XILINX** website and download **Vivado 2018.3** https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2018-3.html
   Note: Use your Vivado license to validate the software

Cygwin

1. Download **Cygwin** and download all the necessary **make**, **git** and **nano**(optional) packages
   (Page) https://www.cygwin.com/
   (Download) https://www.cygwin.com/setup-x86_64.exe

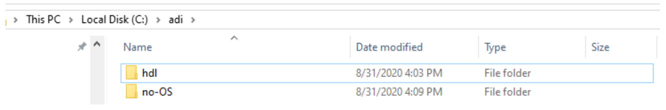
Analog Devices

AD-FMCDAQ2-EBZ

1. Go to your **C** folder and create a folder named **adi**. Inside **adi** folder, git clone the following repositories in **Cygwin**

```
cd /cygdrive/c/adi/
git clone https://github.com/analogdevicesinc/hdl
git clone https://github.com/analogdevicesinc/no-OS
```

   Note: Both repositories should be in the same folder



```
> This PC > Local Disk (C:) > adi >
Name                        Date modified        Type          Size
hdl                         8/31/2020 4:03 PM    File folder
no-OS                       8/31/2020 4:09 PM    File folder
```

2. Before we get officially started with Cygwin terminal, first thing that we must do is

```
export PATH=$PATH:/cygdrive/c/Xilinx/Vivado/2018.3/bin/
export PATH=$PATH:/cygdrive/c/Xilinx/SDK/2018.3/bin/
```

   Note: To permanently add to your PATH, you can do the following

   Go to your directory and open with a text editor the .bash_profile

```
  nano ~/.bash_profile
```

This script should appear once you open it

```
# To the extent possible under law, the author(s) have dedicated all
# copyright and related and neighboring rights to this software to the
# public domain worldwide. This software is distributed without any warranty.
# You should have received a copy of the CC0 Public Domain Dedication along
# with this software.
# If not, see <http://creativecommons.org/publicdomain/zero/1.0/>.

# base-files version 4.3-2

# ~/.bash_profile: executed by bash(1) for login shells.

# The latest version as installed by the Cygwin Setup program can
# always be found at /etc/defaults/etc/skel/.bash_profile

# Modifying /etc/skel/.bash_profile directly will prevent
# setup from updating it.

# The copy in your home directory (~/.bash_profile) is yours, please
# feel free to customise it to create a shell
# environment to your liking.  If you feel a change
# would be benifitial to all, please feel free to send
# a patch to the cygwin mailing list.

# User dependent .bash_profile file

# source the users bashrc if it exists
if [ -f "${HOME}/.bashrc" ] ; then
  source "${HOME}/.bashrc"
fi

# Set PATH so it includes user's private bin if it exists
# if [ -d "${HOME}/bin" ] ; then
#   PATH="${HOME}/bin:${PATH}"
# fi

# Set MANPATH so it includes users' private man if it exists
# if [ -d "${HOME}/man" ]; then
#   MANPATH="${HOME}/man:${MANPATH}"
# fi

# Set INFOPATH so it includes users' private info if it exists
# if [ -d "${HOME}/info" ]; then
#   INFOPATH="${HOME}/info:${INFOPATH}"
# fi
```

At the end of the file insert the lines below and then press Ctrl+S then Ctrl+X

```
export PATH=$PATH:/cygdrive/c/Xilinx/Vivado/2018.3/bin/

export PATH=$PATH:/cygdrive/c/Xilinx/SDK/2018.3/bin/
```

Now finally, we're back at the terminal, to activate these changes immediately, run the following command

```
source ~/.bash_profile
```

3. Go to hdl folder and change to the branch hdl_2018_r2

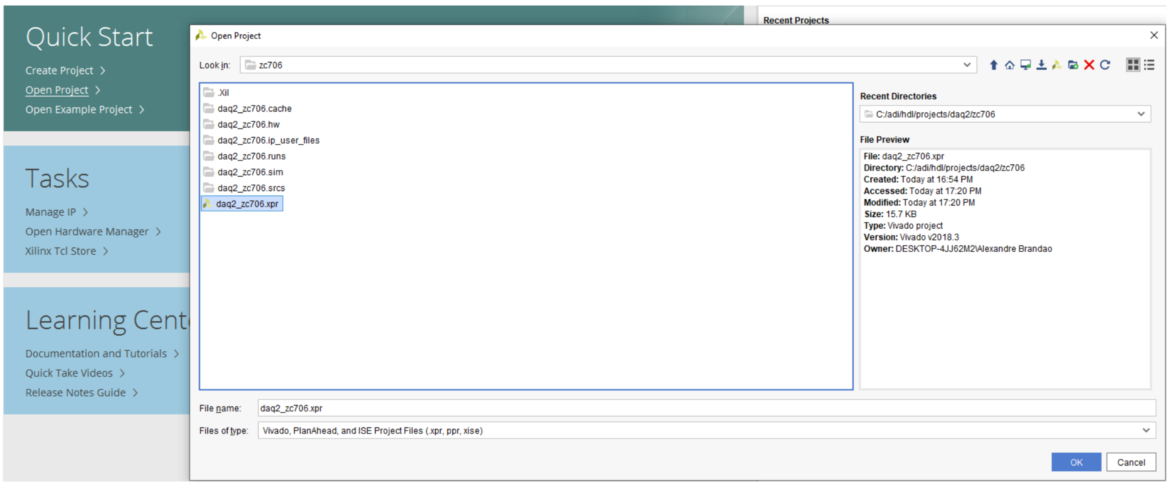
```
cd /cygdrive/c/adi/hdl
git checkout hdl_2018_r2
```

4. Build the project using the Makefile

```
cd projects/daq2/zc706/
make -C .
```

Note: This will take some time

5. Open Vivado 2018.3, Open Project and open daq2_zc706.xpr



Look in: zc706

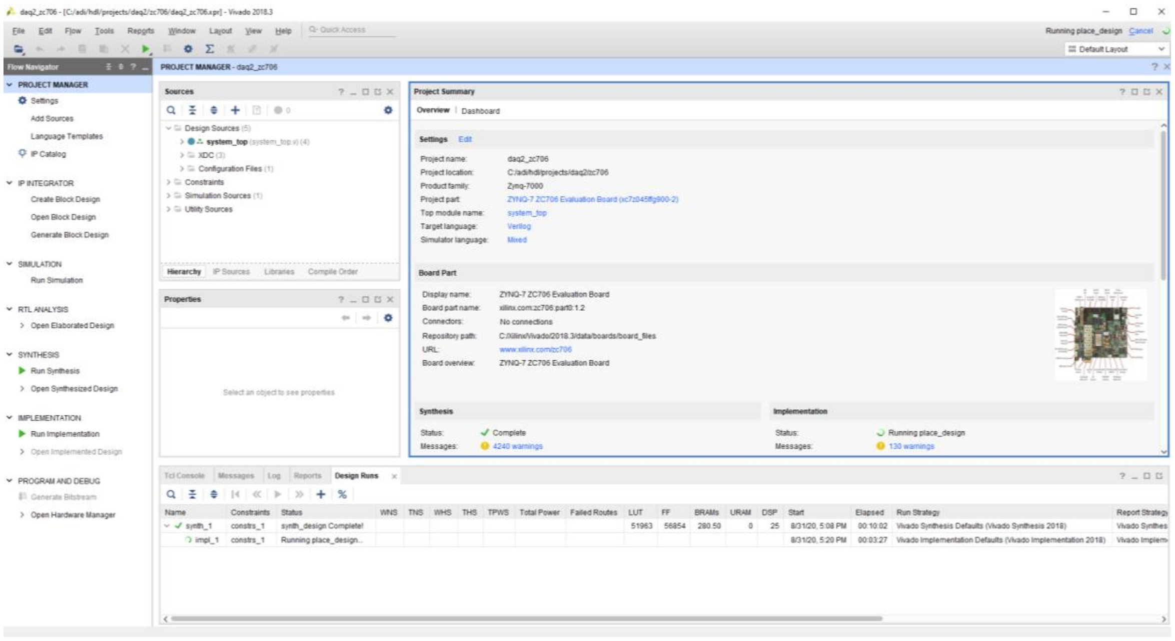
.Xil
daq2_zc706.cache
daq2_zc706.hw
daq2_zc706.ip_user_files
daq2_zc706.runs
daq2_zc706.sim
daq2_zc706.srcs
daq2_zc706.xpr

Recent Directories
C:/adi/hdl/projects/daq2/zc706

File Preview
File: daq2_zc706.xpr
Directory: C:/adi/hdl/projects/daq2/zc706
Created: Today at 16:54 PM
Accessed: Today at 17:20 PM
Modified: Today at 17:20 PM
Size: 15.7 KB
Type: Vivado project
Version: Vivado v2018.3
Owner: DESKTOP-4JJ62M2\Alexandre Brandao

File name: daq2_zc706.xpr
Files of type: Vivado, PlanAhead, and ISE Project Files (.xpr, ppr, xise)

6. On the design run section, right-click synth1, launch runs and once ready, right click again and generate bitstream



daq2_zc706 - [C:/adi/hdl/projects/daq2/zc706/daq2_zc706.xpr] - Vivado 2018.3

PROJECT MANAGER - daq2_zc706

Design Sources (5)
system_top (system_top.v) (4)
XDC (3)
Configuration Files (1)
Constraints
Simulation Sources (1)
Utility Sources

Settings Edit

Project name:	daq2_zc706
Project location:	C:/adi/hdl/projects/daq2/zc706
Product family:	Zynq-7000
Project part:	ZYNQ-7 ZC706 Evaluation Board (xc7z045ffg900-2)
Top module name:	system_top
Target language:	Verilog
Simulator language:	Mixed

Board Part

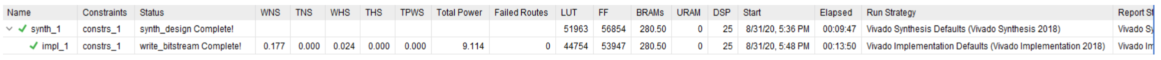
Display name:	ZYNQ-7 ZC706 Evaluation Board
Board part name:	xilinx.com:zc706:part0:1.2
Connectors:	No connections
Repository path:	C:/Xilinx/Vivado/2018.3/data/boards/board_files
URL:	www.xilinx.com/zc706
Board overview:	ZYNQ-7 ZC706 Evaluation Board

Synthesis
Status: ✓ Complete
Messages: 4240 warnings

Implementation
Status: Running place_design
Messages: 130 warnings

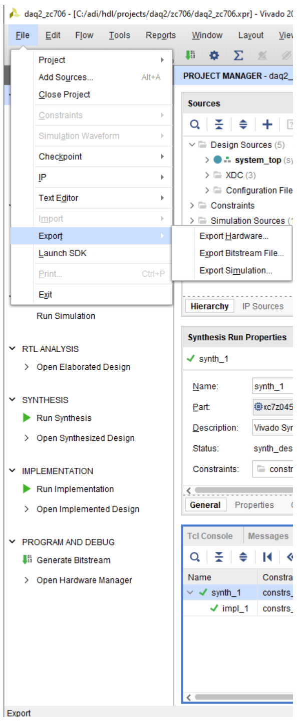
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
✓ synth_1	constrs_1	synth_design Complete!								51963	56854	280.50	0	25	8/31/20, 5:08 PM	00:10:02	Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Synthes
impl_1	constrs_1	Running place_design...													8/31/20, 5:20 PM	00:03:27	Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Implem

Note: During the building of the files, sometimes it may already run the implementation, if you see 2 ticks on the design run tab(Image below) then you may skip to the next step.

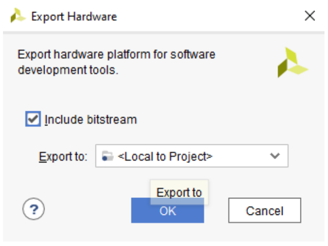


Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report St
✓ synth_1	constrs_1	synth_design Complete!								51963	56854	280.50	0	25	8/31/20, 5:36 PM	00:09:47	Vivado Synthesis Defaults (Vivado Synthesis 2018)	Vivado Sy
✓ impl_1	constrs_1	write_bitstream Complete!	0.177	0.000	0.024	0.000	0.000	9.114	0	44754	53947	280.50	0	25	8/31/20, 5:48 PM	00:13:50	Vivado Implementation Defaults (Vivado Implementation 2018)	Vivado Im

7. On the File section, export HARDWARE



8. Tick the include bitstream and press OK


Export Hardware

Export hardware platform for software development tools.

[✓] Include bitstream

Export to: <Local to Project>

9. On the File section click launch SDK

10. Now back to no-OS folder and checkout to the branch 2018_R2 on the **Cygwin terminal**

```
cd /cygdrive/c/adi/no-OS
git checkout 2018_R2
```

11. Building the software

```
cd /cygdrive/c/adi/no-OS/fmcdaq2/zc706
make -C .
```

Note: Make sure that the FPGA is powered on and connected to the PC and then run the command:

12. Running the software

```
make -C . run
```

Note: The **make run** will download the bitstream on the FPGA and after that program the board with the elf file.

The software is started before the memory debugger disconnects.

Use Tera Term to see what the UART is sending

13. Evaluating the result

```
make capture
```

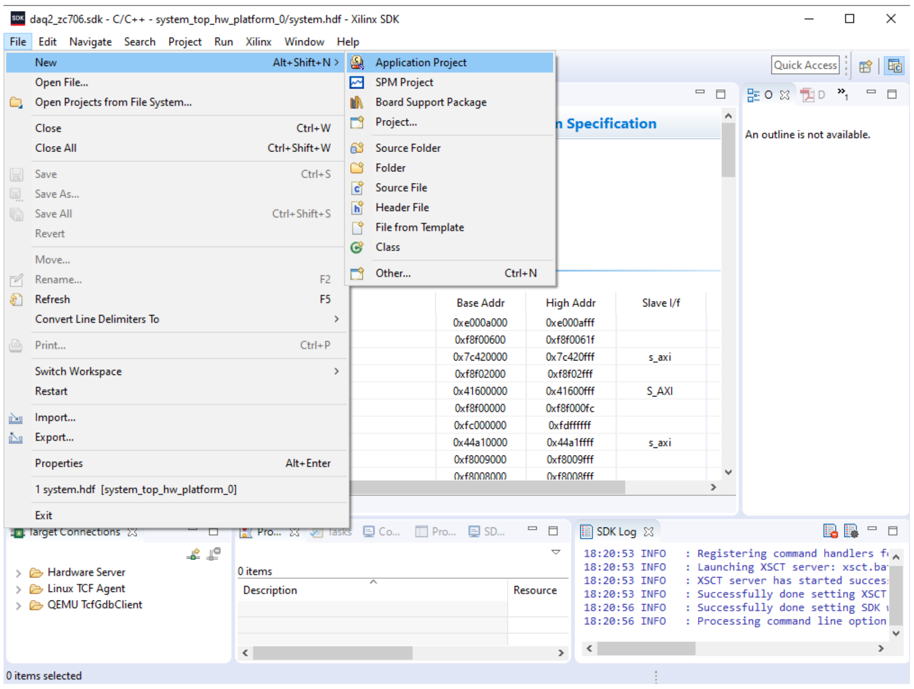
Note: By default, the software captures (in case of ADC based projects) the data received from the device in the RAM.

```
rx_xfer.start_address = *_MEM_BASEADDR + OFFSET;

rx_xfer.no_of_samples = value;

dmac_start_transaction(ad_core_dma);
```

Note: The CSV files are in 2’s complement format, they are located at

```
/cygdrive/c/adi/no-OS/fmcdaq2/zc706
```

14. Back to the SDK, create a new Application Project, **Empty Application**

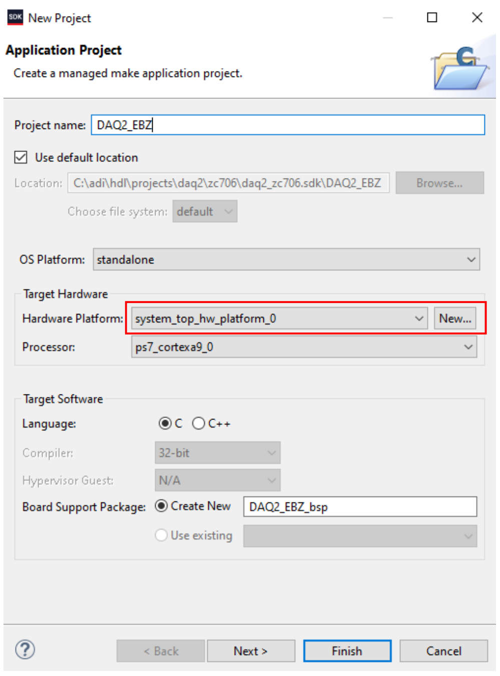


Base Addr	High Addr	Slave I/f
0xe000a000	0xe000afff	
0xf8f00600	0xf8f0061f	
0x7c420000	0x7c420fff	s_axi
0xf8f02000	0xf8f02fff	
0x41600000	0x41600fff	S_AXI
0xf8f00000	0xf8f000fc	
0xfc000000	0xfdffffff	
0x44a10000	0x44a1ffff	s_axi
0xf8009000	0xf8009fff	
0xf8008000	0xf8008fff	

0 items

```
18:20:53 INFO  : Registering command handlers f
18:20:53 INFO  : Launching XSCT server: xsct.ba
18:20:53 INFO  : XSCT server has started succes
18:20:53 INFO  : Successfully done setting XSCT
18:20:56 INFO  : Successfully done setting SDK
18:20:56 INFO  : Processing command line option
```

15. Create your hardware platform with system_top.hdl file and name it in project name section.



Project name: DAQ2_EBZ

[✓] Use default location

Location: C:\adi\hdl\projects\daq2\zc706\daq2_zc706.sdk\DAQ2_EBZ

Choose file system: default

OS Platform: standalone

Hardware Platform: system_top_hw_platform_0

Processor: ps7_cortexa9_0

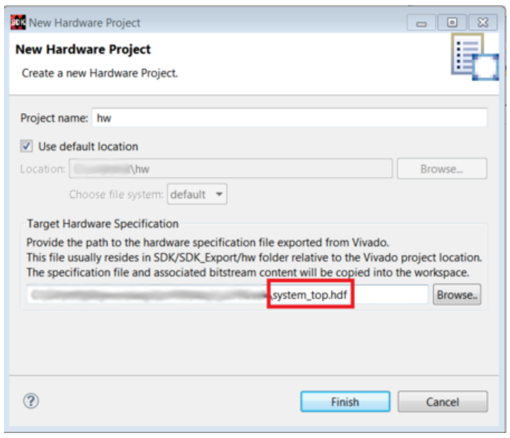
Language: [✓ C] C++

Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [✓ Create New] DAQ2_EBZ_bsp

Use existing



Project name: hw

[✓] Use default location

Location: [redacted]\hw

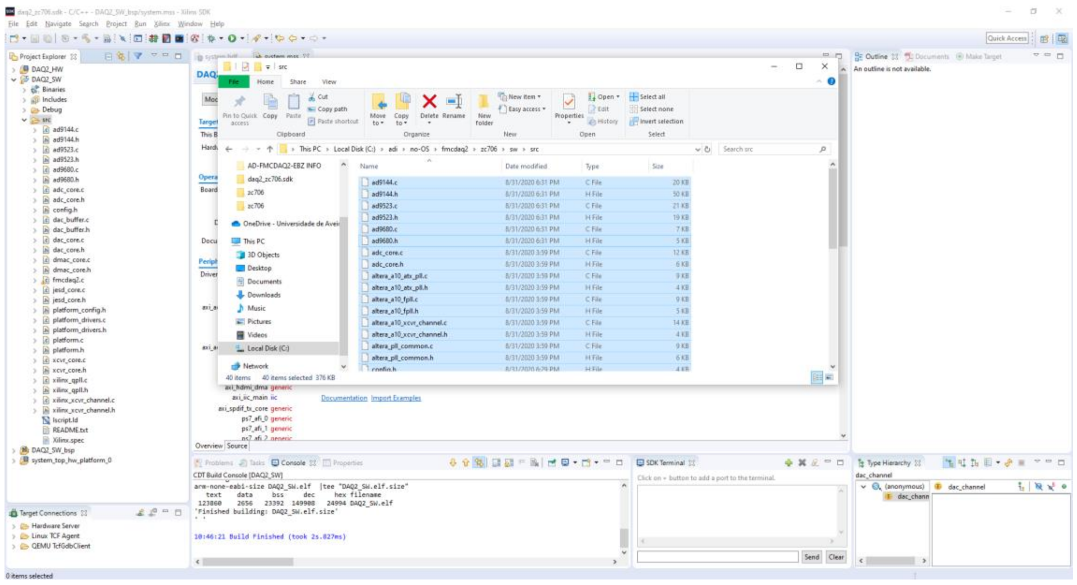
Choose file system: default

Provide the path to the hardware specification file exported from Vivado.
This file usually resides in SDK/SDK_Export/hw folder relative to the Vivado project location.
The specification file and associated bitstream content will be copied into the workspace.

[redacted]system_top.hdf

16. After having created your project, copy/drag all the files in the directory(below) to the src files of your application project in SDK.

```
/cygdrive/c/adi/no-OS/fmcdaq2/zc706/sw/src/
```



Local Disk (C:) › adi › no-OS › fmcdaq2 › zc706 › sw › src

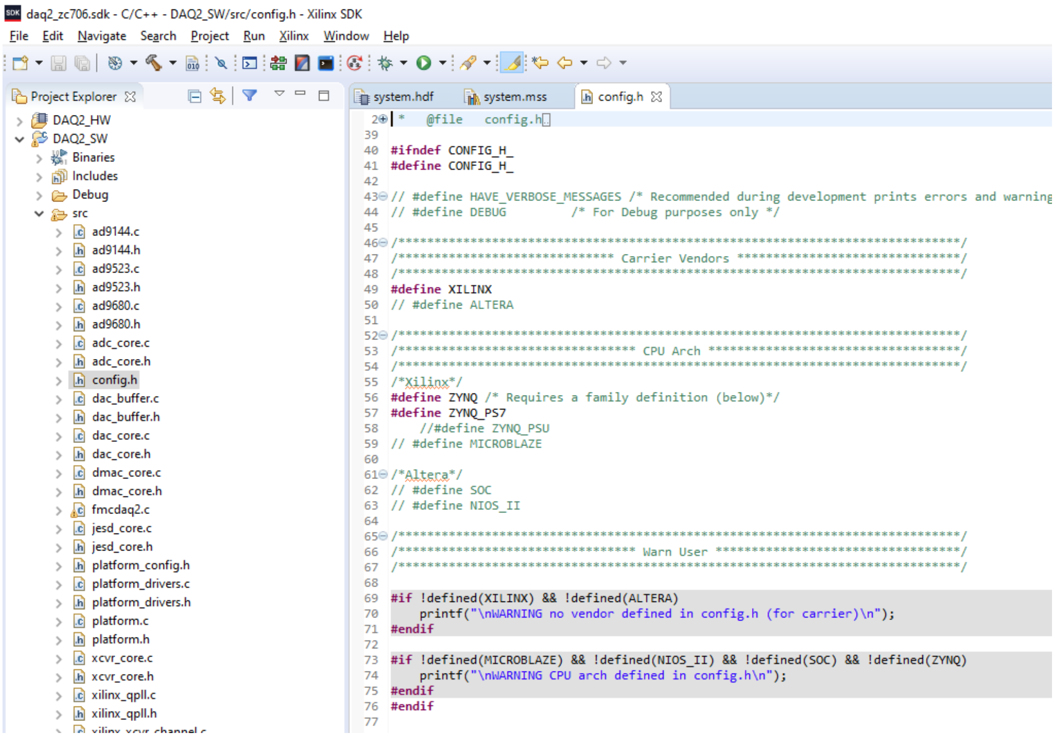
Name	Date modified	Type	Size
ad9144.c	8/31/2020 6:31 PM	C File	20 KB
ad9144.h	8/31/2020 6:31 PM	H File	50 KB
ad9523.c	8/31/2020 6:31 PM	C File	21 KB
ad9523.h	8/31/2020 6:31 PM	H File	19 KB
ad9680.c	8/31/2020 6:31 PM	C File	7 KB
ad9680.h	8/31/2020 6:31 PM	H File	5 KB
adc_core.c	8/31/2020 3:59 PM	C File	12 KB
adc_core.h	8/31/2020 3:59 PM	H File	6 KB
altera_a10_atx_pll.c	8/31/2020 3:59 PM	C File	9 KB
altera_a10_atx_pll.h	8/31/2020 3:59 PM	H File	4 KB
altera_a10_fpll.c	8/31/2020 3:59 PM	C File	9 KB
altera_a10_fpll.h	8/31/2020 3:59 PM	H File	5 KB
altera_a10_xcvr_channel.c	8/31/2020 3:59 PM	C File	14 KB
altera_a10_xcvr_channel.h	8/31/2020 3:59 PM	H File	4 KB
altera_pll_common.c	8/31/2020 3:59 PM	C File	9 KB
altera_pll_common.h	8/31/2020 3:59 PM	H File	6 KB
config.h	8/31/2020 6:29 PM	H File	4 KB

40 items 40 items selected 376 KB

CDT Build Console [DAQ2_SW]

```
arm-none-eabi-size DAQ2_SW.elf  |tee "DAQ2_SW.elf.size"
   text    data     bss     dec     hex filename
 123860    2656   23392  149908   24994 DAQ2_SW.elf
'Finished building: DAQ2_SW.elf.size'
' '

10:46:21 Build Finished (took 2s.827ms)
```

17. On your src directory in SDK, edit the config.h file and uncomment XILINX, ZYNQ and ZYNQ_PS7



daq2_zc706.sdk - C/C++ - DAQ2_SW/src/config.h - Xilinx SDK

```
 *   @file   config.h

#ifndef CONFIG_H_
#define CONFIG_H_

// #define HAVE_VERBOSE_MESSAGES /* Recommended during development prints errors and warning
// #define DEBUG          /* For Debug purposes only */

/******************************************************************************/
/****************************** Carrier Vendors ******************************/
/******************************************************************************/
#define XILINX
// #define ALTERA

/******************************************************************************/
/********************************* CPU Arch **********************************/
/******************************************************************************/
/*Xilinx*/
#define ZYNQ /* Requires a family definition (below)*/
#define ZYNQ_PS7
    //#define ZYNQ_PSU
// #define MICROBLAZE

/*Altera*/
// #define SOC
// #define NIOS_II

/******************************************************************************/
/********************************* Warn User *********************************/
/******************************************************************************/

#if !defined(XILINX) && !defined(ALTERA)
    printf("\nWARNING no vendor defined in config.h (for carrier)\n");
#endif

#if !defined(MICROBLAZE) && !defined(NIOS_II) && !defined(SOC) && !defined(ZYNQ)
    printf("\nWARNING CPU arch defined in config.h\n");
#endif
#endif
```

18. Everything should be ready, you should be free to edit **fmcdaq2.c** freely.

Appendix B

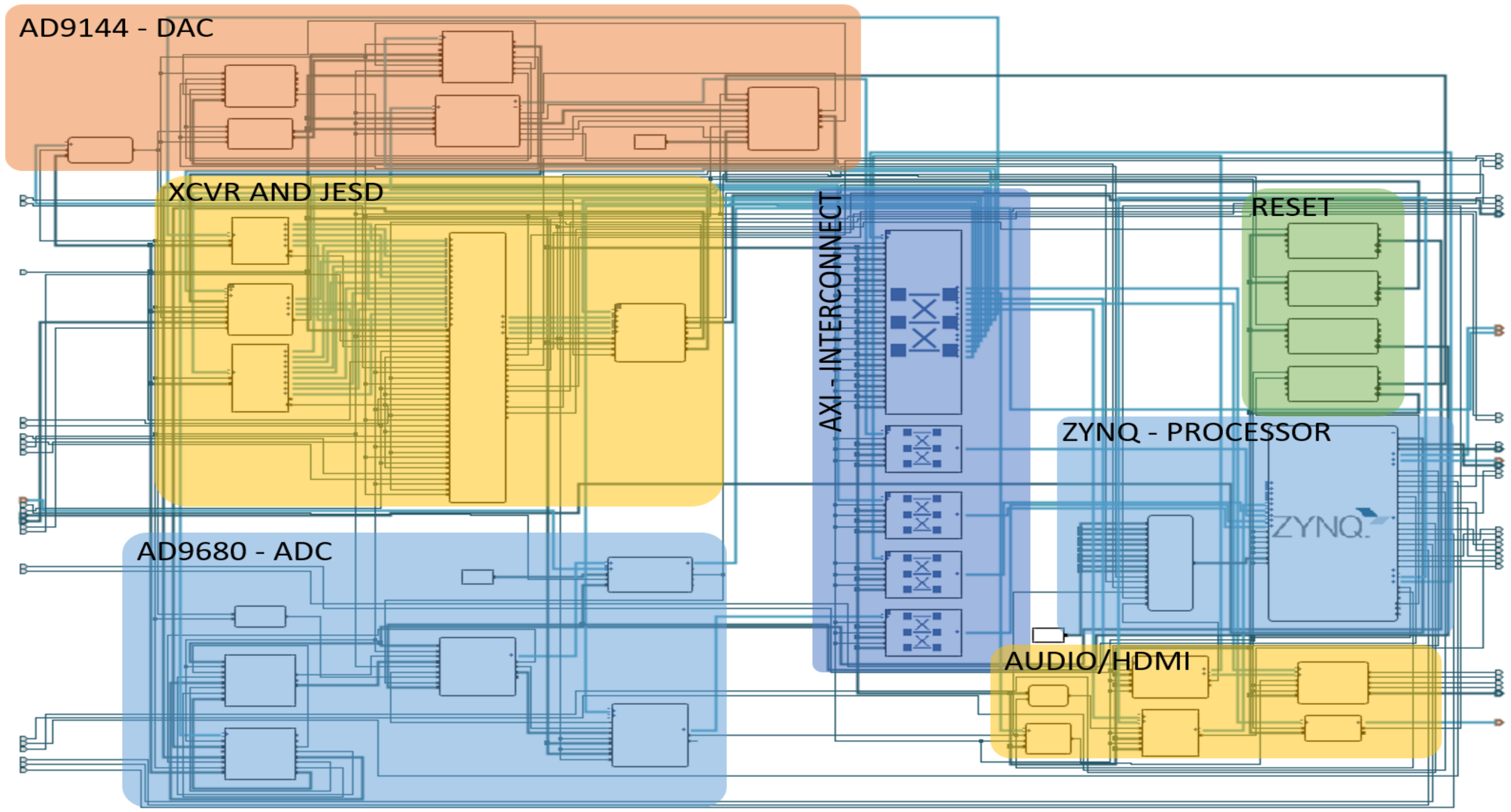


Figure 1: Vivado DAQ2 reference design.

The item list ahead summarizes the main components in the figure 1.

- **AD9144:** The systems transmitter, it will deal with all the translation from the digital domain to the analog domain.
- **AD9680:** The systems receiver that will deal with the translation from the Analog domain to the digital domain.
- **Axi-Interconnect:** Used to connect one or more Memory Mapped Axi-Master to one or more memory mapped slaves devices.
- **ZYNQ:** The Zynq architecture is Xilinx's all-programmable SoC, combines a dual-core Advanced RISC Machine (ARM) Cortex-A9 with a traditional FPGA whose interface is based on the AXI standard [65].
- **Reset:** This block is important when designing an FPGA project. A resets main function is to clear any pending errors and unknown states and bring a system back into its normal state.
- **XCVR and JESD:** The JESD is a multigigabit serial data link between converters and receiver, in our case the FPGAs. The purpose of the xcvr is to implement a JESD204B device link using a transceiver core [66].
- **Audio/HDMI:** This block wasn't used during the whole project, however it's function served for in case somebody wanted to use the FPGA with an Operating System (OS).