Investigation of White Rabbit based Frequency Synchronisation for Distributed MIMO Testbed

Hugo Lindholm & Jonathan von Bergen
hu2288li-s@student.lu.se & jo8766vo-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Liang Liu & Lina Tinnerberg

Examiner: Pietro Andreani

June 23, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

The demand for better networks and faster speeds within telecommunications has been an ever-growing ambition for the last decade. A promising candidate for dealing with this demand is distributed multiple-input multiple-output (D-MIMO) systems that combine large-scale antenna arrays and decentralized node architectures. However, evaluating the possibilities and limitations of such system requires real-time testbeds.

This thesis investigates White Rabbit as a potential alternative for synchronization in the Lund University Large Intelligent Surface (LuLIS) testbed. The testbed is a scalable field programmable gate array (FPGA) based testbed with 16 distributed AMD Zynq UltraScale+ ZCU216 nodes that can operate up to 256 antennas. The testbed currently achieves frequency synchronization by distributing an external reference clock over coaxial cables. Although effective over short distances, this becomes increasingly difficult as the node separation grows. An alternative is White Rabbit, which is an open-source synchronization technology that achieves sub-nanosecond time synchronization via fiber-optic communication over large distances.

A White Rabbit node was developed and integrated on the AMD Zynq UltraScale+ ZCU216 evaluation board. The implemented design is evaluated in terms of synchronization accuracy and frequency stability. The results shows that White Rabbits presents a highly stable frequency stability, while remaining within the cyclic prefix. Therefore, posing as a scalable and viable solution for replacing traditional coaxial cables in a distributed wireless communication system.

Popular Science Summary

Imagine an orchestra, where the musicians are scattered across an entire city. For the orchestra to sound good, each musician has to start at the same beat and have the correct tempo. Even a small delay would impair the performance.

This challenge is similarly faced by future wireless networks. Researchers are developing a system where many antennas corporate to send and receive radio signals. Instead of gathering all antennas at a stationary point, they can be distributed across buildings, streets and public spaces while still functioning as one coordinated system. This technology, called distributed multiple-input multiple-output (DMIMO), could increase network capacity and improve coverage.

The difficulty is keeping all antennas operating at the same pace and agree on exactly when to transmit and receive signals, much like the musicians following the same tempo and starting on the same beat. However, the precision required for these systems is so high that a measured difference in billions of a second would impact the performance.

Today, many researchers solve this problem by distributing a reference signal through a dedicated cable. While this is proven effective in small area it can be increasingly difficult as the system grows larger and antennas are being placed further apart.

This thesis investigates whether a technology developed by CERN called White Rabbit could provide a better solution. White Rabbit was originally developed for the Large Hadron Collider where scientific instruments must be synchronized with high precision over large distances.

The results show that White Rabbit can provide the necessary synchronization accuracy while offering greater flexibility and scalability than traditional cable-based solutions.

Acknowledgments

First, we would like to express our sincere gratitude to our supervisors, Liang Liu and Lina Tinnerberg, for their guidance and support throughout the project. We are also grateful to them for giving us the opportunity to work on this project and for helping us when we needed it most.

We also thank Dumitra Iancu for taking an interest in our project and for helping us while Lina was away on her honeymoon.

Special thanks go to Anders Johansson and Vilgot Snygg for their valuable insights and assistance during the measurements. Finally, we thank Sirvan Abdollah Poor for his support in the laboratory.

Contents

List of Figures

List of Tables

List of Acronyms

ADC Analog-to-Digital Converter

BMC Best Master Clock

BRAM Block Random-Access Memory

BS Base Station

BUFG Global Clock Buffer

CDR Clock and Data Recovery

CERN European Organization for Nuclear Research (Conseil Européen pour la Recherche Nucléaire)

CP Cyclic Prefix

CPU Central Processing Unit

DAC Digital-to-Analog Converter

D-MIMO Distributed Multiple-Input Multiple-Output

DMTD Dual Mixer Time Difference

DSP Digital Signal Processor

EEPROM Electrically Erasable Programmable Read-Only Memory

EIT Department of Electrical and Information Technology

FFT Fast Fourier Transform

FIFO First-In, First-Out

FPGA Field-Programmable Gate Array

GPIO General-Purpose Input/Output

GPS Global Positioning System

LC Inductor-Capacitor

LHC Large Hadron Collider

LM32 LatticeMico32

LuLIS Lund University Large Intelligent Surface

LUT Look-Up Table

LUTRAM Look-Up Table Random-Access Memory

MAC Media Access Control

MIMO Multiple-Input Multiple-Output

MMCM Mixed-Mode Clock Manager

MU-MIMO Multi-User Multiple-Input Multiple-Output

NIC Network Interface Controller

NTP Network Time Protocol

OOB Out-of-Band

OFDM Orthogonal Frequency-Division Multiplexing

OHWR Open Hardware Repository

PCS Physical Coding Sublayer

PFD Phase-Frequency Detector

PLL Phase-Locked Loop

PPS Pulse Per Second

PTP Precision Time Protocol

PTPv2 Precision Time Protocol Version 2

QAM Quadrature Amplitude Modulation

QPLL Quad Phase-Locked Loop

RF Radio Frequency

RFSoC Radio Frequency System-on-Chip

RISC-V Reduced Instruction Set Computer Five

RMS Root Mean Square

RTU Routing Table Unit

RX Receive

SDM Sigma-Delta Modulator

SFD Start Frame Delimiter

SFP Small Form-factor Pluggable

SMA SubMiniature version A

STM System Timing Master

Sync-E Synchronous Ethernet

TBI Ten-Bit Interface

TIE Time Interval Error

TX Transmit

UART Universal Asynchronous Receiver-Transmitter

UE User Equipment

UFL Ultra-Miniature Coaxial Connector

UTC Coordinated Universal Time

VCO Voltage-Controlled Oscillator

VCXO Voltage-Controlled Crystal Oscillator

WR White Rabbit

WRE White Rabbit Endpoint

WRF White Rabbit Fabric

WR PLL White Rabbit Phase-Locked Loop

WRPC White Rabbit PTP Core

WRPTP White Rabbit Precision Time Protocol

8b/10b 8-bit/10-bit encoding

Chapter 1
Introduction

The demand for better networks and faster speeds within telecommunications has been an ever-growing ambition for the last decades. The most recent addition to this was 5G, but the technology is still evolving to set the new standard for 6G. One of the key technologies for 5G is Massive MIMO [18]. This technique has improved the performance of beamforming, spatial multiplexing, single/multiple-user MIMO and null forming. In addition, this technology relies heavily on accurate synchronization within the system [19]. One of the next steps after Massive MIMO is believed to be Distributed MIMO, or D-MIMO. Compared to a Massive MIMO system, in a D-MIMO system, the BS (base station) antennas are spread out over a wider area, which improves the spectral efficiency and coverage [3].

At Lund University, one of the largest D-MIMO testbeds has been built by the Department of Electrical and Information Technology (EIT). It is called the Lund Univeristy Large Intelligent Surface, or LuLIS, testbed [4]. It supports 4 uplink UE (user equipment) and 256 base-station antennas. It can be configured to act as a typical Massive MIMO setup, or its antenna panel nodes can be distributed to create a D-MIMO system. Each antenna panel is connected to an AMD Zynq UltraScale+ RFSoC ZCU216 evaluation kit. These boards are daisy-chained together for data transfer and timing synchronization. Furthermore, they are frequency synchronized with three OctoClocks. The first OctoClock is connected to a 10 MHz reference signal, which is distributed to the other OctoClocks. The same two OctoClocks are then connected to each RFSoC ZCU216 board using a coaxial cable.

The clock synchronization for the current system setup works. However, if the system expands in terms of the distance between the BS nodes, it may not work or may be suboptimal. The reason being that the coaxial cable used in frequency synchronization has some physical limitations, such as damping, environmental distortion, and propagation impairment that are proportional to the coaxial cable length. Therefore, a new synchronization technique is needed. Synchronizing frequency over large distances is not a new problem, and there are several solutions that may be appropriate for this problem. One of these solutions is using White Rabbit [20] for synchronization. It offers both frequency and time synchronization with sub-nanosecond accuracy over large distances, up to 10 km [21], using fiber optic cables. It was originally developed by CERN to replace the old timing synchronization system for its particle accelerator. Today, it is open-source hardware and there are several FPGA implementations that are available online and free to

use. This makes White Rabbit a promising candidate for synchronization on the current LuLIS testbed.

1.1 Purpose and aims

- Investigate how White Rabbit works, as well as its requirements and performance.
- Implement White Rabbit on an AMD Zynq UltraScale+ RFSoC ZCU216 evaluation board and investigate whether the requirements for the LuLIS testbed are met. The requirements are the following:
  - Minimum use of external peripherals other than the board itself
  - Synchronization within the Cyclic Prefix time of 2.34 µs
  - Stable clock frequency
  - Frequency drift that does not interfere with the 60 kHz subcarriers
  - Hardware utilization that allows the White Rabbit implementation to be placed together with a BS node

The first goal is an in-depth investigation of how White Rabbit works. It includes how synchronization occurs, the hardware and peripherals needed for it, and how it has been implemented on the FPGA. The second goal is to create a working implementation of White Rabbit on the same FPGA boards that the LuLIS testbed uses. The implementation is then evaluated to determine whether it meets the synchronization requirements currently imposed on the LuLIS testbed.

1.2 Previous work

There are protocols and solutions other than White Rabbit that can handle system synchronization. The Network Time Protocol (NTP) [22] is one of them. It synchronizes all clocks in a system to within milliseconds of each other. However, this protocol does not address frequency synchronization, which is needed in our case.

A paper published two years ago explored a scalable synchronization technique for distributed multi-RFSoC server systems and Massive MIMO testbeds [23]. In the paper, they explored an alternative synchronization method instead of White Rabbit. Compared to White Rabbit, an external clock and a trigger distribution network are used. Based on the figure in the paper, it appears that the network is connected using SMA/coaxial cables. The paper claims that the reason White Rabbit was not used was the project cost overhead, the protocol overhead, and the extra implementation complexity. Furthermore, their experiment uses closely located lab equipment, which means that the paper likely did not take into account longer distances when synchronizing.

None of the mentioned solutions are good enough to handle frequency synchronization over long distances. The solution with the external clock and trigger distribution could work, but the use of long coaxial cables over longer distances is not optimal.

1.3 Outline

The thesis is organized as follows. Chapter 2 provides background information about wireless systems, synchronization, hardware and protocols used by White Rabbit. Chapter 3 focuses on White Rabbit, which presents the investigation of how the White Rabbit protocol works, the required hardware and software, and its implementation on the RFSoC ZCU216. Chapter 4, Measurements & Results, explains the measurement setup and presents the results. This is followed by the next chapter, Chapter 5, which discusses the results. Chapter 6, presents the conclusions and suggests future work.

Chapter 2

Theory & Background

In this chapter, the necessary core concepts for the project will be presented. Initially, an overview of wireless systems and the Lund University's large intelligent surface (LuLIS) testbed will be provided. Thereafter, the principles of synchronization and its associated instabilities will be discussed. Last, the foundational components and protocols of White Rabbit will be introduced.

2.1 Wireless System

2.1.1 OFDM

OFDM or Orthogonal frequency-division multiplexing is a widely used digital modulation scheme for transmitting data. It is adopted to the 5G standard and will be the primary frequency division multiplexing for future 6G as well. The scheme is based on splitting a high speed serial data stream into several slower parallel data streams that are transmitted on separate subcarriers. The subcarriers are evenly spaced and orthogonal to each other, eliminating the need for guard bands. As seen in Fig. 2.1, by choosing the correct spacing the zero crossing occurs at each other frequency center. By doing this, several signals can overlap without causing interference when sampling [1].

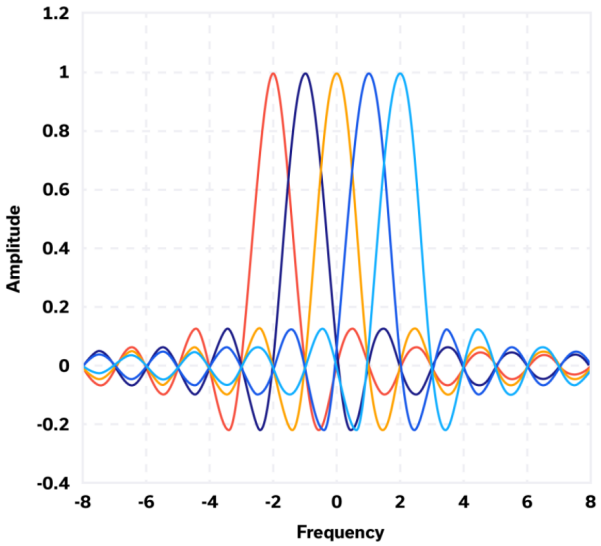

Figure 2.1: Five normalized subcarriers with the amplitude of 1 and frequency of 1 orthogonal to each other. [1]

The sinc functions of the subcarriers seen in Fig. 2.1 originate from a rectangular signal in the time domain that has been transformed with an inverse FFT. In practice, the data are stream bits mapped to modulation symbols, such as QAM symbols. These symbols are then divided over the subcarriers, and the inverse FFT of the signal is taken to create the OFDM signal that is then transmitted. On the receiver side, the received signal is transformed back with the FFT, which separates the subcarriers, and the transmitted data is recovered [1].

One problem that occurs when transmitting OFDM symbols is that adjacent blocks may overlap and interfere. A solution to this problem is, as seen in Fig. 2.2, to take a partial copy of the OFDM symbol and place it in the beginning. The partial copy acts as a guard interval between different OFDM symbols, and is called cyclic prefix [2].

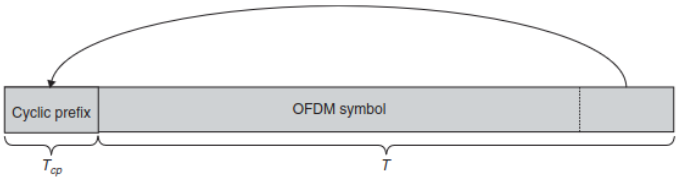

Figure 2.2: Cyclic prefix with OFDM symbol [2]

2.1.2 Multiple Input Multiple Output (MIMO)

In most modern wireless technology, multi-antenna systems are used to increase throughput and improve coverage. A widely adopted multiple-antenna technol-

ogy is MIMO, Multiple-input and multiple-output, it exploits the spatial domain to increase the robustness and throughput of a radio link. The technology uses spatial multiplexing for both the receiving and transmitting antennas to receive and deliver several streams of traffic at the same time and can be applied to multiple separate users within the system. An illustration of a Multi-user MIMO, MU-MIMO, system can be seen in Fig. 2.3 [3].

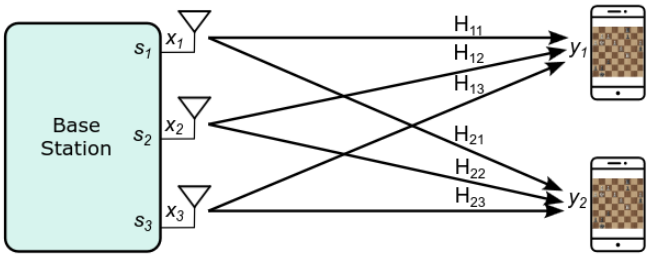

Figure 2.3: MU-MIMO system [3]

By distributing the antennas in the MIMO system over a wider area can spectral efficiency and coverage be improved [3]. An illustration of a distributed MIMO is shown in Fig. 2.5.

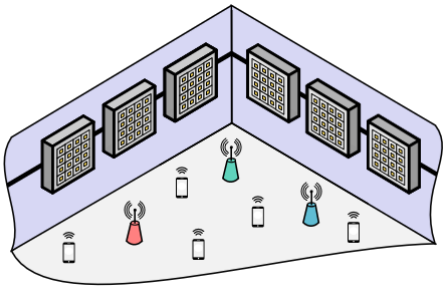

Figure 2.4: Distributed massive MIMO [3]

2.1.3 LuLIS Testbed - A D-MIMO testbed

The department of Electrical and Information Technology (EIT) at Lund University, have in recent time been investigating a D-MIMO setup. The main focus have been to investigate the developing processing architectures and hardware solutions to enable a D-MIMO system. This has been done with their testbed, the Lund University Large Intelligent Surface, or LuLIS, testbed presented in [4]. The system is built using 16 panels that act as one BS (base station). Each of the antenna panels has 16 antenna ports arranged as a 4x2 array of eight dual-polarized antennas, resulting in a total of 256 antennas across the system. Each panel is connected to an RFSoC ZCU216 evaluation kit which represents one node. All nodes are connected to each other with a daisy-chain topology to act as one BS.

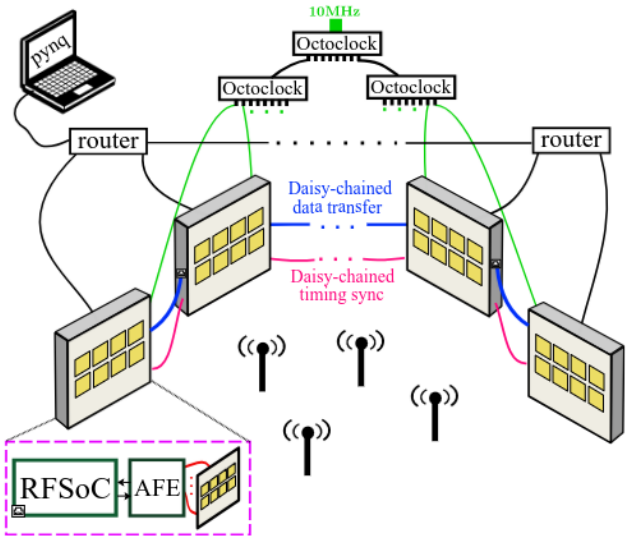

Figure 2.5: The LuLIS testbed setup [4]

The frequency synchronization is performed using a 10 MHz reference signal that is distributed using three CDA-2990 OctoClocks in a tree structure. These are connected using high frequency SMA-cables. The reference clock is fed to each FPGA board via the CLK104 add-on card, which consists of multiple PLLs that provide up-converted clock signals for the data converters. Timing synchronization is done by the UE sending a time synchronization signal through its GPIOs. The signal is then propagated to other BS nodes using UFL to SMA cables in a daisy chain fashion. The timing delay of each BS node is measured, and compensation is applied so that the BS node can process the received OFDM frames correctly.

The testbed follows the 5G standard of using OFDM as the transmission scheme and its system parameters can be seen in Table 2.1. From the table, the cyclic prefix time can be derived using the OFDM symbol duration of 16.67 µs, the cyclic prefix length of 144 samples, and the symbol size of 1024 samples. The result is a cyclic prefix time of 2.34 µs, see (2.1).

$$T_{cp} = \frac{16.67\mu s \cdot 144}{1024} = 2.34421875\mu s \tag{2.1}$$

The hardware utilization of a BS node is stated in Table 2.2. These values were extracted from the Vivado design tool after implementation. The node utilizes both SFP ports 2 and 3.

Table 2.1: System parameters for the LuLIS testbed [4]

Parameter	Value
Center frequency [GHz]	3.84
Channel bandwidth [MHz]	50
ADC/DAC sampling rate [GS/s]	2.457/4.915
Baseband sampling rate [MS/s]	61.44
FFT size [number of samples]	1024
OFDM symbol duration [us]	16.67
Subcarrier spacing [kHz]	60
CP length [number of samples]	144
FPGA clock frequency [MHz]	153.6
Number of antennas/RF chains per panel	16
Number of panels (J)	16

Table 2.2: Hardware utilization for BS node on the RFSoc ZCU216

Resource	Utilization	Available	Utilization %
LUT	125630	425280	29.54
LUTRAM	13176	213600	6.17
FF	158928	850560	18.69
BRAM	511.50	1080	47.36
DSP	1316	4272	30.81
IO	32	408	7.84
GT	2	16	12.50
BUFG	30	696	4.31
MMCM	1	8	12.50

2.2 Synchronization

2.2.1 Clock Synchronization

There are two concepts in synchronization, frequency and time. Time synchronization refers to a system in which all the nodes within it are running at the same time. The nodes may drift from each other over time. Synchronizing the time ensures that all nodes are at the same time in the space.

Frequency synchronization, or syntonization, refers to a system in which all the nodes clock is running at the same rate. In a system, each node's clock may run at different rates. Syntonization ensures that these rates are matched.

An analogy to this concept is two cars on a road. The speed of a car is the frequency, and the position of it is the time. An illustration of this analogy is presented in Fig. 2.6.

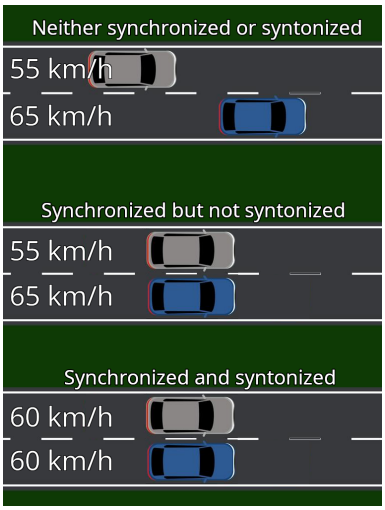

Figure 2.6: An illustration of synchronization vs syntonization. The system are represented by two nodes as cars. A cars speed is the frequency and its position is the time.

2.2.2 Jitter

A key factor affecting synchronization performance is jitter. Jitter is the deviation of a set signal edges from their ideal values. It falls into two main categories:

- **Random jitter:** Is a stochastic Gaussian process that is present in every system. In a phase noise plot, the jitter is characterized by the smooth sections along the bottom. Random jitter is unbounded and intrinsic to each system, and can be a significant contributor to the overall system jitter.

- **Deterministic jitter:** Is not random or intrinsic, and has a specific cause. The jitter is often periodic and narrowband, and therefore repetitious at one or more frequencies.

Fig. 2.7 illustrates the three primary types of jitter measurement: period jitter, cycle-to-cycle jitter and time interval error (TIE) jitter. Period jitter is the variation of one clock period compared to the average. Cycle-to-cycle jitter measures the cycle time between adjacent clock periods. TIE jitter, also known as phase jitter, is the deviation of time from the ideal clock over all clock periods [6].

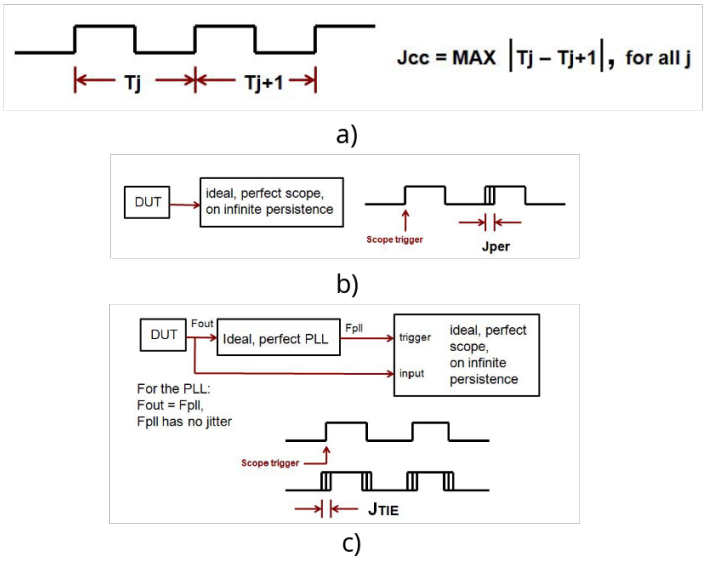

Figure 2.7: a) Cycle-to-cycle jitter b) Period jitter c) TIE jitter [5]

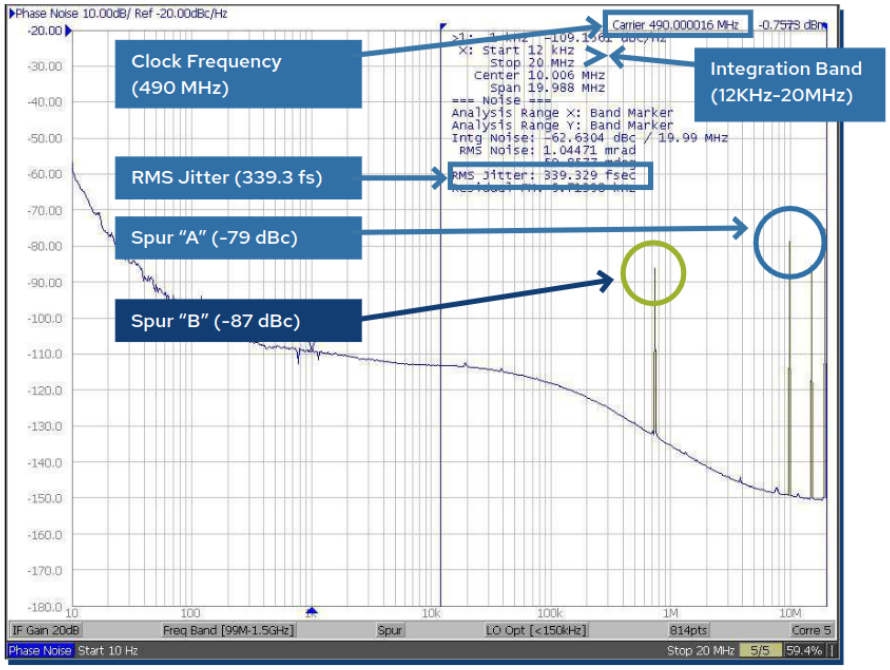

Figure 2.8: 490 MHz Clock Phase Noise Plot [6]

Phase noise is measured in the frequency domain and jitter in the time domain.

However, in theory, if the phase noise is measured to an infinite carrier offset, it would provide the same value as jitter. Phase noise is, as shown in Fig. 2.8, a single single-sideband plot of the spectrum that can be measured by either a spectrum analyzer or a phase noise analyzer. The spurs in a phase noise plot are the deterministic jitter, while the smooth sections, as discussed previously, are the random jitter. The phase noise plot also displays the root mean square, RMS, jitter, which is the integration of a specified frequency band [6].

Allan variance

Allan variance is a measure of frequency stability over time in clocks and oscillators. It estimates the stability of oscillator noise, such as white noise and flicker noise. The systematic errors, such as temperature or drift, should be considered separately. There are several versions of Allan variance, a widely used version is the Overlapping Allan variance. Its equation can be seen below in Equation 2.2, where $\sigma_y(\tau)$ is the Allan variance over time τ, τ is the averaging time, N is the number of phase/time-error samples, m is the averaging factor, and x_i is the time or phase error at index i [24].

$$\sigma_y^2(\tau) = \frac{1}{2(N-2m)\tau^2} \sum_{i=1}^{N-2m} (x_{i+2m} - 2x_{i+m} + x_i)^2 \tag{2.2}$$

Allan variance is a measurement used to compare the performance of different oscillators and clocks. A lower Allan variance measurement compared to another oscillator indicates better frequency stability at the corresponding time τ. The Allan deviation is the square root of the Allan variance; in other words, $\sigma_y(\tau) = \sqrt{\sigma_y^2(\tau)}$.

2.3 Hardware components and protocols

2.3.1 Precision Time Protocol - PTP (IEEE-1588)

PTP or Precision Time Protocol is an IEEE standard protocol (IEEE-1588). It is designed to synchronize and syntonize clocks within a distributed system of nodes that communicate using a network. Compared to other protocols, such as NTP, is PTP targeted to have both time synchronization and syntonization. The synchronization accuracy is achieved by hardware-based timestamps, and is the foundation for synchronization in White Rabbit. A PTP network is built upon the master/slave hierarchy, where the slaves free running oscillator is locked to the reference oscillator in the master. Furthermore, a network which implements PTP have three types of clocks,

- Ordinary clock: A node that either can be a master or a slave. The former is the source of the synchronization, and the latter is the follower of the source.
- Boundary clock: A device which synchronizes with the upstream reference clock and distributes it downstream, usually a network switch.

- Transparent clock: A node without an oscillator that matches with the upstream reference clock, and passes it down stream. The PTP package is modified to add the residency time before it is passed on.

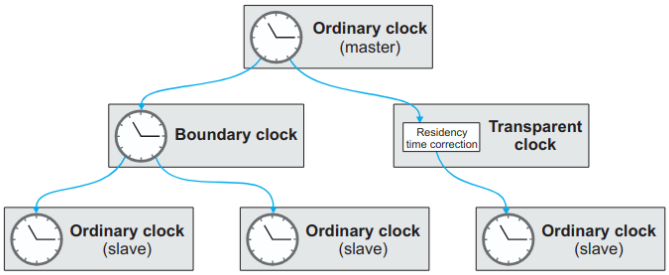

Figure 2.9: PTP clock hieracrchy example [7].

The node's role, either master or slave, is decided dynamically based on the Best Master clock (BMC) algorithm. The algorithm can in broad terms be explained as: if a node's neighbors have worse clock than itself, it becomes a master; otherwise it becomes a slave and synchronizes to the best available source.

The synchronization is done by sending Ethernet packets between two nodes which contains timestamps. These are then used to measure the delay and offset between the nodes. One of these schemes can be seen in Fig. 2.10. PTP defines two types of message: the first is a timestamped event message (can be seen in red in Fig. 2.10), and the second is a general message. The event message is used to communicate the offset and drift measurement, while the general measurement is used to identify other PTP nodes, create clock hierarchy, and exchange data such as settings and parameters. It can also include timestamps in certain setups. Illustrated in Fig. 2.10, the master node periodically sends an announce message that contains information about the master, its clock quality, and the data for the BMC algorithm.

The synchronization of the slave's clock is done using four timestamps (t_1, t_2, t_3, t_4). The gathering of these timestamps on the slave side is done by the event messages *Sync* (accompanied by *Follow_Up* in certain cases), *Delay_Up* and *Delay_Resp*. Timestamp t_1, is either embedded in the *Sync* message or the following message *Follow_Up*. Timestamp t_2 is saved when the *Sync* message is received, and in response, the message *Delay_Req* is sent back to the master. Timestamp t_3 is saved when *Delay_Req* message is sent. The last message *Delay_Resp* is the response of the previous message and contains timestamp t_4 which indicates when the message *Delay_Req* was received.

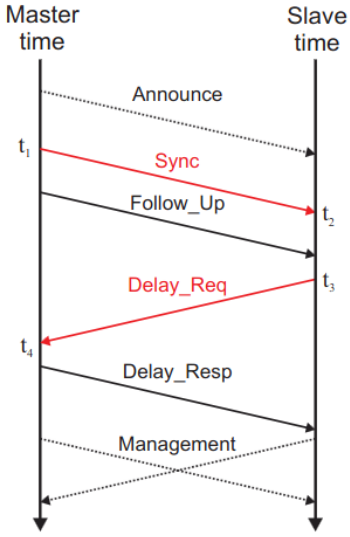

Figure 2.10: PTP clock synchronization message exchange with a master and slave node [7].

With the four timestamps, a roundtrip delay ($delay\_course$) and a one-way delay (one_way_delay) are calculated (see (2.3) and (2.4)). The slave corrects its clock using (2.5), where the goal is the sum to be zero. To maintain synchronization and syntonization, the event message *Sync* is sent out periodically, enabling the slave to maintain (2.5).

$$delay\_course = (t_4 - t_1) - (t_3 - t_2) \tag{2.3}$$

$$one\_way\_delay = \frac{delay\_course}{2} \tag{2.4}$$

$$t_2 - t_1 + one\_way\_delay = 0 \tag{2.5}$$

After the initial synchronization and syntonization are fixed, the main concern is maintaining the clock offset. If the slave is syntonized to the master, the synchronization can be maintained by the master only sending the *Sync* message without awaiting a response. The reason being the link delay, or $delay\_course$ in (2.3), depends only on the slow changes, for example temperature changes. In typical applications are these negligible, meaning a new one way delay is not needed to be calculated. Therefore, the slave does not need to respond with a *Delay_Req* message [7].

PTP has evolved with several versions. The first iteration being IEEE 1588-2002 also called PTP, followed by IEEE 1588-2008 called PTPv2. PTPv2 introduced new features such as the transparent clock, but kept the same message format as the one mentioned above [25]. The newest PTP version is IEEE 1588-2019 which is called PTPv2.1. White Rabbit is based on PTPv2 which led to the

creation of the standard PTPv2.1, where White Rabbit is the low latency profile within it [26].

2.3.2 DMTD phase detector

There are several ways to measure phase difference. One simple way is to use a fast digital counter. However, this approach has the problem of needing a much higher frequency for the counter than the measured clock frequency. Therefore, if an already high frequency is used, having an even higher frequency for the counter may be difficult.

Analog DMTD

The Dual Mixer Time Difference (DMTD) phase detector is a much more convenient way of measuring the phase difference [7]. Illustrated in Fig. 2.11 is an analog DMTD. It consists of a local oscillator and two identical mixers with image-rejection filters connected to a time interval counter.

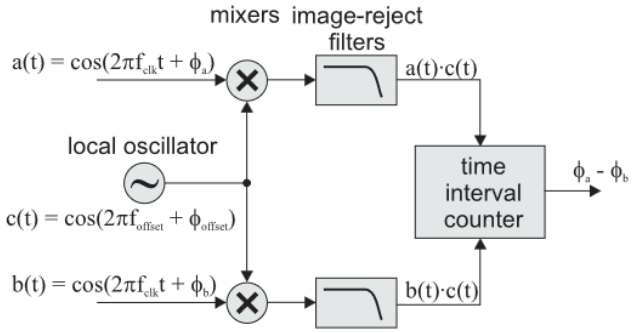

Figure 2.11: Block diagram of an analog DMTD phase detector [7]

The input is two measured clocks $a(t)$ and $b(t)$ which have the same frequency (f_{clk}) and an amplitude of 1. The first operation of the DMTD is to multiply the input clocks with the local oscillator ($c(t)$), which has a frequency (f_{offset}) offset by a few Hertz of f_{clk}. The result consists of two components: one with high frequency (2.7) and one with low frequency (2.8). The low pass filter filters out the high frequency product, and left is a down-conversion of the two clocks with the same phase relation. Thus, the phase relation between $a(t)$ and $b(t)$ can be measured using a simple counter [7].

$$a(t) \cdot c(t) = cos(2\pi t f_{clk} + \phi_a) \cdot cos(2\pi t f_{clk} + \phi_{offset}) \tag{2.6}$$

$$= \frac{1}{2} cos(2\pi t(f_{clk} + f_{offset}) + \phi_a + \phi_{offset}) \tag{2.7}$$

$$+ \frac{1}{2} cos(2\pi t(f_{clk} - f_{offset}) + \phi_a - \phi_{offset}) \tag{2.8}$$

Digital DMTD

The digital counterpart of the analog DMTD is presented in Fig. 2.12. The digital DMTD works just like its analog counterpart, it converts a phase shift from a high frequency domain to the low frequency domain. By doing so, it provides a large multiplication effect and yields the possibility of measuring the phase difference between two frequencies that are almost identical. Furthermore, it is also possible to implement it on an FPGA [8]. This is one of the biggest advantages of it compared to its analog counterpart. The analog DMTD needs discrete external components, such as mixers and filters, which makes it an inconvenience for multiport-port applications, like White Rabbit [7].

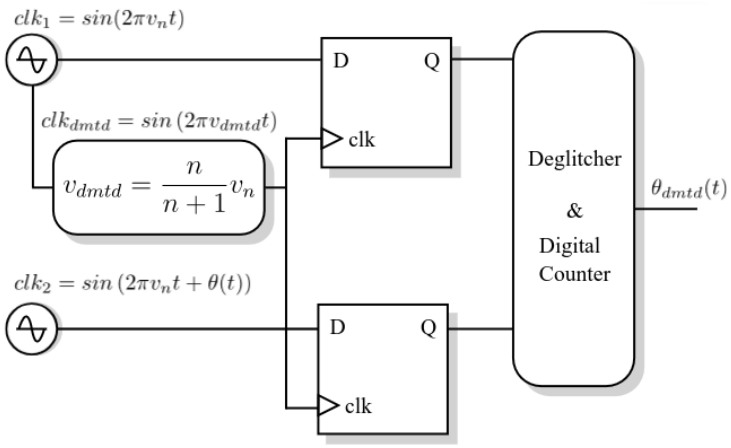

Figure 2.12: Digital DMTD phase detector schematic [8]

In Fig. 2.12 a block diagram of the Digital DMTD can be seen. It consists of two separate clocks, clk_1 and clk_2, where clk_2 has a slight phase offset from clk_1. These two clocks are inputted as D to one D flip-flop each, which acts as multipliers (see previous subsection 2.3.2). One of the clocks, in this case clk_1, is fed to a smaller block that creates an offset clock by a couple of Hertz from the input clock (clk_{dmtd}). The DMTD clock (clk_{dmtd}) then acts as the clock input for both D flip-flops. The DMTD clock is used to create the low frequency beat note, which is represented in (2.9). The beat frequency, clk_{beat}, is what dictates the overall performance of the system [8].

$$v_{beat} = |v_n - v_{dmtd}| \tag{2.9}$$

The beat frequency, clk_{beat}, is directly linked to the DMTD's time step, τ_{dmtd}, meaning a smaller clk_{beat} also leads to τ_{dmtd} being smaller. The DMTD's time step is the time the clk_{dmtd} has moved away from clk_1 and clk_2, as seen in Fig. 2.13 [8].

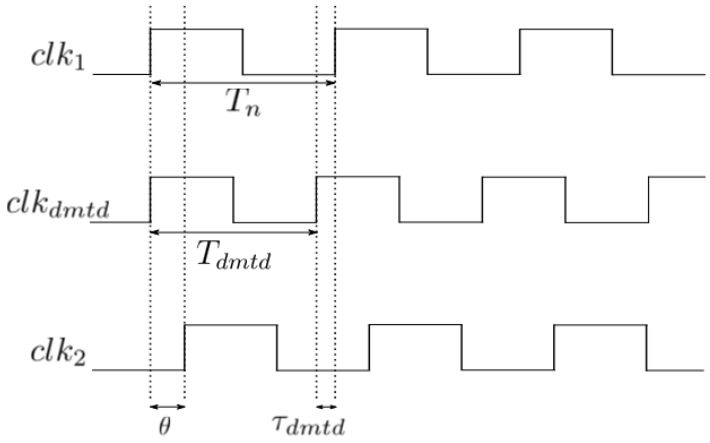

Figure 2.13: An example of the clock signals in the Digital DMTD [8]

By measuring the time difference between the rising edges of the D flip-flop outputs, the phase difference between clk_1 and clk_2 can be determined. The measurement is effectively amplified by the ratio of the original frequency to the beat frequency. The phase difference between clk_1 and clk_2 can therefore be modeled as in (2.10), where T_{beat} is the period time of clk_{beat}, $\Delta\varphi$ is the phase difference and Δt_{dmtd} is the time difference of the D flip-flops, as seen in Fig. 2.14 [8].

$$\Delta\varphi = \frac{2\pi\Delta t_{dmtd}}{T_{beat}} \tag{2.10}$$

$$\Delta t_{dmtd} = n_{cycles} \cdot T_{dmtd} \tag{2.11}$$

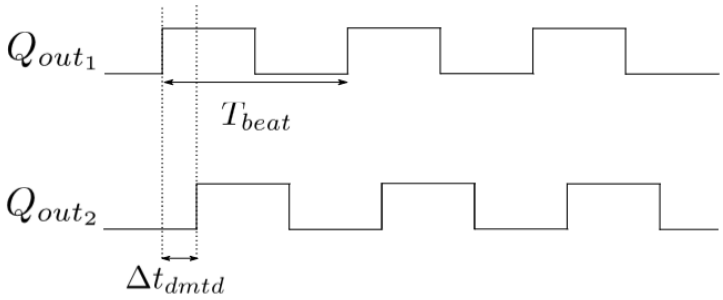

Figure 2.14: The digital DMTD's D flip-flop output signals. [8]

The time difference Δt_{dmtd} can be further explained by (2.11), where n_{cycles} denotes the number of cycles of clk_{dmtd} that occur before the other flip flop be-

comes high. This is not shown in Fig. 2.14 but can be seen in Fig. 2.15 for a better understanding.

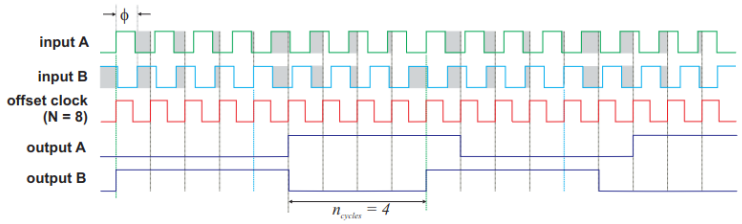

Figure 2.15: Another example of the DMTD's flip-flop signal with an offset clock [7] (this case being equivalent to clk_{dmtd}).

The achievable measurement precision can potentially be very precise. According to (2.12), where δ_{dmtd} is the measurement precision, v_n is the clock frequency, v_{beat} is the beat frequency and δ_c is the accuracy of the counter, a precision of 80 fs can be achieved. This assumes a clock frequency of 125 MHz, a beat frequency 10 kHz and time interval counter resolution of 1 ns [8].

$$\delta_{dmtd} = \frac{v_{beat}}{v_n} \cdot \delta_c \Rightarrow \frac{10kHz}{125MHz} \cdot 1ns = 80fs \tag{2.12}$$

2.3.3 Optical Gigabit Ethernet

Ethernet is widely used in today's networks for sending large amounts of data between machines and computers. Ethernet is well standardized by the IEEE 802.3 working group and has evolved over the years, offering higher and higher data rates.

The Ethernet version used by White Rabbit is the 1000BASE-BX10 [27]. This Ethernet version is a part of the 802.3ah standard. The protocol supports 1 Gb/s data transfer (at a 1.25 Gb/s clock rate) over full duplex, fiber-optic link, and uses 8b/10b encoding when transmitting data. Its communication medium is through a single fiber-optic cable, where it uses two wavelengths, 1490 nm and 1310 nm, and can transmit up to a distance of 10 km [28].

Low level encoding

Before reaching the fiber, Ethernet frames need to be properly encoded and serialized to match the properties of the transmission channel. The low-level data encoding and decoding is depicted in Fig. 2.16 [9].

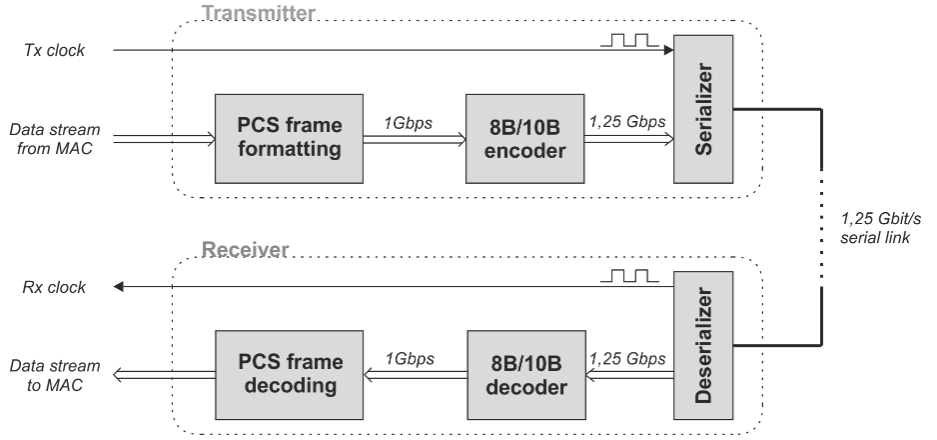

Figure 2.16: Block diagram of the encoding/decoding of Base1000-X data [9].

PCS - Physical Coding Sublayer The first step is encapsulate the data stream in the PCS, or Physical Coding Sublayer. It maps the bytes from the MAC data stream into data characters (Dx.y), and inserts control characters (Kx.y). There are a total of 256 data characters and 12 control characters, mapped out according to the 8b/10b encoding standard. A couple of these control characters, such as start of frame, end of frame and idle, can be seen in table 2.3 [9]. The column marked as "Symbols" is the PCS-level symbolic name of the 8b10b encoding.

Symbols	8B10B symbol	Description
/C1/	K28.5 / D12.5	Configuration 1
/C2/	K28.5 / D2.2	Configuration 2
/I1/	K28.5 / D5.6	Idle (correcting disparity)
/I2/	K28.5 / D16.2	Idle (preserving disparity
/R/	K23.7	Carrier extend
/S/	K27.7	Start-of-packet
/T/	K29.7	End-of-packet
/V/	K30.7	Error

Table 2.3: A set of control characters used by 1000Base-X [9]

/C1/ and /C2/ are used to exchange the devices capabilities during the auto-negation process. /I1/ and /I2/ symbolizes idle characters being sent out when no data are transmitted. /R/ is used instead of /I1/ and /I2/, if the PCS operates in half-duplex mode and transmits a burst of frames to fill the gaps. /S/ and /T/ indicates where the frame starts and ends, while /V/ indicates when an error has occurred during the transmission. The frame should then be dropped since it is

prone to errors [9].

8b/10b encoding As stated above, the PCS maps the MAC data stream to data character (Dx.y) and inserts the control character (Kx.y). The 8-bit input data is encoded to 10 bit data groups from a standardized lookup table [9]. A more thorough explanation of the encoding can be found in Appendix C.

Serializer/Deserializer The final step before sending the 8b/10b encoded data is the serializer. It takes the parallel 10-bit data and converts it into a serial data stream. The data stream is then transmitted over the optical fiber at a serial bit rate of 1.25 Gb/s. The stream is clocked at 1.25 GHz by a reference clock of 125 MHz, which is multiplied by a factor of ten in a PLL. The internal structure of the serializer and deserializer is illustrated in Fig. 2.17 [9].

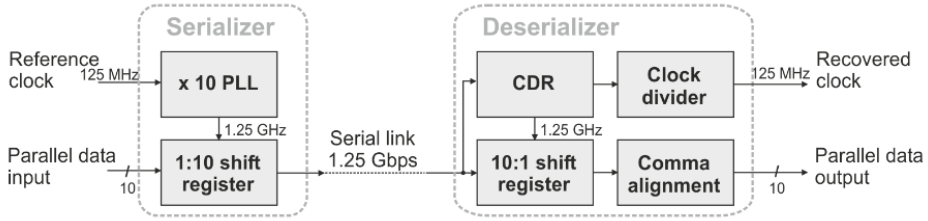

Figure 2.17: The internal structure of the serializer/deserializer. [9]

The deserializer recovers the 1.25 GHz clock from the incoming data stream with the help of the CDR (Clock and Data Recovery). The CDR drives a 10:1 shift register with the recovered clock of 1.25 GHz and converts the received serial data to parallel 10-bit words. The 1.25 GHz recovered clock is at the same time divided by a factor of 10 to output a recovered clock signal of 125 MHz. However, the shift register does not know the word boundaries and only outputs 10-bit chunks. The alignment may therefore be incorrect, and a *Comma alignment* component is used to detect the correct word boundaries. This component looks for the code group /K28.5/, and when found, shifts the data by a number of bits (depending on how misaligned the shift register is) to align the data with the inter-word boundaries. This can also be seen as adjusting the phase of the recovered 125 MHz clock to match the inter-word boundaries. The process can be seen in Fig. 2.18.

The 8b10b encoding have two requirements that makes comma alignment possible. First, the receiver clock recovery circuits require that a 10-bit code group cannot have more than 5 consecutive ones or zeros. Second, the "comma" code group pattern cannot appear across the boundary between two code groups. The comma code group is /K28.5/ and appears in sequence with the symbols /I1/, /I2/, /C1/ and /C2/. The reason for this requirement is that it prevents confusion in the receiver when detecting the boundaries of received 10-bit code groups [9].

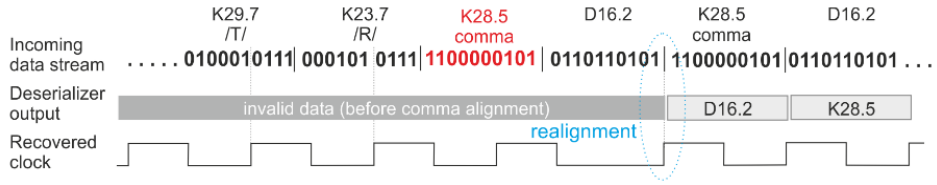

Figure 2.18: The comma alignment process. [9]

PCS frame decoder and 8b/10b decoder The whole data encoding process is then reversed on the receiving side. As seen in Fig. 2.16, the 8b/10b decoders and PCS frame decoders reverse the process by processing the frames back to raw Ethernet frames. They also perform a first level error checking, such as detection of faulty 8b/10b code groups and loss of communication [9].

2.3.4 Synchronous Ethernet - Sync-E

Typical Ethernet networks are asynchronous, meaning every node uses its own free-running oscillator. As a result, differences in clock frequencies exist between transmitting and receiving nodes when packets are exchanged. These frequency differences are compensated for using asynchronous packet buffers.

Synchronous Ethernet, or Sync-E, extends the Ethernet standard by frequency synchronizing (syntonizing) the clocks across the whole network, or in other words ensuring that all nodes operate at the same clock rate. This is similar to traditional telecommunications network, such as SDH/SONET. Sync-E uses a network hierarchy structure with the top node referred to as the System timing master (STM). The STM interoperates the primary clock, for example, an atomic clock or a GPS clock, and outputs the reference clock to its slaves. The slaves then uses a PLL to recover the clock from the upstream and propagates it downstream for additional nodes. See Fig. 2.19 for a comparison between Standard and Synchronous Ethernet [7].

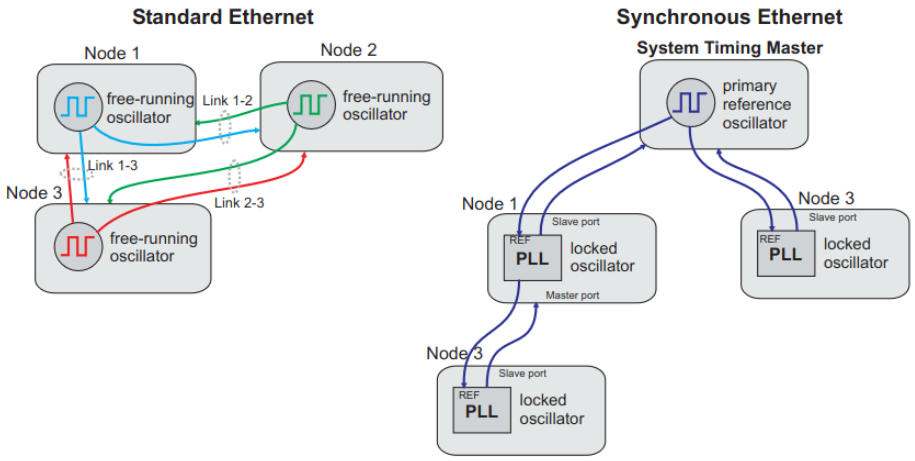

Figure 2.19: Clock network comparison with Standard Ethernet and Synchronous Ethernet [7].

Sync-E responsibility within White Rabbit is syntonizing the clocks. Therefore, the role of the PTP protocol is reduced to measurement and compensation of the clock offset. Sync-E measures the delays in term of phase shift, which facilitates a more precise measurement than direct time measurements. Sync-E is therefore a crucial part for achieving sub-nanoseconds synchronization in White Rabbit [7].

2.3.5 Fractional-N synthesizer

Fractional-N synthesizer allows for frequency resolutions that are a fractional part of the reference frequency. For modern electronics such as cell phones that support multiple standards, this is a must. A cell phone must synthesize frequencies for standards such as GSM, Bluetooth and WiFi where most of these frequency channels do not bear an integer ratio with respect to the reference frequency [10]. The basic structure of a fractional-N synthesizer is illustrated in Fig. 2.20. The fractional division is achieved by dynamically changing the division ratio from M to M+1 such that the average division ratio becomes M+K/F. Therefore, dividing by M+1 K times and by M F-K times, the output frequency becomes $F_{out} = F_{REF}(M + K/F)$, where F is the fractional resolution of the device and K is the fractional channel of operation [29].

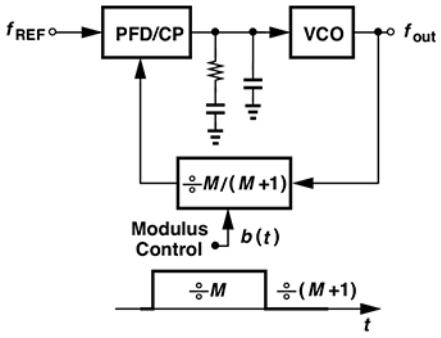

Figure 2.20: Fractional-N synthesizer [10]

One may be tempted to toggle periodically between M and M+1. However, by doing this, we get fractional spurs. The VCO settles at the average of the toggled values since the loop is slow and the VCO does not have time to lock to Mf_{REF} or $(M+1)f_{REF}$. However, the divider voltage is not periodic and experience periodic phase modulation thereby exhibiting sidebands. These sidebands are shifted up and down by f_{REF} becoming components referred to as fractional spurs. To reduce the magnitude of these spurs we can switch between the two values randomly while preserving the dividing ratio. However, due to the randomization, additional phase noise arises. To minimize this phase noise a method called noise shaping can be used to randomize the dividing ratio such that the phase noise within the loop bandwidth is small. A sigma delta modulator is such a digital system whose quantization noise is shaped into high-pass spectrum pushing the energy to high frequencies. A basic $\Sigma\Delta$ fractional-N synthesizer is depicted in Fig. 2.21 where α is the fractional ratio [10].

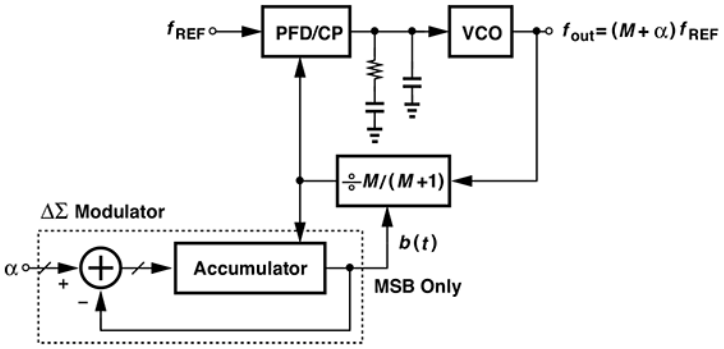

Figure 2.21: Basic $\Sigma\Delta$ fractional-N synthesizer [10]

The individual components for the fractional-N synthesizer are described in details in Appendix A.

Chapter 3
White Rabbit

This chapter presents our investigation of White Rabbit. First, the White Rabbit protocol is explained. Second, the hardware that is used within White Rabbit is explained, along with an explanation of how the protocol and the components described in the background section are integrated. In the same section, there is a small subsection in which the software is described. Third, a specific implementation of White Rabbit called Light Rabbit is analyzed. The Light Rabbit implementation is used as a reference for our own implementation on the RFSoC ZCU216 board, which is presented in the final section.

3.1 The White Rabbit Protocol

White Rabbit is a protocol that combines IEEE 1588-2008 (PTPv2), Synchronous Ethernet, digital phase measurement (Digital DMTD), and self calibration techniques. It was developed by CERN (The European Organization for Nuclear Research) to meet the stringent timing and control requirements of the Large Hadron Collider (LHC) [9]. The protocol makes it possible to achieve the following [21]:

- Sub-nanosecond synchronization accuracy and picosecond synchronization precision.
- Synchronization of large systems over distances of up to 10 km between nodes.
- Support for connecting thousands of nodes.

It is also open source, allowing anyone to implement it in their own systems. Several implementations are freely available and ready to use, and can be found on White Rabbits GitLab [12].

Fig. 3.1 is an example of the synchronization performance for White Rabbit. This specific example is the very first White Rabbit Switch that was created [7].

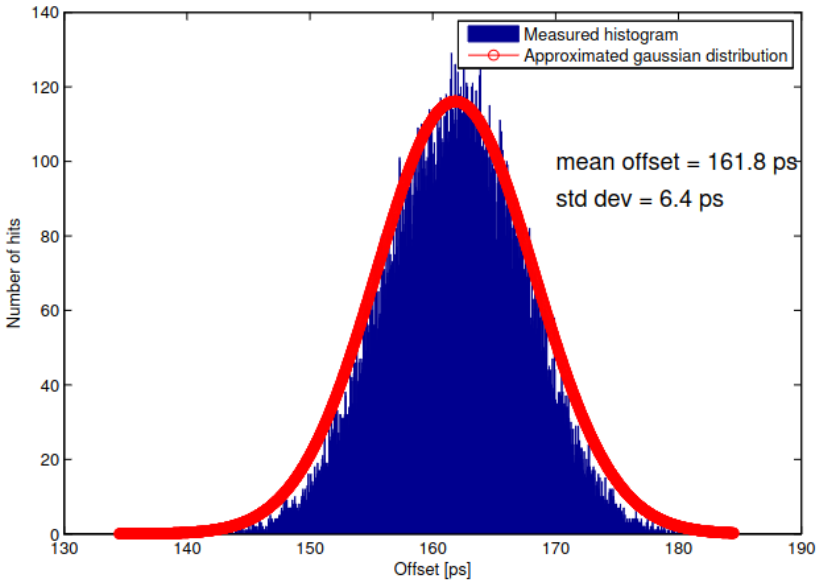

Figure 3.1: Time offset measurement showing the performance of a White Rabbit switch [7].

3.1.1 Network topology

The White Rabbit protocol is based on Ethernet and therefore allows it to send both data and timing packets. For regular data, White Rabbit network operates just like a standard Ethernet-based network. For the timing, it differs a bit, however, the structure is quite similar to Synchronous Ethernet (see Section 2.3.4). A picture of a typical White Rabbit network can be seen in Fig. 3.2.

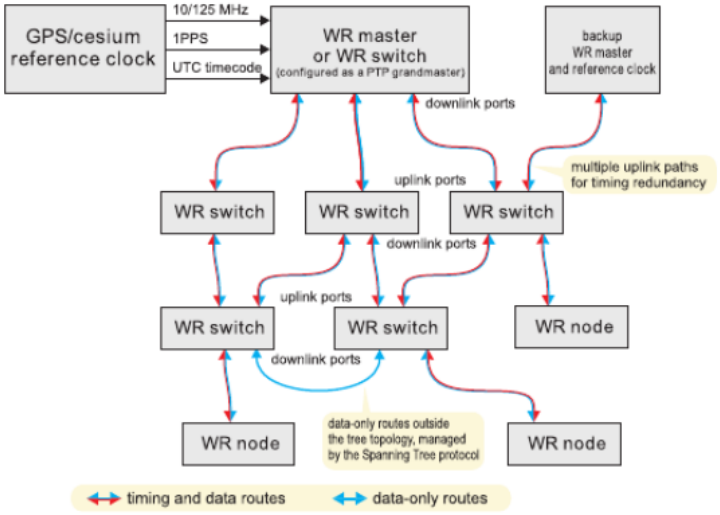

Figure 3.2: White Rabbit network topology [9]

Observing the figure, it can be seen that it has at least one White Rabbit master node (the System Timing Master, or STM) at the top and below it several layers of White Rabbit switches and slave nodes. The White Rabbit master node can be a master or a grandmaster. A grandmaster node is synchronized to an external 1-PPS and 10 MHz clock signal (that is very precise), while a master node uses its own free-running oscillator as the reference clock [12]. The reason for having two kinds of master modes is that one system may only need a common timescale, and not a reference to UTC. Thus, not having the need to use an external reference oscillator, and therefore save some resources. As seen by the picture it may exist more than one master. In such a situation, only the primary master is used while the others serve as backup, in case the primary master stops working. The White Rabbit switches are responsible for creating a tree structure. A White Rabbit switch is a multi-port device and is analogous to the PTP boundary clocks (see Section 2.3.1). The switch, like a PTP boundary clock, synchronizes with the upstream reference clock, and propagates its clock and precise timing to other nodes downstream. A White Rabbit node, both master and slave, is a single-port device, like a PTP ordinary clocks. They can either synchronize their local oscillators to a timescale from a reference clock through the network, or feed the network with their own local oscillators as a reference clock [9].

3.1.2 Syntonization

The process of syntonization two White Rabbit nodes, a master and slave, is outlined as follows. Initially, when the nodes are connected with a fiber-optic cable they will start to receive each other idle pattern. By receiving an idle pattern the nodes recognize that they are connected. The master is the one that initializes the communication by broadcasting the *ANNOUNCE* message. The slave node

observes this message and responds with *SLAVE_PRESENT* to indicate it is also a WR-compatible node. After this message exchange has been successful, a White Rabbit link has been established and syntonization can begin.

The syntonization process is very similar to the one used by Synchronous Ethernet (see section 2.3.4). The process starts with a slave receiving a *LOCK* message from the master. The slave recovers the master reference clock from the data downstream, and locks it clock to it by a PLL. After, the lock has occurred the subordinate responds back with a message *LOCKED*, and the syntonization process is completed [9]. See section 3.1.5 for a diagram of these exchanges.

3.1.3 Link delay model

White Rabbits ability to achieve sub-nanosecond accuracy and picosecond precision synchronization comes from its link delay model. The PTP protocol assumes symmetric link delay, which reduces the synchronization accuracy (see section 2.3.1). White Rabbit instead considers a few different sources of link asymmetry to achieve better synchronization accuracy. The protocol estimates both master-to-slave and slave-to-master delays [9]. It achieves this through the PTP protocol together with the Digital DMTD phase detector. This is done by attaining correct fixed delays and variable delays according to Fig. 3.3 below.

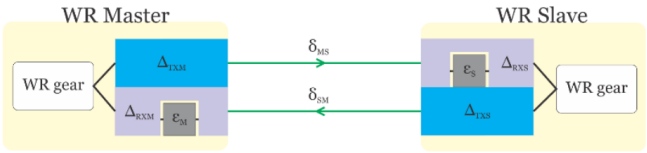

Figure 3.3: Link delay model used in White Rabbit [11].

The fixed delays in the figure above are Δ_{TXM}, Δ_{RXM}, Δ_{TXS} and Δ_{RXS}. These delays represent the master/slave fixed transmission circuit delay. They are measured during the calibration of the transceiver by following the calibration guide, and set as constants within the node [30].

There are also delays which vary between connections. They are the bitslide delays ϵ_M and ϵ_S. These are measured every time a link between two White Rabbit nodes is established. The measurement is performed by turning off the comma alignment component in the transceiver (see section 2.3.3) [7]. Manual bit-shifting is performed until a valid 8b10b code is detected. The amount of bits shifted is then saved into the bitslide delays variables. See Fig. 3.4 for an example of the bitslide delay.

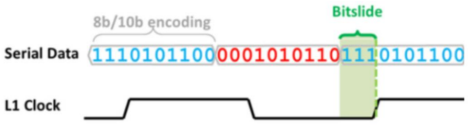

Figure 3.4: Bitslide delay of the comma alignment [11]

The variables δ_{MS} and δ_{SM} are the delays in transmission through the medium, in this case a bidirectional fiber optic cable. These delays vary depending on environmental factors such as temperature, but also link asymmetry since the bidirectional fiber optic uses two different wave lengths. Therefore, the propagation velocity of the different directions are not same. To account for the asymmetry, a formal calibration can be done to find the fiber asymmetry, α, of the cable [11]. The fiber asymmetry is defined as the ratio of δ_{MS} and δ_{SM}, as seen in (3.1).

$$\alpha = \frac{\delta_{MS}}{\delta_{SM}} - 1 \tag{3.1}$$

It is important to note that the calculation of the variables δ_{MS} and δ_{SM} only takes the fiber asymmetry into account, and not environmental factors.

As mentioned above, White Rabbit does not assume that the delay from master to slave, $delay_{MS}$, and from slave to master, $delay_{SM}$, are equal. Instead, it calculates the delays according to (3.2), based on the model shown in Fig. 3.3. The master-to-slave delay is used in the calculation of $offset_{MS}$, which is used to adjust the slave clock to match the master during synchronization. The slave-to-master delay can be expressed similarly but with Δ_{rxm}, δ_{SM}, and Δ_{RXS} instead [9].

$$delay_{MS} = \Delta_{TXM} + \delta_{MS} + \Delta_{RXS} \tag{3.2}$$

3.1.4 Synchronization

Utilizing the previously discussed sections: PTP, DMTD phase detector and the Link delay model (see Section 2.3.1, 2.3.2 and 3.1.3) can synchronization of a slave to its master be accurately performed. The synchronization process can be divided into two parts as illustrated in Fig. 3.5 [7].

1. Initial synchronization: This step sets the initial value for $offset_{MS}$, from which the slave's phase shift, $phase_S$, is derived to compensate the phase difference.

2. Phase tracking: The system continuously measures $offset_{MS}$ and updates the slave's phase shifter, $phase_S$, to keep the slave synchronized with the master.

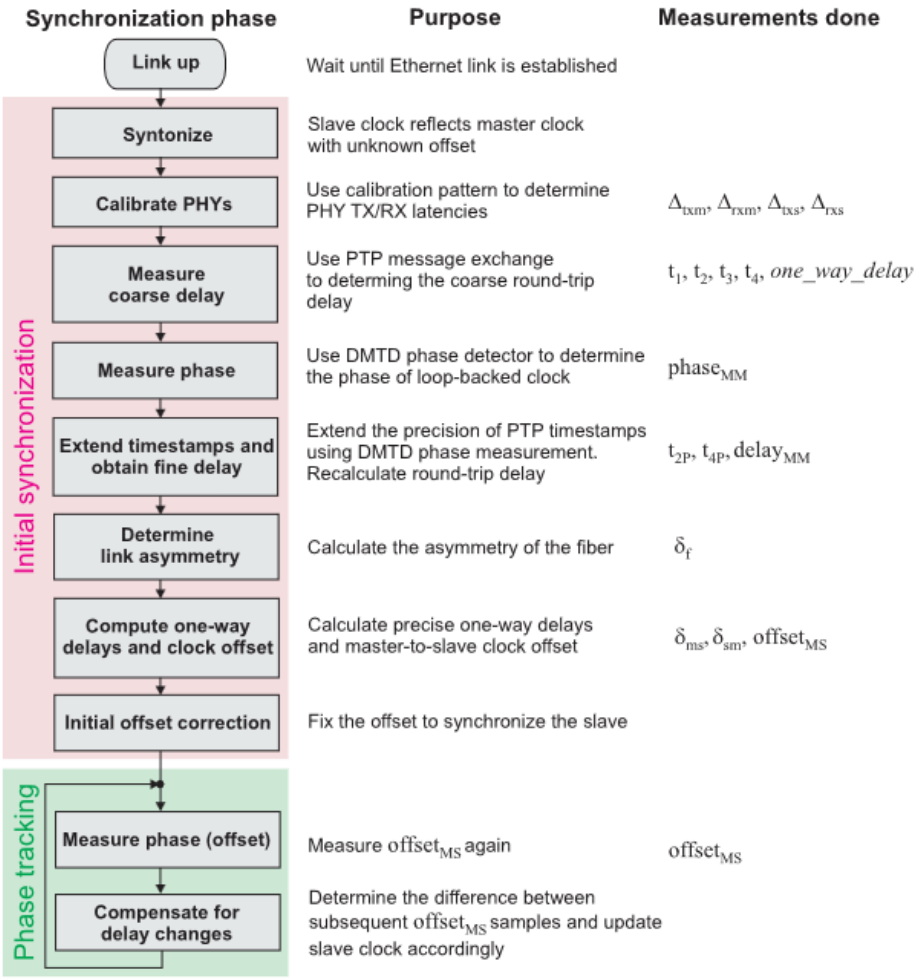

Figure 3.5: White Rabbit synchronization flow. [7]

The execution of the phase tracking process can be explained with Fig. 3.6, Fig. 3.7, and the following steps in broad strokes:

1. The master's reference clock is encoded to its data stream and transmitted to the slave.
2. The slave receives the clock signal that is the same as the reference clock but with the delay, $delay_{MS}$.
3. The recovered clock is phase shifted using a Digital DMTD-enabled PLL according to $phase_S$, resulting in a phase compensated clock at the slave. The slave encodes the clock to the data stream which is transmitted back to the master.
4. The master receives the data stream and recovers the clock signal within the receiver. A Digital DMTD is used to measure the phase difference

($phase_{MM}$) between the recovered clock from the slave and the reference clock. The phase difference is used to calculate a more precise timestamp that is sent back to the slave.

5. The slave utilizes the new timestamp to adjust its phase so that it remains synchronized with the master.

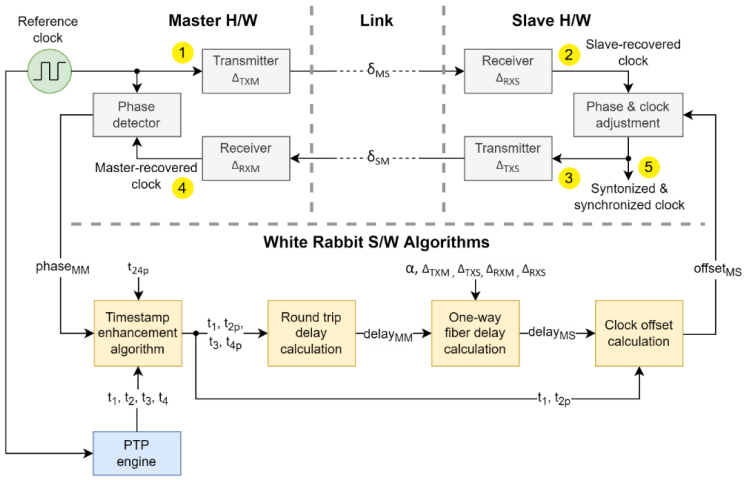

Figure 3.6: Synchronization block diagram for White Rabbit [11].

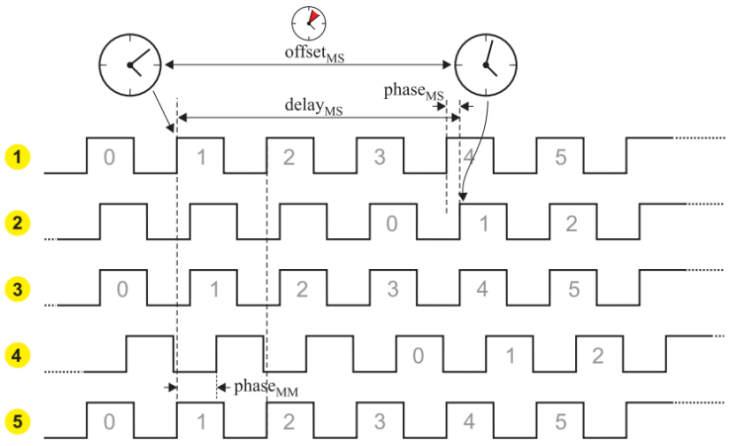

Figure 3.7: Clock diagram when synchronization occurs in White Rabbit [11].

The variable $phase_{MM}$ is a phase measurement and can be modeled as (3.3). In the equation, Δ represents the sum of the fixed delays described in 3.1.3

($\Delta = \Delta_{TXM} + \Delta_{RXM} + \Delta_{TXS} + \Delta_{RXS}$), while δ_{MS} and δ_{SM} represents the variable delays (see Section 3.1.3). In practice, $phase_{MM}$ is measured by the master's Digital DMTD. The variable $phase_S$ denotes the programmable phase shift performed in the slave. It represents the phase shift that must be applied to align the slave's clock with the master's reference clock (see (3.8)). Finally, T_{ref} represents the period of the 125 MHz clock used in Gigabit Ethernet [11].

$$phase_{MM} = (\Delta + \delta_{SM} + \delta_{MS} + phase_S) \mod T_{ref} \tag{3.3}$$

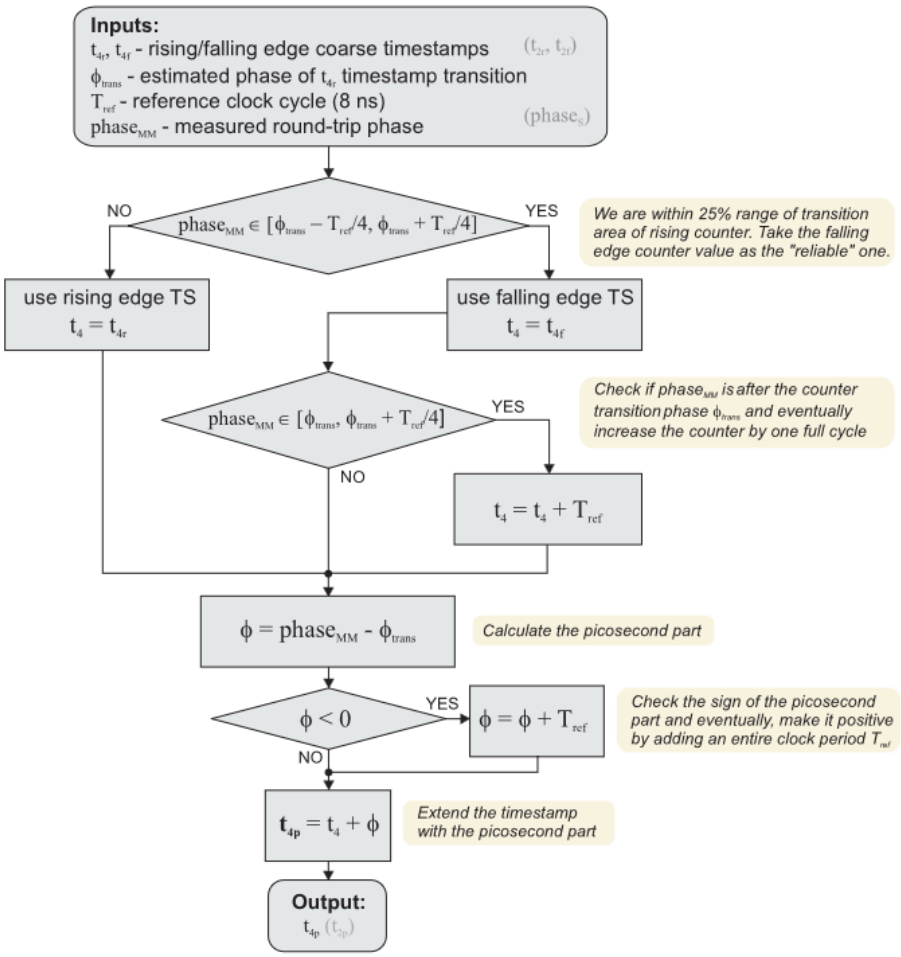

Figure 3.8: The algorithm to calculate the more precise timestamps t_{4p} and t_{2p} [7]

As mentioned above, are $phase_{MM}$, $phase_S$ and PTP packets used in the White Rabbit algorithm to accurately syntonized and synchronize the slave to its the master. More specifically, $phase_{MM}$ is used to create a more accurate t_4 timestamp, called t_{4p}. In a similar fashion is the slave's phase shift, $phase_S$, used

to create a more accurate timestamp called t_{2p} [9]. The timestamps are taken from the timestamping unit which takes two timestamps for t_2 and t_4, one on the rising edge (t_{xr}) and one on the falling edge (t_{xf}). The algorithm in Fig. 3.8 is then used to determine which timestamp to use [7].

The more precise timestamps, along with the other PTP timestamps, t_1 and t_3, are then used to calculate the round trip delay, $delay_{MM}$, as normal PTP (see Section 2.3.1, (2.3)). The roundtrip delay is defined as in (3.4)[11].

$$delay_{MM} = (t_{4p} - t_1) - (t_3 - t_{2p}) \tag{3.4}$$

The roundtrip delay, $delay_{MM}$, can also be described with the master-to-slave fiber delay δ_{MS}, slave-to-master fiber delay δ_{SM}, and the sum of the fixed delays Δ, since the calibrated fiber asymmetry, α (see section 3.1.3), is know. The equation is seen in (3.5)[30]. Utilizing (3.5) and (3.1) can the one way fiber delay, δ_{MS}, be calculated, as seen in (3.6) [9].

$$delay_{MM} = \Delta + \delta_{MS} + \delta_{SM} \tag{3.5}$$

$$\delta_{MS} = \frac{1+\alpha}{2+\alpha}(delay_{MM} - \Delta) \tag{3.6}$$

With δ_{MS} determined, (3.2) from the link delay model can be used to calculate the one way delay, $delay_{MS}$. Finally, the phase offset adjustment can be calculated to feed the adjustment algorithm in the slave's clock servo. The calculation uses the more precise timestamps and the one way delay, $delay_{MS}$, as shown in (3.7) [9].

$$offset_{MS} = t_1 - t_{2p} - delay_{MS} \tag{3.7}$$

The offset, $offset_{MS}$ is used to adjust three components. The first component to be adjusted is the UTC time. The current value of the UTC counter is adjusted with full seconds of $offset_{MS}$. The second component to be adjusted is the PPS counter, which is adjusted by the remaining number of full 8 ns (T_{ref}) cycles. The last component to be adjusted is the phase shifter, $phase_S$, which is adjusted with the remaining, picoseconds part of $offset_{MS}$ [9]. In other words, is $phase_S$ calculated with the formula seen in (3.8)[13].

$$phase_S = offset_{MS} \mod T_{ref} \tag{3.8}$$

As mentioned above, $phase_S$ is a programmable value, and its initial value is zero. Although this is not explicitly stated, it can be found in the White Rabbit software code [31].

3.1.5 Message exchange pattern

The process of syntonization and synchronization (see Section 3.1.2 and 3.1.4) can be summarized to the message exchange pattern seen in Fig. 3.9.

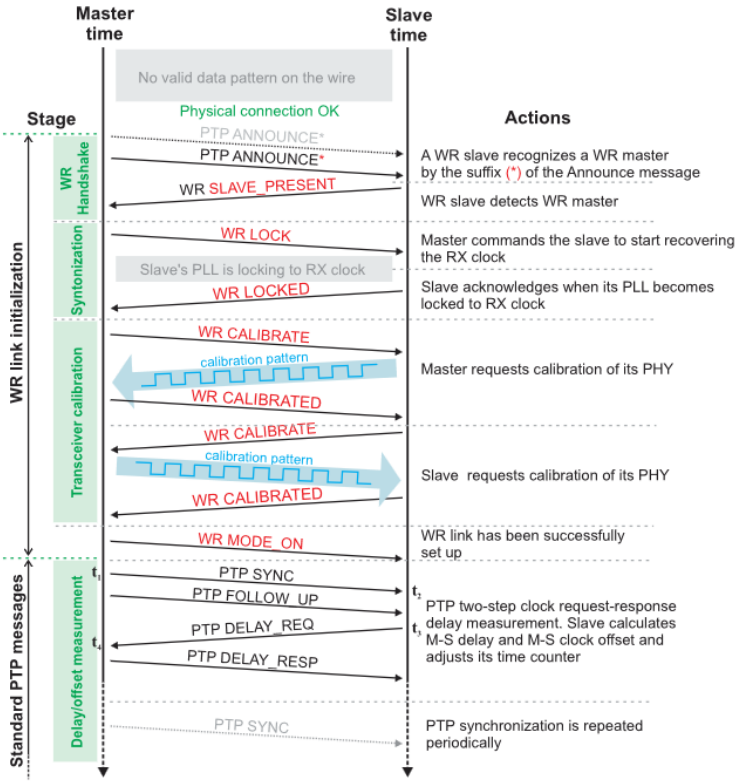

Figure 3.9: The White Rabbit message flow when initializing and upkeep of the White Rabbit PTP (WRPC). [7]

3.2 White Rabbit hardware and software

The implementation of the White Rabbit protocol is referred to as the White Rabbit PTP Core or WRPC. The core acts as a black box for the system in which it is implemented, as illustrated in Fig. 3.10.

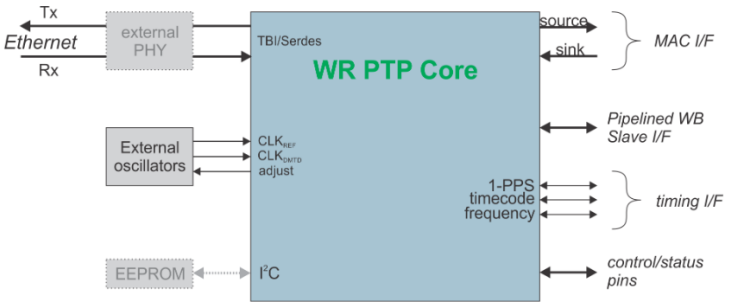

Figure 3.10: Black box of the WRPC [9]

WRPC has a single Gigabit Ethernet interface, which supports 1000Base-BX optical link, that is used to communicate with the other nodes in a system. The WRPC requires only two external components outside of the FPGA, being two digitally tunable oscillators, VCXO (voltage controlled crystal oscillators). One of the oscillators creates the 125 MHz reference frequency, and the other one sets the offset frequency for the Digital DMTD. The rest of the blocks outside of core are not essential, but they can be very useful. The I^2C interface, for instance, can be used to connect to an external EEPROM. The EEPROM can store the device's configuration data and calibration parameters. The core also includes a generic Ten-Bit Interface, which is useful for non-Xilinix device using an external PHY different from the default one used in [9]. The core includes four interfaces for communication with the rest of the system. These may differ depending on the specific implementation. The four interfaces are:

- **Control/Status pins:** These GPIO signals are used for basic monitoring and control of the core. It includes a UART port, which is used to interface a basic monitor of the core, status LEDs, input switches and also a 1-wire interface.
- **Ethernet MAC interface:** Its purpose is to pass non-PTP Ethernet traffic between a user application and the integrated Ethernet MAC. It is also used to provide cycle accurate timestamps at the receiver/transmitter, which is required for White Rabbit PTP operation.
- **Wishbone slave:** This interface provides access to the core's internal control and status registers, and can also be used for debugging purposes and for loading firmware onto the embedded CPU.
- **Timing port:** This port is responsible for the 1-PPS, timecode (UTC and nanoseconds) and the reference frequency. Depending on the operation

mode of the core, the input and output behavior changes. In master mode, it can be used as an input for an external reference, whereas in slave mode, it can be used as an output for the recovered and synchronized signals [9]. In practical setups, the master clock and 1-PPS signal may also be accessible, making it possible to compare the slave's synchronized signals with the master signals.

The White Rabbit core is split into two sections: gateware and software. The gateware section is the hardware implementation, while the software section is the PTP daemon which has been altered for White Rabbit [9].

3.2.1 Gateware

Fig. 3.11 shows the block diagram of the WRPC. The modules seen in the figure uses a Wishbone interface to communicate with each other (more about Wishbone can be read in Appendix B). The WRPC also has two Wishbone crossbars that handles the interconnect between the modules, which is controlled by a soft core processor, for example a Lattice32 [9] or a RISC-V processor [32]. The core also contains two clock domains: REF and SYS. The REF clock domain operates at 62.5 MHz, and the SYS clock domain operates at 125 MHz. Newer versions of White Rabbit have both clock domains operating at 62.5 MHz. The latter domain is used for the 1-PPS generation, Gigabit Ethernet (see section 2.3.3) and timestamping, while the former domain is used for the rest of the system [9].

Fig. 3.12 shows a more modern WR PTP core, while Fig. 3.11 shows the block diagram of the first iteration for the White Rabbit SPEC board. The biggest difference is that the more modern WR PTP core includes the RISC-V processor uRV [33], which is connected directly to the RAM instead of through the crossbar. As a result, the design requires only one crossbar. Depending on the implementation, not only the soft/hard core processor can change, but the actual core can as well. However, in general, they are mostly the same, with some slight variation in the modules that are included.

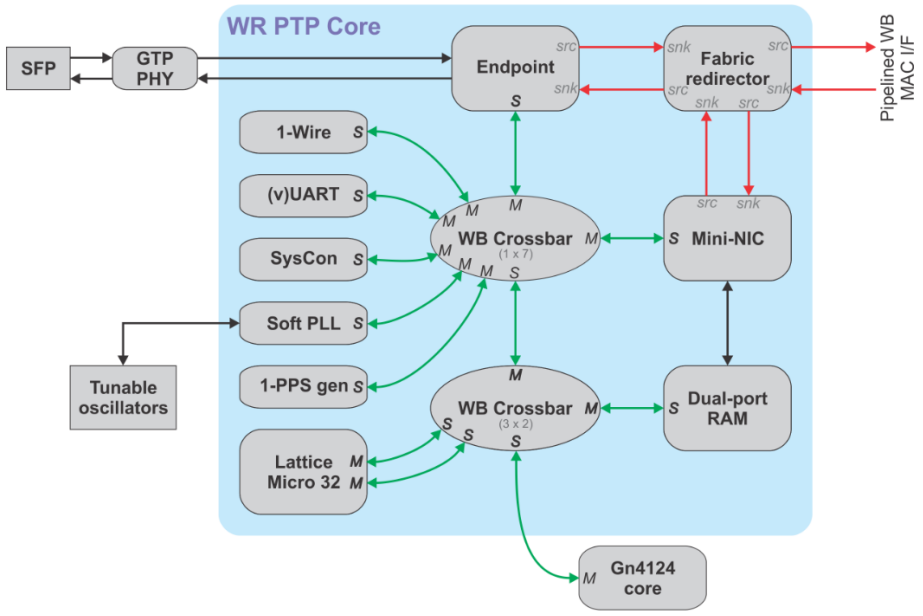

Figure 3.11: Block diagram of the White Rabbit PTP core hardware design, for the first iteration of the SPEC board [9]

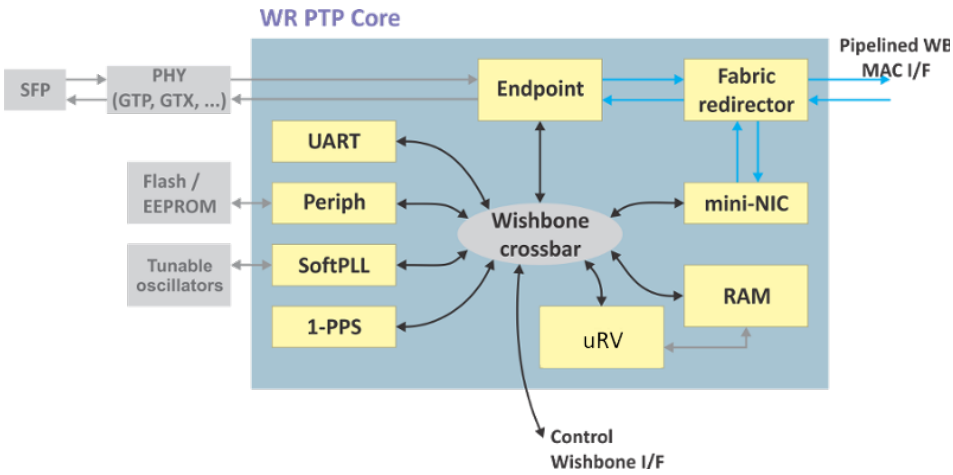

Figure 3.12: Block diagram of a modern WR PTP core hardware design [12]

WR Endpoint

The White Rabbit Endpoint, or WRE, controls the low-level Ethernet communication. A block diagram of it can be seen in Fig. 3.13 [7].

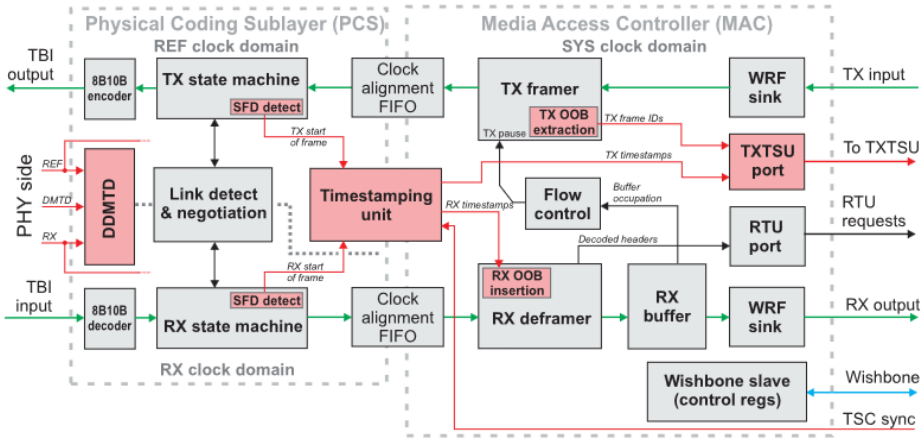

Figure 3.13: Block diagram of White Rabbit Endpoint, from a switch [7]. Observe that the RX output should be a WRF source, and not sink.

There are several parts in this component; however, to keep the scope narrow, only a few of them will be explained. Additionally, certain parts are only important in a White Rabbit Switch, such as the routing table.

This implementation differs from a normal typical Gigabit Ethernet (see section 2.3.3) PCS and MAC, since some additional modules, such as the Digital DMTD (see section 2.3.2) and a timestamping unit are included in it. They are necessary since they generate precise timestamps and phase timestamps respectively. Communication with the PHY can occur either through a ten bit (TBI) interface, commonly used in White Rabbit switches, or through a 8 bit parallel bus, often used with Xilinx FPGA transceivers such as GTP [7] and GTH [12].

Furthermore, the WRE has several different interfaces. They are the following [7]:

- **WR fabric (WRF):** The sink inputs packages to be sent, and the source outputs the received packets.
- **TX timestamping unit (TXTSU)**: Emitting the collected timestamps for outgoing packets.
- **Routing Table Unit (RTU):** Delivers decoded packet headers such as MAC addresses, priorities, and VLAN IDs to the filtering engine (used by the White Rabbit Switch).
- **Wishbone slave:** Enabling access to internal configuration and status registers.

As seen in Fig. 3.13, the endpoint is split into two clock domains, with the PCS block in the REF clock domain and the MAC block in the SYS clock domain. To retime the data between the clock domains, two asynchronous FIFO buffers are

used. The PCS block provides the standard functionality of a typical Gigabit Ethernet interface, such as 8b10b encoder/decoder, low-level framing, autonegotiation and link detection.

The PCS also contains the timestamping unit, which generates timestamps that are then forwarded to the MAC. In addition, the PCS performs Digital DMTD phase measurements, which can be read from the Wishbone registers [7].

White Rabbit Fabric All incoming and outgoing packets are transmitted through the White Rabbit Fabric (WRF) interface. The WRF interface is a Wishbone-based unidirectional interface created for the White Rabbit system. It is a point-to-point connection between a packet source and a packet sink [7]. This can be seen in Fig. 3.11 as the Fabric redirector/Endpoint and in Fig. 3.13 as the WRF sink/source. The WRF interface is used for the NIC and WRE, while the rest of the WRPC uses the normal Wishbone interface. The interface uses a 16-bit data bus and a 2-bits address field. The address bits describe the type of data being sent, as shown Table 3.1 [9].

Type	Code	Description
Regular data	0x0	Packet header, payload
Out Of Band data	0x1	Frame reception timestamp
Status word	0x2	Packet descriptor for a Wishbone bus cycle
User data	0x3	Not used by WRPC, can be used for custom implementation

Table 3.1: Address field types in the WRF interface [9]. *ADDR* in Fig. 3.14.

Fig. 3.14 shows an example of a packet transfer. The sink and the source can both control the flow of traffic. The sink can do this by asserting STALL, and the source by deasserting WE. The sink also contains three standard registers, as follows [7]:

- **FLAGS:** This register is used for passing status information generated by the WRF source. The WRE can use it to mark the result of an address flickering or provide an error code for a corrupted packet.

- **DATA:** This is a FIFO register to which the source sequentially writes the packet data.

- **Out of band register (OOB):** This is another FIFO used to transfer packet-specific user-defined data alongside the corresponding packet, such as timestamps.

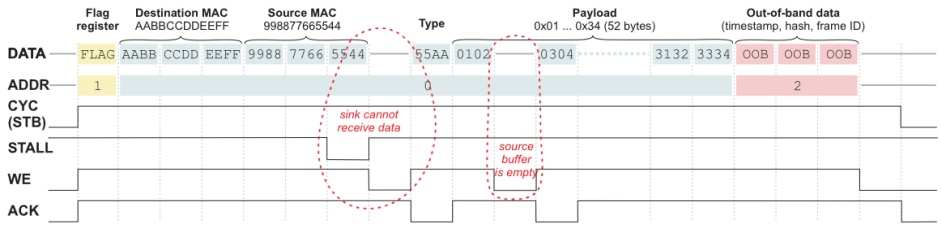

Figure 3.14: An example of packet transfer using the WRF interface [7]

Timestamping unit The timestamping unit is used to produce the PTP timestamps (see Section 2.3.1). These are generated using hardware in the PCS, as illustrated in Fig. 3.15. The timestamp is taken in the PCS when it detects the SFD (start of frame delimiter) of the incoming or outgoing data streams. When the PCS detects the SFD, it sends a trigger pulse that causes the timestamp registers (`DREG_R` and `DREG_F1`) to capture a timestamp snapshot of `CTRL_R` and `CTRL_F`.

`CTRL_R` is a free-running counter that counts from 0 to 124999999 or 62499999 depending on whether the reference clock is 125 MHz or 62.5 MHz. The counter is configured to have a period of one second and operates synchronously with the reference clock in the master and the compensated clock in the slave.

Because the timestamping unit crosses time domains, a synchronization chain of D flip-flops is required. The pulse extender widens the trigger pulse from the PCS, ensuring that no pulses are missed due to metastability in the synchronization chain [7].

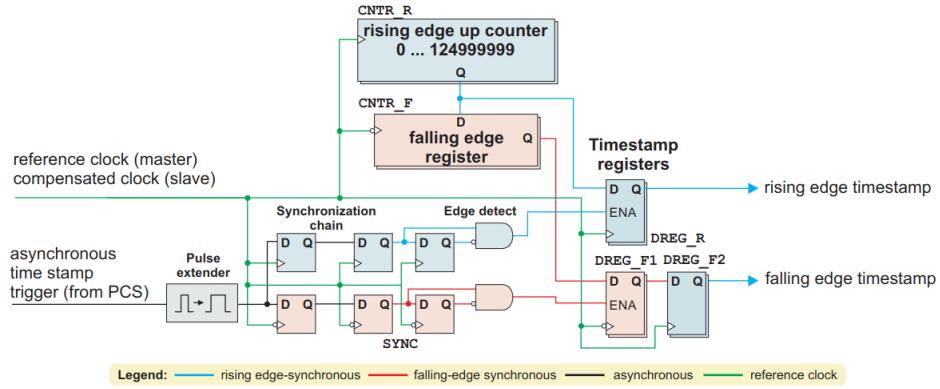

Figure 3.15: Structure inside of the timestamping unit [7]

The blue boxes in Fig. 3.15 are used for taking the timestamp during a rising edge of the clock, while the pink boxes are for the falling edge. The reason for this design is jitter in the clock signals when crossing domains. When the receiver clock and the reference clock are almost in phase, an error of ± 1 LSB can occur. Only the timestamps t_2 and t_4 need enhancement, since they are created in a clock

domain which is asynchronous to the reference (or compensated) clock, see Table 3.2 [7].

Timestamp	Origin	Trigger clock	Timestamping clock
t_1	Master TX	reference	reference
t_2	Slave RX	slave recovered	compensated
t_3	Slave TX	compensated	compensated
t_4	Master RX	master recovered	reference

Table 3.2: Timestamping clock domains [7]. Notice the difference between the trigger clock and timestamping clock is different for t_2 and t_4, and the same for t_1 and t_3.

An example of this problem can be seen in Fig. 3.16. Ideally, the timestamp trigger clock should occur slightly after the reference clock, but due to jitter, the timestamp trigger clock can occur before hand, thereby capturing the previous value of `CNTR_R`. One solution to this is to also capture the timestamp on the falling edge of the reference clock. This is done by latching the counter value from `CNTR_R` into `CNTR_F` on the falling edge, thereby delaying the timestamping by half a period. This ensures that at least one of the timestamps is correct [7]. The timestamp used is decided by the algorithm in Section 3.1.4 (Fig. 3.8).

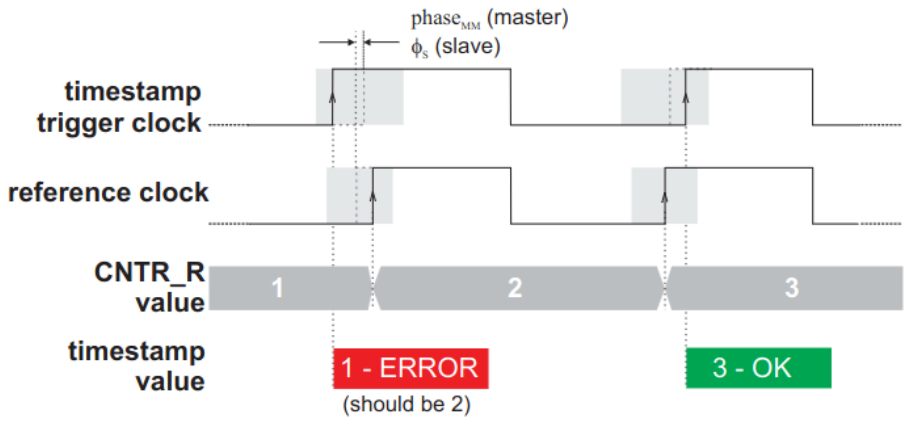

Figure 3.16: The timestamping error caused by jitter. The dashed lines represnts the ideal jitter free clock. [7]

Only the sub-second part of the timestamp is captured in the timestamping unit. The rest of the timestamp, the UTC part, is appended by software. A pseudo code of this algorithm can be seen in Algorithm (1).

Algorithm 1 Producing rest of the timestamp from UTC [7]

```
timestamp ts ← get_hardware_timestamp();
time t ← get_utc_time();
# Check if there has been a change of the UTC time from the
# time attaining the timestamp and the readout of the software.
if t.milliseconds < ts.rising_edge then
    ts.seconds ← t.seconds - 1
else
    ts.seconds ← t.seconds
end if
```

Timestamp delivery The OOB FIFO register is used to deliver all of the timestamps used by the processor for the PTP algorithm. The timestamps are used to calculate the $delay_{MM}$ (3.4) and $offset_{MS}$ (3.7), as shown in Section 3.1.4. Fig. 3.17 shows a block diagram of how this is done [7].

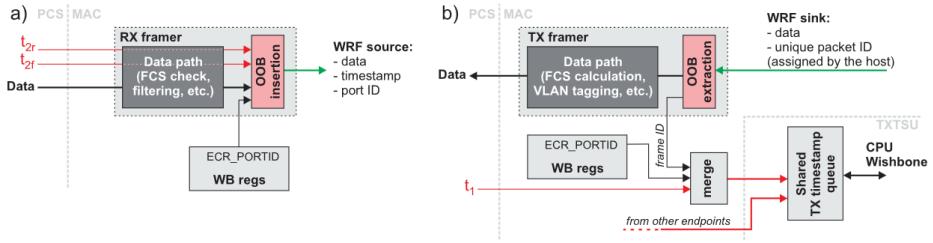

Figure 3.17: Passing the PTP timestamps from RX and TX using the OOB registers and WRF [7].

First, the receiver side (Fig. 3.17 a)) will be explained. As mentioned above, only the receiver needs to take the timestamps on the rising and falling edge of the reference clock. The timestamps produced in the PCS are sent to the RX framer. In the RX framer, the data packet consists of three 16-bit blocks, which are then transferred to the OOB block. The processor can then access them from the OOB block via the NIC [7].

For the transmitter side (Fig. 3.17 b)), the process is slightly more complex. The reason is that the WRF sink cannot deliver the timestamp to the processor due to the WRF interface being unidirectional. Moreover, whenever a PTP packet is transmitted, it is broadcast to all of the ports, meaning that a PTP packet may receive different timestamps from each port.

The solution is to assign a unique packet ID to each transmitted PTP packet. The frame ID is then merged together with the timestamp and placed in the TX timestamping unit queue with the timestamps from the other WREs. The TX timestamping unit collects all the timestamps with their respective ID in a shared FIFO register from all the WREs. They are then deliverers through the Wishbone interface to the processor [7].

1-PPS generator

The WRPC contains a block called the 1-PPS generator. Its responsibility is to generate a 1-PPS output and provide the current UTC time. A block diagram of it can be seen in Fig. 3.18 [7].

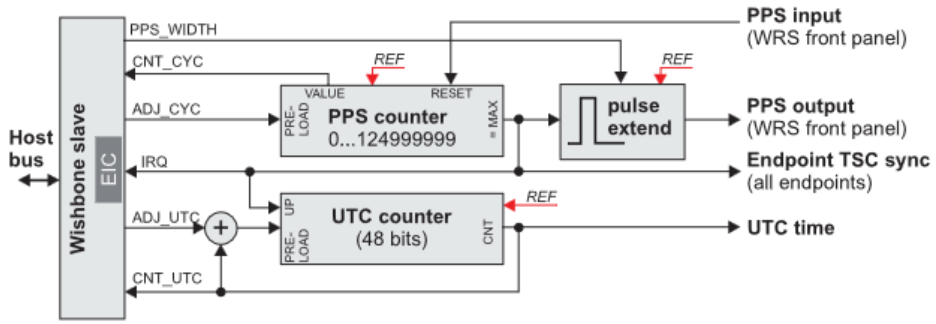

Figure 3.18: Block diagram of the 1-PPS generator [7]

Two counters are used to keep track of time and generate the PPS signal: one for PPS and one for UTC. The PPS counter works as the one mentioned in the timestamping unit, except that its pulse width can be modified. The UTC counter is a 48 bit counter that is incremented every time the PPS counter produces a pulse. The UTC counter therefore contains the full seconds, which are used by the WRPC as its local UTC clock. The current value of the UTC counter is read by software when the current time is required. The UTC time and the sub-nanosecond timestamp from the timestamping unit are merged to create a complete timestamp [9].

White Rabbit Phase-Locked Loop (WR PLL)

The WR PLL is a software based PLL that is executed on an embedded CPU. It utilizes a DDMTD phase detector to produce lower frequency clock signals. These signals are timestamped at the rising edge by a time counter that is incremented by the DDMTD clock signal. The timestamps produced by the counter are called phase tags and are forwarded into the Soft PLL that is a software implementation of a Proportional-Integral, PI, controller. The PIs adjust the Voltage-Controlled Crystal Oscillators, VCXO, through a Digital-to-Analog Converter, DAC. Fig. 3.19 shows the simplified block diagram for the WR PLL. The WR PLL consists of two PLL: a Helper PLL and a Main PLL [34].

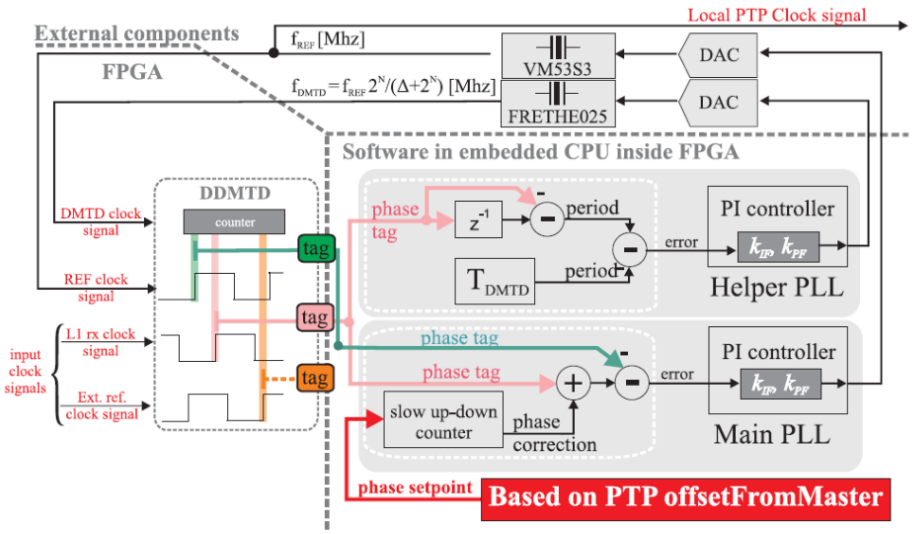

Figure 3.19: Overview of the WR PLL [13]

The Helper PLL controls the DMTD clock signal, which is close but not the same frequency as the reference clock. The DMTD clock is determined by,

$$f_{DMTD} = f_{REF} \frac{2^N}{2^N + \Delta} \tag{3.9}$$

For most implementations Δ is set to 1 and N to 14. Therefore, having a beat frequency of 3.814 KHz and a DMTD frequency of 62.496185 MHz for a reference frequency of 62.5 MHz. The Helper PLL works by comparing the difference between subsequent phase tags to the ideal period of the DMTD clock [34].

The Main PLL controls the REF clock signal which is a phase corrected copy of the recovered clock or the external clock in grandmaster mode. The phase corrected clock is achieved by applying a programmable phase shift ($phase_S$) obtained from the value of $offset_{MS}$ with the WRPC. It works by comparing the phase tags of the chosen input clock with the correct setpoint applied to it, against the phase tags of the REF clock signal. The phase setpoint is not applied directly to avoid ringing in the PI. An up-down counter is used to slowly ramp the correction value [7].

Other modules in WRPTP core

There are several other modules, which can be seen in Fig. 3.11, that have not yet been mentioned in this chapter. These modules are very common in most systems, and several of them even critical. A brief explanation of these modules is given below.

- **1-Wire:** This block is not necessary for the WRPC core and can be removed. It can be used as a digital thermometer, for example [9].

- **UART:** The UART module can be used for debugging purposes. It is used to output information from the WRPTP daemon in the WRPC [9].
- **SysCon:** The SysCon is a combination of several peripherals integrated into one module. It contains the reset registers, GPIOs, hardware feature registers and the timer counter. The timer counter is a simple counter that operates independently of the synchronization process and is not adjusted. Its purpose is to output timer intervals for the software layer [9].
- **Wishbone crossbar:** The Wishbone Crossbar connects several wishbones master with wishbone slaves. The WRPC with LM32 processor (the one in Fig. 3.11) uses two crossbars. The reason is that the soft core processor has two wishbone masters, one for instruction memory and one for data memory. Only the instruction memory requires access to the RAM, therefore, if only one crossbar were used, it would result in a lot of unused logic [9]. The number of crossbars may differ between implementations. Most current implementations use only one crossbar, as seen in Fig. 3.12.
- **Mini-NIC:** The purpose of the Mini-NIC is to be a packet transfer block between the processor, dual-port RAM and the WRE Ethernet MAC. It uses a DMA (Direct Memory Access) mechanism, meaning the processor does not manually push and fetch PTP packets over the WRF interface. For transmission to occur, the processor places a packet in the Dual-port RAM and instructs the NIC to send it to the WRE. For reception, the NIC writes an incoming Ethernet packet into the RAM and signals the processor that a new packet is available.
- **Dual-port RAM:** Both the NIC and the processor need access to the same memory and must be able to use it simultaneously. The solution is a Dual-port RAM or DPRAM, implemented using FPGA Block-RAM (BRAM).

3.2.2 Software

The WRPC has a hardware layer that contains a processor used to control synchronization. The processor specification varies between WRPC versions. Older versions use an LM32 soft-core processor [9]; however, newer versions implement either a RISC-V processor [32] or an ARM processor [35]. The software is a large project and is beyond the scope of the thesis; therefore, it will only be briefly presented.

White Rabbit relies on dedicated software to control the hardware and perform synchronization. The WRPC includes the WRPTP daemon software. A daemon is a program that runs as a background process. The WRPTP daemon is a modified version of the PTPv2 daemon. The core contains three software layers: the PTP daemon, a wrapper, and device drivers. The wrapper contains custom implementations of operating-system mechanisms, certain library functions, and monitoring and debugging functions. The device drivers manage the HDL modules through a set of functions inside the WRPC [9]. The structure of the processor can be seen in Fig. 3.20.

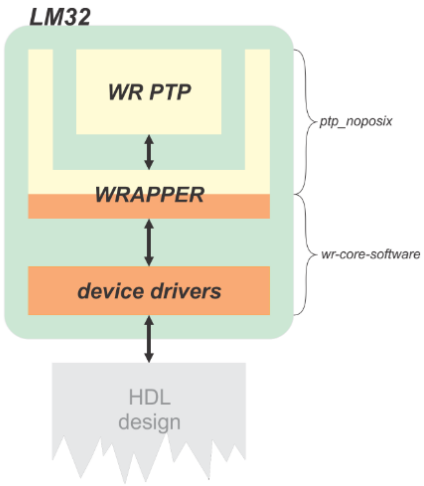

Figure 3.20: The structure of the processor, more specifically the SPEC board that uses a LM32 soft-core processor [9].

.

There are multiple device drivers, one of which is the UART driver. The UART can monitor, set values, and take measurements on a node. The processor can output the status of the WRPC and the synchronization quality by sending single bytes. It can also read incoming bytes by polling the UART receiver. Another device driver is the I^2C bus. The I^2C bus is connected to two GPIO lines, SDA and SCL. The processor can perform read and write operations on the EEPROM, for example, to store device configuration data such as the MAC address and calibration parameters [9].

The firmware is crucial for the WRPC to operate. There are two ways to upload the firmware to the processor. The first method uses a firmware loader program on a separate computer that loads the firmware using PCIe [9]. The second method, which is more commonly used when working with an FPGA on an evaluation board, is to place the firmware in the BRAM before initializing the FPGA. A firmware loader requires additional hardware, which most FPGA evaluation boards do not have [9].

3.3 Light Rabbit

Light Rabbit is a fully implemented White Rabbit node on an AMD evaluation board. The main purpose of Light Rabbit is to eliminate all extra hardware needed to implement White Rabbit. Light Rabbit is currently supported for ZCU102, ZCU106 and ZC706. ZCU102 and ZCU106 are for now the only ones pushed to the master branch. ZCU102 and ZCU106 are both based on Xilinx application note, XAPP1276 [15], of using fractional PLLs, described in 2.3.5, instead of external VCXOs. The transceivers for ZCU102 and ZCU106 are GTHE4. The LC-based fractional PLL are provided inside the transceivers of the UltraScale evaluation boards, and are called Quad PLL (QPLL). Each Quad on the board have access to two QPLL located in GTHE3/4_COMMON primitives. The GTHE3/4_COMMON primitives needs only to be instantiated when a QPLL is used. Either QPLL can be shared by the serial transceivers channels within the same Quad. A Quad is a cluster of four GTHE3/4_CHANNEL primitives together with one GTHE3/4_COMMON, shown in Fig. 3.21 [14].

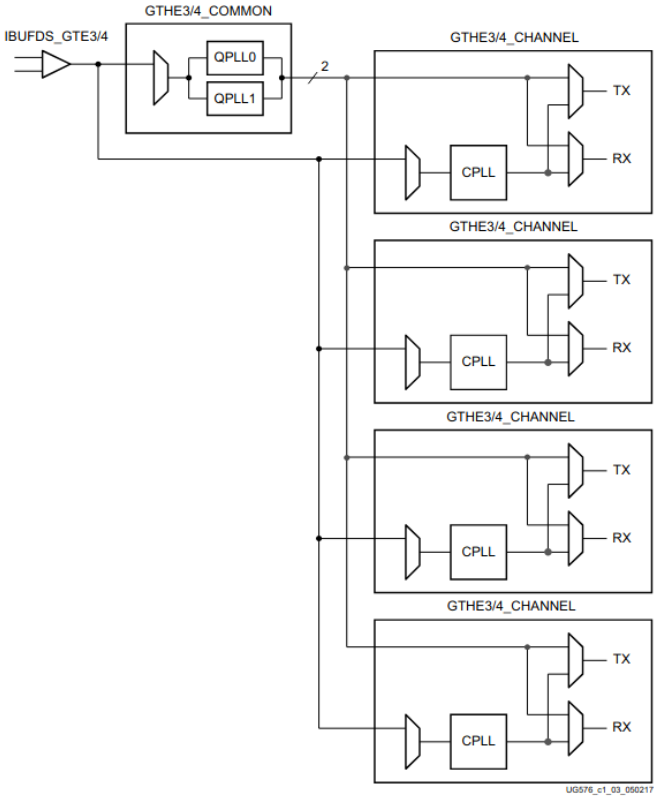

Figure 3.21: Four channel configuration [14]

The Quad have two dedicated external reference clock pins pair, MGTREFCLK, and a dedicated reference clock routing. The Quad can source its reference

clock from up to two Quads below and above. The GTHE3/4_CHANNEL consists of a channel PLL, a transceiver and a receiver illustrated in Fig. 3.22. The clock for the transceiver and receiver in the channel are fed either from the QPLL0/1 or the CPLL to the TX and RX clock dividers as shown in Fig. 3.23 [14].

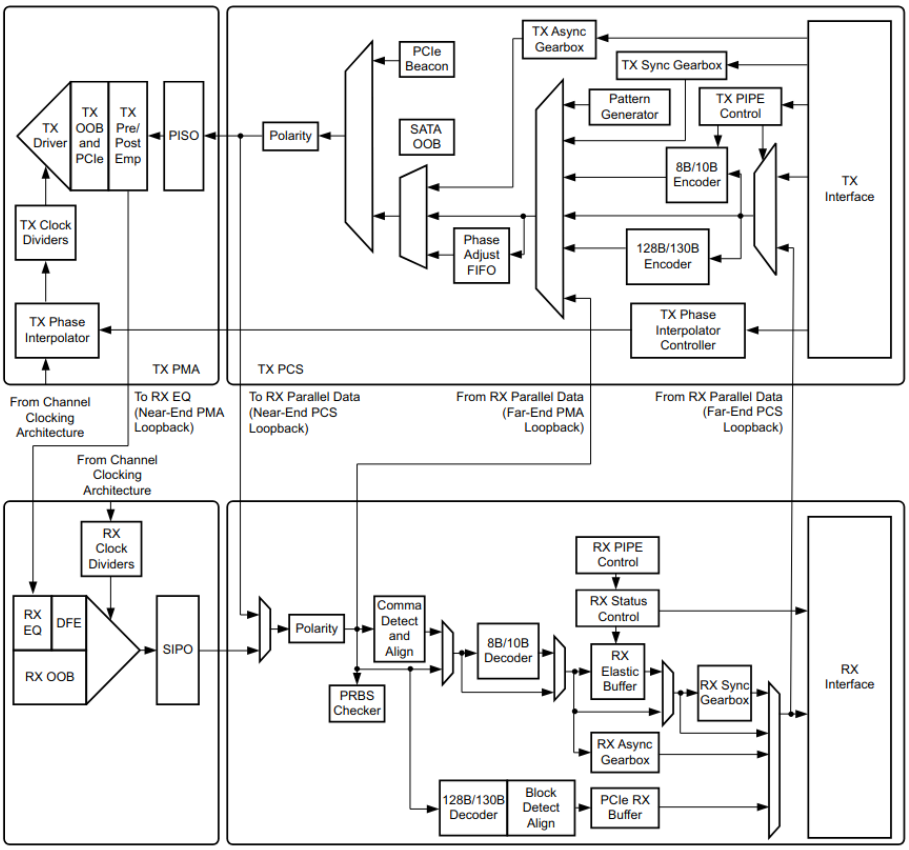

Figure 3.22: GTHE3/4_CHANNEL primitive topology [14]

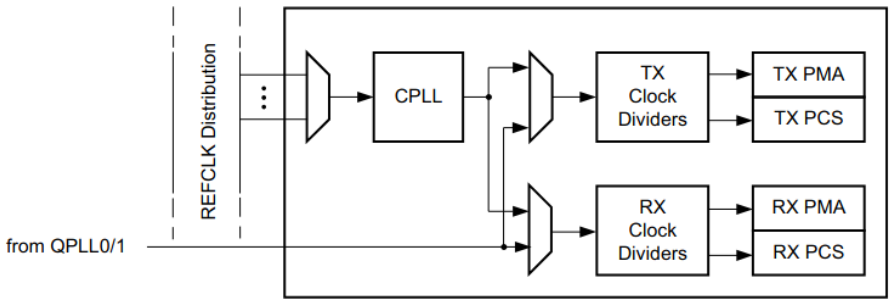

Figure 3.23: Internal channel clocking architecture [14]

The conceptional view of the QPLL architecture is illustrated in Fig. 3.24. The architecture is similar to the one described in 2.3.5 except that a lock indicator has been introduced to determine if a frequency lock has occurred. The QPLL also has an interface that controls the sigma delta modulator to provide the fractional feedback. The control input, SDMDATA, is typically set static but can be dynamically controlled which is utilized by Light Rabbit. The fractional part can be calculated by $\text{SDMDATA}/2^{\text{SDMWIDTH}}$, where the maximum allowed value for SDMDATA is $2^{24} - 1$. SDMWIDTH can be set to 16, 20 and 24. The default value is 24.

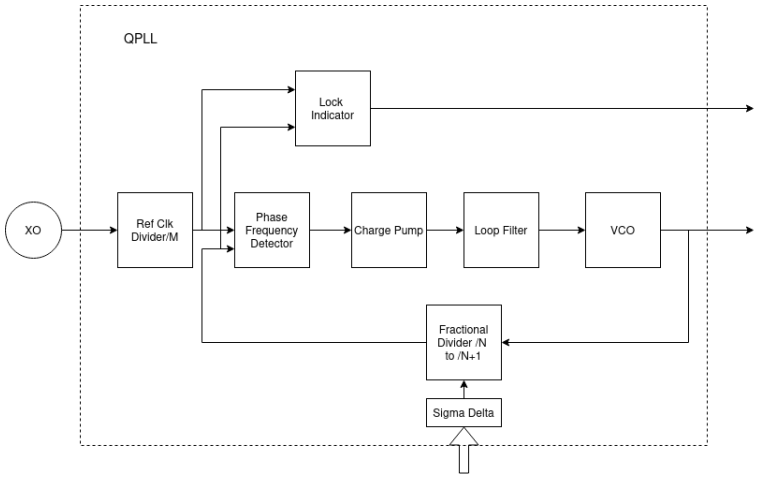

Figure 3.24: QPLL block diagram

An example when Dynamic Frac-N is used is illustrated in Fig. 3.25. In the example, the fabric-based PLL provides the controls around the GTY Frac-N to lock to the external reference clock. The SDM functionality is enabled, allowing the circuit to modulate the fractional divider between N and N+1 with high precision. The UltraScale+ is restricted to a maximum tuning range of ±200 ppm. The Dynamic Fractional Divider is controlled by a user operated strobe, SDMTOGGLE,

that controls the SDMDATA transfers from the fabric to the transceiver.

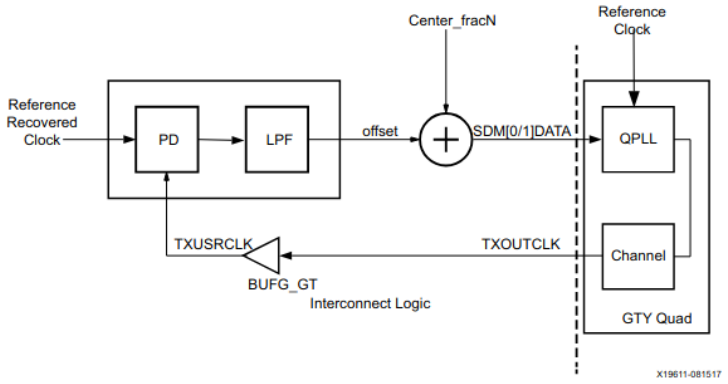

Figure 3.25: Dynamic frac-N example [15]

A architectural overview of Light Rabbit is provided in Fig. 3.26. The QPLLs are used in a similar fashion as Fig. 3.25. The reference clock, MGTREFCLK, to the QPLLs is provided by two Si570 slightly tuned below 125 MHz to a frequency of 124.975605 MHz. The integral part of the fraction, N, is set to 80 and the TX clock divider, TXPROGDIV, is set to 2*80. As illustrated in Fig. 3.25, can the fractional part only be increased, and thereby lowering the VCO feedback. This in turns increases the VCO frequency meaning that in order to have tuneable range around 125 MHz, MGTREFCLK have to be down-tuned. For the stated reference frequency and the integral part, N, can the VCO frequency be calculated according to (3.10).

$$f_{VCO} = f_{REF} \cdot (N + \frac{SDMDATA}{2^{24}}) \tag{3.10}$$

The output from the soft PLL is 16 bits, therefore in order to be able to almost entirely use the ±200 ppm tuning range the signal has to be padded by 3 bits. For a signal bigger than 19 bits the tuning range becomes larger than the maximum allowed range. In this design the 3 bits were decided to be 011_2 and consequently the minimum VCO frequency becomes 9.99805 GHz and the maximum 10,00195 GHz, thereby meaning that we have a tuning range of ±195 ppm. This results in a ±195 ppm range almost exactly around 125 MHz.

The architectural overview in Fig. 3.26 is a simplified representation. In reality, the Quad connected to the helper uses two channels as well as two QPLLs. However, only the channel and QPLL that produce the output used in the design are illustrated in Fig. 3.26.

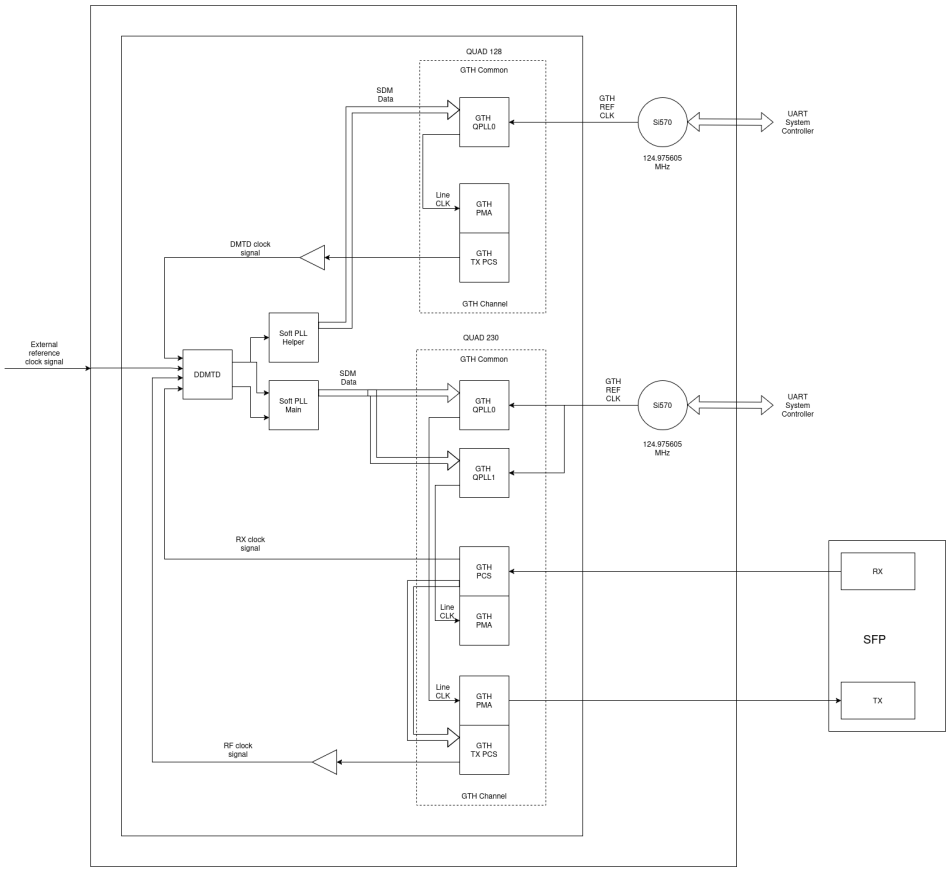

Figure 3.26: Architecture of Light Rabbit for ZCU102

The company that implemented Light Rabbit is called MLE [16]. The synchronization performance of their implementation on the ZCU102 platform is shown in Fig. 3.27.

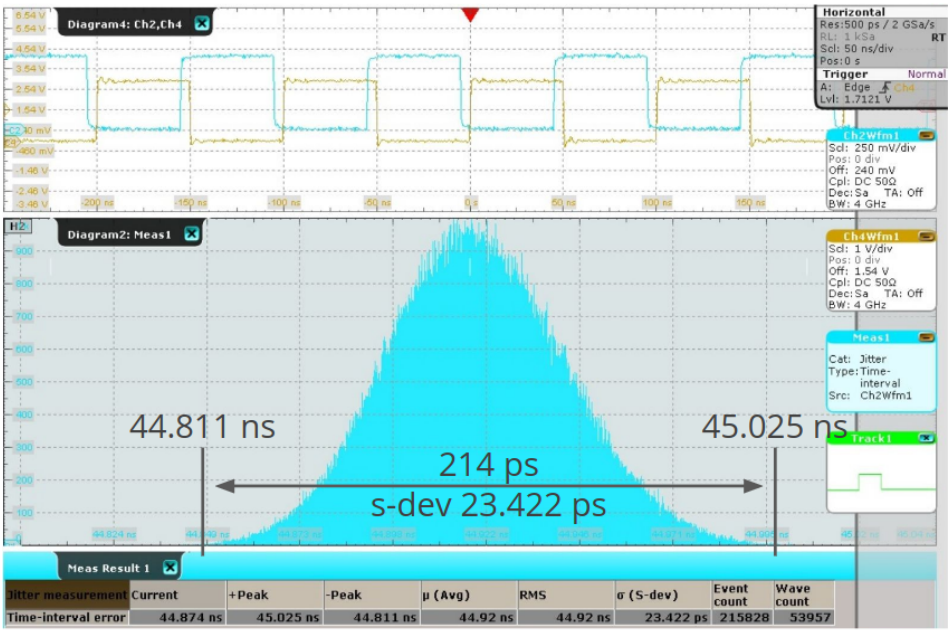

Figure 3.27: MLE measurement of their Light Rabbit implementation on the RFSoC ZCU102, no calibration [16].

3.4 Implementation on RFSoC ZCU216

The architectural overview of the implemented design for the ZCU216 is illustrated in Fig. 3.28. The design is based on the ZCU102 version of Light Rabbit. The ZCU216 uses GTYE4 transceivers instead of GTHE4. Therefore, the instantiation of the transceiver IPs had to be changed and modified for the ZCU216. As illustrated in Fig. 3.28 is ZCU216 restricted to one Si570 located as a reference clock for Quad 131. Therefore, the usable SFP ports are restricted to SFP2/3. SFP0 and SFP1 are located at Quad 128 more than two Quads below Quad 131 meaning that neither of them can use Si570 as a frequency reference. The extra channel and QPLL discussed in Light Rabbit were also removed. The only output used from the Quad is the one producing the DMTD frequency, and therefore the extra channel and QPLL is unnecessary.

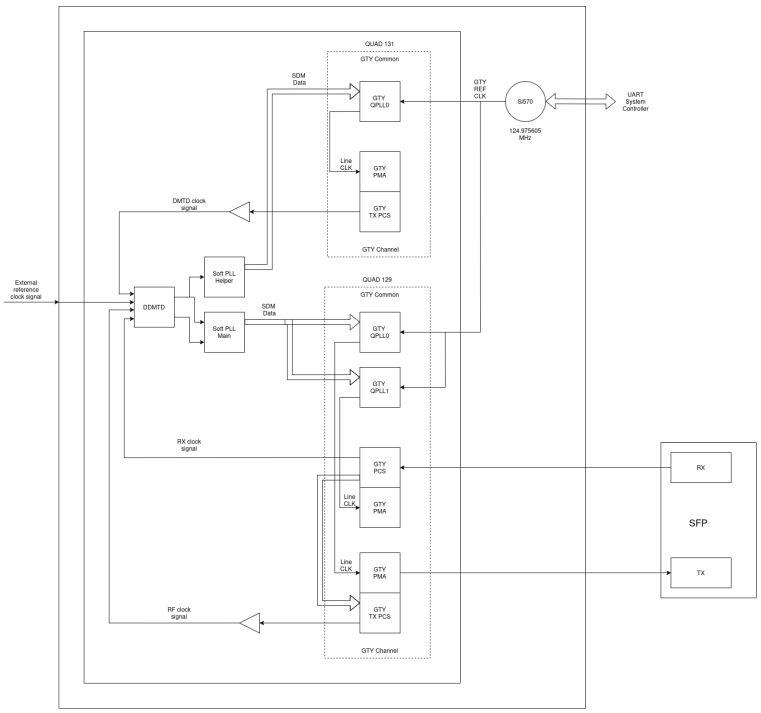

Figure 3.28: Architecture of Light Rabbit for ZCU216

The fractional part described in section 3.3 were also changed. In Light Rabbit the output from the soft PLLs were padded with a 3 bit signal of 011_2. By padding the signal with 3 bits the fractional resolution is reduced to 21 bits. To get a finer resolution the soft PLL output could have not been padded, and it center frequencies could have been set to the DMTD frequency and the White Rabbit reference frequency. However, this significantly reduces the tuning range, and the ZCU216 board also only have access to one Si570 making this impossible. Therefore, an adjustment to the three LSB were made to more accurately represent the

frequencies with a resolution of 24 bits. Using (3.10) can the specific SDMDATA for the DMTD frequency and White Rabbit reference frequency be calculated. The DMTD frequency are set as described in section 3.2.1 to 62.496185MHz, and the White Rabbit reference frequency to 62.5 MHz. The calculated SDMDATA value for the DMTD frequency is 180060 meaning that the three LSB are 100_2. Therefore, the lower three bits were changed to 100_2. For the main PLL the value of SDMDATA was calculated to 261990, which means a value of 110_2 for the lower three bits. The frequency could also have been changed to 124.975591 MHZ to have a more accurate center frequency of 125 MHz. However, because this makes a very small difference the reference frequency were kept at 124.975605 MHz.

The Light Rabbit implementation for the ZCU102 board utilized the HDMI EEPROM to store calibrations and other data. However, this was not possible on the ZCU216, as it does not include an HDMI EEPROM. Instead, the main EEPROM on the ZCU216 was used. The EEPROM is connected to the I2C0 bus through the switch at address 0x74, port 0, with the EEPROM located at address 0x54.

Before using the EEPROM, its content were examined by dumping it using PYNQ. The dump is shown in Fig. 3.29.

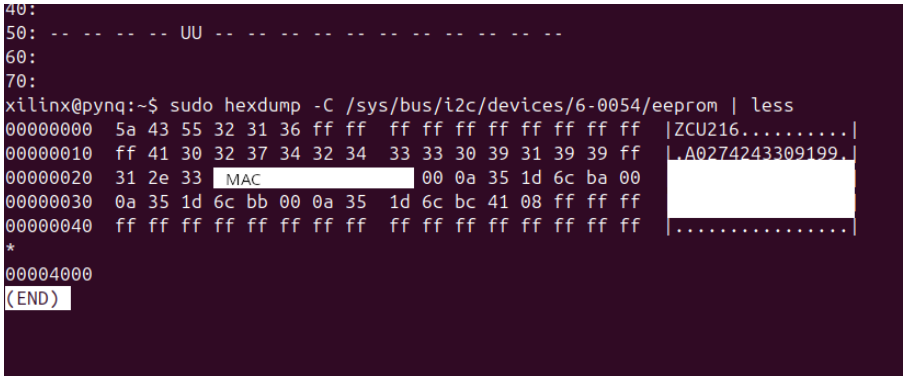

Figure 3.29: Dump of the ZCU216 main EEPROM before making adjustments. The MAC address is censored.

Fig. 3.29 shows that the EEPROM mainly stores basic information about the board, such as the board type and its MAC address. Preserving this data is crucial, not only because it may be required by other FPGA projects, but also because the stored MAC-address is used in our implementation. Fortunately, the size of the EEPROM is relatively large, and as seen in Fig. 3.29, the memory from address 40_{16} to 4000_{16} is unused. This made it possible to store our data in the same EEPROM.

To ensure that the basic information was not overwritten, the data was written starting at an offset of 100_{16}. The original board information occupies the address range from 0_{16} to 40_{16}. A slight modification was also required when reading the MAC address. On the ZCU102, the MAC address is stored at offset 20_{16}, whereas on the ZCU216 it is stored at offset 23_{16}. All of these changes were implemented in the firmware, which was then recompiled and loaded into the BRAM.

Chapter 4
Measurements & Results

4.1 Measurements setup

All measurements were performed with a 1 km fiber optic cable. The fiber was a single-mode 9/125 µm G.652.D LC/UPC fiber connected between the master and slave using two specific SFPs. The SFP used by the master was the BO15C4931620D-WR, and the one used by the slave was the BO15C3149620D-WR. All measurements were performed at room temperature. The frequency counter used was an `Agilent 53230A`, and the signal source analyzer was an `R&S FSUP 8 Signal Source Analyzer`. Whenever an external clock was connected to the master, it was provided by the signal generator `RIGOL DG1032Z` using the settings shown in Table 4.1. The cables used to connect to and from the signal generator, frequency counter, and signal source analyzer were 1 m coaxial cables. An external 10 MHz GPS clock signal was connected to all instruments during all measurements. The gate time on the frequency counter was 100 ms for the 62.5 MHz measurement and 10 s for the PPS measurement.

Signal	**10 MHz**	**1-PPS**
Waveform	Sinus	Pulse
Frequency	10.000 MHz	1.000 Hz
Amplitude	4.000 Vpp	4.000 Vpp
Offset	5.000 V DC	5.000 V DC
Phase	0.000°	0.000°
Duty Cycle	-	5.000%

Table 4.1: `RIGOL DG1032Z` signal generator settings.

4.2 Time-offset

4.2.1 Setup

Time-offset measurements were performed by measuring the skew between the PPS signals and the output reference clocks, 62.5 Mhz, of the master and slave using the frequency counter. Two measurements were performed for both the PPS

and 62.5 MHz signals: one where an external clock was connected to the PPS and 10 MHz inputs of the master, corresponding to grandmaster mode, and one where the master's own internal clock was used, corresponding to master mode. The full setup can be seen in Fig. 4.1. The PPS signals were measured overnight, while the samples for the 62.5 MHz signals were collected for roughly one hour.

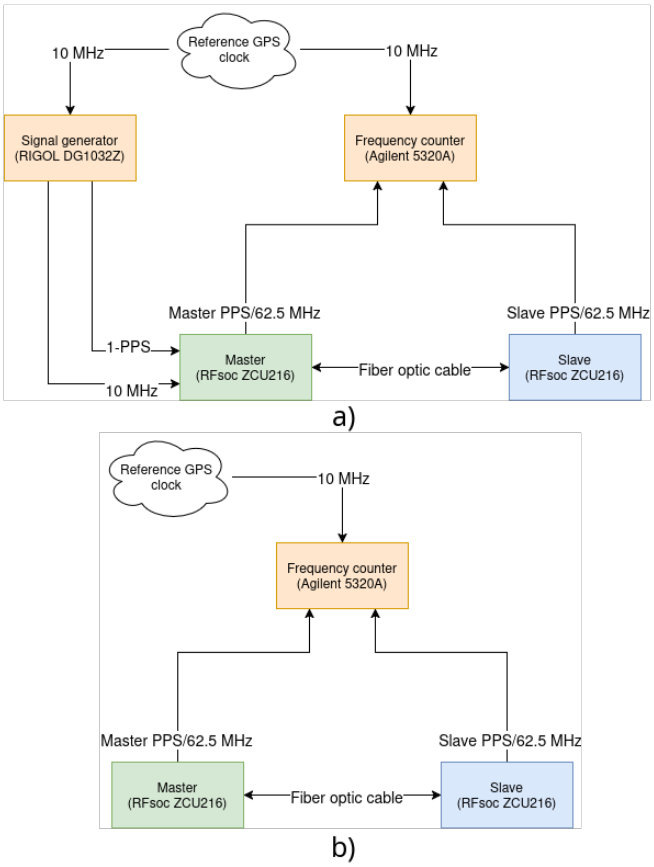

Figure 4.1: Time-offset measurement setup when connected to the frequency counter. a) The master clock is in grandmaster mode b) The master clock is in master mode.

4.2.2 Results

The measured PPS and 62.5 MHz skew between master and slave can be seen in Fig. 4.2/4.4 and Fig. 4.3/4.5. The former figures show the results when White Rabbit is running in master mode, while the latter figures show the results when it is running in grandmaster mode. Table 4.2 contains the values from the plots.

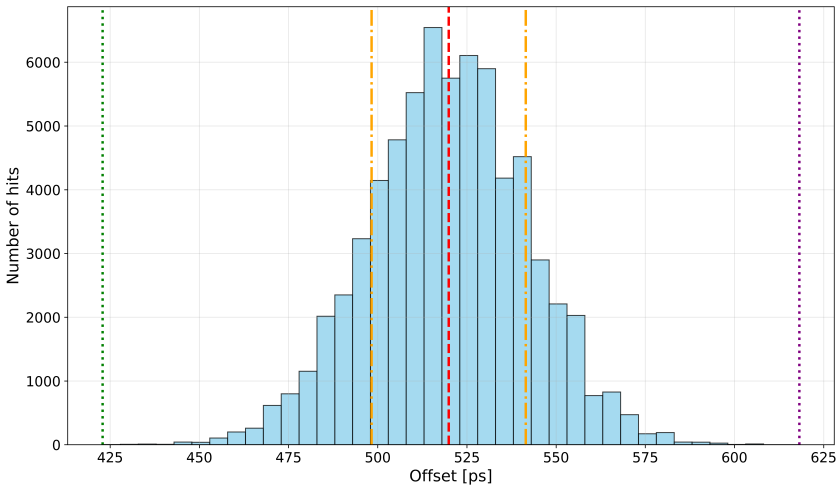

Figure 4.2: Measurement of the master-slave PPS skew in master mode

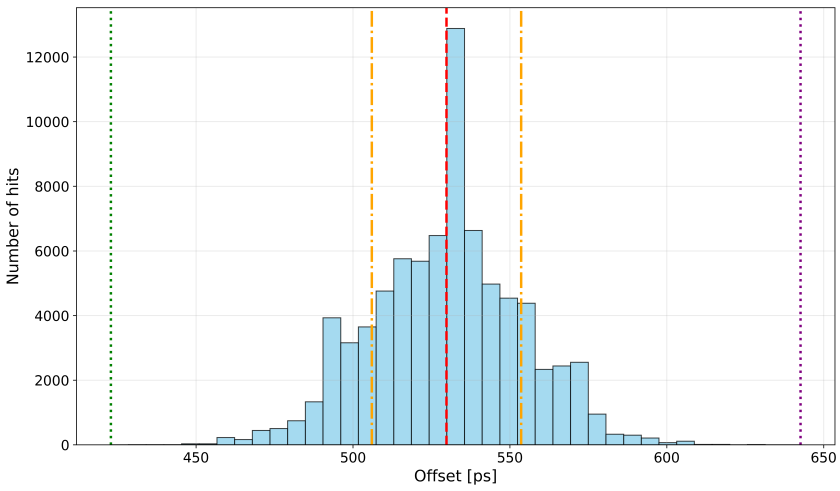

Figure 4.3: Measurement of the master-slave PPS skew in grand-master mode

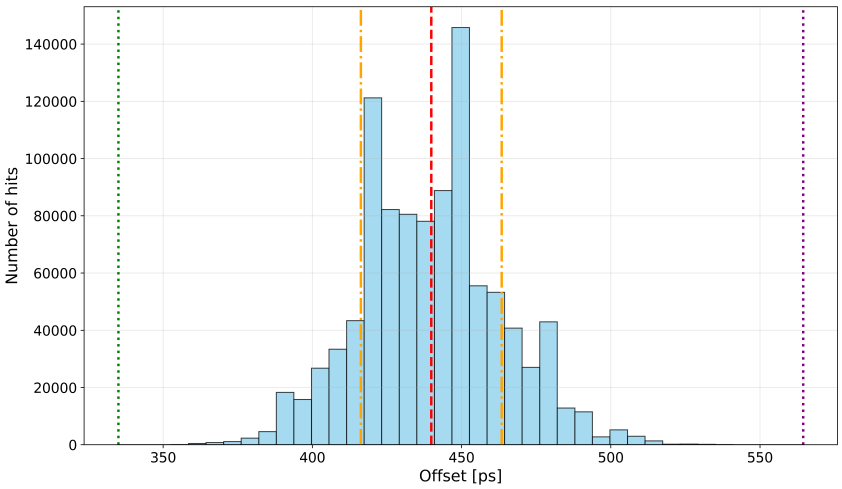

Figure 4.4: Measurement of the master-slave 62.5 MHz skew in master mode

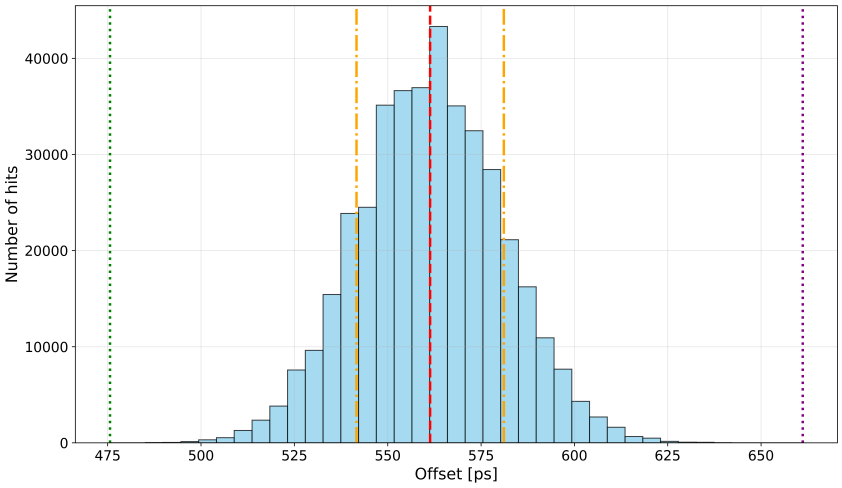

Figure 4.5: Measurement of the master-slave 62.5 MHz skew in grandmaster mode

Table 4.2: Summary of time offset measurement statistics.

Measurement	Samples	Mean [ps]	Std. dev. [ps]	Min [ps]	Max [ps]	Max–Min [ps]	Fig.
PPS	67960	519.880	21.604	422.867	618.179	195.313	4.2
PPS external clock	79623	529.792	23.777	422.867	642.593	219.727	4.3
62.5 MHz	403373	561.355	19.730	475.607	661.154	185.547	4.4
62.5 MHz external clock	1000008	439.852	23.605	334.976	564.468	229.492	4.5
Average	387741	512.720	22.179	414.079	621.599	207.520	

4.3 Frequency stability

4.3.1 Setup

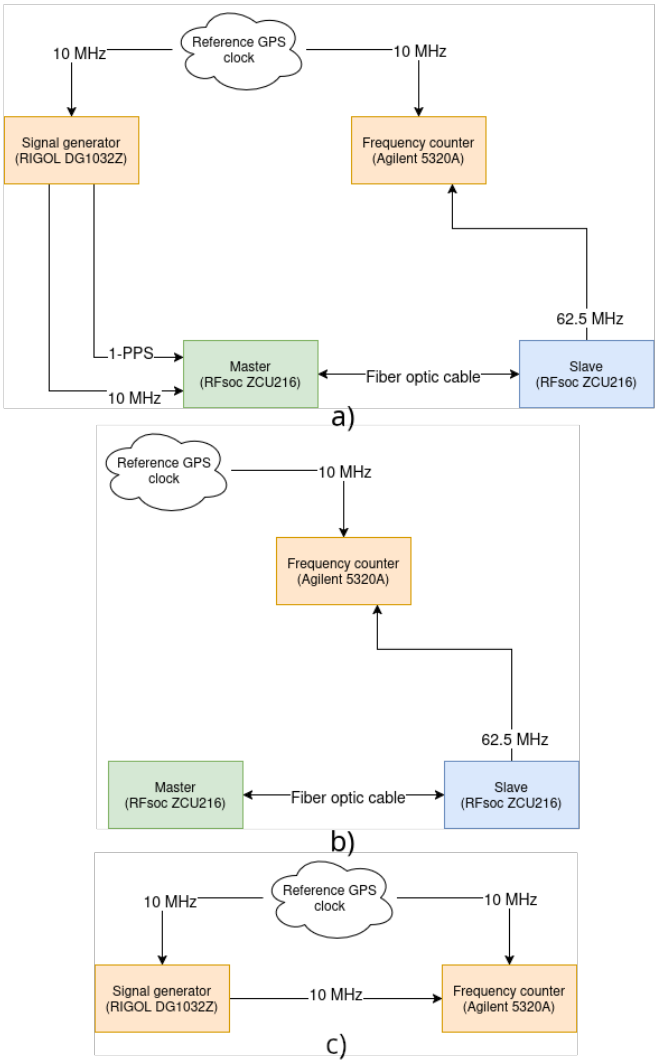

Figure 4.6: Frequency stability measurement setup of a) the slave with the master using its internal clock b) the slave with the master using an external clock c) the signal generator.

Frequency stability was measured by connecting the 62.5 MHz output of the slave to the frequency counter. Two measurements were performed: one where the master was connected to an external clock, corresponding to grandmaster mode, and one where the master used its internal clock, corresponding to master mode.

For comparison, the frequency stability of the signal generator was also measured. This was done by connecting the signal generator to the frequency counter. The instrument measured for approximately 20 minutes, producing 10 000 samples for each measurement. Illustrations of the measurement setup can be seen in Fig. 4.6. Allan deviation measurements were performed with the same setup as described above for the slave, where the master was in master mode and grandmaster mode, and for the signal generator. Additionally, a reference from a Cesium CS4000 clock [36] was added to the plot.

4.3.2 Result

Fig. 4.7 shows the slave 62.5 MHz signal with the master using an external clock, while Fig. 4.8 shows the slave 62.5 MHz signal with the master using its internal clock. Fig. 4.9 shows the 10 MHz signal from the signal generator. Table 4.3 contains the values from the plots. There are two plots of the Allan deviations. The first, Fig. 4.10, contains all measured Allan deviations, while the second, Fig. 4.11, leaves out the slave with the master using its internal clock.

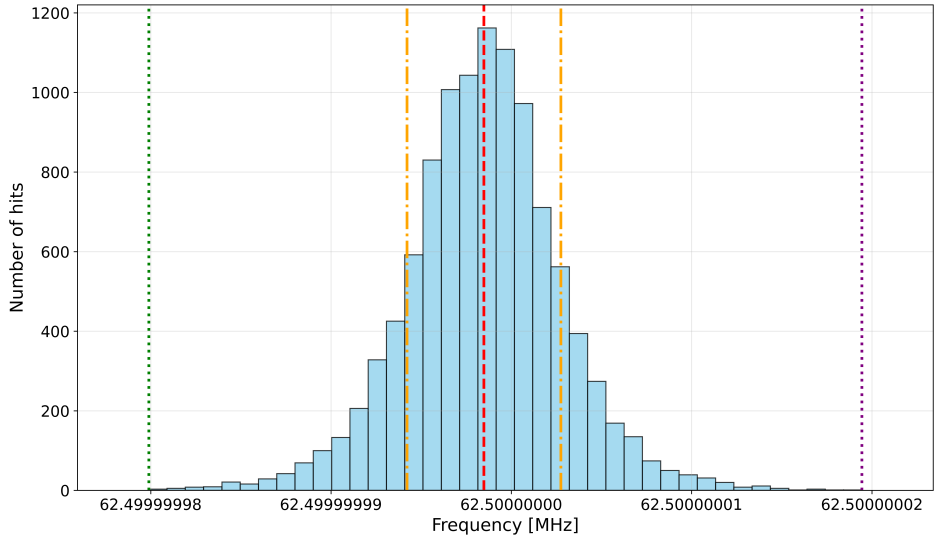

Figure 4.7: Slave 62.5 MHz signal with the master using an external clock

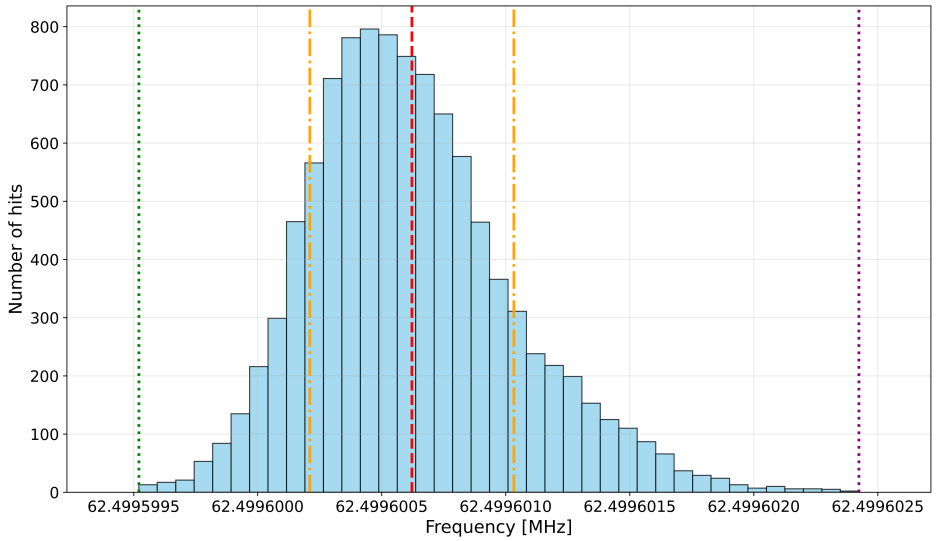

Figure 4.8: Slave 62.5 MHz signal with the master using its internal clock

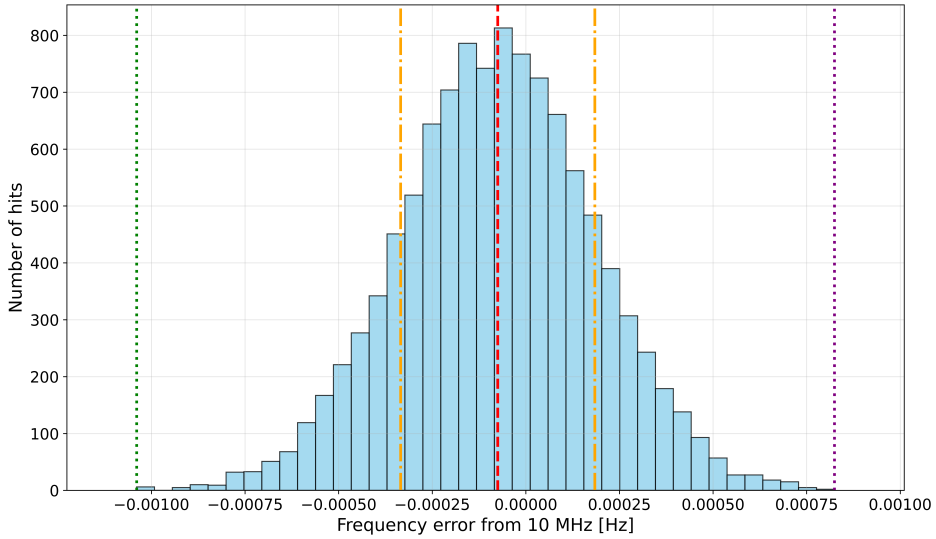

Figure 4.9: 10 MHz signal from the signal generator connected to an external GPS clock signal

Table 4.3: Frequency stability summary for the slave 62.5 MHz with the master using an external clock, the slave 62.5 MHz signal with the master using its internal clock, and the 10 MHz signal from the signal generator, from Fig. 4.7, Fig. 4.8, and Fig. 4.9.

Metric	Internal clock	External clock	Signal gen.
Samples	10113	10597	10699
Theor. frequency [MHz]	62.500000000	62.500000000	10.000000000
Mean frequency [MHz]	62.499600621	62.499999998477	9.999999999925
Mean error [Hz]	-399.379	-0.001523	-0.000075
Standard deviation [Hz]	0.411	0.004265	0.000259
Min. frequency [MHz]	62.499599521	62.499999979893	9.999999998960
Max. frequency [MHz]	62.499602423	62.500000019442	10.000000000824
Mean − Min. [Hz]	1.100	0.018584	0.000964
Max. − Mean [Hz]	1.802	0.020965	0.000899
Max. − Min. [Hz]	2.903	0.039550	0.001864
Frequency stability [ppb]	±28.8321842395	±0.335440000008	±0.0964000000007

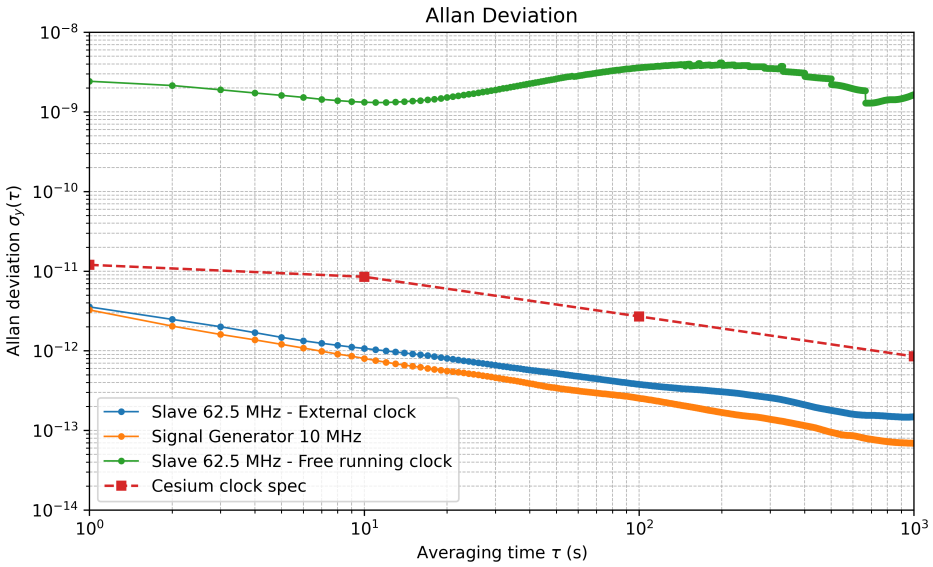

Figure 4.10: Allan deviation measurements of the slave 62.5 MHz output signal with the master using its internal clock and using an external clock, and the signal generator 10 MHz signal.

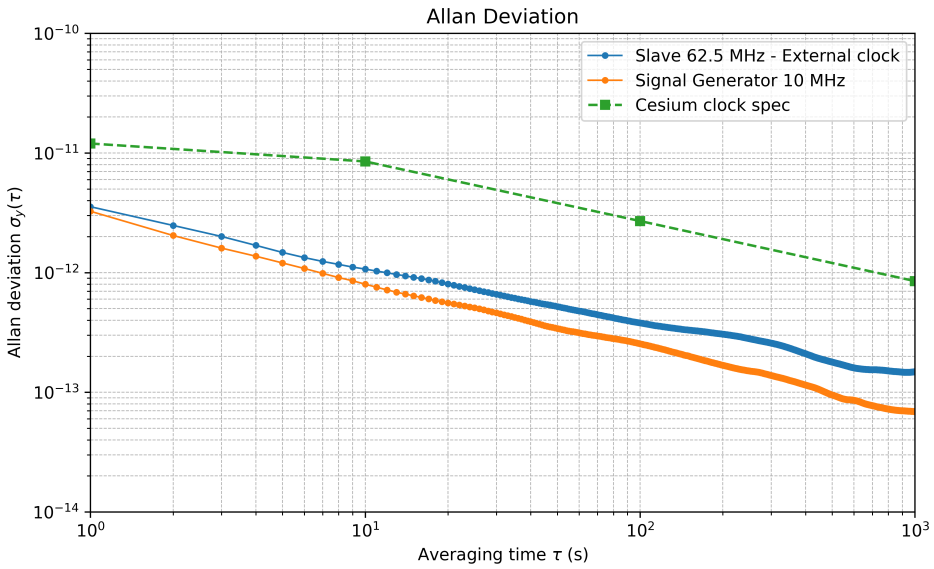

Figure 4.11: Allan deviation measurements of the slave 62.5 MHz output signal with the master using an external clock and the signal generator 10 MHz signal.

4.4 Phase noise

4.4.1 Setup

The phase noise measurements were performed on the slave using both a master with an external clock, corresponding to grandmaster mode, and a master with its internal clock, corresponding to master mode. The signal generator and the GPS clock signal were also measured. The Radio Bandwidth was set to 0.1% for all measurements. The offset-frequency range was set from 1 Hz to 3 MHz, except for one additional measurement of the slave, where the span was set from 30 Hz to 30 MHz. The setup for all measurements, except for the GPS signal, is illustrated in Fig. 4.12.

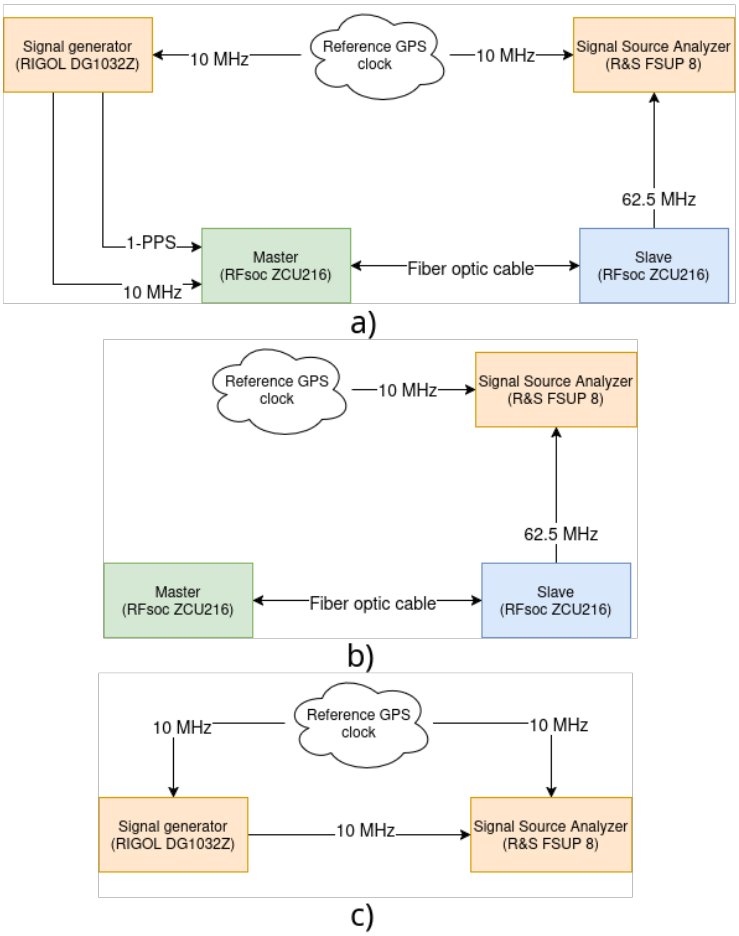

Figure 4.12: Phase noise measurement of the a) slave connected to the master which is using an external clock b) slave connected to the master which is using its internal clock c) signal generator.

4.4.2 Result

Fig. 4.13 shows the phase noise plot of the slave with the master using an external reference clock. The corresponding phase noise plot of the slave with the master using its internal clock is seen in Fig. 4.14. The phase noise plot of the signal generator and the GPS reference clock is shown in Fig. 4.15 and Fig. 4.16, respectively. All figures shows the phase noise over an offset-frequency range of 1 Hz to 3 MHz, except for Fig. 4.17. Fig. 4.17 shows the phase noise of the slave with the master using its internal clock over an offset-frequency range of 30 Hz to 30 MHz, with an integration band of 12 kHz to 20 MHz.

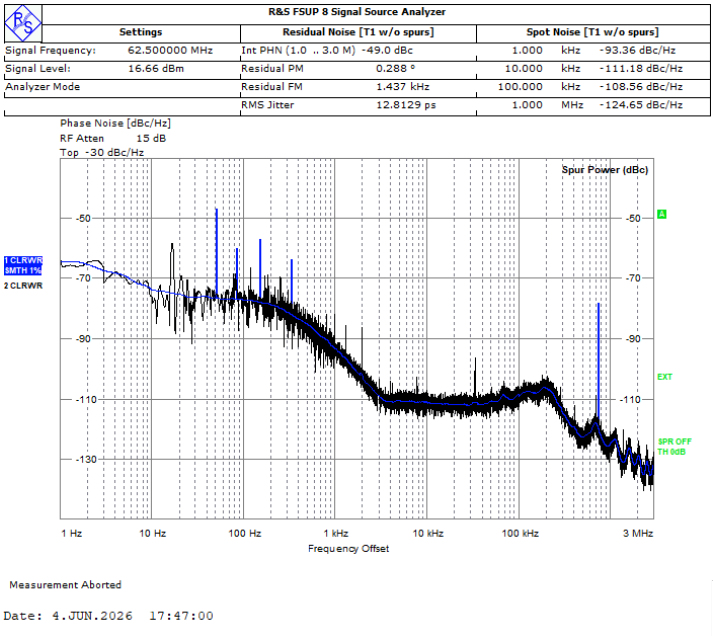

Figure 4.13: Phase noise measurement of the slave with the master using an external clock over a frequency span 1 Hz to 3 MHz.

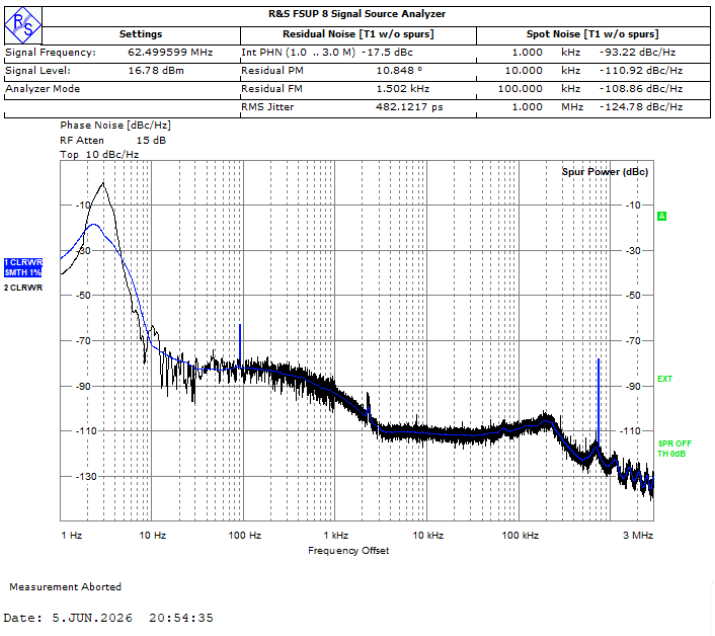

Figure 4.14: Phase noise measurement of the slave with the master using its internal clock over a frequency span of 1 Hz to 3 MHz.

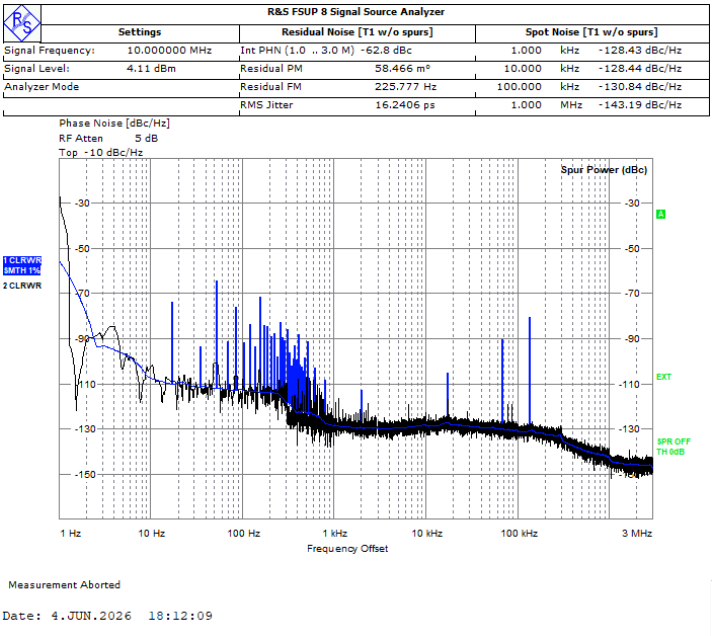

Figure 4.15: Phase noise measurement of the signal generator 10 MHz signal over an offset-frequency range of 1 Hz to 3 MHz.

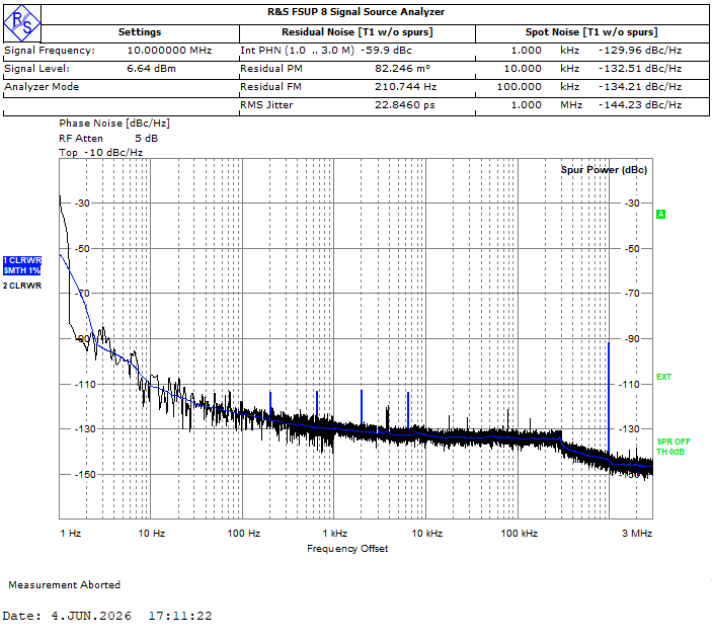

Figure 4.16: Phase noise measurement of the 10 MHz GPS clock signal over an offset-frequency range of 1 Hz to 3 MHz.

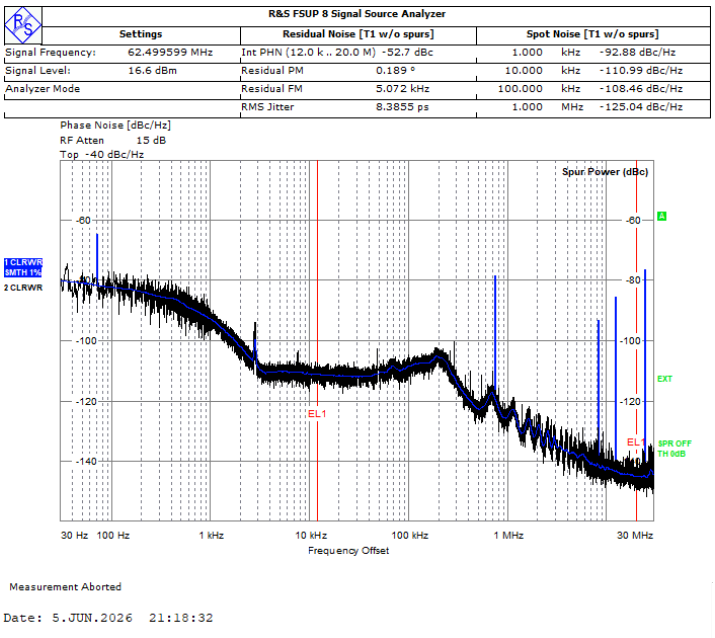

Figure 4.17: Phase noise measurement of the slave with the master using its internal clock over a offset-frequency range of 30 Hz to 30 MHz. The red lines marks the integration band (12 kHz to 20 MHz).

4.5 Hardware utilization

A table showing the hardware utilization of the White Rabbit implementation on the RFSoC ZCU216 can be found in Table 4.4. These values were extracted from the Vivado design tool after implementation.

Table 4.4: Resource Utilization of the White Rabbit port to AMD ZCU216

Resource	**Utilization**	**Available**	**Utilization (%)**
LUT	6371	425280	1.50
LUTRAM	225	213600	0.11
FF	8182	850560	0.96
BRAM	68.5	1080	6.34
DSP	5	4272	0.12
IO	22	408	5.39
GT	3	16	18.75
BUFG	19	696	2.73
MMCM	2	8	25.00

Chapter 5
Discussion

The measurements will be discussed in this chapter. First, the synchronization results will be discussed. Next, the results of the frequency stability analysis will be examined. Finally, the hardware utilization will be discussed to determine whether it can fit on the current LuLIS testbed.

5.1 Synchronization performance

The results are consistent with what is expected from a successful implementation of White Rabbit. The standard deviations of the skew for the PPS signal and 62.5 MHz signal are almost identical (see Fig. 4.3 and Fig. 4.5). This is expected since the signals are generated by the same source.

The measured jitter in master mode and grandmaster mode is almost identical (see Fig. 4.4 and Fig. 4.5). This is most likely due to the jitter being mainly produced by the fractional-N PLL, making the difference not visible. Thus, the limiting factor for the jitter in the system is the fractional-N PLL. This is further supported by the corresponding phase noise plots (see Fig. 4.14 and Fig. 4.13). The phase noise plots follow the typical plot of a fractional-N PLL [37]. It looks like the oscillator dominates at the lower offset-frequencies, as seen in [38], and at higher offsets it is dominated by the VCO. The specified RMS jitter for the offset-frequency range of 12 kHz to 20 MHz is ∼0.68 ps for the oscillator, Si570 [39]. The measured RMS jitter for that range is ∼8.3 ps, which further shows that the jitter is not dominated by the Si570 in this region. The phase noise could potentially be reduced by a couple of picoseconds by replacing the current oscillator with a more stable oscillator connected through the FPGA Mezzanine Card Plus, FMCP, interface. The phase noise plot for Fig. 4.14 has large values at lower offset frequencies, which is due to the instability of the clock. During the measurement, the center frequency changed, thereby increasing the phase noise at lower frequency offsets. Despite this, the system meets the requirement for the cyclic prefix. The calculated cyclic prefix time of 2.34 µs is much larger than the offset. The offset is ∼5700 times smaller than the cyclic prefix time and is also within the 1 ns requirement for White Rabbit. The jitter distribution is also almost identical to that of the Light Rabbit implementation (see Fig. 3.27), and falls within the specification of precision accuracy for White Rabbit. However, the mean is much larger in the Light Rabbit implementation, which is most likely

because the Light Rabbit device is uncalibrated.

The spurs observed in Fig. 4.13 are due to the signal generator. The same spurs appear in the phase noise plot of the signal generator, as seen in Fig. 4.15. Compared with the 10 MHz signal, these spurs are not present (see Fig. 4.16). These spurs could introduce problems in the ADCs and DACs. However, this should not be an issue, since the current testbed uses a signal generator to distribute the frequency to the nodes. The spurs are most likely filtered out by the clock board. The RMS jitter for the 10 MHz signal and the signal generator is quite high. This is most likely due to the length of the 10 MHz clock cable, which caused reflections in the cable. However, this does not affect the results, since the phase noise in grandmaster mode and master mode is the same. This indicates that another component is the main contributor to the noise, probably the fractional-N PLL as discussed previously.

The mean offset of all measurements was between 439 ps and 561 ps. The offset could potentially be lower for all measurements. The reason it is not lower is most likely related to the calibration. The calibration guide [30] could not be followed completely, meaning that some values in the transceivers may not be optimal. This results in a loss of synchronization accuracy, which can be observed in the mean offset.

5.1.1 Frequency measurements

From Table 4.3, it can be seen that when White Rabbit is in master mode, the frequency output is less stable. The standard deviation is 0.41 Hz for master mode compared to 0.0043 Hz in grandmaster mode, which is almost 100 times worse. This result is expected since the external clock connected to the master has higher frequency stability than the master node's own on-board oscillator.

The Allan deviation further proves that the frequency stability of the slave is determined by the selected master clock (see Fig. 4.10 and Fig. 4.11). In grandmaster mode, the slave is almost on par with the signal generator. In contrast, the slave shows significantly worse performance with the master using its internal clock.

Applying the previous result to the LuLIS testbed, with a center frequency of 3.84 GHz and a subcarrier spacing of 60 kHz, the worst-case frequency offset can be calculated using (5.1). The calculated frequency deviation for the slave with the master using its internal clock, master mode, is ±110.42 Hz, while in grandmaster mode it is ±1.28 Hz. Therefore, master mode has a higher risk of causing inter-carrier interference than grandmaster mode. However, the deviation is probably negligible in both cases and will most likely not cause problems when sending and receiving OFDM frames.

$$frequency_{max\_offset} = calculated\_frequency\_stabilty \cdot center\_frequency \tag{5.1}$$

5.2 Hardware utilization

Comparing the hardware utilization (see Table 4.4) with that of the LuLIS testbed BS node (see Table 2.2) shows that the implementation should fit. However, this is based solely on the numbers in the two tables. It is therefore not certain that the timing or routing requirements for the combined implementation are met. Furthermore, the implementation connects its SFP to the transceiver port SFP2. The BS node also uses SFP2, as well as SFP3. The reason for this design choice was that the oscillator needed for the SFP was placed more than two quads above SFP0/1, as discussed in Section 3.4. This means that if the design is implemented on the BS node, the SFP port in either design must be changed.

Another important aspect is the current topology of the testbed. The testbed currently uses a daisy-chain topology for data transfer. At the start of the project, the plan was to implement White Rabbit so that the frequency synchronization could be daisy-chained as well. However, the current implementation does not support this. It is an ordinary clock in PTP terms (see Section 2.3.1), which means that a node can either be a master or a slave. It cannot propagate a signal like a boundary clock or a switch. Therefore, if the implementation is integrated into the testbed, a White Rabbit switch will be needed. The switch would act as the (grand)master instead of the OctoClock. An illustration of this can be seen in Fig. 5.1.

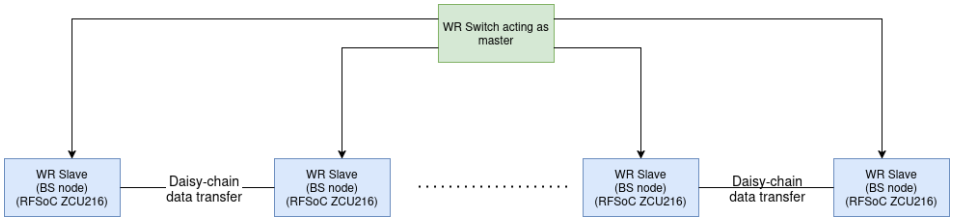

Figure 5.1: A suggestion for frequency synchronization topology if White Rabbit is used instead of the OctoClocks.

Chapter 6
Conclusion & Future work

In this chapter, the conclusion and future work of the thesis are presented. First, the conclusion goes through the whole project of what has been done, whether the goals of the project is met, and if the requirements for the LuLIS testbed are fulfilled. Second, the future work of this project is presented.

6.1 Conclusion

This thesis has investigated what White Rabbit is and how it works. The thesis goes in depth, in Chapter 3, into the synchronization process for both time and frequency, as well as the different hardware and software parts needed. From this, it was concluded that an implementation of White Rabbit without external VCXOs was possible. A version of White Rabbit called Light Rabbit was used as a framework, and from it, an implementation for the RFSoC ZCU216 was created.

Various measurements, such as the time offset between the master and slave, the frequency stability of the slave, the Allan deviation of the slave and the external clock (signal generator), and the phase noise of the slave, signal generator and GPS clock, were performed. All measurements also included a comparison between the White Rabbit master and grandmaster modes. Both modes were used to determine whether our implementation was successful and whether it could be implemented on the LuLIS testbed.

From the previous chapter, it can be concluded that most requirements were met to implement White Rabbit on the LuLIS testbed. In our implementation of White Rabbit, no external VCXOs are needed, since we use the FPGA's QPLLs. The peripherals needed were certain SFPs and a certain fiber-optic cable. The time synchronization remained within the cyclic prefix by a factor of 5700. It can also be concluded that the slave's 62.5 MHz output, with the master using an external clock, is most likely stable enough, since it is almost on par with the signal generator results in the Allan deviation. The chance of frequency drift interfering with the 60 kHz subcarrier should also be low, since the calculated maximum frequency offsets were negligible.

However, combining our implementation with the BS node could cause some problems. One problem is that both use the same SFP port (port 2). Furthermore, even if the combined hardware utilization of the implementation and a BS node does not exceed the available resources, it does not necessarily mean that the

combined implementation will work, since certain requirements, such as timing and routing requirements, can change with a combined design.

Overall, White Rabbit is a promising candidate for replacing the current frequency synchronization on the LuLIS testbed. However, more work needs to be done to determine whether it can actually work, such as placing the White Rabbit implementation together with the BS node.

6.2 Future work

A more thorough investigation should be conducted to uncover the source of the phase noise. A more high-performance oscillator could be connected to the FMCP interface to quantify the amount of noise contributed by the oscillator.

The SFP ports needs also to be looked over, and changed in either the current testbed or our implementation. Our implementation also has to be integrated in the current testbed, which could provide some additional problems such as timing issues.

Further research should also investigate how the current implementation can be modified to function as a switch node. Such an approach could simplify the integration to the current testbed by allowing synchronization between the nodes in a daisy-chained manner. The architecture would also reduce the number of cables and external devices needed in the testbed.

A potential starting point would be to instantiate two nodes, one as a slave which provides the recovered clock to a second node configured as grandmaster. The grandmaster node would then forward the clock to the subsequent node. However, the synchronization performance of this architecture needs to be thoroughly evaluated to determine if it could provide sufficient stability and timing accuracy.

Bibliography

[1] Mini-Circuits, "The Basics of Orthogonal Frequency-Division Multiplexing (OFDM)," Blog post, Feb. 2024, accessed: 2026-05-31. [Online]. Available: https://blog.minicircuits.com/the-basics-of-orthogonal-frequency-division-multiplexing-ofdm/

[2] T. J. Rouphael, "Common digital modulation methods," in *RF and Digital Signal Processing for Software-Defined Radio: A Multi-Standard Multi-Mode Approach*. Newnes, 2009, pp. 25–85.

[3] M. Attari, "Application Specific Instruction-set Processors for Massive MIMO Systems," Doctoral thesis, Lund University, Lund, Sweden, Oct. 2024, department of Electrical and Information Technology. Accessed: 2026-05-31. [Online]. Available: https://lucris.lub.lu.se/ws/portalfiles/portal/197410452/Application_Specific_Instruction-set_Processors_for_Massive_MIMO_Systems.pdf

[4] D. Iancu, V. Snygg, S. Cheng, L. Tinnerberg, M. Henriksson, E. Bergman, A. J. Johansson, B. Behmanesh, O. Edfors, and L. Liu, "A Scalable 256-Antenna Distributed MIMO Testbed with Real-Time Fully Digital Beamforming," arXiv preprint arXiv:2605.02388, May 2026, submitted May 4, 2026. Accessed: 2026-05-31. [Online]. Available: https://arxiv.org/pdf/2605.02388

[5] Silicon Laboratories Inc., "A primer on jitter, jitter measurement and phase-locked loops," Silicon Laboratories Inc., Application Note AN687, May 2012, rev. 0.1. [Online]. Available: https://www.all-electronics.de/files/2025/10/15/an687.pdf

[6] J. Wilson, "Timing jitter dictionary and measurement guide," https://www.skyworksinc.com/-/media/skyworks/sl/documents/public/white-papers/timing-jitter-dictionary-and-measurement-guide-ebook.pdf, 2022, accessed: 2026-06-07.

[7] J. Serrano, T. Włostowski, P. Alvarez, M. Lipiński *et al.*, "Precise time and frequency transfer in a white rabbit network," CERN, Tech. Rep., 2009, accessed: 2026-03-25. [Online]. Available: https://white-rabbit.web.cern.ch/documents/Precise_time_and_frequency_transfer_in_a_White_Rabbit_network.pdf

[8] P. Moreira, P. Alvarez, J. Serrano, I. Darwezeh, and T. Wlostowski, "Digital dual mixer time difference for sub-nanosecond time synchronization in ethernet," in *2010 IEEE International Frequency Control Symposium*, 2010, pp. 449–453.

[9] G. Daniluk, "White rabbit ptp core: The sub-nanosecond time synchronization over ethernet," Master's thesis, Warsaw University of Technology, Faculty of Electronics and Information Technologies, Institute of Electronic Systems, Warsaw, Poland, 2012.

[10] B. Razavi, *LC Oscillator Design*. Cambridge University Press, 2020, p. 138–177.

[11] A. Scarbro, "Sub-nanosecond network synchronisation: An introduction to white rabbit," https://www.armms.org/media/uploads/sub-nanosecond-network-synchronisation---an-introd.pdf, 2022, aRMMS RF & Microwave Society presentation/document, Issue 2, Apr. 2022. Accessed: 2026-04-10.

[12] Open Hardware Repository (OHWR), "Wrpc core (white rabbit ptp core) wiki," https://gitlab.com/ohwr/project/wr-cores/-/wikis/Wrpc-core, n.d., gitLab wiki page describing the WRPC (White Rabbit PTP Core) and related HDL cores. Accessed: 2026-04-09.

[13] M. M. Lipinski, "Methods to increase reliability and ensure determinism in a white rabbit network," Ph.D. dissertation, Warsaw University of Technology, Faculty of Electronics and Information Technology, 2016, cERN-THESIS-2016-283, EuCARD-2 Project. [Online]. Available: http://cds.cern.ch/search?p=CERN-THESIS-2016-283

[14] *UltraScale Architecture GTH Transceivers User Guide*, v1.7.1 ed., AMD Xilinx, Aug. 2021, release date: 2021-08-18. [Online]. Available: https://docs.amd.com/v/u/en-US/ug576-ultrascale-gth-transceivers

[15] V. V. David Taylor, Matt Klein and A. D. Fresco, "All digital vcxo replacement using a gigabit transceiver fractional pll," AMD Xilinx, Tech. Rep. XAPP1276, Jan. 2022, application Note. [Online]. Available: https://docs.amd.com/v/u/en-US/xapp1276-vcxo

[16] F. Pfautsch and U. Langenbach, "Light Rabbit: Implementing a White Rabbit Node on COTS AMD Development Boards Without Relying on External VCXOs," Presented at the 13th White Rabbit Workshop and WR Collaboration, Geneva, Switzerland, Mar. 2024, presentation, Mar. 20, 2024. Accessed: 2026-05-28. [Online]. Available: https://www.missinglinkelectronics.com/wp-content/uploads/2024/03/MLE-Light-Rabbit-Presentation-at-13th-White-Rabbit-Workshop.pdf

[17] Lattice Semiconductor Corporation, "8b/10b encoder/decoder," Lattice Semiconductor, Reference Design RD1012 RD1012, January 2015, version 1.4. [Online]. Available: https://www.latticesemi.com

[18] Ericsson, "Massive MIMO Explained: Unlocking 5G Efficiency," Web page, accessed: 2026-06-06. [Online]. Available: https://www.ericsson.com/en/ran/massive-mimo

[19] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for Next Generation Wireless Systems," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 186–195, Feb. 2014. [Online]. Available: https://lucris.lub.lu.se/ws/portalfiles/portal/3993364/5323045.pdf

[20] M. Lipiński, T. Włostowski, J. Serrano, and P. Alvarez, "White Rabbit: A PTP Application for Robust Sub-Nanosecond Synchronization," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS)*, Munich, Germany, 2011, pp. 25–30.

[21] Open Hardware Repository (OHWR), "White rabbit project wiki," https://gitlab.com/ohwr/project/white-rabbit/-/wikis/home, n.d., gitLab repository documentation for the White Rabbit project. Accessed: 2026-04-09.

[22] D. L. Mills, *Computer Network Time Synchronization: The Network Time Protocol*. Boca Raton, FL, USA: CRC Press, 2006.

[23] C. Karle, M. Neu, B. Nuss, L. Witte, A. Scheder, E. Waldner, E. Shkurtaj, T. Harbaum, and J. Becker, "Scalable Multi-Level Synchronization Technique of Distributed Multi-RFSoC-Server Systems for 6G," in *2024 IEEE 37th International System-on-Chip Conference (SOCC)*, Dresden, Germany, 2024, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10737777

[24] W. J. Riley and D. A. Howe, "Handbook of Frequency Stability Analysis," National Institute of Standards and Technology, Gaithersburg, MD, USA, NIST Special Publication 1065, Jul. 2008. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=50505

[25] Beckhoff Automation, *EL6688: IEEE 1588 PTP External Synchronization Interface*, Beckhoff Automation, Oct. 2024, version 3.1.0, Oct. 15, 2024. Accessed: 2026-06-08. [Online]. Available: https://infosys.beckhoff.com/content/1033/el6688/index.html

[26] White Rabbit Project. (n.d.) Ieee1588 standard. White Rabbit Collaboration. Accessed: 25 March 2026. [Online]. Available: https://www.white-rabbit.tech/ieee1588-standard/

[27] Open Hardware Repository, "SFP," White Rabbit Project GitLab Wiki, accessed: 2026-05-30. [Online]. Available: https://gitlab.com/ohwr/project/white-rabbit/-/wikis/SFP

[28] IEEE, "IEEE Std 802.3ah-2004: IEEE Standard for Information Technology—Telecommunications and Information Exchange Between Systems—Local and Metropolitan Area Networks—Specific Requirements—Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications—Amendment: Media Access Control Parameters, Physical Layers, and Management Parameters for Subscriber Access Networks," IEEE Standard, Sep. 2004, published Sep. 7, 2004. Accessed: 2026-05-30. [Online]. Available: https://www.ieee802.org/21/doctree/2006_Meeting_Docs/2006-11_meeting_docs/802.3ah-2004.pdf

[29] C. Barrett, “Fractional/integer-n pll basics,” Texas Instruments, Technical Brief SWRA029, Aug. 1999, accessed: 2026-06-01. [Online]. Available: https://www.ti.com/lit/an/swra029/swra029.pdf

[30] White Rabbit Collaboration, “White rabbit calibration (version 1.1),” https://gitlab.com/ohwr/project/white-rabbit/-/wikis/uploads/76cdbdbadccc9d6c54d5caf246550fbf/WR_Calibration-v1.1-20151109.pdf, 2015, technical documentation describing calibration procedures for White Rabbit devices. Accessed: 2026-04-10.

[31] Open Hardware Repository, “Software for White Rabbit PTP Core,” GitLab repository, accessed: 2026-05-30. [Online]. Available: https://gitlab.com/ohwr/project/wrpc-sw

[32] Open Hardware Repository (OHWR), “White rabbit ptp core v5.0 released,” https://ohwr.org/news/wr-cores-8/, 2023, published Dec. 23, 2023. Accessed: 2026-04-10.

[33] Open Hardware Repository, “uRV Core,” OHWR project page, accessed: 2026-06-01. [Online]. Available: https://ohwr.org/projects/urv-core/

[34] M. Rizzi, “Digital dual mixer time difference: Phase noise stability,” CERN / Open Hardware Repository, Tech. Rep., Feb. 2017, accessed: 2026-05-28. [Online]. Available: https://gitlab.com/ohwr/project/wr-low-jitter/-/wikis/uploads/62c4f265c806154bef2a067600d3e15e/DDMTD_report_final.pdf

[35] Missing Link Electronics (MLE), “Mle upstreamed an implementation for vcxo-less white rabbit nodes on an amd zynq ultrascale+ mpsoc zcu102 evaluation board,” https://www.missinglinkelectronics.com/company/news/mle-upstreamed-an-implementation-for-vcxo-less-white-rabbit-nodes-on-an-amd-zynq-ultrascale-mpsoc-zcu102-evaluation-board/, 2024, published Oct. 21, 2024. Accessed: 2026-04-10.

[36] Symmetricom, “Cs4000 Cesium Frequency Standard,” Data sheet, accessed: 2026-06-04. [Online]. Available: https://testequipment.center/Product_Documents/Symmetricom-CS4000-Specifications-A990E.pdf

[37] G. Giust. (2024, Mar.) How to identify the source of phase jitter through phase noise plots. SiTime Corporation. [Online]. Available: https://www.sitime.com/company/newsroom/blog/how-identify-source-phase-jitter-through-phase-noise-plots

[38] Silicon Laboratories Inc., “Si57x Presentation,” PDF presentation, 2010, archived by the Internet Archive (Wayback Machine), accessed Jun. 11, 2026. [Online]. Available: https://web.archive.org/web/20151129152821/http://www.silabs.com/Marcom%20Documents/Resources/Si57xPresentation.pdf

[39] Skyworks Solutions, Inc., “Si570/Si571: 10 MHz to 1.4 GHz I2C Programmable XO/VCXO,” Data sheet, Mar. 2022, rev. 1.6, Mar. 4, 2022. Accessed: 2026-05-29. [Online]. Available: https://www.skyworksinc.com/-/media/SkyWorks/SL/documents/public/data-sheets/Si570-71.pdf

[40] OpenCores Organization, "WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores," https://cdn.opencores.org/downloads/wbspec_b3.pdf, revision B.3, released September 7, 2002. Accessed: 2026-05-28.

[41] LiveAction, "Gigabit ethernet and fibre channel technology," n.d., accessed: 2026-04-29. [Online]. Available: https://www.liveaction.com/glossary/gigabit-ethernet-and-fibre-channel-technology/

Appendix A

Components of Fractional-N synthesizer

Voltage-Controlled Oscillator

By exploiting the property that negative feedback in a system can become unstable, we can construct an oscillator using a feedback system. For a simple feedback system in Fig. A.1, we can write the closed-loop transfer function as,

$$\frac{Y(s)}{X(s)} = \frac{H(s)}{1 + H(s)} \tag{A.1}$$

We note that if X(s) is a sinusoidal wave, $s = jw_0$, and $H(jw_0) = -1$, the open-loop frequency response exhibits a 180° phase shift and a unity magnitude at w_0. For $(Y(jw_0)/X(jw_0))$ the system provides an infinite gain. The return signal exhibits another 180° phase shift due to the nominally negative feedback. This results in a total phase shift of 360° at w_0 meaning the signal circles around the loop enhancing it self. These conditions for oscillation can be summarized as,

$$|H(jw_0)| = 1 \tag{A.2}$$

$$\angle H(jw_0) = 180^\circ \tag{A.3}$$

These conditions are called the Barkhausen's criteria. $H(jw_0)$ is called the startup condition. Typically are oscillators designed with $|H(jw_0)| > 1$ and $\angle H(jw0) = 180^\circ$ due to temperature, PVT, and gain drop due to non-linearity. However, $|H(jw0)| > 1$ does not always guarantee oscillation but is sufficient for typically oscillators.

Figure A.1: Feedback system [10]

LC Oscillator LC oscillators have the advantage of lower noise and the ability to operate at high frequencies. By placing and inductor and capacitor in parallel we exhibit some interesting property. We know for such a circuit the impedance is dominated by L in lower frequencies and C in higher frequencies. The impedance

for the circuit can be expressed as $Z = (Ljw)||[1/(Cjw)]$. For a resonance frequency, w_0, we can see that the impedance grows indefinitely, and therefore the LC-tank exemplifies a resonator. In reality a more realistic inductor model is incorporated with a series resistance. However, this give little intuition of the behavior of the "lossy" tanks compared to an ideal LC tank. Therefore, we often model it as a parallel resistor. This model isn't equivalent for all frequencies but can approximate for some frequency range. The impedance for this model at resonance rises only to $|R_p|$, while L and C cancel each other out.

By using the described LC-tank we can create an LC oscillator. An example of an LC oscillator is the two cascade tuned CS stages in a loop, shown in Fig. A.2. The first CS stage exhibits a $-g_m R_p$ gain and a 180° phase shift at resonance frequency. Therefore, by cascading, the output exhibits an open-loop gain of $(g_m R_p)^2$ and a phase shift of $-360°$. The closed-loop circuit oscillates therefore as long as $(g_m R_p)^2 \geq 1$.

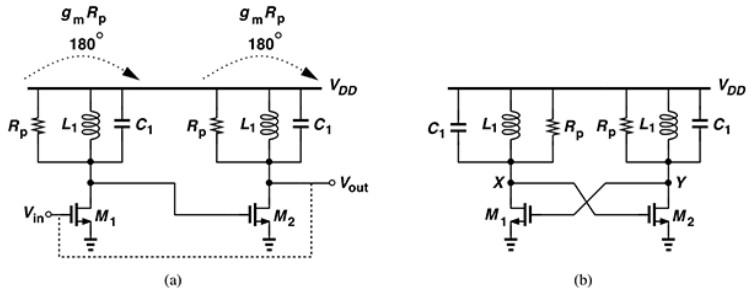

Figure A.2: a) Two cascaded tuned CS stage in a loop, b) Redrawn as cross-coupled [10]

VCO A oscillator that can be controlled by a voltage is called a voltage-controlled oscillator, VCO. The tuning characteristic of an ideal VCO is a straight line given by $w_{out} = K_{VCO} V_{cont} + w_0$, where K_{VCO} is called the gain. Fig. A.3 shows a simple LC VCO based on the previously described circuit. The frequency is tuned by varactor capacitors with a voltage, V_{cont}. A varactor capacitor is a variable capacitor controlled by a voltage.

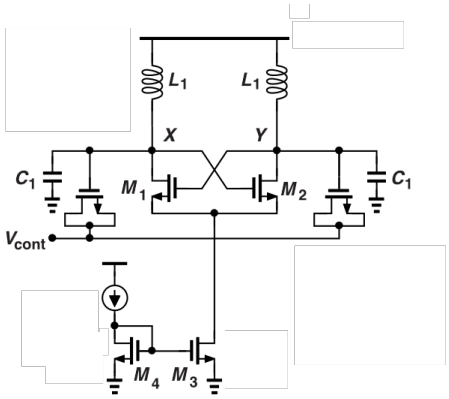

Figure A.3: LC VCO [10]

Frequency Divider

A ÷2 divider can be realized as a master-slave D flipflop placed in a negative-feedback loop, see Fig. A.4. A tracks B when CK goes high and is frozen when CK goes low. Similarly B tracks $\bar{A}$ when CK falls and is frozen when CK goes high. Therefore, A and B toggles half of CK with a phase difference of 90°.

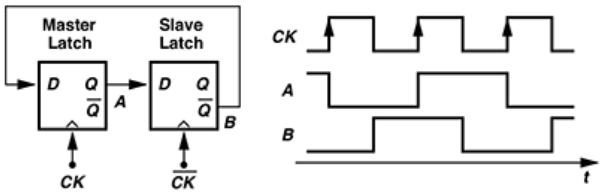

Figure A.4: Div2 circuit [10]

A ÷3 circuit requires two D flipflops to support three different states. Shown in Fig. A.5 is an ÷3 circuit, if $Q_1\bar{Q}_2 = 00$ and CK goes high, $\bar{Q}_2$ goes high while Q_1 remains zero. In the next cycle Q_1 goes high and makes $Q_1\bar{Q}_2 = 11$, generating a 1 in the AND gate forcing $\bar{Q}_2$ next cycle to zero. Next cycle Q_1 stays high while $\bar{Q}_2$ goes to zero. The pattern after that are repeating, yielding a duty cycle of 1/3 for both outputs.

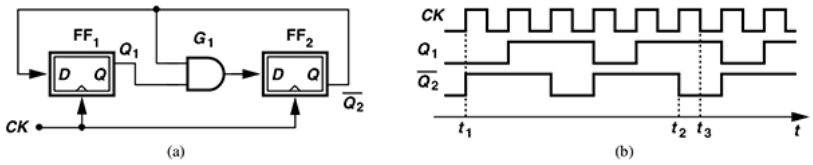

Figure A.5: Div3 circuit [10]

Dual-Modulus Prescalar By modifying the ÷3 circuit a dual-modulus ÷2/3 can be created, as illustrated in Fig. A.6. The circuit adds an OR gate to reduce the circuit to one flipflop when the circuit needs to divide by two. If the control input MC, Modulus Control, is low the circuit works as previously but when MC is high, Q_1 has no effect and the circuit acts as a ÷2 circuit.

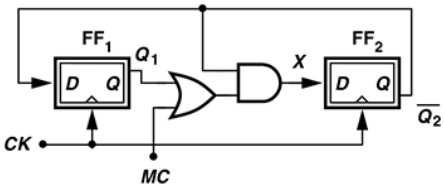

Figure A.6: Dual-modulus ÷2/3 [10]

Phase Frequency Detector

In a PLL, we wish to lock signals regardless of the initial value of the output frequency and phase. Such a PLL would have an acquisition range equal to the VCO tuning range. A Phase Frequency Detector, PFD, is such a circuit that can resolve phase difference in the range of +/- 2π or more. The operation of the PDF is based on the principle that if Q_A and Q_B are low and a rising edge occurs on A then Q_A goes high while Q_B remains low. If an rising edge occurs on B while A is high, then Q_A is reset. The same happens for the opposite. In the case of equal frequency but different phase the average value of Q_A minus Q_B is linearly propositional to the phase error. For $f_A > f_B$, Q_A exhibits a greater average than Q_B.

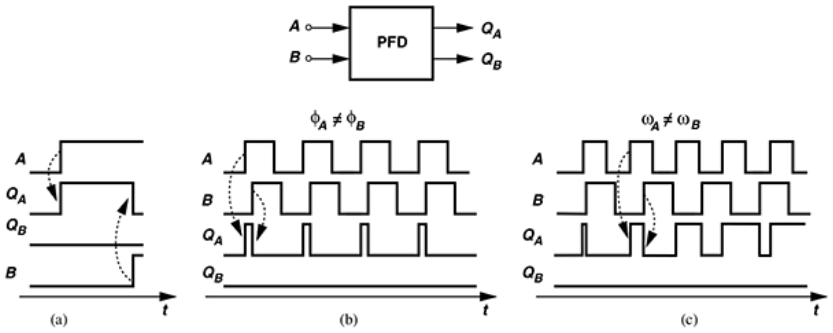

Figure A.7: Operation of PFD [10]

An implementation of the PFD described above can be shown in Fig. A.8. It consists of two resettable D flipflops and an AND gate. The D input of the flipflops are held high while sensing the clock pulse of A and B. When both Q_A and Q_B are high the AND gate forces Q_A and Q_B to zero. Unlike the ideal case where both Q_A and Q_B will be high simultaneously for a brief time.

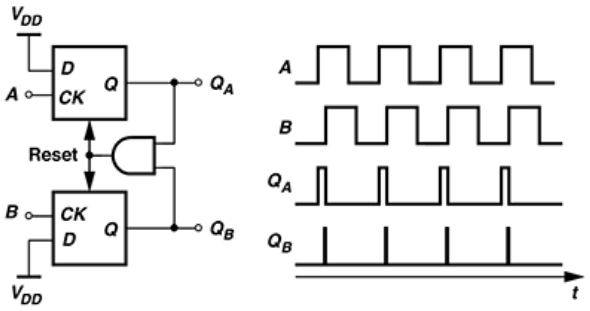

Figure A.8: Implemented PFD [10]

Charge Pump

The principle of a charge pump is to charge and discharge for controlled amount of time. A simple realization of charge pump shown in figure A.9, charges C_1 when S_1 is on with I_1, and discharges it with I_2 when S_2 is on. The signals Up and Down controls respectively S_1 and S_2. In the circuit we assume $I_1 = I_2$.

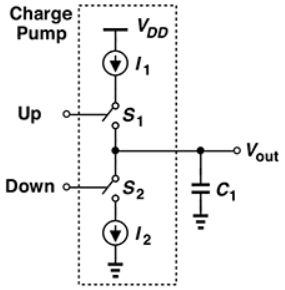

Figure A.9: Simple charge Pump [10]

Combining the charge pump with the previously discussed PFD, we have a circuit shown in Fig. A.10. From the figure, one can observe that each time a phase comparison is made, V_{out} rises by $\Delta V = (\Delta\phi_1/2\pi)T_{in}(I_{cp}/C_1)$. If the phase difference remains constant V_{out} goes towards infinity. Therefore, for V_{out} to be finite the phase error must be zero so the CP injects zero net charges.

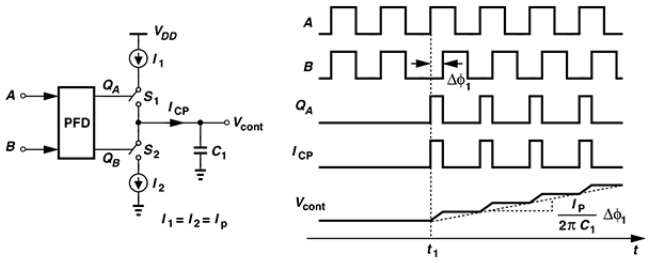

Figure A.10: PFD/CP/Capacitor circuit with associated waveform [10]

Transfer function

The transfer function for the PDF/CP/Capacitor cascade can be calculated by applying a phase step as seen in Fig. A.10 and taking the derivative of the output to find the impulse response. By approximating V_{count} as a continuous ramp shown by the dashed line, the attribution of V_{count} can be determined as $\Delta V/T_{in} = (\Delta\phi_1/2\pi)(I_{cp}/C_1)$. The impulse response can therefore be determined as $V_{count}/\Delta\phi_1(s) = I_p/(2\pi C_1 s)$.

Constructing a linear phase model of a PLL with PFD/CP/capacitor cascade depicted in Fig. A.11, we obtain an open-loop transfer function of $[I_p/(2\pi C_1 s)](K_{VCO}/s)$.

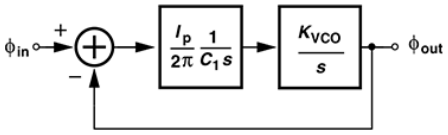

Figure A.11: Transfer function for a PDF/CP/Capacitor cascade [10]

The closed-loop transfer function contains two imaginary poles implying an unstable loop. To stabilize the loop, we can add a resistor in series with C_1 to add an open-loop zero to the transfer function. However, the series resistor makes the control voltage jump by an amount of $I_p R_1$ when the charge pump turns on or off, an effect absent in the circuit without a series resistor. To reduce the ripple, a simple solution is to add a parallel capacitor, C_2, to force the initial charge current to flow through C_2, therefore lowering the peak to peak ripple to $(I_p/C_2)T_{sk}$. A side effect of adding C_2 is the degradation of the phase margin for adding an extra pole to the transfer function. Using this arrangement we have created a "charge-pump PLL" shown in Fig. A.12. The R_1C_1 and C_2 branches are called the "loop filter".

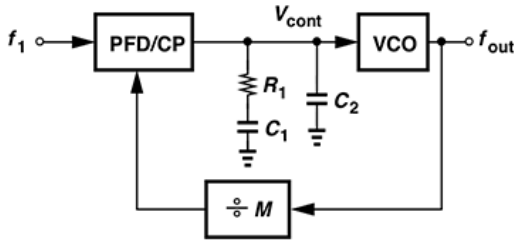

Figure A.12: Second capacitor in the loop filter of a integer-N frequency synthesizer [10]

Appendix B
Wishbone

Wishbone is an open source protocol used for buses and interfaces. It is developed by OpenCores.org and has numerous of features which includes the following [40]:

- It is a logical IP core hardware that is simple and it requires few logical gates.
- It utilizes master/slave architecture for flexible system designs.
- There is support for common data transfer for read and write. Such as typical read/write cycle, block transfer cycle and RMD (read modify write) cycle.
- The interface supports both big-endian and little-endian data ordering.
- Both data and address fields width is configurable.
- The handshaking protocol allows IP to throttle transfer speed.
- A wide range of interconnection methods is supported. They are point-to-point, crossbar, shared bus, switch, and switch fabric interconnection.
- User defined tags can also be utilized. Such tags can apply information to a address bus, data bus or a bus cycle. The tags can be helpful when identifying data transfer, interrupt vectors, cache control operations and data transfers.

The White Rabbit PTP core (WRPC) uses it to communicate with all of its modules inside of it. A wishbone bus can communicate in two ways, either in pipeline mode or in standard mode. The pipeline mode offer better transferring performance when transferring multiple data words in a single wishbone cycle. Both of these mode could be used by the WRPC. On one hand, there is a relative low amount of data exchanged, meaning both mode sends data at the same rate anyways. However, on the other hand, the processor is constantly fetching instruction and data from the RAM. Therefore, the pipeline mode is used for the buses throughout the whole WRPC design [9]. As mentioned above, the Wishbone bus does have a master/slave architecture, where the master initialize the contact with a read/write cycle and the slave responds. The Wishbone master/slave bus has the following signals in White Rabbit [9]:

- NAME_X/Y: Example notation, not a part of the protocol. NAME represents the signal name, X indicates whether the signal is an input or output for the master, and Y indicates whether the signal is an input or output for the slave.

- CYC_O/I: When the master asserts this line, a Wishbone cycles starts. The cycle is valid until the masters deasserts it.

- STB_O/I: Is a strobe signal, which indicates only a valid data transfer. When this signal is asserted by the master, the rest of the signal are considered valid.

- WE_O/I: Shows whatever the current transfer is a read or write cycle. Asserted during writes and negated during reads.

- ADR_O/I: The ADR signal is used to desribe which word the master wants to read/write from the memory or register.

- DAT_O/I: The master passes the data it wants to write to the slave.

- DAT_I/O: The data the slave passes to the master from a read operation.

- SEL_O/I: Indicates which byte lanes of the data bus that are valid during a transfer, allowing smaller data units, such as a single byte, to be transferred over a wider bus.

- ACK_I/O: An acknowledgment signal from the slave, which indicates the read/write operation was completed.

- STALL_I/O: This signal is only used in pipeline mode. It can be used by the slave to indicate that it cannot accept anymore transfers at the moment. The master can not continue until the slave deasserts the signal.

- ERR_I/O: Whenever a slave detects an error it can report it. This indicates that the cycle should be terminated. An example of this is when a master tries to access an address that is outside of the slaves address range.

The actual wishbone interface have more signals [40], however the ones above are the ones that are used within White Rabbit.

An example of a pipelined Wishbone write cycle can be seen in Fig. B.1 [9]. The cycles starts by the master asserting the STB signal, and sends a burst of write and read requests. The slave then process each write/read request and sends an ACK for a single clock cycle for every processed write/read request. The slave can also stall the burst by asserting the STALL signal. Whenever the cycle is done, the master deasserts the CYC signal. Comparing pipelined mode to standard mode, the master needs to wait for a ACK from the slave every time it does a read/write operation before proceeding with the next data word [9].

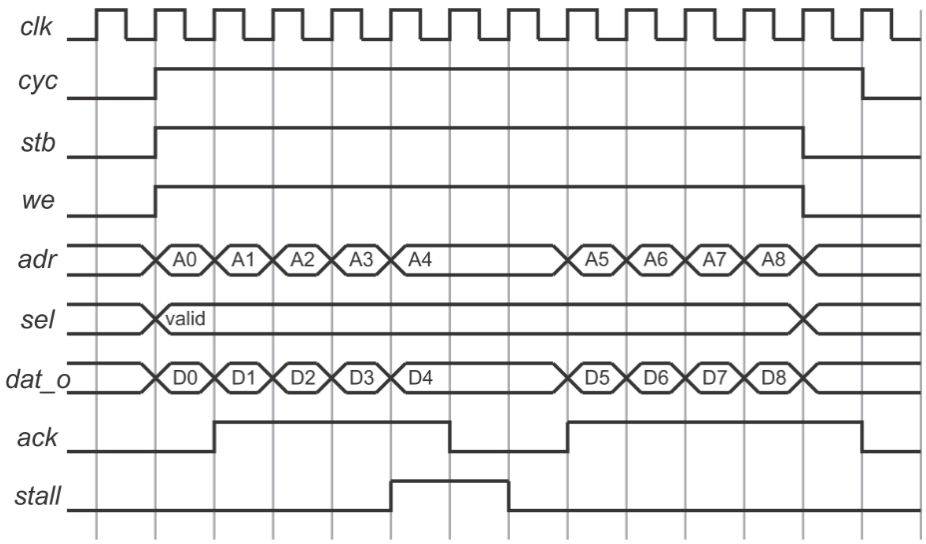

Figure B.1: An example of a pipelined Wishbone write cycle [9]

Beside the standard wishbone bus, a modified version of the wishbone bus, called White Rabbit Fabric (WRF), is also used in the WRPC [7]. A more in-depth explanation of how it works can be found in Section 3.2.1.

Appendix C

8b10b encoding

As stated in Section 2.3.3, the PCS maps the MAC data stream to data character (Dx.y) and inserts control character (Kx.y). The 8-bit input data is encoded to 10 bit data groups from a standardized lookup table, hence the name 8b/10b encoding.

The control characters (Kx.y) are inserted depending on the control signals provided by the GMII (Gigabit Media Independent Interface), which accompanies the MAC data stream [41]. The mapping is done by breaking the original 8 bit group into two blocks, the three most significant bits (y) in one block and the five least significant bits (x) in another. Each block is mapped to a lookup table. Thereafter, the blocks are concatenated together and they have a ten-bit encoded value. A more comprehensive view of the scheme is shown in Fig. C.1 [17].

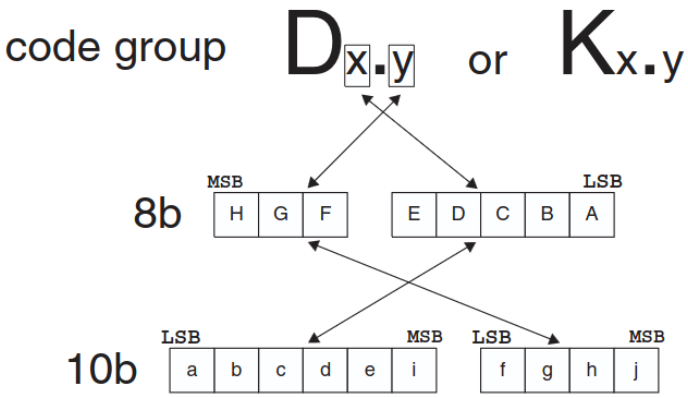

Figure C.1: 8b10b coding scheme [17]

There are a couple of reason for splitting the bits into two blocks, and mapping them to a lookup table. One of them is to be able to create a DC-balanced stream. This is done by controlling the running disparity, which is the difference between the number of ones and zeros transmitted in a code group (Dx.y or Kx.y). Disparity is calculated as the number of ones minus the number of zeros in a code group, while the running disparity is tracked over time. If the disparity is zero,

the code group is said to be disparity neutral. The goal is to keep the running disparity bounded [17].

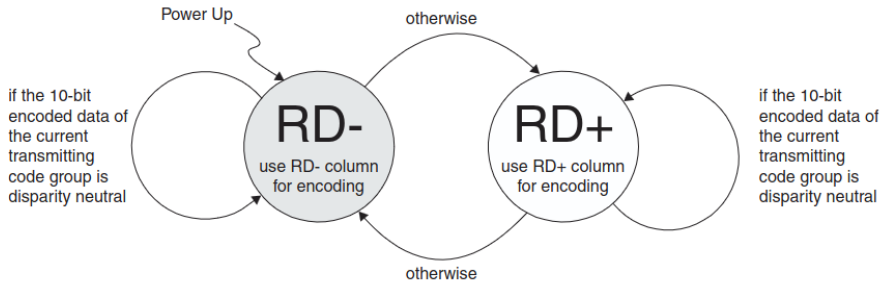

Figure C.2: State machine of the 8b/10b running disparity [17].

3b Decimal	3b Binary (HGF)	4b Binary (fghi)
0	000	0100 or 1011
1	001	1001
4	100	0010 or 1101
7	111	0001 or 1110 or 1000 or 0100

5b Decimal	5b Binary (EDCBA)	6b Binary (abcdei)
0	00000	100111 or 011000
1	00001	011101 or 100010
4	00100	110101 or 001010
9	01001	110101

Code Group (Dx.y/Kx.y)	8-bit data HGF EDCBA	10-bit data (RD-) acbcdei fghj	10-bit data (RD+) abcdei fghj
D0.0	000 00000	100111 0100 (+0)	011000 1011 (-0)
D1.1	001 00001	011101 1001 (+2)	100010 1001 (-2)
D4.4	100 00100	001010 1101 (+0)	110101 0010 (-0)
D9.4	100 01001	100101 1101 (+2)	100101 0010 (-2)
K28.5	101 11100	001111 1010 (+2)	110000 0101 (-2)

Table C.1: A couple values from the standard 8b/10b lookup table. The 3 bits is the y block and 5 bits is the x block. The (RD-) column contains disparity of 0 or +2, and the (RD+) column contains disparity of 0 or -2. The RD+/- is representing the current state of the state machine in Fig. C.2 [17].

Keeping the disparity neutral for every code group is impossible. Instead, the 8b/10b encoding uses code groups, which has different disparities and controls the overall balance through the running disparity. There are encoding of values which

have two possible disparities, either positive or negative. The encoder chooses between the values based on current running disparity with the help of a state machine, see Fig. C.2. Thus keeping the overall disparity bounded over time. Code groups may have a disparity of ± 2, while the running disparity is kept within a limited range. Table C.1 shows examples of how the encoding works with disparity [17].