Warsaw University of Technology
Faculty of Electronics
and Information Technologies
Institute of Radioelectronics

Academic year 2010/2011

Master of Science thesis

Tomasz Włostowski

Precise time and frequency transfer in a White Rabbit network

Supervisor:
prof. nzw. dr hab. Zbigniew Kulka

.....
Evaluation

.....
Signature of the Head of
Examination Committee

Acknowledgements:
Mr Javier Serrano,
Mr Maciej Lipiński,
Mr Pablo Alvarez Sanchez,
Mr Alessandro Rubini,
Mr Emilio Garcia Cota,
the members of the BE-CO-HT section at CERN
and all White Rabbit project contributors,

Mr Grzegorz Kasprowicz,

Professor Zbigniew Kulka

Thank you for your help and support!

I dedicate this thesis to my Mom.
Tomek

Abstract

The White Rabbit [5] project is a multi-laboratory and multi-company effort to design a control network based on Ethernet, which provides deterministic data transfer capabilities and an accurate clock synchronization mechanism.

This thesis describes in detail the mechanisms used in the White Rabbit network for precise time and frequency transfer (with a master-slave time offset below 1 ns).

The first chapter contains an introduction to timing systems with a focus on systems used in particle accelerators. In the 2nd chapter, the reader is presented the basic theory (phase noise, jitter, phase-locked loops) required to understand the operation of such systems. The 3rd chapter describes the White Rabbit network structure and the employed synchronization and syntonization methods (Precision Time Protocol — IEEE 1588 in conjunction with Synchronous Ethernet). The 4th chapter presents a prototype hardware and software design of the key component in a White Rabbit network — the White Rabbit Switch.

The last two chapters summarize the results, discuss possible applications and plans for further development.

Streszczenie

Precyzyjny transfer czasu i częstotliwości w sieci White Rabbit

White Rabbit [5] jest projektem opartej na standardzie Ethernet sieci komputerowej, umożliwiającej deterministyczne transfery danych oraz precyzyjny transfer czasu i częstotliwości. Projekt powstaje przy współudziale kilku instytucji naukowych oraz firm.

W niniejszej pracy szczegółowo opisano mechanizmy wykorzystywane w sieci White Rabbit do precyzyjnej synchronizacji czasu (z błędem poniżej 1 ns) i transferu wzorca częstotliwości.

Pierwszy rozdział zawiera wprowadzenie do systemów synchronizacji czasu, z naciskiem na systemy stosowane w akceleratorach cząstek. Następnie, w rozdziale drugim czytelnik poznaje podstawy teoretyczne (szumy fazowe, jitter, pętle fazowe PLL) niezbędne do zrozumienia zasad działania takich systemów. Rozdział 3 opisuje strukturę sieci White Rabbit oraz zastosowane mechanizmy synchronizacji czasu (Precision Time Protocol – IEEE 1588 w połączeniu z synchronicznym Ethernetem). W rozdziale 4 przedstawiono projekt prototypu sprzętu i oprogramowania centralnego elementu sieci White Rabbit — przełącznika Ethernetowego WR.

Ostatnie dwa rozdziały zawierają podsumowanie pracy, omawiają możliwe aplikacje systemu oraz dalsze plany jego rozwoju.

Contents

List of abbreviations

1588	IEEE 1588 standard
802.3	IEEE 802.3 Ethernet
AMC	Advanced Mezzanine Card
BMC	Best Master Clock (algorithm)
CDR	Clock Data Recovery
DAQ	Data Acquisition System
DDMTD	Digital Dual Mixer Time Difference
DDS	Direct Digital Synthesis
GMT	General Machine Timing
GMRP	Generic Multicast Registration Protocol
HAL	Hardware Abstraction Layer
HP	High Priority (Ethernet frame)
IPMI	Intelligent Platform Management Interface
MAC	Media Access Controller
MCH	Management Carrier Hub
NTP	Network Time Protocol
PCS	Physical Coding Sublayer
PI	Proportional-Integrating (regulator)
PD	Phase Detector
PDV	Packet Delay Variation
PHY	Physical Layer (device)
PLL	Phase-Locked Loop
PPS	Pulse Per Second (timing signal)
PSD	Power Spectral Density
PTP	IEEE 1588 Precision Time Protocol
RTU	Routing Table Unit
SMF	Single-Mode Fiber
SNMP	Simple Network Management Protocol
SSH	Secure Shell
STM	System Timing Master
Sync-E	Synchronous Ethernet
TAI	*Temps Atomique International* - International Atomic Time
TXTSU	TX Timestamping Unit

TDC	Time-to-Digital Converter
TLV	Type-Length-Value
TSU	Time Stamping Unit
UI	Unit Interval
UTC	Universal Coordinated Time
VCO	Voltage Controlled Oscillator
VCXO	Voltage Controlled Crystal Oscillator
TCVCXO	Temperature Compensated VCXO
WDM	Wavelength Division Multiplexing
WR	White Rabbit
WRS	White Rabbit (Ethernet) Switch

1. Introduction

1.1. Background

The Large Hadron Collider (LHC) is arguably the biggest and most complex scientific instrument built by humankind. It is located in a 27 km-long underground tunnel, crossing the Swiss-French border near the city of Geneva.

The foremost purpose of the LHC is to extend our understanding of the current theory describing elementary particles, called the Standard Model, to verify the existence of the Higgs boson and to study "new" physics beyond the Standard Model such as supersymmetry or extra dimensions. There are also expectations for clarifying the origin and properties of the so-called dark matter which is believed to constitute about 25% of all the matter in the Universe. Theoretical expectations place these phenomena within the teraelectronvolt energy range, accessible by the LHC with its 14 TeV center-of-mass collision energy. The probability of the creation of interesting particles in a single collision is however very small. This justifies the need for a collider which can simultaneously provide high beam energy and high luminosity (expressed as the number of particles in the beam per unit area per unit time).

The LHC is not a standalone device. It needs an injection chain consisting of smaller accelerators, which deliver a beam of particles with the required energy, luminosity and spatial structure. A simplified diagram of the CERN accelerator complex is depicted in Figure 1.1.

The LHC can collide either protons or heavy ions, but for the purpose of this introduction, only the proton system will be discussed. The proton beam originates from a small gas canister, containing pure hydrogen. Hydrogen atoms are supplied to a proton generator which strips off the electrons in a high voltage electric field. The protons are accelerated to an energy of 50 MeV in Linac 2. Then, the beam enters the Proton Synchrotron Booster (PSB) where its energy is increased to 1.4 GeV. The PSB is made of four superimposed rings whose aggregate circumference is equal to the circumference of he next machine (Proton Synchrotron – PS). The beam is injected sequentially into separate rings and ejected into a single PS ring but with increased bunch luminosity. The PS output energy reaches 32 GeV. The last machine in the injector chain, the 7 km-long Super Proton Synchrotron (SPS), provides two 450 GeV beams which are fed in opposite directions to the LHC.

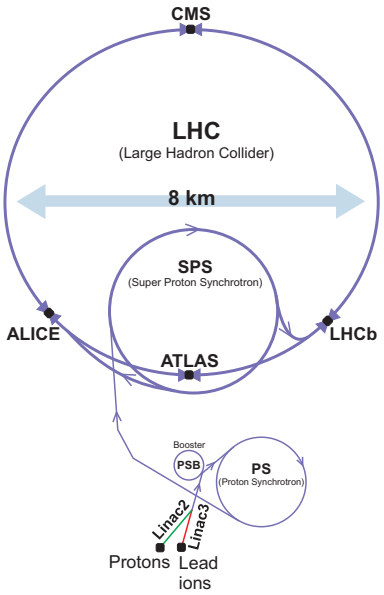

Figure 1.1. Simplified schematic of LHC accelerator complex (not to scale)

The 27 km-long LHC tunnel is split into 8 segments, with a length of 3.5 km each. Between the segments, there are 8 underground caverns. Half of them host the collider equipment:

- point 4 contains the superconducting RF cavities. This is the only place in the LHC where the beams are accelerated.
- point 6 is a beam dump station, where the beams can be safely ejected from the LHC and their energy dissipated in a graphite target.
- points 3 and 7 host the beam collimators – a series of magnets forming the beam cross-section into the required shape.

In the remaining 4 caverns, called ”interaction points” where the counter-running beams collide, there are 6 particle detectors.

- ATLAS (A Toroidal LHC Apparatus) and CMS (Compact Muon Solenoid) are general-purpose experiments, capable of detecting a wide range of particles produced in LHC collisions. The data from these detectors is used to search for new phenomena – the Higgs boson, supersymmetry and extra dimensions
- ALICE (A Large Ion Collider Experiment) is a specialized heavy ion detector, where heavy ions collisions at extreme energy densities will be analysed. It is expected that a new phase of matter, called the quark-gluon plasma will form, the existence and properties of which are key issues in quantum chromodynamics.
- LHCb (LHC Beauty Experiment) is used to study the asymmetry between matter and antimatter (called CP violation) present in iteractions with B-particles (particles which contain the bottom or “beauty” quark).
- TOTEM and LHCf - two smaller detectors. TOTEM is dedicated to precise measurement of proton-proton interaction cross-section and the deep study of proton

structure. LHCf uses forward particles created in LHC collisions to simulate and study cosmic rays in laboratory conditions.

All the machines in the complex are able to interoperate thanks to the sharing of a common and highly precise notion of time. Providing this *timing service* in a reliable and accurate fashion is the role of timing systems, of which one—White Rabbit—is the subject of this thesis.

1.2. Scope

Timing systems are specialized networks which provide a common notion of time in a distributed environment. In other words, they ensure that all clocks in the system are ticking at the same speed and showing identical time at every instant.

Accurate time synchronization is a major requirement in many real-time applications. Distributed control systems, factory automation and data acquisition systems all require the execution of operations with very tight time constraints. This becomes particularly difficult in large-scale systems such as CERN's accelerator complex, where distances between nodes are in the range of tens of kilometers, causing long and often unpredictable transmission delays.

Most of the available off-the-shelf commercial timing solutions do not meet the requirements of large complexes such as CERN. They are not easily scalable to support thousands of nodes and cannot be used over large distances as their real-time performance level would be severely reduced. Therefore almost all timing networks in the accelerator world (even those commercially available) are custom-made.

Despite their cutting-edge performance, custom-made timing systems still suffer from various issues:

- Lack of compatibility with other control systems and conformance to standards.
- Scalability problems.
- Architectural limitations. For example, the General Machine Timing system at CERN is unidirectional; a slave cannot send any data using the timing network.
- Usage of application-specific hardware which has to be supported "in house" for the entire lifetime of the control system.

The White Rabbit project began as a potential successor for several legacy control and timing systems, such as the GMT which is currently in operation at CERN. Afterwards, it was also chosen as the timing system for new accelerators at GSI in Darmstadt, Germany, and it is currently being considered by other institutes and companies. The main goals of the project are to provide:

- A deterministic, large-scale monitoring and control network based on IEEE802.3 Synchronous Gigabit Ethernet, supporting more than 1000 nodes spread over tens of kilometers, all synchronized to sub-nanosecond level.

- A scalable and modular platform that does not require specialized configuration and maintenance.
- A mechanism for robust delivery of high priority messages with low and limited transport delay under fault conditions.
- A fully open design, not tied to any particular hardware or software vendor.

In summary, White Rabbit (WR) aims to combine the accuracy of dedicated timing systems with the real-time performance of industrial networks and flexibility and scalability of Ethernet, as depicted in Figure 1.2 .

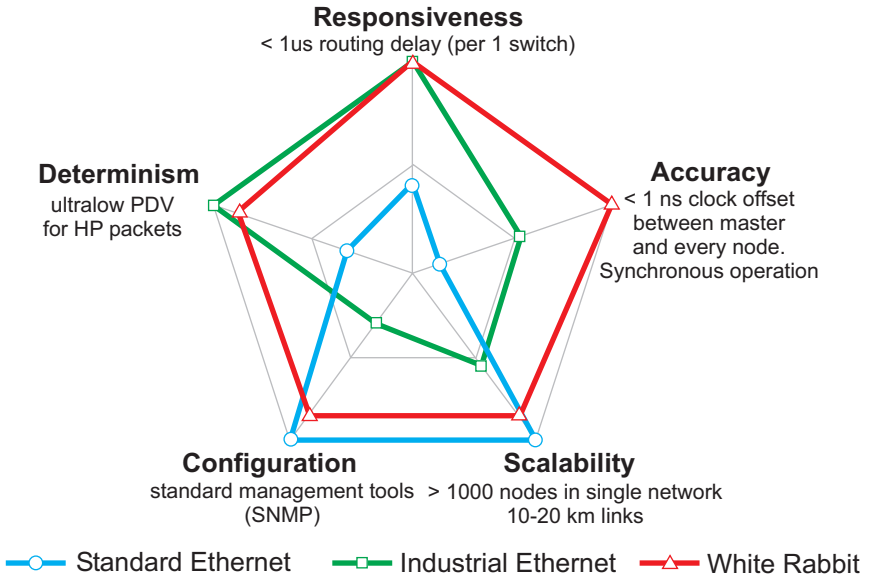

Figure 1.2. Standard and Industrial Ethernet versus White Rabbit

The following chapters provide a detailed description of the clock synchronization method employed in White Rabbit: Synchronous Ethernet for the physical layer and link latency compensation using a Digital Dual Mixer Time Difference (DDMTD) phase detector and the IEEE 1588 Precision Time Protocol. Also the hardware, FPGA and software design of the core element of the White Rabbit network, the WR Ethernet switch, are presented. Finally, preliminary clock quality measurements and possible applications of the WR timing system are discussed.

2. Theoretical background

The subject presented in this thesis requires the reader to have basic theoretical knowledge about oscillators, time references, phase noise, jitter and phase-locked loops. This chapter presents the minimal theoretical background necessary to understand the problems and solutions discussed in the thesis.

2.1. Time and frequency references

Every timing system needs a primary time source which serves as a reference for all the nodes. In most systems, the reference clock must fulfill rigorous requirements for the long-term stability, so it can be traceable to a common time standard, such as UTC (Universal Coordinated Time) or TAI (Temps Atomique International). Therefore, atomic clocks or Global Positioning System (GPS) disciplined oscillators are commonly chosen as primary clock sources.

2.1.1. Atomic clocks

Atomic clocks count time by measuring the characteristic resonance frequencies of atoms excited by lasers or microwave RF signals [22] and are among the most stable time references available. The two most widespread atomic clocks (although not the most stable) are cesium and rubidium clocks. Figure 2.1 shows the simplified structure of a typical cesium clock.

Cesium clocks operate by locking the frequency of a local oscillator to the resonance frequency of the transition between hyperfine states $|4,0\rangle$ and $|3,0\rangle$ in a cesium-133 atom.

A sample of cesium is heated in an oven to a gaseous state. The atoms, heated to high temperature, can exit the oven through a small cavity, forming a beam. The beam then passes through an assembly of electromagnets called "state selector", where it is split by magnetic field into multiple beams depending on the spin energy of the atoms. Only the atoms which are in the low energy state will follow a trajectory leading to exit the state selector. Atoms having other states are absorbed by the getter.

Selected cesium atoms pass through a U-shaped Ramsey cavity in which they are exposed to microwave radiation with a frequency of about 9.19 GHz, causing transitions to the higher energy state. Next to the cavity, in another set of magnets, only the atoms

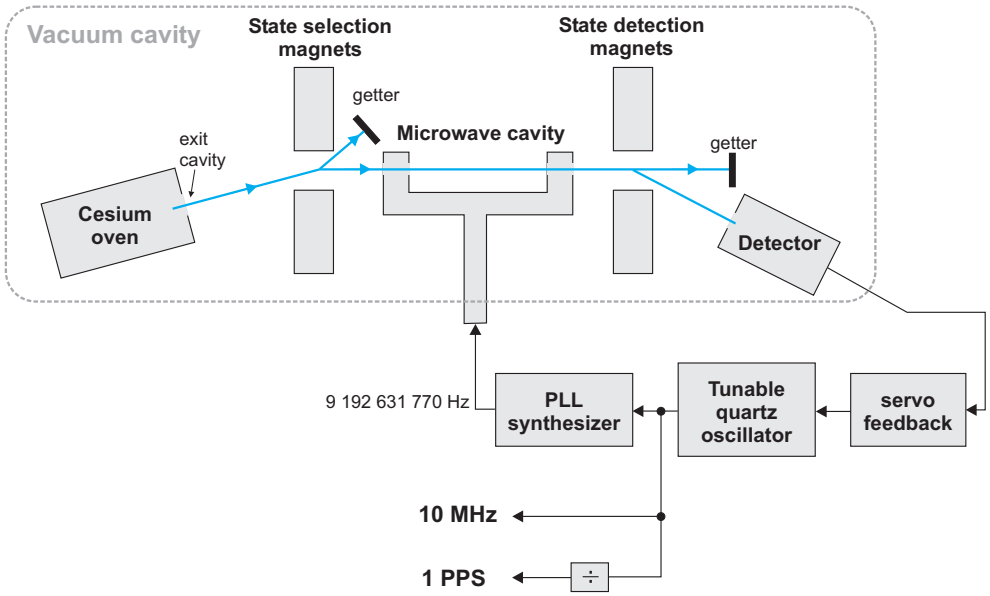

Figure 2.1. Structure of cesium beam atomic clock

which have undergone the required state transition are directed to an electron multiplier detector.

The signal from the detector is fed to the servo controller which maintains such frequency of the tunable quartz oscillator, at which the detector output is at the maximum level. In such conditions, the maximum number of cesium atoms get into the resonance state and the quartz oscillator becomes locked to the discrete frequency of the atomic resonance. For Cesium-133, this frequency is exactly 9,192,631,770 Hz which is also the definition of the second unit in the SI unit system.

Usually, the time information is outputted as a 10 MHz high-speed clock, PPS (one Pulse Per Second) signal and a UTC time code outputted using a serial port. Modern atomic clocks fit in an enclosure of the size of a suitcase.

2.1.2. GPS-synchronized clocks

Another way of obtaining accurate time, traceable to a primary atomic clock is to use a GPS time receiver. In fact, all GPS navigation receivers are also timing receivers, as the synchronization is a prerequisite for estimating the geographical position of the receiver.

The GPS system incorporates 24 to 32 satellites in the medium earth orbit and a few control and monitoring stations on the surface of the Earth. Each of the satellites is continuously transmitting two carriers, as depicted in Figure 2.2: L1 at 1575.42 MHz and L2 at 1227.6 MHz. The carriers are modulated with pseudo-random code sequences (Gold codes): the coarse/acquisition (C/A) code, available for everyone and the precise (P) encrypted code, reserved for military users. As all the satellites use the same carrier frequencies, the codes transmitted by the satellites are mutually orthogonal to prevent

interference. Also, a slow (50 bits/second) navigation message is added to both carriers, containing satellite orbit data (ephemeris), error corrections and other data necessary for the receiver to properly determine its position and time.

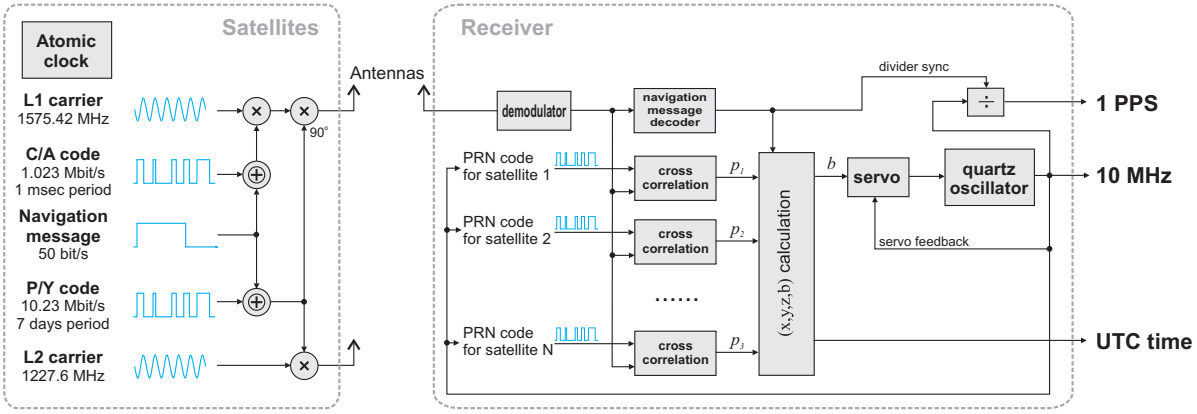

Figure 2.2. Synchronization using GPS

Signals generated by each satellite are locked to its local primary clock (two cesium and two rubidium atomic clocks in a redundant configuration in each of the satellites). The satellites' clocks are synchronized and corrected by the terrestrial control stations. The C/A code sequence repeats every 1 millisecond and its beginning is aligned with the millisecond transition of the reference clock. The receiver knows the code sequences sent by each satellite and attempts to determine an offset p_i for each of the locally generated PRS codes at which the cross-correlation with the received code sequence gives the maximum value. These offsets are used to solve the system of navigation equations 2.1:

$$p_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i{}^2)} - bc \quad \text{for } i \in 1,2,3,4 \tag{2.1}$$

where p_i are the correlation offsets (called *pseudoranges* in GPS terminology), (x, y, z) is the position of the receiver, (x_i, y_i, z_i) are the positions of the satellites, c is the speed of light and b is the clock offset, the value we are looking for. Note that as there are 4 unknowns in 2.1, the visibility of at least 4 satellites is required for the synchronization.

Time receivers ignore the position information and feed the value of offset b into a servo adjusting the rate and offset of the local tunable quartz oscillator. The quartz output signal is used to generate the local PRS codes and also output as a reference frequency. The PPS signal is generated by dividing the 10 MHz clock, where the offset of PPS pulses and UTC time are determined by the navigation message decoder (transmission of each navigation subframe begins at round seconds).

2.2. The imperfect clock signal

The goal of a timing system is to provide a common notion of time in a distributed environment. This notion of time is usually the result of counting ticks of a clock signal from an arbitrary instant. The clock signal is ideally of perfect periodicity and stability. Real-world clocks, however, present imperfections in both amplitude and phase as expressed in eq. 2.2.

$$x(t) = A(1 + m(t))\sin(\omega t + \varphi(t)) \tag{2.2}$$

In case of digital systems, such as the White Rabbit, most of clock signals are output by digital gates with hard amplitude limiters. These square signals do not suffer from amplitude modulation, so the $m(t)$ term can be ignored from now on. The random variations in the zero-crossing of the pseudo-periodic signals arise from the $\varphi(t)$ term, usually called phase noise. Ignoring amplitude modulation, eq. 2.2 can be re-written as

$$x(t) = A\sin\left(\omega\left(t + \frac{\varphi(t)}{\omega}\right)\right) \tag{2.3}$$

showing that the $\frac{\varphi(t)}{\omega}$ term, which has dimension of time, represents the time deviations in zero-crossing between the perfect and the imperfect periodic waveforms. $\varphi(t)$ is a random signal whose rms value is in principle a good indicator of clock quality. Dividing that rms value by ω gives the clock jitter.

Why is phase noise so important? Because this imperfect clock is typically distributed to many receivers, where local counting is done and the common notion of time is generated. In order to compensate for delays in cables and fibers, a constant correction is applied to the local time base, but this assumes the clock is a perfect copy of itself T seconds ago, where T is any multiple of the clock period. If this is not the case, as in all real-life clocks, the delay compensation mechanism does not fully achieve its goal.

2.3. Phase noise and jitter

Unfortunately, all clocks ultimately diverge in phase and even frequency, in such a way that the rms calculation of jitter gets bigger and bigger as the averaging time grows. In order to tackle this problem, it is useful to work in the frequency domain. The Fourier transform of $\varphi(t)$, noted $\Phi(f)$ has the same energy as the time-domain signal. This result, expressed mathematically in eq. 2.4, is known as Parseval's theorem [1]:

$$\int_{-\infty}^{+\infty} |\varphi(t)|^2 \, dt = \int_{-\infty}^{+\infty} |\Phi(f)|^2 \, df \tag{2.4}$$

The units of the left-hand side of eq. 2.4 are $rad^2 \cdot s$. A real-life signal would be bounded in time, and hence the integration limits would be finite. Dividing the result

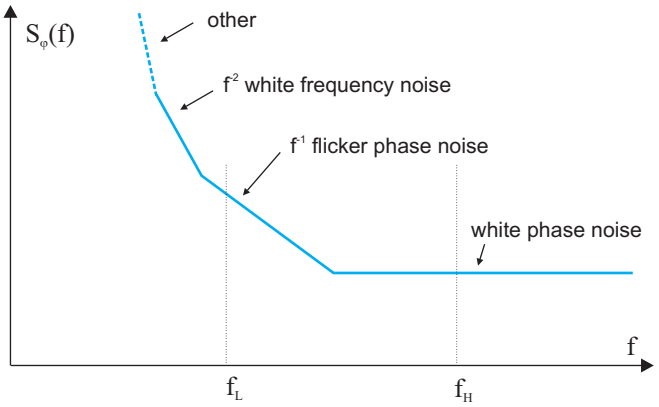

Figure 2.3. One-sided Power Spectral Distribution of phase noise for a typical oscillator.

of the integral by the time span of the signal would give an average of $|\varphi(t)|^2$, and taking the square root would give the phase noise rms value. Dividing that rms value by the nominal frequency results in jitter. Unfortunately, increasing the integration limits will cause infinite growth of measured jitter. This is also true for the integral in the right-hand side, which can be interpreted as that of Power Spectral Density (PSD) of phase noise [26] over all frequencies.

In reality, however, a particular application is only sensitive to jitter generated between two certain limits in the PSD curve. Figure 2.3 shows a typical plot of one-sided PSD ($S_\varphi(f)$) of the phase noise of an oscillator. Integration limits are set between f_L and f_H. Phase noise below f_L corresponds to variations which are so slow as to be common mode for all timing receivers under all circumstances. For example, if accelerators at CERN change beam every 1.2 seconds, phase noise below say 1 mHz will give an almost constant contribution during the 1.2-second span and therefore will not affect the performance of the timing system. Reasons for establishing an upper limit in integration come mainly from the inability of some systems to react to such fast variations, due to limitations in bandwidth. These limitations can be in electronics, such as the bandwidth of the input stage of a digital gate, or in electro-mechanical systems such as an RF accelerating cavity. It is important to justify lower and upper integration limits for a given application based on both requirements and detailed knowledge of the system.

Figure 2.3 also illustrates different types of noise, which can be identified by the different slopes of their PSDs in a double logarithmic-scale graph [3]. White phase noise dominates the high frequency area and has a flat distribution. Moving towards lower frequencies, one can find flicker (pink) phase noise, which is characterized by a PSD scaling with f^{-1}. Since frequency is the derivative of phase, white frequency noise – arising from white noise in the frequency-setting elements of an oscillator – features an f^{-2} slope in the phase noise PSD diagram. Higher order f^{-n} terms can also be present. This low-frequency area of the graph will feature quick divergence under integration, and

corresponds to the problematic long time-spans mentioned earlier for the time-domain representation.

2.3.1. Sampled phase noise considerations

As the the phase noise signal can be digitally recorded, it becomes desirable to calculate overall jitter using a computer program. Given a sampled version of $\varphi(t)$, called $\varphi[n]$, and its Discrete Fourier Transform (DFT), $\Phi[k]$, Parseval's theorem becomes [2,21]

$$\sum_{n=0}^{N-1} |\varphi[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |\Phi[k]|^2 \tag{2.5}$$

where N is the number of acquired samples. Dividing eq. 2.5 by N and taking the square root will give an estimate of rms phase noise.

2.3.2. Relationship with oscilloscope jitter measurements

In many cases, the parameter of interest is the time-domain first-difference jitter between a certain rising edge of a clock signal and another rising edge k clock periods later. This is typically measured with an oscilloscope triggered at the first edge of interest, observing on the screen k ticks later in infinite persistence mode. Oscilloscope time measurements, however, are inherently less precise than frequency-domain measurements of phase noise as described in 2.4. Fortunately, one can take the first-difference jitter problem and cast it into phase noise language so that first-difference jitter information can be extracted from phase noise measurements in the frequency domain [8]. Let's first define a discrete time deviation signal which measures the deviation between the zero-crossing of a real oscillating signal and that of an ideal one. From eq. 2.3, discretizing, we have

$$x[n] = \frac{\varphi[n]}{\omega} \tag{2.6}$$

where x is now a time deviation and ω is the nominal frequency of the oscillating signal (which is also the sampling frequency of this discrete-time system). The oscilloscope measurement is equivalent to applying a filter to the sequence $x[n]$ and measuring the rms value at the output of the filter. This filter implements the difference equation

$$y[n] = x[n] - x[n-k] \tag{2.7}$$

whose transfer function in the Z domain is:

$$H_k(z) = 1 - z^{-k} \tag{2.8}$$

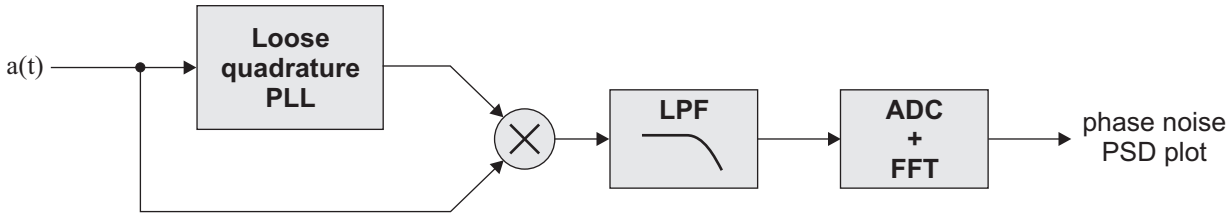

Figure 2.4. Internal structure of a typical phase measurement system.

The transfer function in the frequency domain can be obtained by evaluating the Z transfer function in the unit circle:

$$H_k\left(e^{j\omega_d}\right) = (1-\cos\omega_d k) + j\sin\omega_d k \tag{2.9}$$

where ω_d is now a digital frequency constrained between $-\pi$ and π. This frequency can be converted to the analogue frequency f_a domain using

$$\omega_d = 2\pi\frac{f_a}{f_s} \tag{2.10}$$

where f_s is the sampling frequency. The filter response in the f_a domain will of course be periodic with period f_s as for any digital filter. One possible method to calculate first difference jitter from phase noise PSD data is to filter that data with $\left|H_k\left(e^{j\omega_d}\right)\right|^2$ and then calculate the rms value of the output using the resulting scaled PSD. One remarkable feature of the first difference filter is that it will attenuate phase noise at frequencies close to its zeroes, and in particular very low frequency phase noise will not show up in the result. This is a confirmation of the fact that in many applications, this type of noise is irrelevant because it shows up as common mode in situations where only the differences matter.

2.4. Measuring phase noise

For low-noise conditions, we can re-write eq. 2.2 as

$$x(t) = A(1+m(t))\sin\omega t + A\varphi(t)\cos\omega t \tag{2.11}$$

where second-order terms are dropped. The amplitude and phase noise contributions are now modulating carriers in quadrature, so it is simple to design a device that will discriminate between them for measurement purposes. Figure 2.4 depicts the internal structure of such a device.

The role of the loose quadrature phase-locked loop (PLL) is to track the incoming signal with very low bandwidth, filtering all the high-frequency phase noise we are interested in studying. More on this filtering action will be explained in section 2.5. Phase

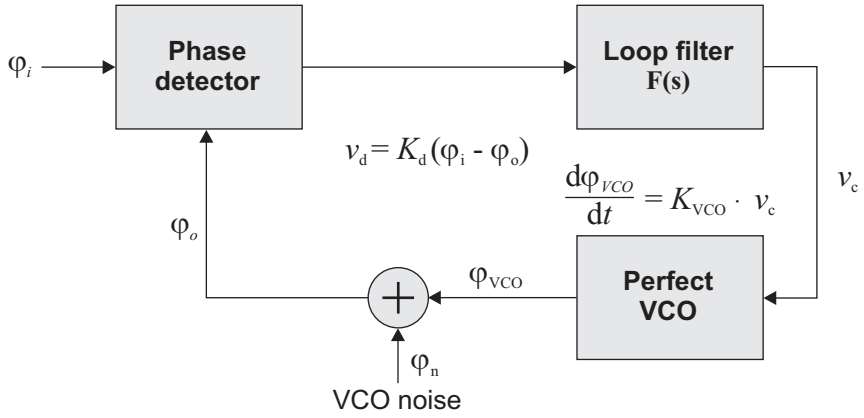

Figure 2.5. Block diagram of a phase-locked loop.

noise of lower frequency than the bandwidth of the loose PLL cannot be measured with this device. In addition, the PLL has to be set up to generate a signal in quadrature with $x(t)$ i.e. $\cos\omega t$ and not $\sin\omega t$ so that at the output of the mixer and low-pass filter we are left with a baseband signal proportional to $\varphi(t)$. Digitization of this signal followed by Fourier analysis results in a plot of phase noise PSD vs. frequency.

For the cases where very low-noise sources have to be characterized, the noise of the loose PLL can represent a significant portion of the overall measured noise, therefore affecting the precision of the measurement. In order to compensate for this, more advanced signal source analyzers feed the signal to two parallel PLL + mixer + low-pass + ADC branches and perform a cross-correlation of measurements. Noise which is uncorrelated between the two branches is attributed to the PLLs and other imperfections and discarded before displaying the PSD.

One last important item to bear in mind when measuring phase noise with commercial instruments is that measurements often get reported as $\mathcal{L}(f)$ which is defined as $\frac{S_\varphi(f)}{2}$. This is recommended in IEEE standard 1139.

2.5. Phase-locked loops

Phase-locked loops [12] are an invaluable tool in cleaning up the jitter of clocks, among many other possible applications. Figure 2.5 depicts the structure of a typical PLL.

The phase detector (PD) block generates an output voltage v_d proportional to the phase difference between the input and output of the PLL. In Laplace space, its output is therefore

$$V_d(s) = K_d(\Phi_i(s) - \Phi_o(s)) \tag{2.12}$$

The next block after the phase detector is the loop filter, which outputs the control signal for the Voltage-Controlled Oscillator (VCO):

$$V_c(s) = F(s) \cdot V_d(s) \tag{2.13}$$

The VCO outputs a signal with a frequency proportional to its input voltage. Since frequency is the derivative of phase, this means that the phase of the signal at the output of the VCO is proportional to the VCO control voltage:

$$\Phi_{VCO}(s) = \frac{K_{VCO} \cdot V_c(s)}{s} \tag{2.14}$$

Since there are no perfect VCOs, the diagram includes a VCO noise source, contributing phase φ_n. Calculating the output phase φ_o from the two sources in the diagram (reference input phase φ_i and VCO phase noise φ_n) again in Laplace space gives

$$\Phi_o(s) = H(s) \cdot \Phi_i(s) + E(s) \cdot \Phi_n(s) \tag{2.15}$$

where $H(s)$ is called the system transfer function, defined as

$$H(s) = \frac{K_{VCO} K_d F(s)}{s + K_{VCO} K_d F(s)} \tag{2.16}$$

and $E(s)$ is the so called error transfer function, defined as

$$E(s) = 1 - H(s) = \frac{s}{s + K_{VCO} K_d F(s)} \tag{2.17}$$

In typical clock-cleaning applications, $H(s)$ is a low-pass filter, while $E(s)$ is high-pass. Cut-off frequencies are dictated by PLL parameters, and most importantly the loop filter $F(s)$. The PSD of the phase noise of φ_i will be filtered by $|H(s)|^2$ while the phase noise PSD of the VCO will be filtered by $|E(s)|^2$. This means that the low frequency noise in the PSD of φ_o will come from the reference φ_i and the high-frequency noise will come from φ_n. The transition from one noise source to the other will be at a frequency determined by the loop parameters. After careful study of the PSDs of φ_i and φ_n it is the task of the designer to choose a cut-off frequency that will minimize overall area under the φ_o PSD curve, and consequently time-domain jitter. In typical systems – like the transmission of a very stable clock over a channel which adds high-frequency noise – the VCO is worse than the reference at low frequencies and better at high frequencies. The point in frequency where the two PSD plots (reference and VCO) cross is in that case an optimum setting for PLL bandwidth, as shown in Figure 2.6.

In Figure 2.5 the phase detector is shown as perfect, with no noise added to it as for the VCO. In practice, phase detector noise is also a concern, but mathematically it

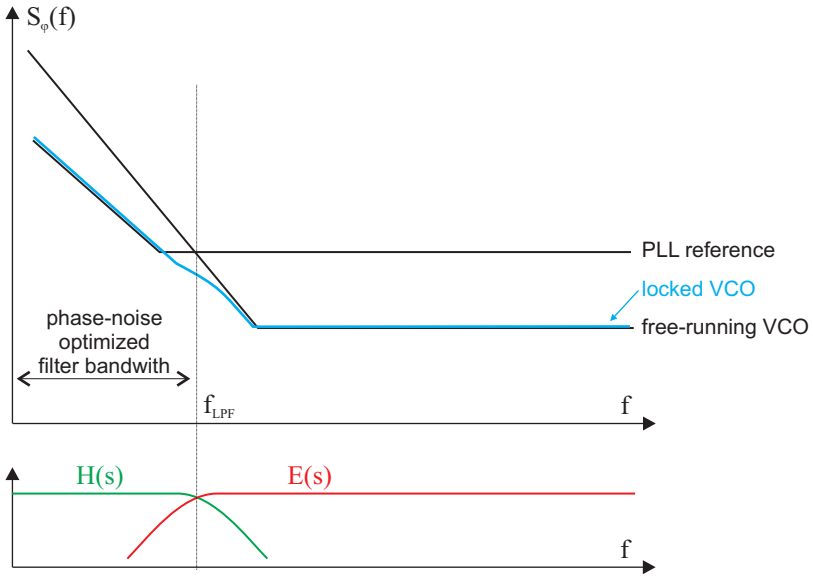

Figure 2.6. Optimal choice of PLL bandwidth for jitter-cleaning applications.

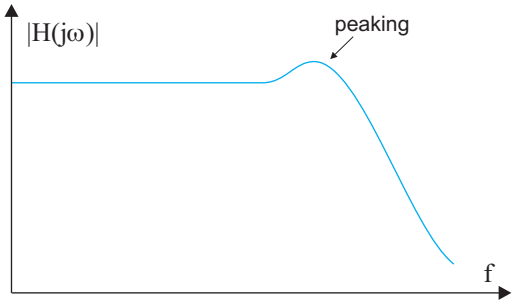

Figure 2.7. Illustration of transfer function peaking in a Bode plot.

is equivalent to reference noise, so that the above formalism can be applied, replacing reference noise by reference + phase detector noise.

One important aspect to bear in mind for applications in which many PLLs are cascaded is that of peaking in the $H(s)$ Bode plot. Peaking is a rise of the magnitude of $H(j\omega)$ before it starts its descent at the cut-off frequency (see Figure 2.7). Peaking is typical in under-damped systems [14]. If many PLLs with the same (or similar) $H(s)$ transfer function are cascaded, the peaking can be exacerbated and affect phase and gain margin, ultimately resulting in potential instabilities of the overall system. For this reason, in cascaded PLL applications, it is better to adjust the PLL parameters so that the resulting PLL is slightly over-damped.

3. Synchronization in White Rabbit

Synchronization in White Rabbit (further abbreviated as WR) results from the combination of IEEE1588-2008 (PTP) [15] with two further improvements: precise knowledge of the link delay model and clock syntonization over the physical layer. This chapter describes all the components involved in the synchronization process and the practical synchronization model used in WR.

3.1. The clock synchronization problem

Before starting the discussion of the WR synchronization model, let's introduce the mathematical principle of synchronization of two spatially separated clocks. Diagram 3.1 shows a simple, linear model of the clock synchronization problem: There are two clocks:

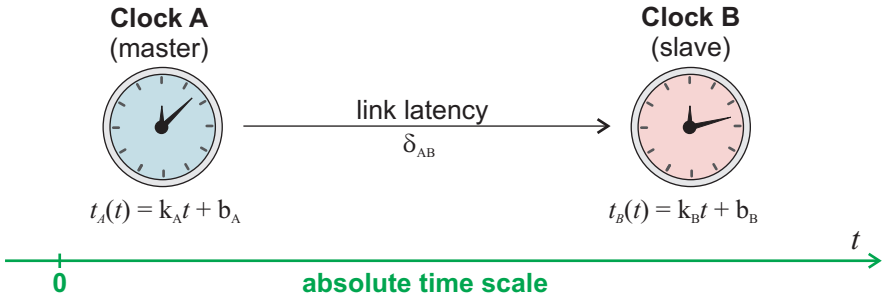

Figure 3.1. The clock synchronization problem.

the master clock which serves as a time reference, and the slave clock which has to be synchronized with the master clock. The clocks can communicate with each other through a data link with transmission latency δ_{AB}. The clocks operate with speed k and offset b with respect to an ideal, absolute time scale, hence an ideal clock would have $k = 1$ and $b = 0$. In order to synchronize the clocks, the following two steps must be performed:

- the rate of the slave clock k_B has to be same as the rate of the master clock k_A. This process is called **syntonization**. In electronics parlance syntonized oscillators run at the same frequency, but may have different phases. In WR, syntonization is provided by the physical layer of the network, thanks to Synchronous Ethernet (see section 3.4).

- the slave offset b_B must be equalized to the master offset b_A in a process called **offset adjustment**. WR adjusts clock offsets by using time-stamped packets, Precision Time Protocol (IEEE 1588) and DMTD phase tracking (see sections 3.3, 3.5.4).

It should be noticed, that the values of k_A, k_B, b_A, b_B are not known by any of the clocks at any moment. These coefficients only express the relation of the time shown by a particular clock to an ideal time scale.

Once the clocks are syntonized, the master can send its actual time value t_A to the slave. The slave will receive this information δ_{AB} later, which is the one-way (master to slave) latency introduced by the link connecting the two clocks. Knowing the values of t_A and δ_{AB}, the slave can calculate the clock offset Δ using the formula 3.1:

$$\Delta = t_A - t_B + \delta_{AB} \tag{3.1}$$

where Δ expresses how much the slave clock should add to its time counter to become synchronized with the master.

This model, however, is not sufficient for a real-world clock synchronization system, which has to deal with a variety of problems discussed further in this chapter. These are: varying link latency (3.3, 3.5.4), asymmetry of link delay (3.5.6), properties of data encoding (3.2, noise and hardware limitations (3.5.5).

3.2. Physical network layer

The accuracy of synchronization in a WR network relies deeply on the properties of the physical medium and the low-level data encoding. Therefore, this section will explain the principles of operation of Gigabit Ethernet physical and data link layers.

3.2.1. IEEE 802.3 optical gigabit Ethernet

WR can operate in virtually any synchronous Ethernet-based network, but the best (sub-nanosecond) synchronization accuracy is currently achievable only with gigabit optical links. Physical layer based on copper (100BaseT, 1000BaseT) can also be used in segments of the network with less stringent accuracy requirements (30 ns). The high accuracy implementation presented in this thesis uses the 1000Base-LX profile of IEEE 802.3 (802.3z) [16] optical Gigabit Ethernet operating in full-duplex mode as the transmission medium between nodes and switches. Figure 3.2 shows a typical configuration of a WR optical link. The medium is standard 9/125 μm single-mode fiber optic cable, up to 20 km in length (depending on the output power and sensitivity of the optical transceivers used). Both sides use the same fiber for transmission and reception and employ wavelength division multiplexing (WDM), so the link can operate in full duplex

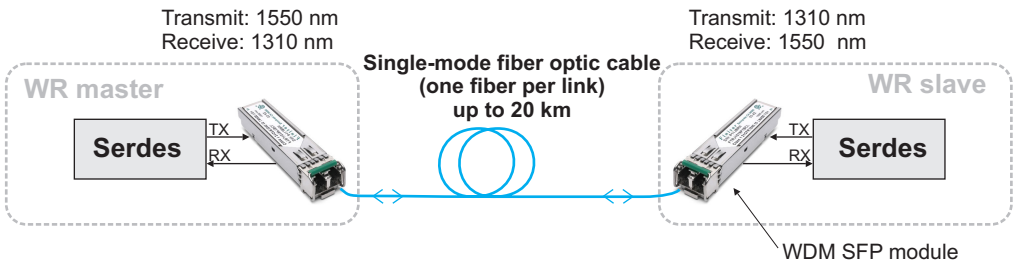

Figure 3.2. Physical link between WR nodes.

mode. The master transmits on 1550 nm wavelength, and the slave on 1310 nm. WDM operation has been chosen for the following reasons:

- deterministic medium asymmetry – the master-to-slave and slave-to-master paths have identical lengths. The asymmetry comes only from the difference of refraction factors for different wavelengths (see 3.5.6).
- price – cables and connectors make the biggest contribution to the price of an optical data link. Using WDM transceivers therefore allows for a 50% reduction of the cabling costs.

3.2.2. Low-level 1000Base-X 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 process is illustrated in Figure 3.3.

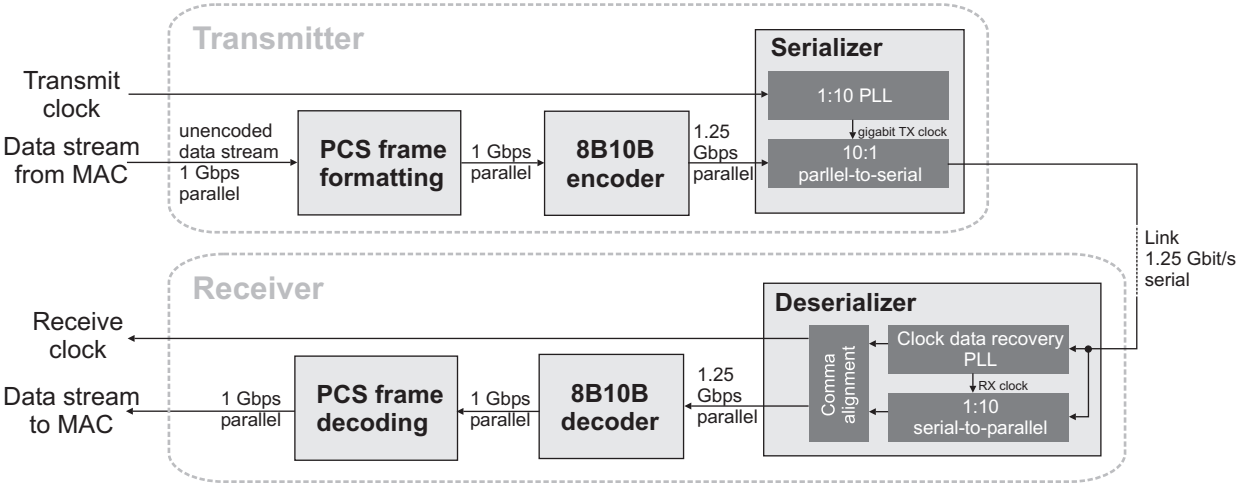

Figure 3.3. Encoding and decoding of 1000Base-X serial data stream.

Physical Coding Sublayer (PCS) encapsulation

The first encoding step is the encapsulation of Ethernet frames in the PCS, depicted in Figure 3.4. The process extends plain Ethernet frames with preambles and start/end of packet control sequences and generates a special idle pattern when no data is being

transmitted. The role of this idle pattern is to always keep the link active, which is necessary to maintain correct operation of the optoelectronics and clock recovery logic in the receiver. Frame payloads are encoded using 256 data symbols, and the control sequences utilize the special control symbols listed in table 3.1 The /V/ ordered set,

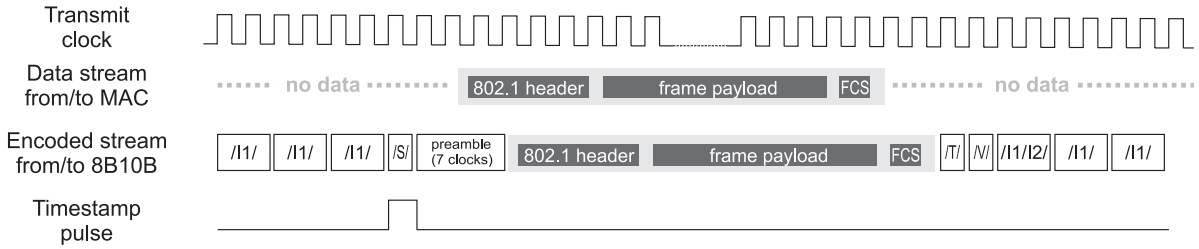

Figure 3.4. Frame encapsulation in the Physical Coding Sublayer (PCS)

Table 3.1. 1000Base-X control characters

Symbol	8B10B ordered set	Description
/C1/	K28.5 / D12.5 / 16-bit config reg	Configuration 1
/C2/	K28.5 / D2.2 / 16-bit config reg	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 propagation

not shown in Figure 3.4, is used to inform the receiver that an error occurred during the transmission (for example an underrun of the TX buffer), and the received packet should be discarded. /C1/ and /C2/ are used to exchange the device capabilities during link autonegotiation. /R/ indicates a carrier extension event: during a burst of frames, the PCS fills the gaps between frames with /R/ symbols instead of /I/ sequences to keep the carrier active. This applies only to the half-duplex links using Carrier Sense Multiple Access (CSMA) mode which cannot be used in a WR network.

It's worth observing that the PCS encapsulation unit also generates a pulse coherent with the transmission of the /S/ ordered set, which is used to gather WR-compatible deterministic packet transmission timestamps.

8B10B encoding

The PCS output data stream is fed to the 8B10B encoder, whose role is to adapt the format of input symbols to match the requirements of the communication channel. Input data and control symbols are converted into 10-bit words having the following properties:

- the maximum bit run length is 5 bits (i.e. there are never more than 5 consecutive ones or zeros in the encoded stream). This is required by the clock recovery PLL in the receiver.
- the output data stream must not contain a DC component. This requirement is fulfilled by using an encoding scheme called "running disparity" to maintain equal number number of transmitted ones and zeros (see [7]).
- The /K28.5/ symbol which is the part of the /I/ and /C/ sequences is called a "comma" character. Its bit sequence (0011111010) can never occur in between two encoded 10-bit symbols. By detecting the comma sequence, the receiver can find the boundary between subsequent words on the serial link (see Figure 3.5b).

Serializer and deserializer

The serializer converts 10-bit parallel words coming from the 8B10B encoder into a 1.25 Gbit/s serial data stream which drives a differential (LVPECL, LVDS, CML) output connected to the fiber optic transmitter. Most serializers use a 10:1 shift register clocked with a 1.25 GHz clock derived by a built-in PLL from a 125 MHz reference, as shown in Figure 3.5a. The deserializer recovers the 1.25 GHz bit clock by using a

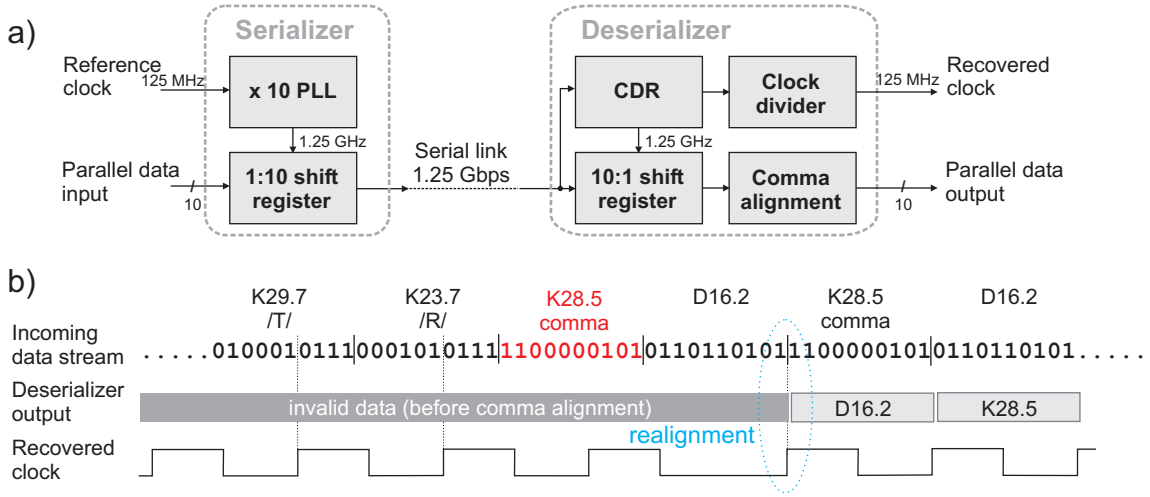

Figure 3.5. 1000Base-X serializer and deserializer (a) and comma alignment (b)

specialized PLL with phase detector capable of locking onto irregular signal transitions (for example a bang-bang phase detector [30]) or a digital oversampling clock and data recovery unit [27]. The recovered high speed clock drives the 10:1 ratio serial-to-parallel shift register, and after being divided by the serialization factor, it is output from the deserializer.

The comma alignment unit ensures that the shift register's parallel output matches the inter-symbol boundary on the serial input. Upon detection of a unique comma pattern (which cannot occur between two unaligned symbols), the data word is shifted

by the necessary amount of bits and the recovered 125 MHz clock is adjusted to match the inter-word boundary. The alignment process is illustrated in Figure 3.5b.

Note that the recovered clock must always track the serializer's reference clock. Eventual delay may be caused only by the varying link latency. Serdes chips which introduce clock delays which change during the operation are not suitable for use in a WR network.

8B10B and PCS frame decoder

The 8B10B decoder and PCS decoder reverse the operations done by their counterparts in the transmitter and provide raw Ethernet frames to the higher layers of the network stack. They also perform low-level error checking (detection of invalid 8B10B symbols and loss of communication). In addition, the receive path PCS outputs a pulse upon detection of the /S/ symbol which is used for timestamping received packets.

Packet timestamping

As mentioned in section 3.1, clock synchronization requires the knowledge of the link latency. In packet networks, such as Ethernet, the latency can be measured by analyzing the packets' send and receive timestamps.

In order to achieve the maximum possible measurement accuracy, timestamping in WR is done by the PCS sublayer, before the data crosses clock domains or is passed to blocks introducing varying latencies (e.g. FIFOs). The timestamps are produced by a free-running counter gated by the pulses coming from transmitter and receiver PCS units. The timestamping process is extensively described in section 3.5.3.

3.3. Precision Time Protocol (IEEE 1588)

The Precision Time Protocol (PTP) is an IEEE-standardized (IEEE 1588) high-precision time protocol designed for use in local area networks [15]. The main aim of PTP is providing a microsecond-level synchronization in Ethernet-based networks. Unlike dedicated synchronization systems (such as IRIG-B), PTP does not require a separate network infrastructure, allowing for mixing precise time information with data transfers. This approach allows for significant cost reduction (no extra cabling, standard hardware) with no compromises in performance. Unfortunately, Ethernet is very nondeterminisic in terms of latency and communications jitter, as the active elements of the network (switches, routers, network cards and finally the operating system's networking stack) all introduce random and unpredictable latencies as illustrated in Figure 3.6. PTP overcomes this issue by timestamping the packets used for synchronization at the physical layer of the network. Hardware-based timestamping minimizes the error of time measurement by performing it at the point from which the latency to the physical port is constant or at least possible to estimate with enough precision. PTP can achieve accuracies of a few microseconds in networks built using standard components

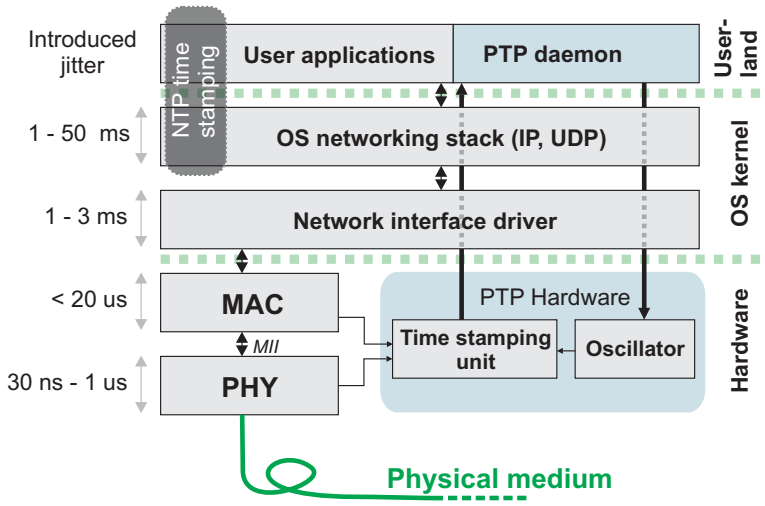

Figure 3.6. Timestamping in PTP and NTP

(switches, hubs), and better than 100 ns in networks where all active components are PTP-compliant.

Unlike the popular Network Time Protocol (NTP) which timestamps the packets in a purely software way, PTP doesn't impose any real-time constraints on the synchronization protocol software, as the priority of the synchronization application no longer affects the accuracy of timestamping. PTP daemons can therefore run as background processes on non-realtime operating systems (such as Linux or Windows) with no penalty on the synchronization accuracy.

The biggest field of PTP applications is in industrial control and automation systems. PTP support has been integrated into several Ethernet-based field buses such as Profinet, Powerlink and Ethernet/IP. There is an increasing demand for PTP-based synchronization from test and measurement, telecom and military users. PTP is also a base protocol for synchronization in WR.

3.3.1. Synchronization in PTP

The PTP standard defines a distributed master-slave synchronization hierarchy where the free running oscillator in the slave node is locked to the reference oscillator in the master node, as shown in Figure 3.7. Network nodes in PTP parlance are called "clocks". PTP defines 3 types of clocks:

- **ordinary clock (OC)** – a single-port device (for example, a network card supporting PTP) which can be either the source of synchronization (master) or a destination for synchronization reference (slave),
- **boundary clock (BC)** – a multi-port device (for example a switch) which bridges the synchronization from one network segment to another by using its local oscillator (locked to a remote master node) as a reference for the adjacent nodes. A BC can

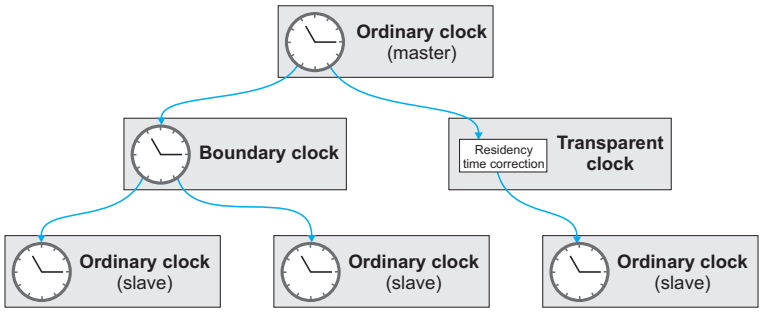

Figure 3.7. An example of PTP network

be seen as a number of ordinary clocks integrated into a single device and sharing a single local oscillator,

- **transparent clock (TC)** – a multi-port PTP node without a servo-controlled oscillator. PTP packets passing through a transparent clock are modified by the TC hardware to include the packet residency time of the TC (i.e. the difference between the departure and the arrival of the packet from/to the device). Due to the lack of a PTP-controlled oscillator, transparent clocks are not susceptible to the chained servo problem (see 2.5) while still maintaining satisfactory accuracy.

The roles of master and slave clocks are assigned dynamically, depending on the actual topology of the network and the quality of each clock's oscillator. If a node detects that all the clocks in the neighbourhood are worse than its local oscillator or there are no other clocks connected, it automatically becomes a master, otherwise it enters slave mode, synchronizing itself to the best available reference. The detailed master/slave selection scheme is defined in the standard as the Best Master Clock (BMC) algorithm [15]. As a result of the BMC, the entire network is synchronized to the best available source. In case of failure of the current master clock, its role will be automatically taken over by another clock in the network.

As mentioned before, synchronization relies on the measurements of the delay introduced by the link connecting the nodes and the frequency drift of the slave's oscillator. Both measurements are done by exchanging Ethernet packets, with the knowledge of accurate packet transmission and reception timestamps. There are several schemes of synchronization available in PTP, one of which is illustrated in Figure 3.8. PTP defines two kinds of messages: timestamped *event messages* which are used for offset and drift measurement (marked red in Figure 3.8) and *general messages*, employed by PTP nodes to identify other PTP nodes, establish clock hierarchy and exchange data, e.g. timestamps, settings or parameters. An *Announce Message* is periodically broadcast by the node being in the *Master* state, carrying information about the master node and the masters's clock source quality. Announce messages provide the necessary data for the Best Master Clock (BMC) algorithm.

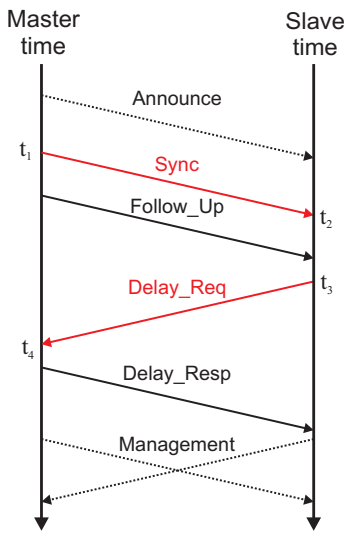

Figure 3.8. PTPv2 message exchange for a two-step End-to-End clock synchronization.

Timestamps ($t_1 ... t_4$) of the event messages (*Sync* and *Delay_Req*) are used to calculate the offset and the frequency drift between the master and the slave. As the slave needs to have knowledge of all 4 timestamp values in order to synchronize itself, the value of t_1 is either embedded into the *Sync* message directly in the hardware (One-step clock) or carried by a separate *Follow_Up* message (Two-step clock). Likewise, the value of t_3 is reported to the slave within the *Delay_Resp* message. Having the values of $t_1 ... t_4$, the slave node performs the following procedure:

- Determines the round-trip (master → slave → master) link delay 3.2

$$delay\_coarse = (t_4 - t_1) - (t_3 - t_2) \tag{3.2}$$

- Calculates the one-way (master → slave) delay 3.3:

$$one\_way\_delay = \frac{delay\_coarse}{2} \tag{3.3}$$

- Corrects the clock offset by subtracting $t_2 - t_1 + one\_way\_delay$ from its time counter
- Maintains syntonization and synchronization of its local oscillator by adjusting its frequency with a servo keeping the value of $t_2 - t_1 + one\_way\_delay$ equal to 0.

Sync messages are periodically broadcast to allow the slave to compensate for the frequency drift of its local oscillator. Clock offset can be measured less often, as (when the clocks are already syntonized), it depends only on the slow changes in the link delay, for example induced by the varying temperature. In typical applications the slight changes in the link delay are usually negligible. Therefore, once the offset is initially fixed, the synchronization can be kept using only the *Sync* messages broadcast by the master node, with no *Delay_Req* requests being sent by the slave.

In 2008, the standard was extended and version 2 (PTPv2, IEEE1588-2008) was published, however incompatible with the original standard. Among the major improvements were:

- Introduction of *management messages*, allowing for clock discovery and state control, and providing access to clock's internal data sets.
- Support for *Type-Length-Value* (TLV) fields for passing implementation-specific data associated with PTP messages.
- Possibility for encapsulating PTP packets within Layer 2 (raw Ethernet) frames, enabling devices such as Ethernet switches to use PTP without a need for an IP stack.

WR employs PTPv2 for measuring coarse (nanosecond level) link delays through the timestamped packet exchange mechanism. PTP also serves as versatile transport layer for all WR-specific calibration and syntonization messages which are encoded in the TLV fields and management messages. The current WR PTP implementation works only over Layer 2 (raw Ethernet), although Layer 3 (UDP) will probably also be supported for compatibility with older PTP slaves.

3.3.2. PTP performance and limitations

The timing performance of every PTP-based system can be expressed with the PSD of phase noise for the slave's recovered clock signal (expressing the syntonization performance) and the histogram of the master-slave clock offset (see Figure 3.9) – a measure of synchronization accuracy. PTP had been primarily designed for use in industrial

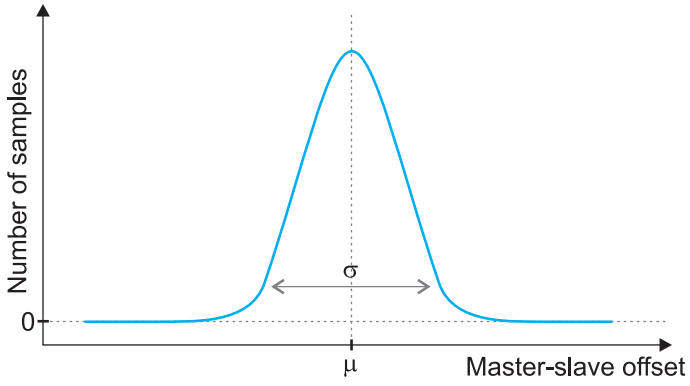

Figure 3.9. An example of clock offset histogram.

control systems, for which microsecond-level accuracy is satisfactory and there are no stringent requirements for the jitter. Therefore, achieving sub-nanosecond accuracy and good phase noise is extremely difficult with standard PTP implementations. The major limiting factors are the following:

- Syntonization based on tracking the difference of *Sync* message timestamps. As timestamps have limited granularity (for example, 8 ns for Gigabit Ethernet), the frequency tracking sensitivity is also limited to a single timestamp granularity unit.
- Recovering a low jitter clock requires either a very stable oscillator in every slave (such as an OCXO) or an increase in the servo bandwidth which also forces higher broadcast rate of *Sync* messages. A large number of PTP messages may interfere with the determinism requirements imposed by control system protocols.
- Being medium-independent, PTP does not provide any mechanisms for measuring the asymmetry of the link which is specific to a certain physical layer. Instead, the synchronization mechanism assumes that the link is perfectly symmetric or that its asymmetry is constant.
- The minimum achievable offset cannot be better than timestamping resolution.

3.4. Synchronous Ethernet

Ethernet is nowadays the most widely used network technology, both in local and metropolitan area networks. However, wide are networks, such as SDH/SONET used in telecommunication systems are based on time-division multiplexing (TDM) techniques. The original task of these networks was transporting voice signals, but over time they have been adapted to carry data streams, including Ethernet traffic. TDM networks are circuit switched – each node is assigned a transmission channel based on time, contrary to Ethernet where nodes are free to send data whenever they want.

TDM systems work synchronously by using a common reference frequency for the entire network to manage the time slots and clock the data transfers. On the other hand, Ethernet devices have independent free running oscillators, as depicted in Figure 3.10. Differences between data transmit rates of neighbouring nodes are compensated by asynchronous packet buffers. Synchronous Ethernet [18] (abbreviated as Sync-E) extends the standard with SDH/SONET-like syntonization and a TDM mechanism. Contrary to the peer-to-peer model of Ethernet, Sync-E imposes a hierarchical network structure with the top node called System Timing Master (STM). The STM contains the primary clock, such as an atomic clock or a GPS receiver and uses its output frequency to encode the outgoing data stream. Slave nodes use PLLs to recover the reference clock from the incoming data. The recovered clock is used to encode the data streams propagated to nodes being lower in the system hierarchy and the data stream sent back to the master node.

Sync-E is a part of WR responsible for syntonizing the clocks. Therefore, the role of the PTP protocol in a Sync-E network is reduced to measurement and compensation of the clock offset. Sync-E also facilitates sub-nanosecond synchronization by opening the door to phase detector technology as a means of measuring delay. Phase shift

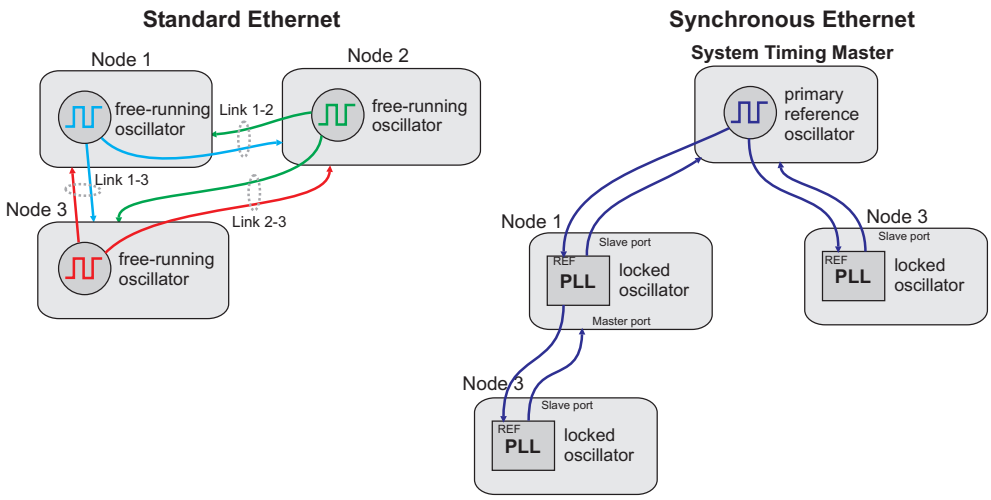

Figure 3.10. Comparison between standard and synchronous Ethernet

measurements are much easier to implement and deliver better precision than direct time measurements.

Since Sync-E uses cascaded phase-locked loops to propagate the reference frequency throughout the network, measures against the peaking effect (see 2.5) must be taken. In WR networks, the frequency responses of the PLLs are precisely controlled with digital filters. Also the network topology is restricted to a balanced tree with the Timing Master being the root node to minimize the PLL chain lengths.

3.5. White Rabbit synchronization model

A WR network is hierarchical timing-wise and flat data-wise (Figure 3.11). WR switches can have two types of ports: uplink ports, which receive the timing information from a remote node and downlink ports, propagating the timing to the nodes being lower in the network hierarchy. Due to the synchronous operation of the physical layer, the topology of a WR network must be a tree with the System Timing Master (STM) being the root of the tree. This restriction only applies to the links which provide the synchronization (marked red in Figure 3.11). Other links can still exist in the network, but they will be used only for data transfers and will be managed by the standard Rapid Spanning Tree (RSTP) protocol.

The root node (WR master) takes its clock from an external source (atomic clock, GPS) and uses it to encode the data streams going to its downlink ports. Each layer of the switches senses this data on its uplink ports and extracts their local clock from that data. The phase of the recovered clock is corrected using the precise link delay obtained by the PTP measurement and DMTD phase detector. The recovered clock is then passed to the downlink ports and recovered by the next layer of switches/nodes which propagate it further, and so on, and so forth.

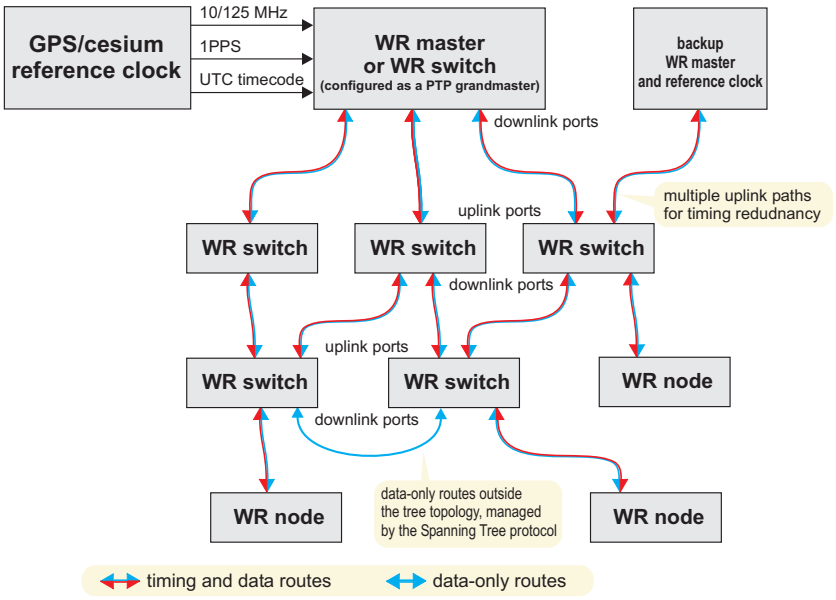

Figure 3.11. Topology of WR network

There may exist more than one STM in a single network, provided that all STMs use the same reference clock. In such situation, only the STM designated as the primary timing master is used, while the others serve as a backup. A node with multiple uplink ports, such as a WR switch, can therefore receive the timing independently on each uplink port and seamlessly switch between reference clocks in case of a failure of one of the synchronization sources.

Such synchronization scheme requires each WR switch to be a PTP boundary clock (although there are slight differences between PTP and WR boundary clocks, which do not affect interoperability between WR an PTP gear. See [11] for details). Since the synchronization takes place only between two adjacent nodes, the WR network can be seen as a set of independently synchronized point-to-point connections. Hence, the ability to synchronize a single link automatically provides an ability to synchronize the entire network.

3.5.1. WR link model

Knowledge of the physical model of links connecting the clocks is a prerequisite for achieving the required synchronization accuracy. The model of a WR optical link is depicted in Figure 3.12. *Delta* values shown in Figure 3.12 express the delays introduced by various elements of the system. Big deltas ($\Delta_{txm}, \Delta_{rxs}, \Delta_{txs}, \Delta_{rxm}$) indicate constant delays (e.g. the delays which never change when the link is active), while small deltas (δ_{ms}, δ_{sm}) express the delays which can vary during the operation of the link.

WR employs the clock loopback technique to measure the round-trip phase shift $phase_{MM}$, which is a starting point for determining the precise two-way delay $delay_{MM}$

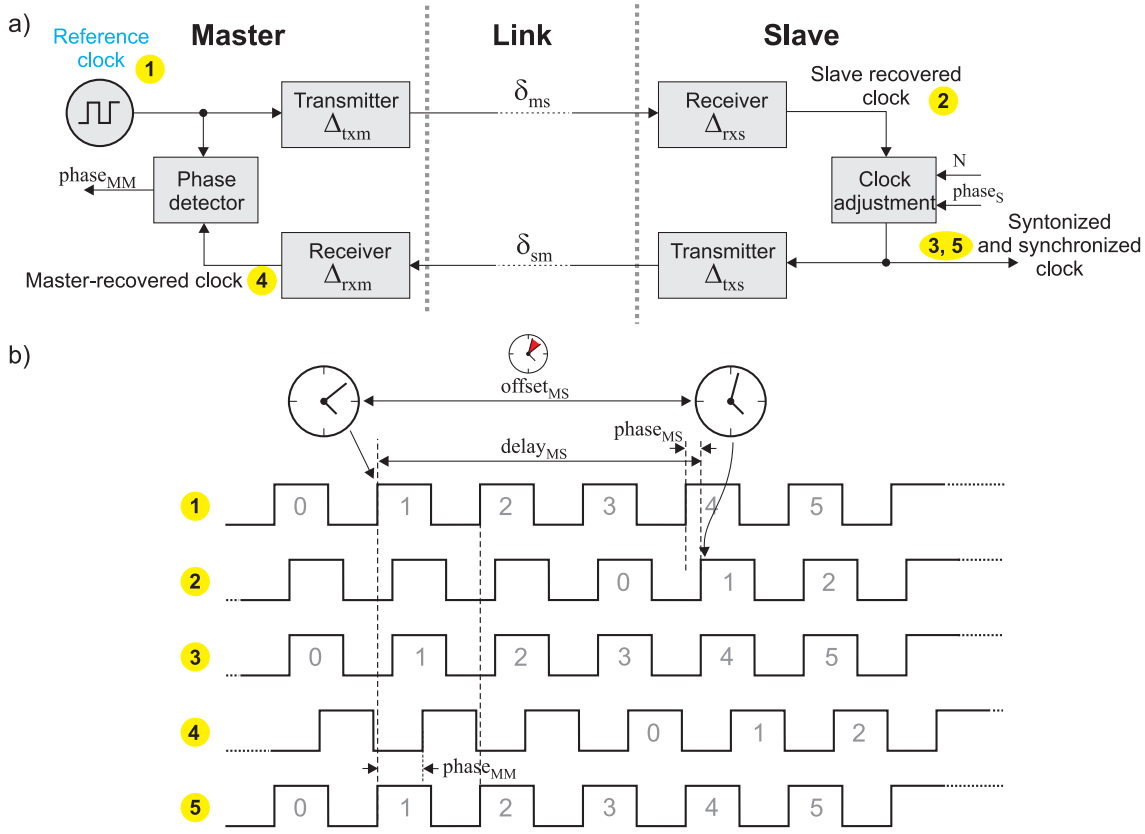

Figure 3.12. Model of a WR link (a) and relations between master and slave clocks (b)

and thus the clock offset $offset_{MS}$. The clock signal is transferred between the master and the slave according to the following scheme:

1. The reference clock (1) is used to encode the master's transmitter output and then extracted from the data stream at the slave's receiver. The recovered clock (2) is a copy of clock (1) delayed by $delay_{MS}$, where

$$delay_{MS} = \Delta_{txm} + \delta_{ms} + \Delta_{rxs} \tag{3.4}$$

   expresses the master-to-slave latency introduced by the transceivers and the link.
2. The recovered clock (2) is fed to a clock adjustment unit which shifts its phase by a programmable value $phase_S$ to obtain the phase-compensated clock (3) – the final result of WR synchronization.
3. Clock (3) encodes the slave's outgoing data stream and is recovered at the master side as signal (4).
4. The master measures the phase shift $phase_{MM}$ between its outgoing (1) and incoming (4) clocks using a phase detector.

The value of $phase_{MM}$ can be expressed as:

$$phase_{MM} = \{\Delta + \delta_{ms} + \delta_{sm} + phase_S\} \bmod T_{ref} \tag{3.5}$$

where $\Delta = \Delta_{txm} + \Delta_{rxs} + \Delta_{txs} + \Delta_{rxm}$ and T_{ref} is the period of Gigabit Ethernet 125 MHz reference clock (8 ns).

The goal of the presented model is to calculate the precise value of master-to-slave offset $offset_{MS}$ by combining a coarse timestamp-based round-trip delay 3.3 with precise phase measurement $phase_{MM}$. Once the offset is computed, the WR slave can phase-shift its recovered clock (deriving $phase_S$ from $offset_{MS}$) to match the phase of the master clock, completing the synchronization. Determining the precise offset, however, is not a

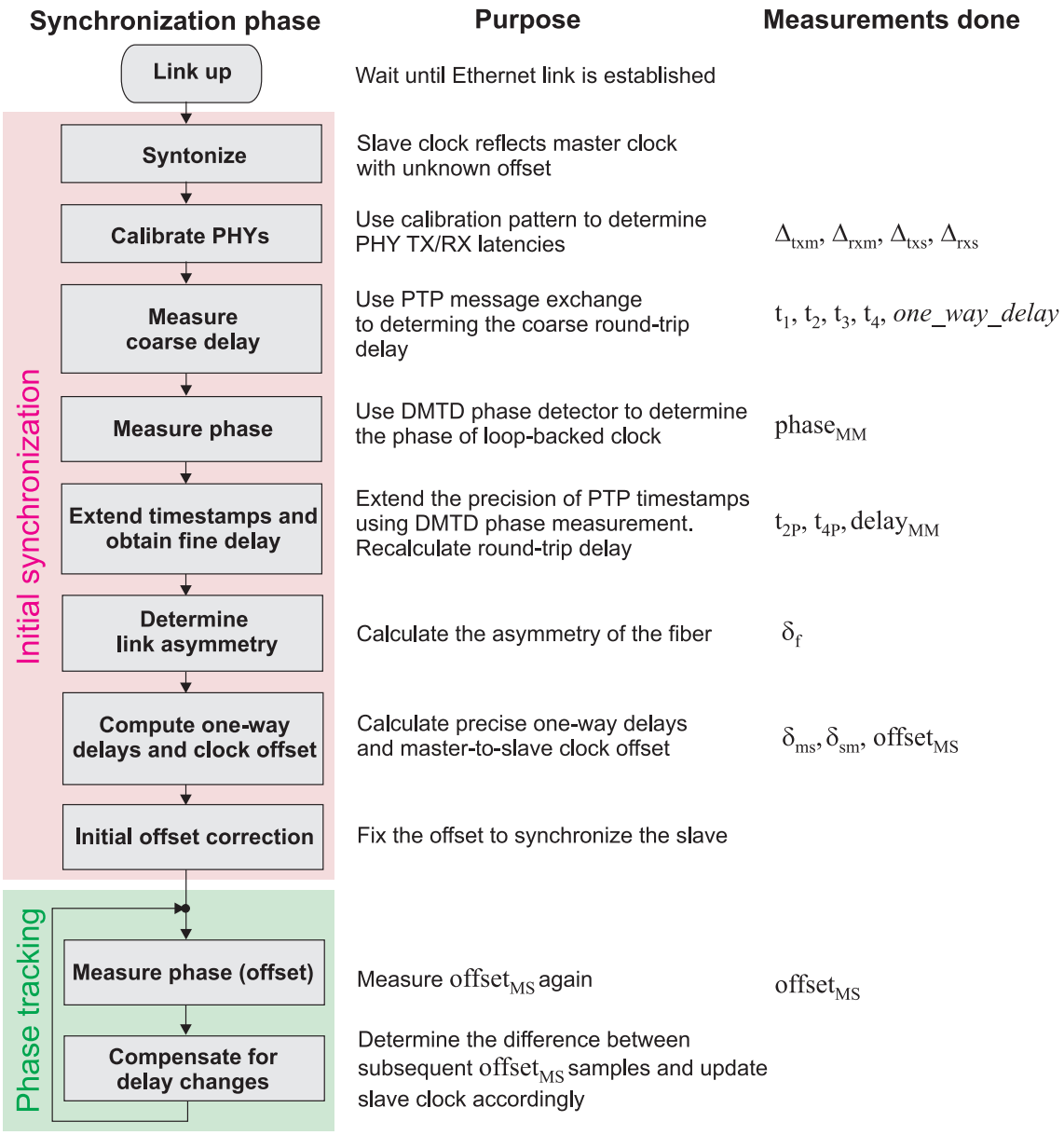

Figure 3.13. WR synchronization flow

trivial task. Figure 3.13 shows the steps needed to achieve and maintain synchronization of a single WR link. The synchronization process can be split into two parts:

- **initial synchronization**, which determines the value of $offset_{MS}$ and fixes the slave's phase shift to compensate the offset.

- **phase tracking** which monitors the changes of $phase_{MM}$ over time and updates the phase shifter in the slave to follow these changes and sustain synchronization.

3.5.2. Link detection and syntonization

During the first two steps of the synchronization flow 3.13, a syntonized WR connection is established. Drawing 3.14 shows the order of hardware operations and message exchanges which result in a syntonized link. At the initial moment, the master and the

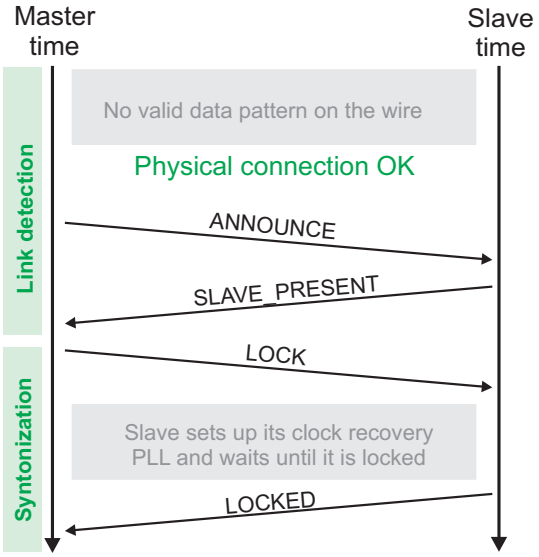

Figure 3.14. WR Link detection and syntonization

slave are not connected to each other. Their PHYs are transmitting an Ethernet idle pattern (see 3.2.2), but are not receiving any meaningful bitstreams. As soon as the physical connection is present, the PHYs will start receiving valid idle patterns and the Ethernet link will become active (see [16] section 36.2.5.2.6). Presence of a valid physical link will trigger the following sequence:

1. The master starts broadcasting *ANNOUNCE* messages to look for a WR slave,
2. Eventually, the slave will respond with a *SLAVE_PRESENT* message, indicating that it supports WR. If no response has been received within a predefined time, the master assumes that the slave is not WR compatible and aborts the synchronization process,
3. The master issues a *LOCK* message, commanding the slave to begin recovering the clock from its received data stream,
4. The slave sets up its PLL to use the RX clock as a reference and as soon as the PLL is locked, responds with a *LOCKED* message.

Upon successful completion of the above process, both nodes are syntonized and the delay measurements can begin.

3.5.3. Coarse delay measurement

The coarse delay measurement produces a set of timestamps $t_1 ... t_4$ which are obtained using the PTP End-to-End, two-way clock method as shown in fig. 3.8. The timestamps are generated in the hardware with a circuit depicted in fig. 3.15, guaranteeing single-cycle timestamping accuracy, which is necessary for reliably merging the coarse delay with the $phase_{MM}$ phase shift during fine delay calculation. Timestamps

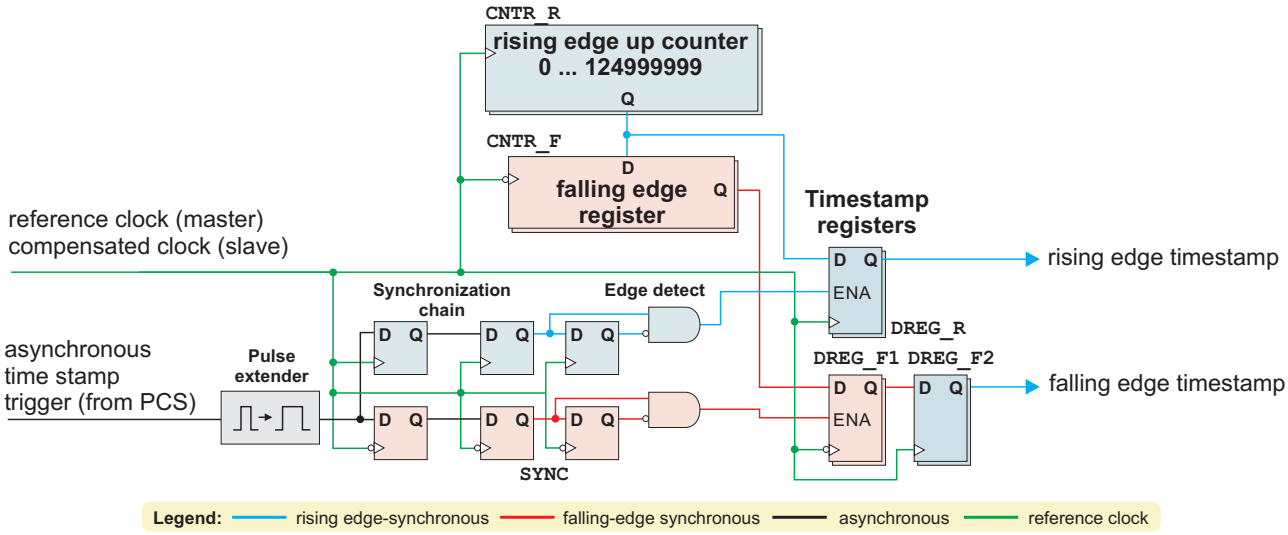

Figure 3.15. Structure of a WR timestamping unit (TSU)

in WR are taken when the PCS detects a Start-of-Frame Delimiter (SFD) character in the incoming (RX timestamps) or outgoing (TX timestamps) data stream (see 3.2.2).

Let's first focus on the blocks marked blue in fig. 3.15. Each time an SFD character is detected, the PCS produces a timestamp trigger pulse which causes the timestaming unit to take a snapshot of the free running counter CNTR_R with the D-type register DREG_R. The counter is counting from 0 to 124999999 which (given the reference clock frequency of 125 MHz) gives a period of one full second.

The counter CNTR_R works synchronously to the reference clock (master side, signal 1 in fig. 3.12) or the compensated clock (slave side, signal 5). Since RX trigger pulses come from different clock domains (2 or 4), they need to be synchronized to the reference clock with a chain of D flip-flops (SYNC). Single-cycle long trigger pulses are widened by the pulse extender before going to the synchronizer chain to ensure that no pulses are missed due to the metastability of synchronizer flip-flops. The counter value is latched in register DREG_R on the rising edge of the synchronizer output. TX timestamp triggers, which are generated within the reference clock domain (clocks 1 or 5) also pass through a synchronizer chain to obtain identical trigger reaction latency.

Unfortunately, due to the jitter of clock signals, crossing clock domains can make the gathered timestamps useless by causing a random ±1 LSB error when the RX clock and the reference clocks are almost in phase. The problem is illustrated in figure 3.16

Dashed lines show the transitions of ideal (jitter-free) clocks. If the jitter is neglected, the reference clock transition should be slightly ahead of the RX clock transition and the gathered timestamp should be equal to 2. However, if the clocks are jittery, the transitions may sometimes occur in reverse order, producing an erroneous timestamp of value 1. One possible way of addressing this issue is to take RX timestamps on both

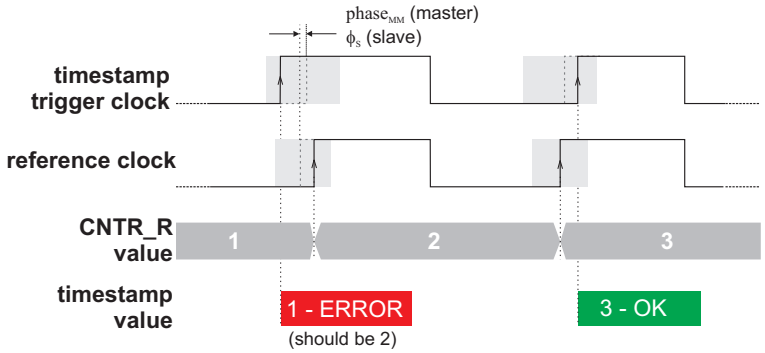

Figure 3.16. Timestamping errors caused by clock jitter

reference clock edges. The falling edge part of the TSU is marked pink in 3.15. It does not have an independent counter – instead, the current value of the rising edge counter is latched in register CNTR_F on the falling edge of the reference clock, making the CNTR_F a copy of CNTR_R delayed by a half of the clock period. This method ensures that at least one of the timestamps is valid at any moment (see 3.17). The correct timestamp is chosen depending on the current phase shift between clocks (see section 3.5.5). In order to simplify the hardware design, the presented TSU can only

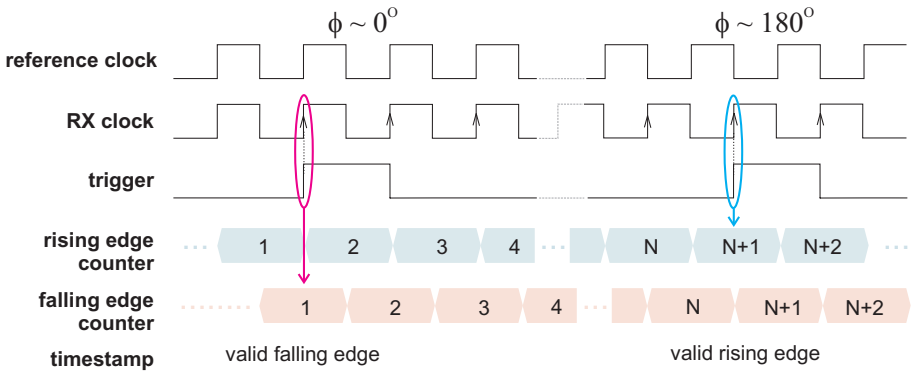

Figure 3.17. Dual-edge timestamping in WR

measure the sub-second part of PTP timestamps. The UTC part is appended in the software using the algorithm shown in listing 3.1.

Listing 3.1. Producing UTC timestamps

```
  timestamp tstamp = get_hardware_timestamp();
  time t = get_utc_time();
// check if there was a transition of the UTC counter between the
```

```
// generation of the timestamp and its readout by the software
  if(t.milliseconds < tstamp.rising_edge)
   tstamp.seconds = t.seconds - 1;
  else
   tstamp.seconds = t.seconds;
```

3.5.4. Digital DMTD phase detector

The fine delay measurements in WR are based on a Dual Mixer Time Difference (DMTD) phase detector. Therefore, in this section a short introduction to DMTD technology will be presented. Figure 3.18 shows an analog DMTD system [9]. Let's

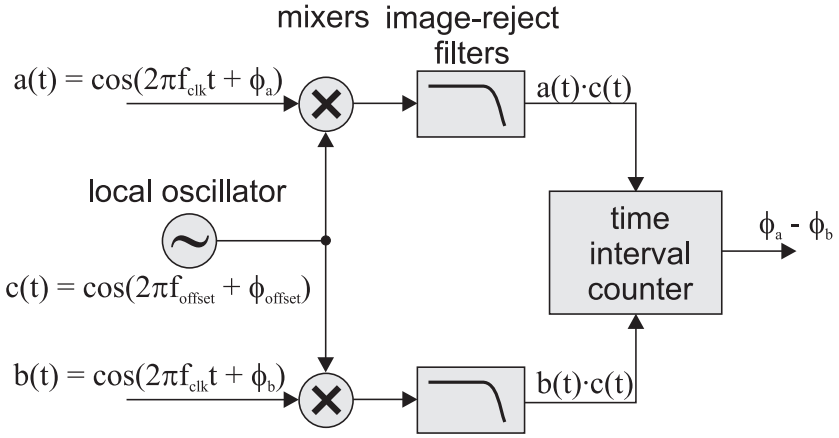

Figure 3.18. Structure of an analog DMTD phase detector

assume that its input clocks $a(t)$ and $b(t)$ have identical amplitudes and frequencies equal to f_{clk}. The phases of both clocks are respectively ϕ_a and ϕ_b. The clocks are multiplied by the mixers with the signal $c(t)$ of frequency f_{offset} and phase ϕ_{offset}. The multiplication result can be expressed as:

$$\begin{aligned} a(t)\cdot c(t) &= cos(2\pi t f_{clk}+\phi_a)\cdot cos(2\pi t f_{offset}+\phi_{offset}) \\ &= \frac{1}{2}cos(2\pi t(f_{clk}+f_{offset})+\phi_a+\phi_{offset}) \qquad (3.6) \\ &+ \frac{1}{2}cos(2\pi t(f_{clk}-f_{offset})+\phi_a-\phi_{offset}) \qquad (3.7) \end{aligned}$$

Term 3.6 has higher frequency than the multiplied signals and it's removed by the low-pass filter, leaving only the low-frequency term 3.7. The downconversion process changes the frequency of the signals, but does not change their phase difference. Therefore, if the offset frequency f_{offset} is close to the input signal frequency f_{clk}, the phase shift can be measured by counting the time between the rising edges of the downconverted clocks. For example, a reference clock of 125 MHz and an offset clock of 124.99

MHz will produce an output signal of 10 kHz. At such frequencies, the phase shift can be very accurately measured using a simple counter.

Analog DMTDs provide excellent resolution and linearity, at the cost of several external discrete components (mixers and filters), which can be troublesome, especially in multi-port applications such as the WR switch. Fortunately, the analog mixing operation can be transformed into a digital sampling operation, resulting in a digital DMTD detector, shown on fig. 3.19 In a Digital DMTD (DDMTD) [19], the input signals are

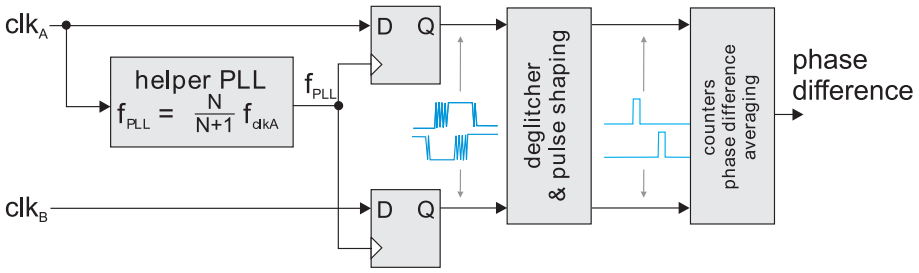

Figure 3.19. Structure of a digital DMTD phase detector

square waves, the mixers are replaced with simple D-type flip-flops and the offset clock is generated with a PLL from one of the input clocks. The sampling operation performed by the flip-flops can be mathematically proved to be equivalent to analog mixing (3.7), but the principle of a DDMTD can be explained in a more intuitive way. Figure 3.20a

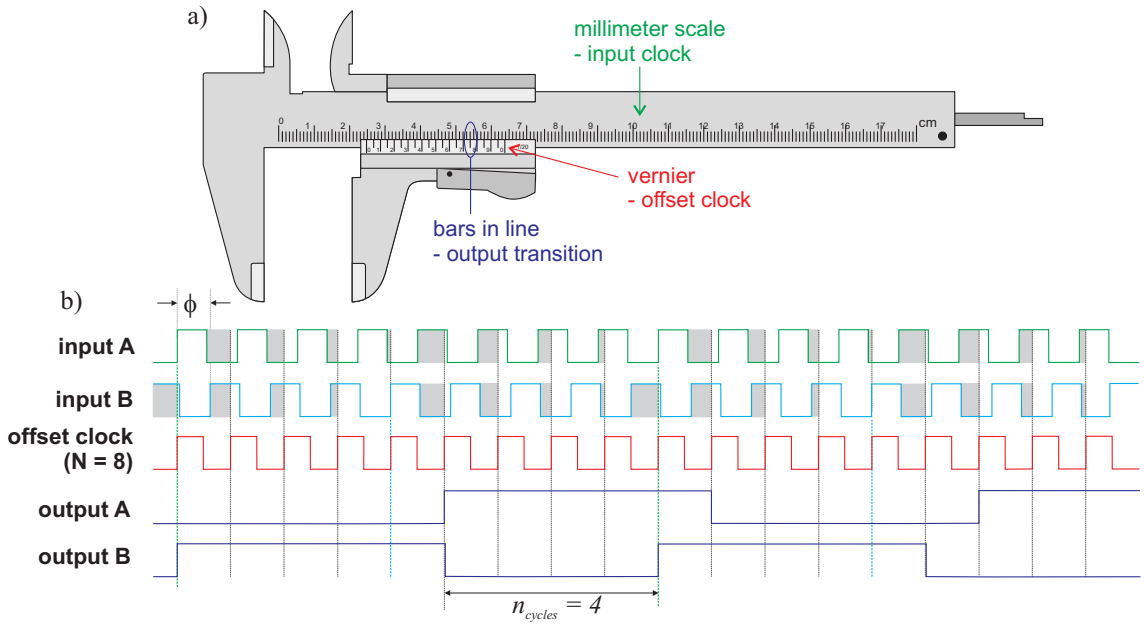

Figure 3.20. A vernier (a) and signals generated by DDMTD (b)

shows a vernier caliper. It has two scales - the big millimeter scale and a small vernier scale, with the units slightly smaller than the main scale. For a typical caliper, the vernier scale is split into 10 intervals with 4.9 mm spacing. If the length of the measured object has a fractional part, one of bars on the vernier scale will be in line with one of

bars from the main scale. Translating to the language of electronics, the main scale is the input clock where each interval represents one cycle, the vernier scale is the offset clock and the transitions in the DDMTD output signals occur when bars on both scales are aligned. An example of signals produced by a DDMTD with $N = 8$ is shown in fig. 3.20b. The output phase shift is proportional to the input phase shift ϕ by a factor of $(N+1)$. A general relation between the phase shift and the cycle interval between the DMTD outputs can be expressed as:

$$\phi[\text{ns}] = \frac{n_{cycles}}{N+1} \cdot \frac{1}{f_{in}} \tag{3.8}$$

Digital DMTDs have all the advantages of their analog counterparts while requiring only one external component (the oscillator producing the offset clock) which is shared among all measurement channels. This opens the way for low-cost FPGA implementations. If a picosecond-level accuracy is not required, even a PLL integrated inside an FPGA can be used, eliminating all external components. In practical DDMTD imple-

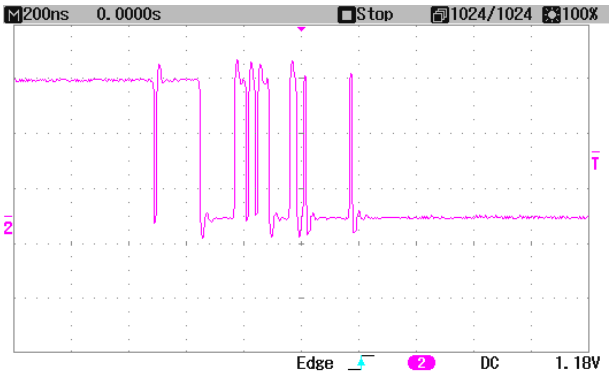

Figure 3.21. Glitches in the DMTD output caused by clock jitter

mentations, the output signals need to be additionally conditioned as the input clock jitter can introduce glitches, as shown on fig. 3.21a. More details about the deglitching and postprocessing algorithm can be found in section 4.3.5.

3.5.5. Fine delay measurement

Having the knowledge of PTP timestamps and round-trip phase shift $phase_{MM}$, we can calculate the precise round-trip delay $delay_{MM}$. The calculation is performed in two steps:

- The accuracy of PTP timestamps t_2 and t_4 is enhanced beyond a single clock cycle using the knowledge of the round-trip phase $phase_{MM}$ and the slave PLL setpoint $phase_S$. As a result, precise timestamps t_{2p} and t_{4p} are obtained.
- The fine round-trip delay is calculated using the standard PTP formula:

$$delay_{MM} = (t_{4p} - t_1) - (t_3 - t_{2p}) \tag{3.9}$$

Only the reception timestamps need to be enhanced, as they are generated within clock domains asynchronous to the reference (or compensated) clock (see table 3.2). Transmission timestamps are always integer because packets are transmitted and timestamped using the same clock.

The flow graph of the algorithm used to merge PTP timestamps with phase measurements is shown on fig. 3.22. Figure 3.23 depicts sample measurements of inputs and results of the enhancement algorithm for t_{4p} timestamps, where the varying link delay was simulated using the slave's phase shifter.

Table 3.2. Timestamping clock domains

Timestamp	Origin	Trigger clock	Timestamping clock	Correction
t_1	Master TX	reference (1)	reference (1)	0
t_{2p}	Slave RX	slave recovered (2)	compensated (3)	$-phase_S$
t_3	Slave TX	compensated (3)	compensated (3)	0
t_{4p}	Master RX	master recovered (4)	reference (1)	$+phase_S$

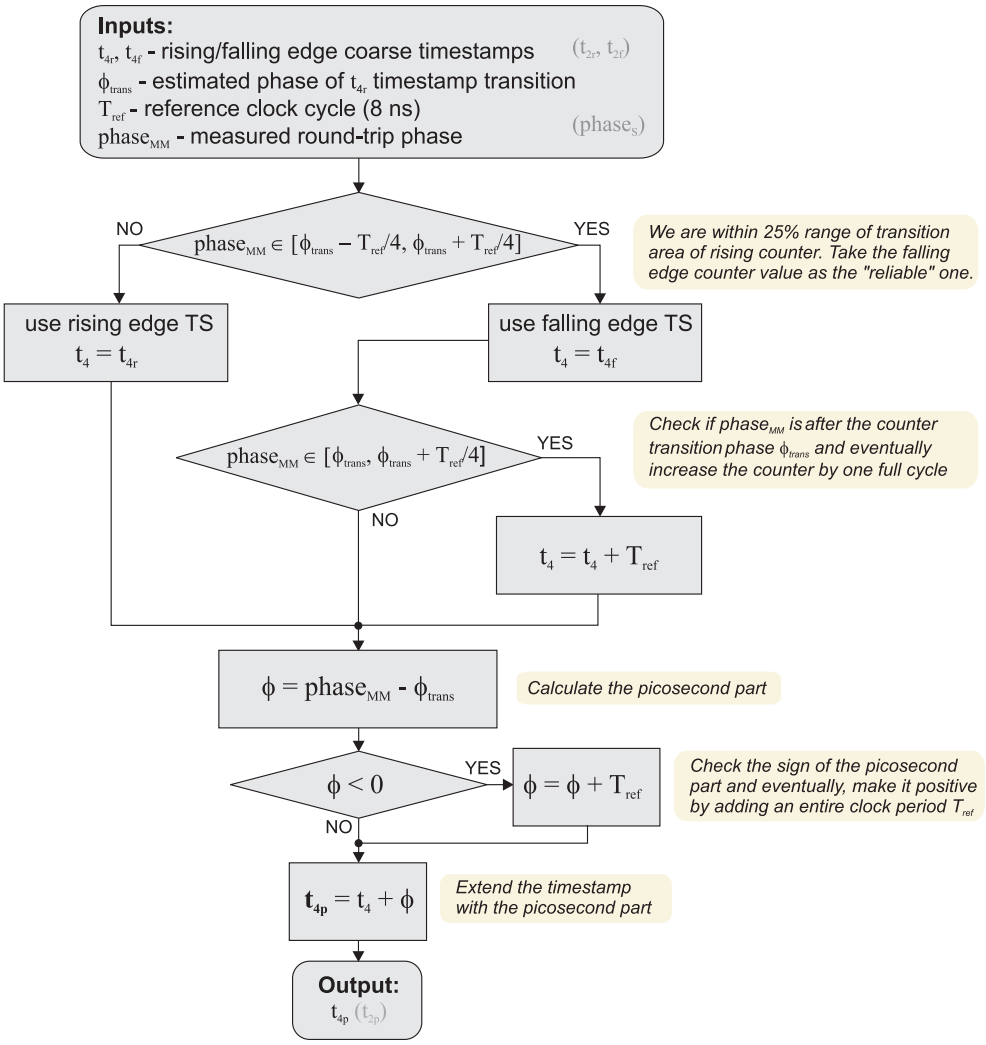

Figure 3.22. Algorithm for enhancing coarse timestamps with DMTD phase.

The first step of the algorithm addresses the problem of glitches in RX timestamps, by choosing either the rising or falling edge timestamp depending on the phase between the RX clock and the reference clock. The key parameter of this step, ϕ_{trans} provides an approximate value of $phase_{MM}$ at which a transition should occur in the value of t_{4r}. In fig. 3.23, it is equal to 6.6 ns - it's the approximate intersection point of $phase_{MM}$ (blue, sawtooth-like trace) and the transition of the rising edge timestamp (red trace). The value of ϕ_{trans} is a device-specific constant. It can be determined once during the factory calibration of a WR device or measured upon every startup by sweeping a full clock period using the built-in phase shifter and searching for a transition in the RX timestamp value.

If the actual value of $phase_{MM}$ lies within a $\pm 25\% T_{ref}$ range from the transition point ϕ_{trans} (green zone in 3.23), the algorithm will use t_{4f} timestamp, otherwise t_{4r} timestamp will be taken (red zone). Note that because $phase_{MM}$ is bounded to $\left[0, T_{ref}\right)$, the range checks must be aware of the jump in the phase between T_{ref} and 0.

The second step checks if the $phase_{MM}$ value is ahead of the transition point ϕ_{trans}, and eventually increases the chosen timestamp by a full cycle (T_{ref}). That ensures the transition in t_4 will always occur when $phase_{MM} = \phi_{trans}$, eliminating the risk of transition glitches.

The last part of the algorithm simply adds the picosecond part (which is the DMTD phase corrected with the transition offset ϕ_{trans}) to the coarse deglitched timestamp t_4. If the resulting picosecond part (ϕ) is negative, an additional full cycle is added to the result. The final output of the merging algorithm is shown in 3.23 as the thick navy trace. The enhancement operation for t_2 timestamp is done in a very similar way by replacing t_4 with t_2 and $phase_{MM}$ with $phase_S$. Note that changing the slave's phase shift $phase_S$ will result in a change of the values of both t_{2p} and t_{4p} (see table 3.2). Increasing $phase_S$ by a certain δ will cause t_{4p} to also increase by δ (assuming that the link delay stays constant). Simultaneously, the value of t_{2p} will be decreased by the same amount, as for the slave's RX timestamps, the timestamping clock is $phase_S$ ahead of the trigger clock (so if the phase shift is increased, the timestamp value goes down). Therefore, the calculated fine delay is not affected by the changes of $phase_S$:

$$delay_{MM} = (t_{4p} - t_1) - (t_3 - t_{2p}) = t_{4p} \cancel{+ phase_S} - t_1 - t_3 + t_{2p} \cancel{- phase_S} \tag{3.10}$$

3.5.6. Link asymmetry estimation

In order to compute an accurate master-slave offset, one must determine the asymmetry of the link delay. This asymmetry cannot be measured directly (as a difference between the overall M-S and S-M delays), as it would require the nodes to be synchronized prior to the measurement. Therefore it is only possible to estimate the asymmetry

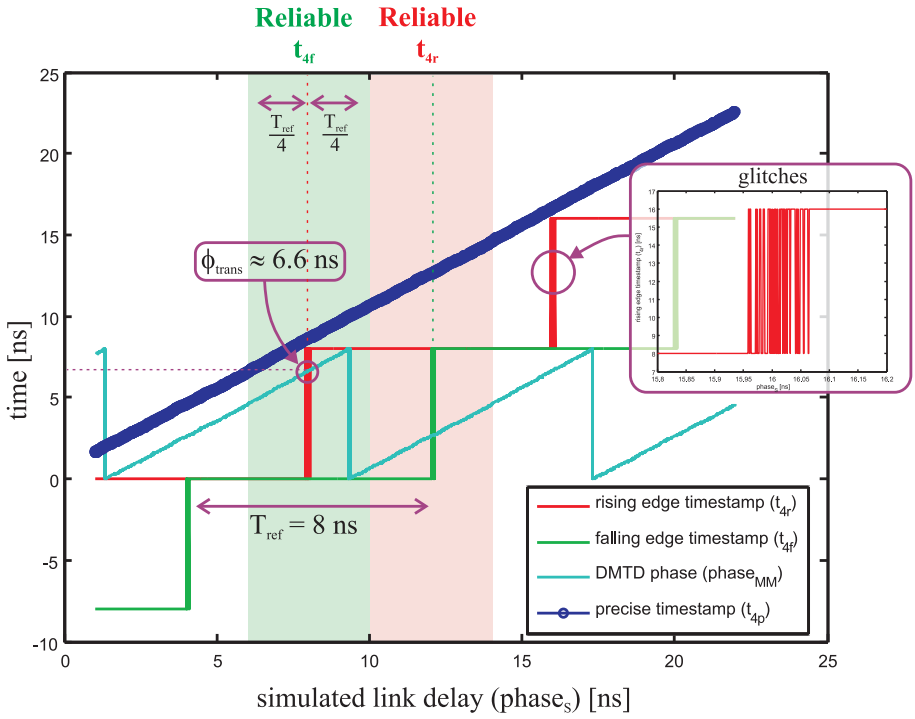

Figure 3.23. Example of t_{4p} timestamp enhancing.

from round-trip delay $delay_{MM}$, using the knowledge of the properties of the components constituting the link.

In the WR optical link model, the following sources of asymmetry were taken into consideration:

- Propagation delays of electronic components and PCB traces (circuit asymmetry),
- Asymmetry of optical transceivers (SFPs),
- Difference between TX and RX wavelengths in the fiber,
- Internal structure and clocking of the PHY (SerDes) chips.

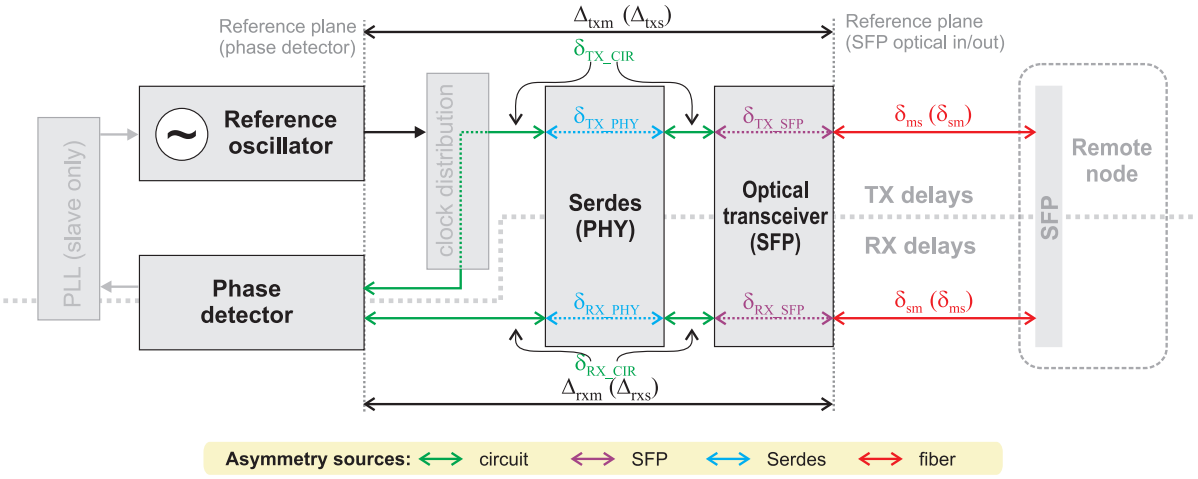

Figure 3.24. Delay asymmetries in WR optical link.

Figure 3.24 shows the reference asymmetry model used in WR devices. The device's asymmetric delays (as shown in fig. 3.12) are expressed as sums of circuit, SFP and PHY delays between the phase detector inputs (the PD measuring $delta_{MM}$ on the master side and the PD in the phase shifting PLL on the slave side) and the SFP optical input/output. The fiber asymmetry is compensated separately in the slave's PTP servo.

$$\begin{aligned} \Delta_{tx(m/s))} &= \delta_{TX\_PHY(M/S)} + \delta_{TX\_CIR(M/S)} + \delta_{TX\_SFP(M/S)} \\ \Delta_{rx(m/s)} &= \delta_{RX\_PHY(M/S)} + \delta_{RX\_CIR(M/S)} + \delta_{RX\_SFP(M/S)} \end{aligned} \tag{3.11}$$

Circuit and SFP asymmetry

Circuit asymmetry stems from differing PCB trace lengths, propagation delays of the electronic components of the clock distribution system and FPGA logic cell placement and routing delays. SFP asymmetry is a result of different reception and transmit delays between the electrical and optical ports of the SFP transceiver.

These asymmetries can vary with changes of the device's operating conditions (i.e. temperature, supply voltage). Depending on the synchronization accuracy requirements, they can be:

- Treated as time-invariant and measured once during factory setup.
- Actively compensated, following the changes in temperature and/or supply voltage read from built-in sensors (requires a model of delay vs. temperature and voltage),
- Eliminated by building a system where both master and slave have identical (or zero) asymmetry and operate in similar conditions.

The last method is particularly useful for compensating the delays inside the FPGA on the way between the PHY (oscillator) and the phase detector, as they cannot be externally measured during the factory calibration process. They are also significantly affected by temperature and voltage changes. Therefore, the only practical way of dealing with the FPGA part of the circuit asymmetry is to equalize the delays on all phase detector inputs. This can be done by constraining the routing delays or manually placing and routing the phase detector block. This approach also allows for reducing the impact of the temperature-voltage changes on the asymmetry to a negligible value, as two equal paths placed next to each other would have very similar temperature/voltage delay coefficients.

SFP asymmetries can be compensated in a similar manner by choosing pairs of SFP transceivers whose TX and RX delays differ by similar values (i.e. $\delta_{TX\_SFPM} - \delta_{RX\_SFPM} = \delta_{TX\_SFPS} - \delta_{RX\_SFPS}$). This can be done in a laboratory system where all other asymmetric delays are already compensated by selecting SFPs that together give minimum clock offset.

WDM fiber asymmetry

As mentioned in section 3.2, WR uses different wavelengths for transmitting and receiving the data (for example 1550/1310 nm). Due to chromatic dispersion of the fiber, the refractive indexes for these wavelengths are slightly different, causing different propagation speeds (and thus, different delays) between the master and the slave.

The refractive index at a given wavelength can be derived using Sellmeier's equation [4] (3.12):

$$n^2(\lambda) = 1 + \frac{B_1\lambda^2}{\lambda^2 - C_1} + \frac{B_2\lambda^2}{\lambda^2 - C_2} + \frac{B_3\lambda^2}{\lambda^2 - C_3} \tag{3.12}$$

where B_i and C_i are material-specific coefficients.

For a standard G.652 telecom fiber, the refractive indexes are respectively: $n_{1550} = 1.467$ and $n_{1310} = 1.466$. In order to simplify the asymmetry calculations in the hardware, the WR specification [11] defines a custom fiber asymmetry coefficient 3.13 expressing the ratio between the M-S and S-M fiber propagation delays:

$$\alpha = \frac{\delta_{ms}}{\delta_{sm}} - 1 = \frac{n_{1550}}{n_{1310}} - 1 \tag{3.13}$$

Unfortunately, refractive indexes may vary slightly between different fiber manufacturers, making the direct calculation of α unreliable. WR can solve this problem by characterizing the fiber asymmetry using laboratory measurements of $delay_{MM}$ (done by PTP) and clock offset $offset_{MS}$ (done using an oscilloscope). The measurements are performed when all the other asymmetries are already compensated. The value of α 3.15 is calculated by solving equation system 3.14:

$$\begin{cases} delay_{MM} = \delta_{ms} + \delta_{sm} + \Delta \\ offset_{MS} = \frac{\delta_{ms} - \delta_{sm}}{2} \end{cases} \tag{3.14}$$

where Δ accounts for all fixed delays in the path (i.e. $\Delta_{txm} + \Delta_{rxs} + \Delta_{txs} + \Delta_{rxm}$). Substitution of δ_{ms} and δ_{sm} with the results of 3.14 gives the final value of α:

$$\alpha = \frac{delay_{MM} - \Delta + 2 \cdot offset_{MS}}{delay_{MM} - \Delta - 2 \cdot offset_{MS}} \tag{3.15}$$

Transceiver asymmetry

Transceiver asymmetry is a result of the internal structure of the SerDes's circuitry. Most of the SerDes chips nowadays are optimized for low power consumption and fast locking to the incoming data stream. Unfortunately, because of these optimizations, PHYs may not keep constant transmit/receive latencies. The problem is illustrated in fig. 3.25. PHY asymmetry manifests itself as a random latency between the rising edge of the TX/RX clock and the inter-symbol boundary in the transmitted (received) serial

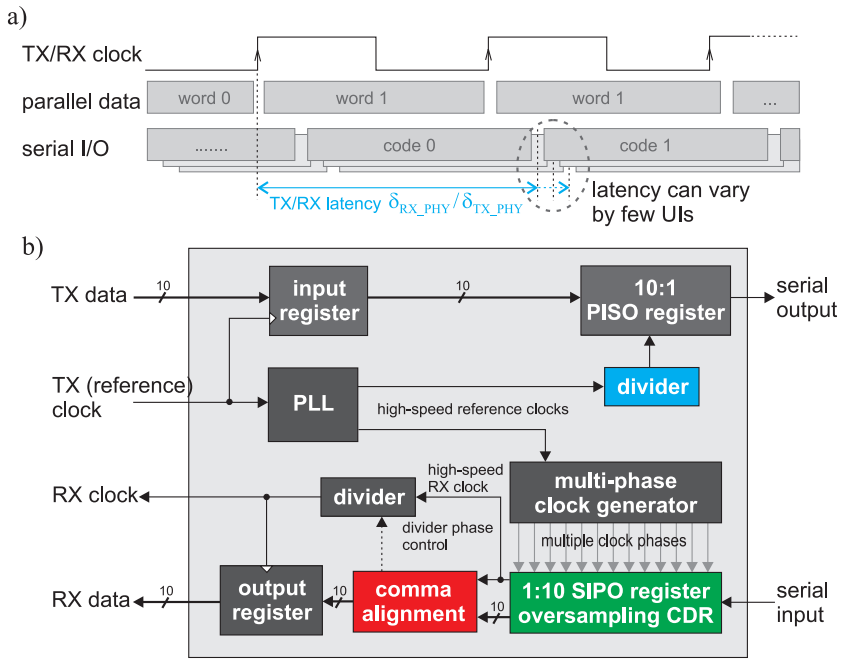

Figure 3.25. Random delays in gigabit SerDes devices (a) and blocks causing them (b).

data stream. In most PHYs, this latency can be different for each PLL/CDR lock cycle, but once the PHY is locked, the delays shall remain constant. PHYs whose TX/RX delays can change when the link is active are unsuitable for WR devices.

In the PHYs which have been evaluated for usability in WR hardware (TLK1221 from Texas Instruments and Virtex-6/Spartan-6 GTP transceivers), the following random delays were identified:

- RX alignment latency, observed in PHYs which do not correct the RX clock phase when aligning to the inter-symbol boundary (red, observed for Xilinx GTP),
- RX latency resulting from the internal structure of the digital oversampling CDR (green, observed for TLK1221),
- TX latency caused by the clock divider between the internal PLL and the parallel-to-serial register (blue, observed for TLK1221).

Alignment latency can be measured every time the link goes up by disabling automatic comma alignment and bit-shifting the unaligned output data until a valid 8B10B code sequence is detected (*bitslip trick*, see [20]). Compensation of the latter two latencies, however, requires additional calibration logic. An example method (used in the WR switch) is shown in fig. 3.26. The PHY transmit path is fed with a sequence of K28.5 characters (1111100000), effectively producing a 125 MHz square wave on the serial outputs. The phase shift between the TX clock and the output bitstream can be measured using a DDMTD phase detector, giving the value of the TX latency. The same method can be used to measure the RX latency by commanding the link partner to send the calibration pattern. Note that since the K28.5 character contains a comma (5 consecutive ones), a burst of subsequent K28.5 symbols will cause improper operation

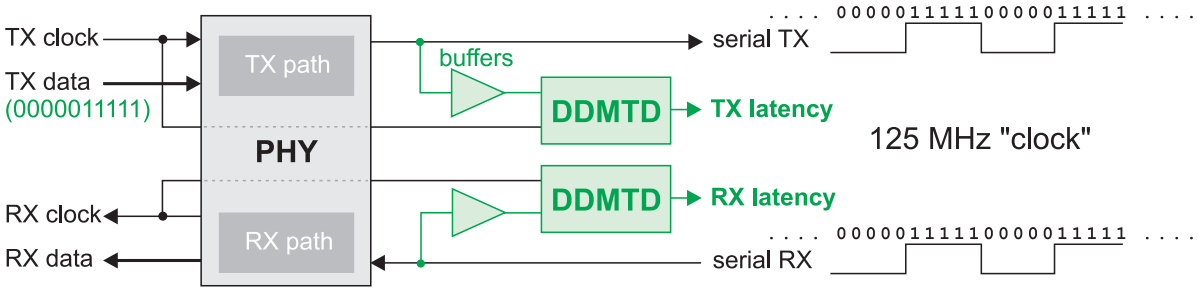

Figure 3.26. PHY latency measurement using calibration patterns.

of the PHY's comma alignment unit. Therefore, comma alignment must be disabled during the calibration process.

3.5.7. Establishing and maintaining synchronization

Having obtained the values of round-trip delay $delay_{MM}$ and link asymmetry, we can calculate the one-way master to slave delay $delay_{MS}$ by solving equation (3.16):

$$delay_{MM} = \Delta + \delta_{ms} + \delta_{sm} \tag{3.16}$$

where $\Delta = \Delta_{txm} + \Delta_{rxs} + \Delta_{txs} + \Delta_{rxm}$. Substituting δ_{sm} with 3.13 and solving for δ_{ms}, we obtain the one-way fiber delay 3.17:

$$\delta_{ms} = \frac{1+\alpha}{2+\alpha}(delay_{MM} - \Delta) \tag{3.17}$$

which after adding the circuit, SFP and PHY asymmetric delays present on the master to slave path gives the final master-slave delay 3.18 and offset 3.19:

$$delay_{MS} = \frac{1+\alpha}{2+\alpha}(delay_{MM} - \Delta) + \Delta_{txm} + \Delta_{rxs} \tag{3.18}$$

$$offset_{MS} = t_1 - t_{2p} - delay_{MS} \tag{3.19}$$

The value of $offset_{MS}$ is the input for the slave's offset adjustment algorithm which controls the slave's clock servo. The flow diagram of the algorithm is shown in fig. 3.27a and an example servo design can be found in fig. 3.27b. The algorithm assumes that the frequency has been already syntonized by means of Sync-E and only the clock offset needs to be corrected. Offset correction is split into 3 steps:

1. **UTC time adjustment**: the UTC counter in the servo is increased (or decreased) by the number of full seconds 3.20 in $offset_{MS}$:

$$corr_{utc} = \lfloor \frac{offset_{MS}}{1\ \mathrm{s}} \rfloor \tag{3.20}$$

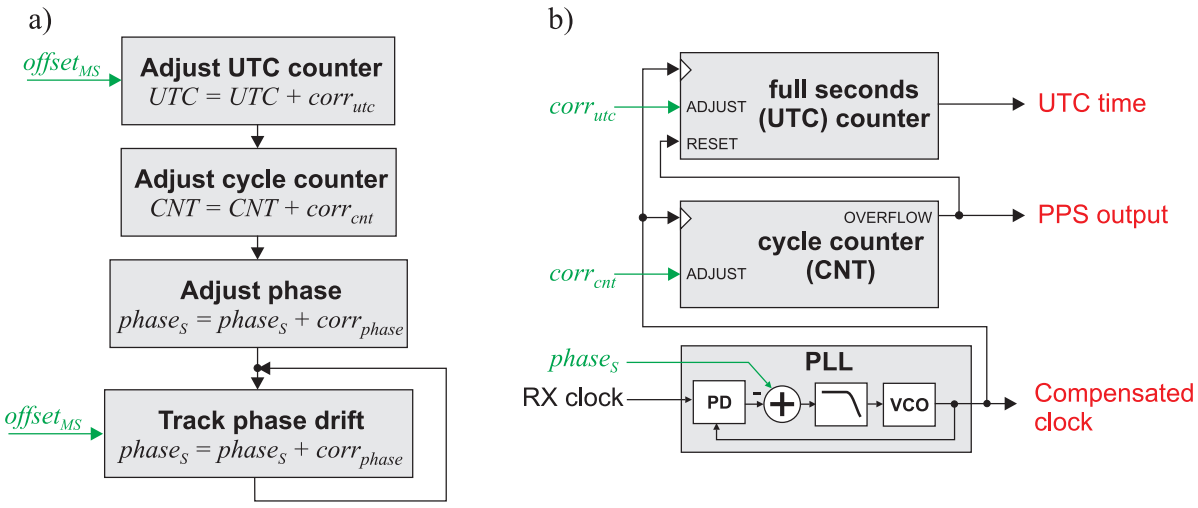

Figure 3.27. WR slave offset adjustment (a) and clock servo (b)

2. **Clock cycle counter adjustment**: The reference clock cycle counter, which produces the PPS signal is adjusted by the number of full T_{ref} (8 ns) cycles 3.21.

$$corr_{cnt} = \lfloor \frac{offset_{MS} - corr_{UTC}}{T_{ref}} \rfloor \tag{3.21}$$

3. **Phase adjustment**: The slave's phase shifter is adjusted with the remaining sub-cycle part of the offset 3.22

$$corr_{phase} = offset_{MS} - [offset_{MS}] \tag{3.22}$$

Voilà! Now the slave's clock and PPS signals are synchronized to the master with sub-nanosecond accuracy. Since the offset can vary with operating conditions, it is measured at regular intervals and the difference between subsequent measurements is added to slave's phase shift to compensate for phase drift:

$$corr_{phase} = offset_{MS} - offset_{MS\_previous} \tag{3.23}$$

Because the phase drift is mainly caused by temperature variations, the rate of subsequent adjustments can be very low, even in the scale of a single adjustment per hour.

Note that the way $corr_{phase}$ is calculated requires the phase shifter to be able to change the phase relatively by any arbitrary value with no "jumps" in the signal when the value of $corr_{phase}$ crosses the inter-cycle boundary. For example, one can use a PLL with a phase detector capable of handling wrap-around phase transitions, but not a programmable delay line. An example design of such phase detector and PLL is described in section 4.3.7.

3.6. Integration of White Rabbit into PTP

As mentioned in section 3.3, WR extends the PTP standard with several custom messages to enable frequency syntonization and PHY delay calibration. These messages utilize the user-definable Type-Length-Value fields in the PTP version 2 Announce and Management messages. A WR master is therefore fully compatible with any PTPv2-compliant device.

Figure 3.28 shows the order of the PTP message exchanges during all the phases of the synchronization process, indicating which messages are standard PTP (black) and WR-specific (red). Note that the calibration stages involving the exchange of calibration patterns are only required if the remote node's PHY has type 2 (CDR) asymmetry.

The drawing is intended to complement the synchronization flow described in the chapter and give the reader an overview of the entire WR-PTP synchronization process. A detailed description of the data flow, packet formats and master's and slave's state machines falls out of the scope of this thesis. A complete description of the protocol can be found in the WR specification [11].

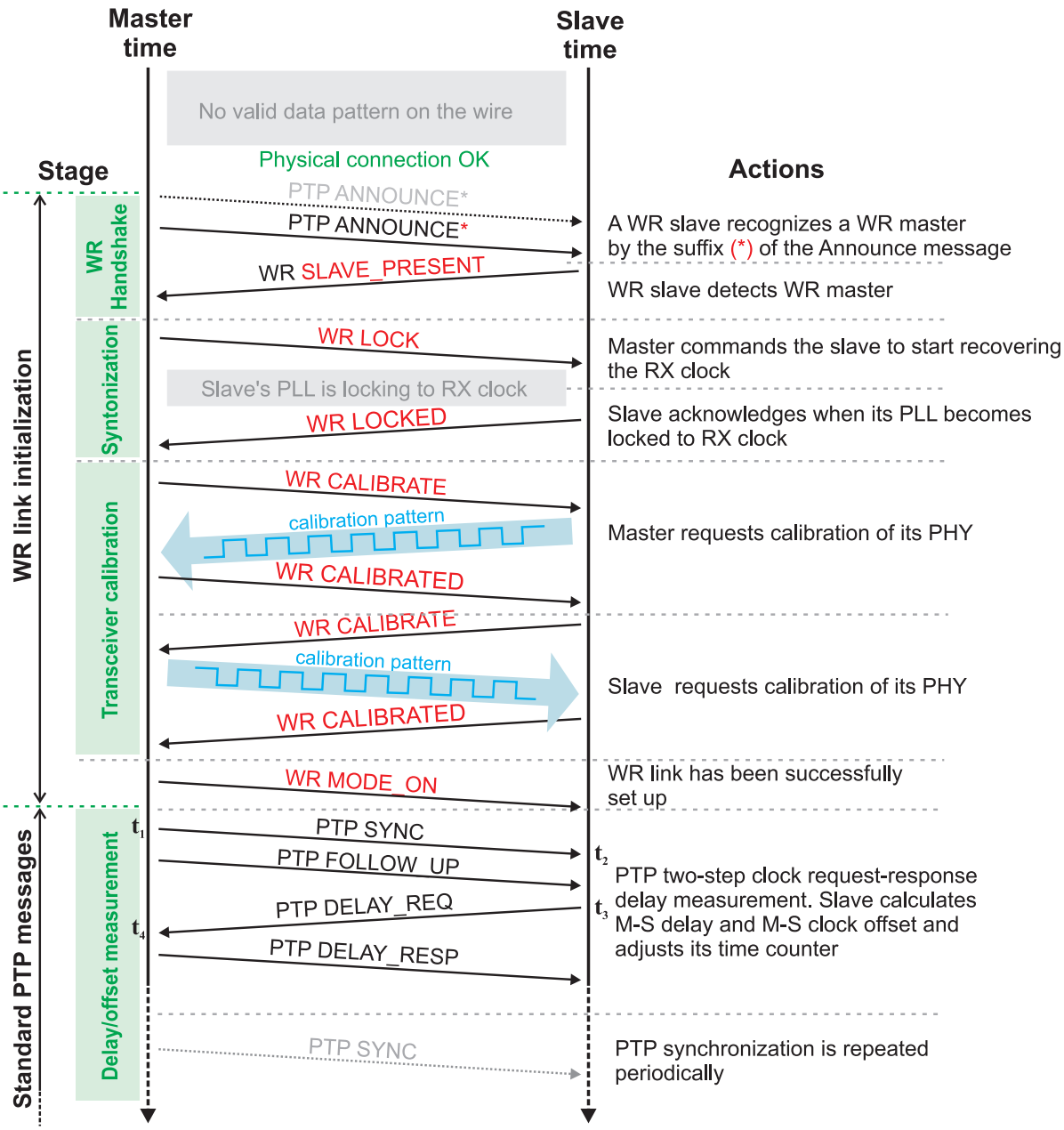

Figure 3.28. PTP message flow during WR synchronization.

4. The White Rabbit Switch

The White Rabbit Ethernet Switch, further abbreviated as WRS, is the central component of every WR network, allowing for multiplexing high precision timing and packet data in a single physical connection. This chapter presents a very brief description of the hardware, FPGA and software design of version 2 of the WRS. Since the scope of this thesis is limited to time and frequency transfer, the focus is on the components of the WRS which play a significant role in these processes. Blocks having little impact on the synchronization are described only to an extent necessary to explain their function in the design of the WRS.

Due to the large size of the WRS codebase (over 20 pages of electrical schematics, 3 complex PCB layouts and more than 80000 lines of VHDL/SystemVerilog/C code), the chapter contains no detailed technical documentation of the WRS in printed form. All WRS design files are available at [31].

4.1. Overview

The WRS is a full-duplex, non-blocking, manageable gigabit Ethernet switch (IEEE802.1D Bridge) with 10 IEEE802.3z SFP (mini-GBIC) ports, supporting both fiber and copper connections. Unlike typical Ethernet switches, the WRS provides a set of enhanced features to fit the requirements of high performance control and timing systems, such as:

- Sub-nanosecond IEEE1588/Sync-E synchronization engine.
- Full routing latency determinism for a certain class of packets (expressed as a fixed upper boundary in the packet residency time).
- 8 priority levels with independently configurable routing mode (store and forward and cut-through).
- Improved network redundancy protocols: hardware-assisted Spanning Tree Protocol allowing for microsecond-range recovery of a broken link.

The functional diagram of the WRS is shown in Figure 4.1. The WRS communicates with other devices using the following interfaces:

- **2 uplink ports** (WR slaves), receiving the time and frequency reference from the upper layers of the network. One of the uplink ports can be designated as primary PTP slave, while the other one can serve as a backup timing receiver.

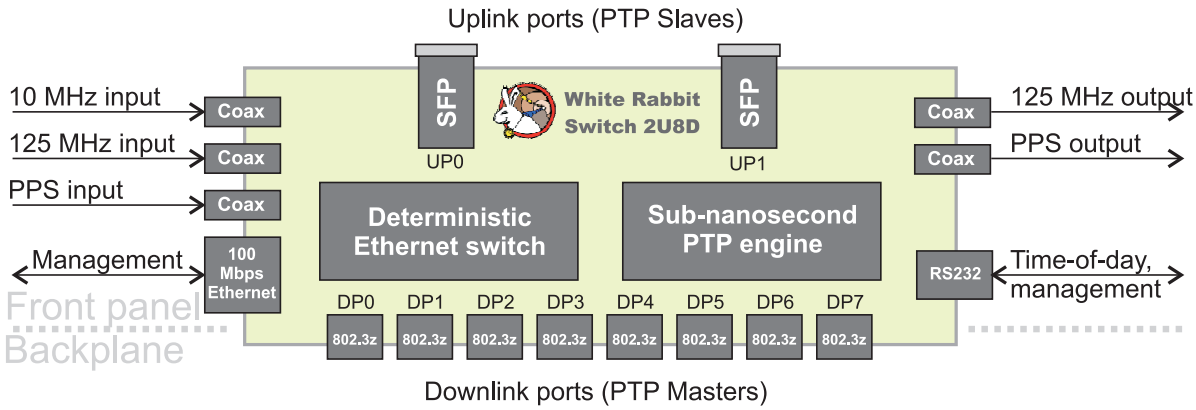

Figure 4.1. Interfaces and functionality of WRS.

- **8 downlink ports** (WR masters), propagating the timing to other switches or nodes. The distinction between the uplink and downlink ports applies only to the synchronization hierarchy, because only uplink ports can recover the Sync-E carrier clock. Data-wise, all the ports are identical as in every Ethernet switch. All the ports are also equal for the non-synchronous PTP implementations.
- **External 10 MHz, 125 MHz and PPS inputs**, allowing the WRS to work as a WR/PTP grandmaster by taking the reference timing from an atomic clock or a GPS receiver. The 125 MHz input can be used as a direct clock input for the PHYs for measurement and testing purposes.
- **PPS and 125 MHz outputs** providing the slave's in-phase recovered clock.
- **100BaseT Ethernet port** foreseen for the device management and administration using an external (non-WR) network.
- **Multi-purpose RS232 port**, which can be used to receive or output the UTC time-of-day information or as a management serial console.

All the ports listed above, except the downlinks, are accessible at the device's front panel. Access to downlink lanes depends on the particular hardware configuration (stand-alone or with a backplane – see the following sections).

4.2. Hardware

4.2.1. Hardware platform

The WRS is designed as a microTCA Management Carrier Hub (MCH) card [24], due to the high throughput, compact size and low cost of microTCA backplanes combined with a vast set of features, such as redundant power supply and sophisticated remote management. Because of the insufficient amount of space available on a single microTCA card, the WRS MCH module is a "sandwich" of two PCBs stacked with dense multipin board-to-board connectors as shown in Figure 4.2a. The PCBs are:

- the **main PCB** (4.2.3), which hosts the packet processing part of the switch (FPGA and CPU with necessary external memories), as well as the power supply block for the entire module.
- the **timing PCB** (4.2.4), carrying the analog timing inputs, clock recovery system (PLLs and oscillators) and the uplink PHYs with SFP sockets. The timing board also delivers the reference clocks to the microTCA backplane.

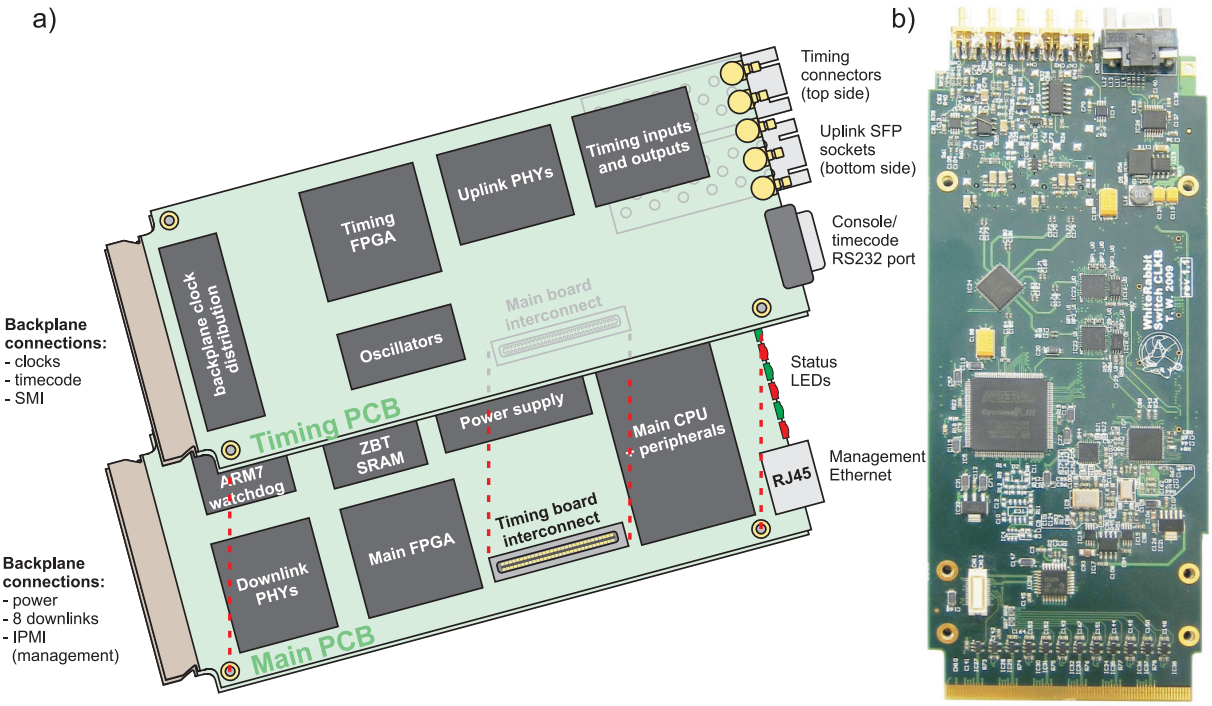

Figure 4.2. WRS module assembly (a) and photograph (b).

Note that some connectors are not placed on the PCB which hosts their drivers (for example the RS232 port or the SFPs). Such layout was necessary to fit all the connectors on a very limited space of a MCH card front panel.

microTCA MCH operation

The WRS was originally designed to operate in a microTCA crate as an MCH module. The main role of the MCH is providing a gigabit Ethernet link and a set of clock signals to each Advanced Mezzanine Card (AMC) in the crate. The MCH must also implement crate management services via the Intelligent Platform Management (IPMI) Interface [17]. The WRS supersedes these requirements by:

- providing **deterministic** Ethernet links for up to 8 AMC cards through the Fabric A (a mandatory lane in every microTCA backplane, dedicated for a GbE link).
- delivering timing signals in the form of a 125 MHz reference clock, transmitted using Telco Clock A lanes and an encoded PPS/timecode signal propagated on Fabric D.

Figure 4.3 shows an example of a microTCA-based WR system, where the timing receiver capabilities of the WRS are used to derive the time base in an A/D converter

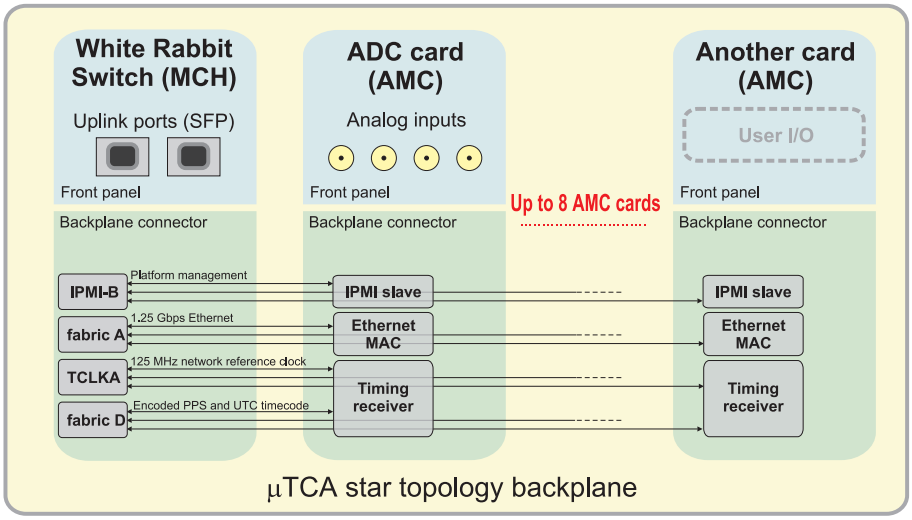

Figure 4.3. WRS operating in a microTCA crate.

AMC card and the Ethernet part deals with the readout of the acquired data. The same crate can host other AMC cards communicating between themselves using WR Ethernet links and spare lanes in the backplane ("fat pipes"). In such way, one can build compact and low-cost distributed data acquisition systems synchronized to a common clock source.

Standalone operation

The microTCA architecture, however, may become an obstacle if only the switching functionality of the WRS is required because of the still significant cost of microTCA backplanes. The MCH module can be therefore used in conjunction with a simple mini-backplane to build a standalone 10-port Ethernet switch, as shown in Figure 4.4.

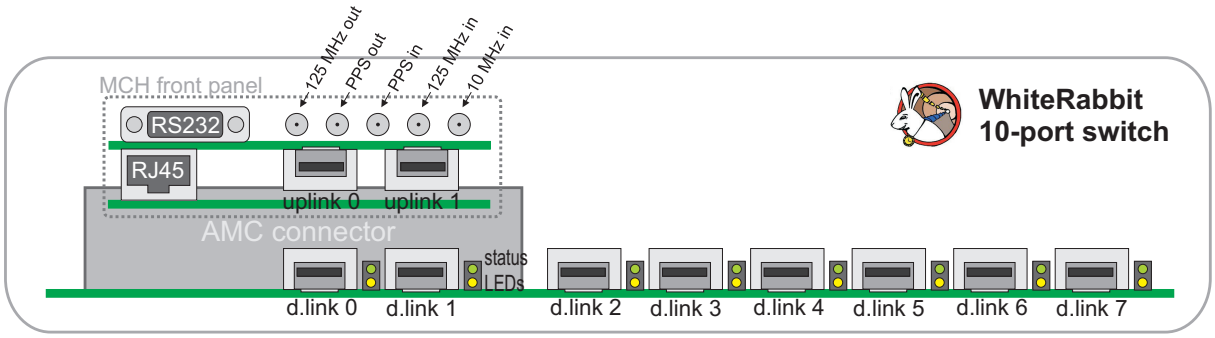

Figure 4.4. Standalone WRS with a mini-backplane.

4.2.2. System design

Having presented the working platform, we can proceed with the description of the MCH hardware. The WRS is a fairly complex design, comprising multiple FPGAs and

CPUs and a number of discrete ICs. Figure 4.5 shows the general block schematic of the MCH, showing all the major components and the interfaces connecting them.

The MCH is built around the two FPGAs: the big **main FPGA** (4.2.3), taking care of the packet processing and the smaller **timing FPGA** (4.2.4) which implements the PLLs for the timing system. The chips communicate with each other via a simple low-speed serial link. Both FPGAs are controlled by the **main CPU** (4.2.3), an ARM processor running the Linux operating system and a set of applications which govern all the aspects of the WRS operation.

The second CPU shown in the diagram (**watchdog MCU**) is a small ARM7 microcontroller, which implements the IPMI stack for microTCA system management and ensures the proper operation of all the logic by means of a challenge-response mechanism. The watchdog also monitors the state of the power supplies and the temperature in various spots on the PCB. The main CPU can configure the watchdog using an SPI interface.

Other components worth noticing are the 10 1000Base-X PHYs and the ZBT (Zero Bus Turnaround) static memory, connected to the main FPGA. There is also a separate backplane clock fanout and a timing line driver, whose role is propagating the reference clock and PPS/timecode to all AMC slots in the crate. The description of the timing components is presented in more detail in section 4.2.4.

4.2.3. Main board

The main board hosts the "digital" part, downlink port PHYs and the power supply of the MCH. Drawing 4.6 shows a snapshot of the main PCB layout with the annotated blocks from the system diagram 4.5. Both PCBs of the MCH have been designed using a 6-layer stackup with 3 signal layers, 2 power planes and one mixed power/signal layer (the stackup, from the top to bottom is: S1, Pwr, S2, Mix, Gnd, S3). Such PCB stackup allowed for achieving good signal integrity and low cost of the board.

Main FPGA

The central chip of the main board is the main FPGA – Altera's EP3C120 in BGA780 package, the biggest Cyclone-3 FPGA series model. This particular chip has been chosen because of the largest amount of the built-in RAM resources (necessary for the packet queues) in the low-cost FPGA segment. The main FPGA is configured directly by the main CPU, using the Passive Serial bootup mode. The same method is used to boot the timing FPGA. There is no dedicated configuration memory, the bitstreams are kept as regular files in the main CPU flash memory. The FPGA is powered from 3 different supplies: +1.2 V for the core, +2.5 V for the PHY interfaces and +3.3 V for the remaining I/O pins.

The FPGA interfaces with the following components:

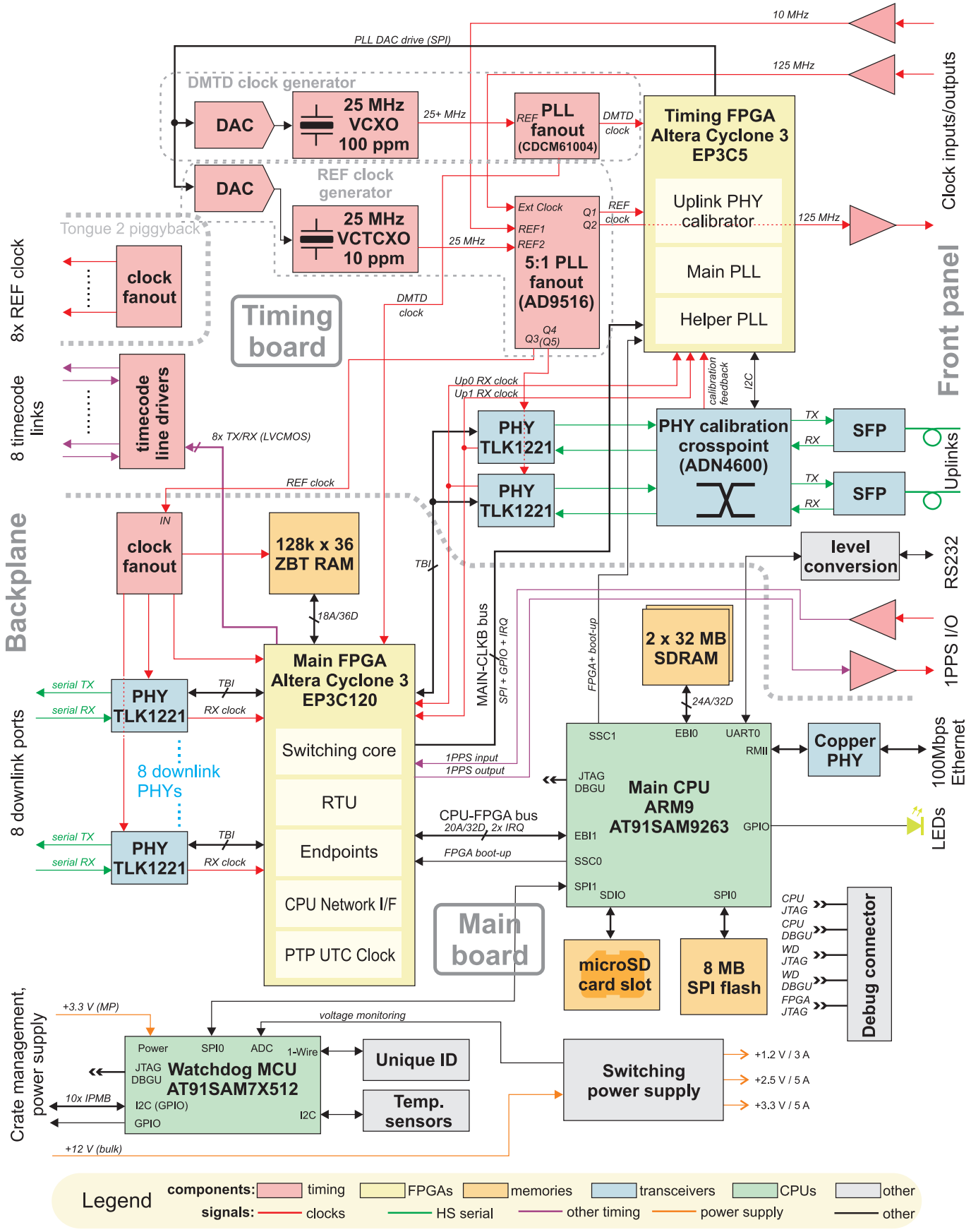

Figure 4.5. WRS MCH system block diagram.

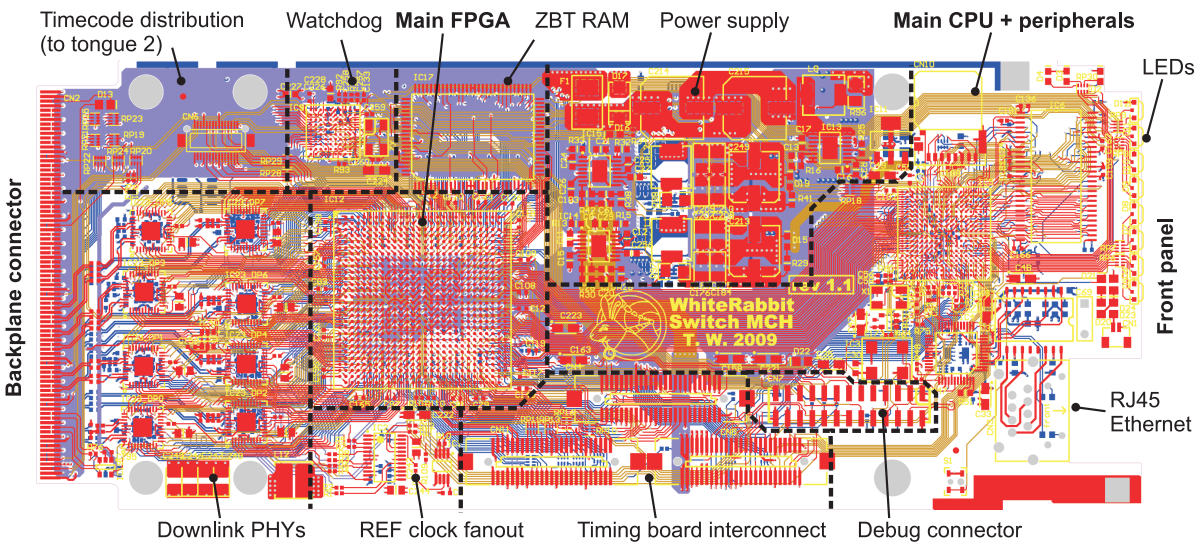

Figure 4.6. WRS MCH main board layout.

- the main CPU (EBI1 controller), using a 32-bit asynchronous bidirectional bus with external wait states. The blocks inside the FPGA are therefore a part of the CPU's memory map.
- the timing FPGA, using a serial SPI-like interface with several extra status and interrupt lines.
- outputs of the PLL oscillators which provide the reference and DMTD clocks.
- 8 downlink PHYs and 2 uplink PHYs (Texas Instruments TLK1221) connected using 8B10B-encoded parallel interfaces.
- 128k x 36 ZBT static RAM (GSI Semiconductor GS840Z36), where the switching engine keeps the MAC lookup database.
- PPS input/output ports on the front panel.
- 8 timecode LVDS drivers.

The most important part of the main FPGA is the deterministic Ethernet switching core incorporating 10 1000Base-X Ethernet MACs (WR Endpoints), a multiport non-blocking packet buffer with multiple priority levels (Swcore) and an address matching and filtering engine (Routing Table Unit – RTU). Another major component is the Network Interface Controller (NIC), enabling the main CPU to access the WR network via the 11th, *virtual* switch port.

Moreover, the main FPGA performs several important synchronization tasks, such as measuring $phase_{MM}$ on all the ports with DDMTD phase detectors, gathering PTP packets' transmission and reception timestamps and generating the PPS pulses and the UTC timecode. It also performs the calibration of the downlink ports when the MCH is used with the mini-backplane.

Ethernet PHYs

The WRS uses Texas Instruments' TLK1221 discrete gigabit transceivers, both for the uplink and downlink ports. The choice of discrete PHYs has been dictated by the high cost of FPGAs with built-in gigabit transceivers at the time when the WRS hardware was specified.

All PHYs use the same 125 MHz clock reference for transmission and all PHY data lines and RX clocks are connected to the main FPGA. In addition, uplink PHY RX clocks are also provided for the timing FPGA as a reference for the phase compensation PLL. Therefore, all ports are identical data-wise and can work as WR masters, but only the uplink ports can operate in WR slave mode.

Unfortunately, the TX/RX latencies of TLK1221 are non-deterministic (2 UI for the TX path, 8 UI for the RX path with 0.5 UI step), hence the PHYs require external calibration circuits. The WRS MCH has built-in calibration logic only for the uplink ports, since in the microTCA configuration, downlink ports are not used to provide timing information. In the standalone mode, the calibration must be performed by the mini-backplane.

Main CPU system

The WRS CPU is an Atmel AT91SAM9263 System-on-Chip microcontroller incorporating an ARM926E core running at 200 MHz. The ARM architecture has been chosen due to extreme popularity, ease of use, low price and excellent Linux OS support. The following CPU peripherals are used:

- External Bus Interface (EBI) 0 connected with a 32-bit bus to 64 MBytes of SDRAM memory operating at 133 MHz.
- EBI1 port, 2 external IRQ lines and DMA request line connected to the main FPGA.
- Serial Peripheral Interface (SPI) port 0, connected to an 8 MB DataFlash memory, containing the system bootloader and the fail-safe OS image.
- SPI1 port, connected to the Watchdog MCU.
- MultiMedia Card (MMCI) interface 0, connected to a micro-SD card slot. The microSD card stores the main OS file system.
- Ethernet MAC (EMAC) port connected through an external 10/100BaseT PHY (Davicom DM9161) to a RJ45 Ethernet port on the device's front panel.
- Two Synchronous Serial Controllers (SSC) connected to the configuration pins of both FPGAs.
- Several General Purpose I/O pins, used to drive the front panel LEDs, configure the AD9516 PLL and drive some FPGA configuration and reset pins.
- UART port routed via a level translator (MAX3232) to a mini-DB9 socket on the front panel.

The primary tasks of the main CPU are running the PTP protocol stack (WR-PTPv2 daemon) and the services handling inter-switch protocols (Spanning Tree, GMRP, etc.). The CPU also runs switch management applications (SSH management console, SNMP agent) and is responsible for the low-level hardware initialization (i.e. booting the FPGAs, setting up the clocking system, etc.).

4.2.4. Timing board

The MCH timing board, whose annotated layout is shown in fig. Figure 4.7 contains the clock recovery system (timing FPGA and oscillators), two uplink ports (with calibration logic), timing signal inputs and outputs, the backplane timecode line drivers and the RS232 port.

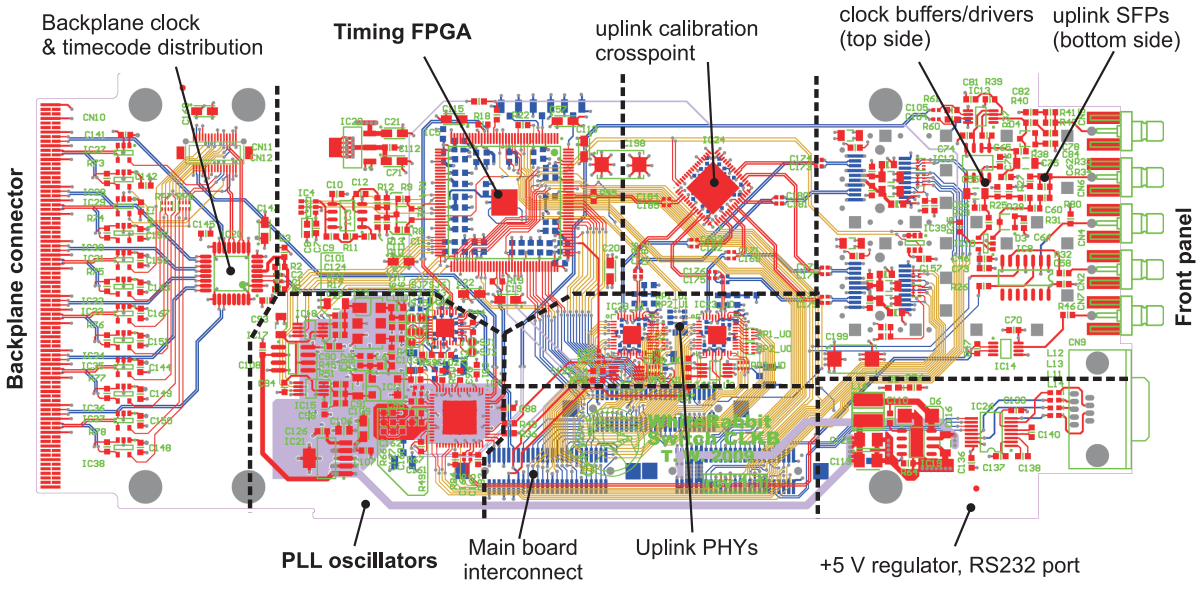

Figure 4.7. WRS MCH timing board layout.

Oscillators and clock distribution

There are two digitally tunable clock generators on the timing board: one producing the clean 125 MHz REF clock locked to the network reference clock and another, delivering the $(125 + f_{offset})$ MHz offset DMTD clock which drives all DDMTD phase detectors in the WRS.

The REF clock is derived from a 25 MHz fundamental frequency generated by a crystal oscillator. In order to fulfill the Sync-E clock stability requirements, a Mercury Crystal VM53S-series VCTCXO has been used, providing ±2.5 ppm center frequency accuracy and ±10 ppm tuning range. The tuning voltage is produced by a 16-bit SPI serial DAC (Analog Devices AD5662).

The VCTCXO drives the reference input of the AD9516 PLL synthesizer, which multiplies its frequency by a factor of 5, producing the 125 MHz REF clock. The second

reference input can be used to lock the REF clock to the clock coming from an external source, such as a cesium or GPS-disciplined oscillator. The AD9516 chip incorporates a 10-output clock fanout which is used to propagate the REF clock to the timing FPGA (as a timing PLL feedback and/or reference), the uplink PHYs, the main FPGA, the microTCA backplane fanout and the SMB connector on the front panel. Furthermore, the built-in fanout provides a programmable delay line on each output, which facilitates compensating the skew between the clock and data traces on the PCB. Since the REF clock drives almost all the logic in both FPGAs, the AD9516 PLL must be initialized by the main CPU before booting up the FPGAs.

The DMTD clock generator has a similar structure but uses different components. The oscillator is a 25 MHz CFPV-45 series VCXO with wider (min. ±100 ppm) pulling range necessary to obtain enough DMTD frequency offset. The PLL multiplier/fanout is a Texas Instruments' CDCM61004, chosen due to lower price and no need for programming. The DMTD VCXO is tuned with another AD5662 DAC.

In order to to protect the sensitive clock signals against interference and improve the signal integrity on complex PCBs, the majority of the clock distribution system uses LVDS/LVPECL differential signalling.

Timing FPGA

The central component of the timing board is a small Cyclone-3 FPGA (EP3C5) which implements two digital PLLs which control the oscillators described in the previous section:

- the Helper PLL, which locks the DMTD clock to the REF clock (in grandmaster mode) or one of the uplink RX clocks (in slave mode) with programmable frequency offset,
- the Main PLL, controlling the REF clock oscillator by locking it to selected uplink RX clock and applying a programmable phase shift to compensate for the phase drift. The main PLL is not used in the grandmaster mode.

The main reason for placing the PLLs in a separate FPGA was the sensitivity of the PLL phase detectors to FPGA internal routing delays, which have to be very tightly constrained in order to keep the same timing for every synthesis. Such constraining can be done much easier (and faster – from the developer's point of view) for a small independent FPGA, rather than for a big, highly populated device.

The other role of the timing FPGA is controlling the PHY calibration crosspoint and measuring the uplink transceivers' TX/RX delays with an independent DDMTD detector.

Uplink ports

The timing board hosts the two uplink PHYs (also TLK1221) and two SFP module sockets, interfacing the transceivers with the physical medium. Both fiber and copper

(1000BaseT) SFP modules are supported, although most copper SFPs cannot operate in synchronous mode, hence significantly reducing the synchronization accuracy.

The serial lanes between the PHYs and the SFPs pass through an Analog Devices' ADN4600 gigabit crosspoint switch which realizes the feedback of the PHY input/output serial streams (see 3.5.6) to the delay calibration DDMTD detector inside the timing FPGA.

4.2.5. Mini-backplane

The mini-backplane (MB) is a simple board providing 8 downlink ports' SFP sockets for the downlink ports of the WRS MCH and the necessary power supplies, enabling the WRS to operate without an expensive microTCA crate. The block diagram of the MB is depicted in Figure 4.8.

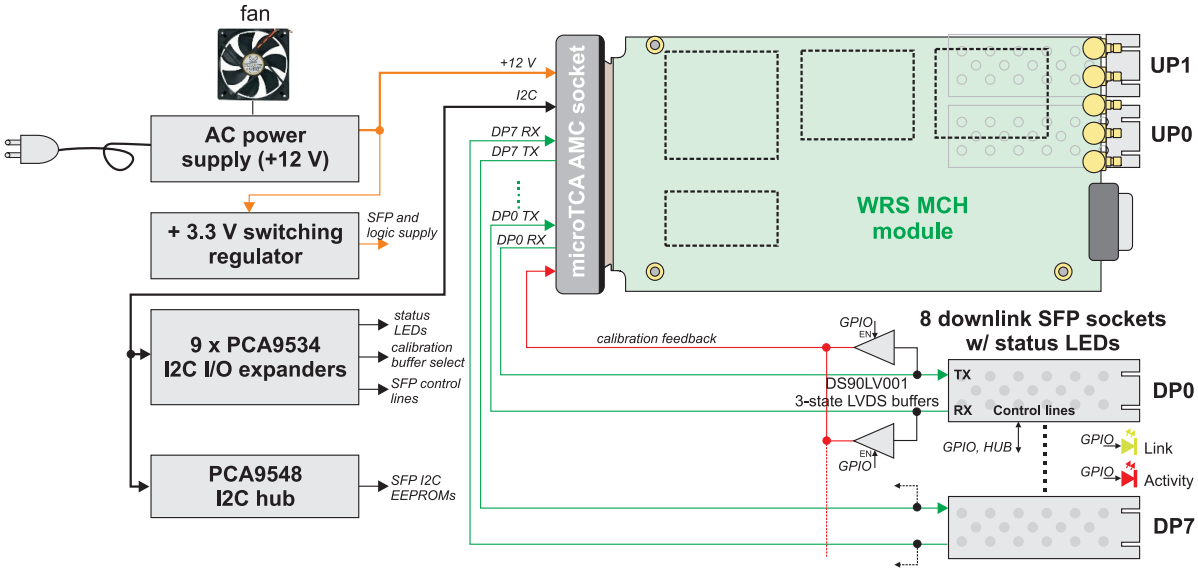

Figure 4.8. Mini-backplane block diagram.

The MCH is attached to the MB through a right-angle 170-pin AMC card connector. All the downlink differential lanes are routed directly to the SFP sockets. The calibration feedback signals for each port are produced by a pair of fast LVDS buffers (National DS90LV001), placed right next to the SFPs to prevent creating stubs on the high speed PCB traces. The tristated outputs of the calibration buffers are connected to a shared bus, thus forming a simple multiplexer. The output of the mux drives one of the very few available lines in the backplane connector, which are routed directly to the main FPGA (microTCA Fabric Update input).

Each port's link status and activity is indicated by two LEDs. The LEDs, along with the SFP control lines (TX_FAULT, LOS, TX_DISABLE, DETECT) and calibration buffers' output enable pins are driven by 9 I2C I/O expanders (NXP's PCA9534). The access to the SFP EEPROM memories is multiplexed with a PCA9548 I2C hub. All

I2C peripherals are hooked to a common I2C bus which uses one of the IPMB lanes to communicate with the MCH main CPU.

The MB includes an AC-DC converter, supplying the MCH board with +12 V. The SFPs, calibration and I2C logic are powered from +3.3 V, produced by another switching regulator. The MCH with MB are mounted in a nice enclosure, thus creating a stand-alone WRS (prototype shown in Figure 4.9). Due to the considerable heat dissipation in the MCH, it is recommended to use a small fan to force the airflow inside the device.

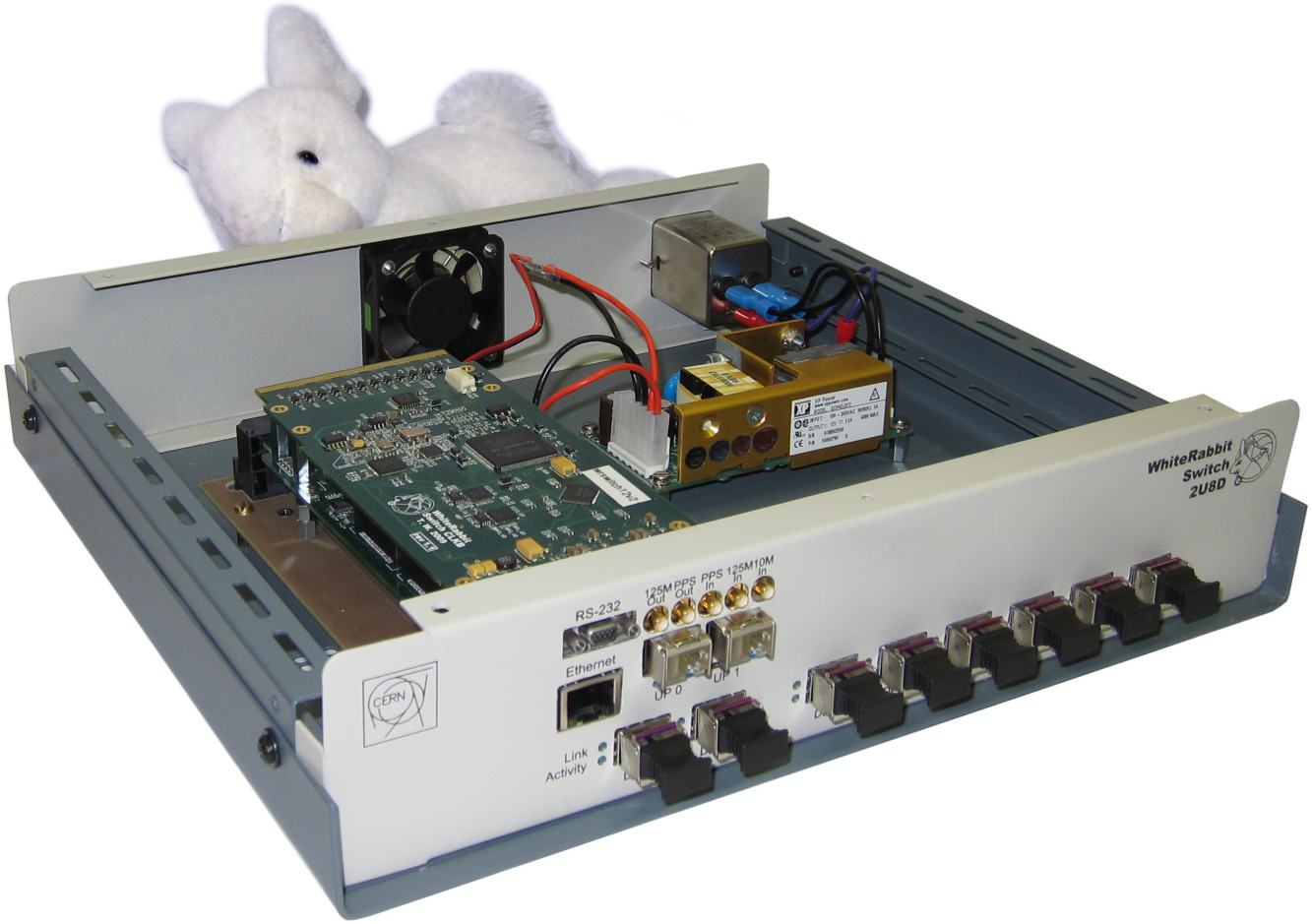

Figure 4.9. Prototype of the stand-alone WRS.

4.3. HDL Design

The FPGA HDL design has been split into several large modules. Figure 4.10 shows the hierarchy and connections of the blocks residing inside the FPGAs.

The system is built upon two interfaces: the memory-mapped Wishbone [23] System-on-Chip bus, through which the CPU can access the internal registers of each

module and the WR Fabric Interface (WRF), used to exchange Ethernet packets, eventually associated with auxiliary data such as PTP timestamps.

Translation between the asynchronous CPU bus and Wishbone is performed in the WB-CPU bridge block (`wb_cpu_bridge`). Some peripherals (RTU, NIC, TXTSU and the PPS generator) can also produce interrupts which are multiplexed into a single CPU IRQ line by the Vectored Interrupt Controller (`wb_vic`).

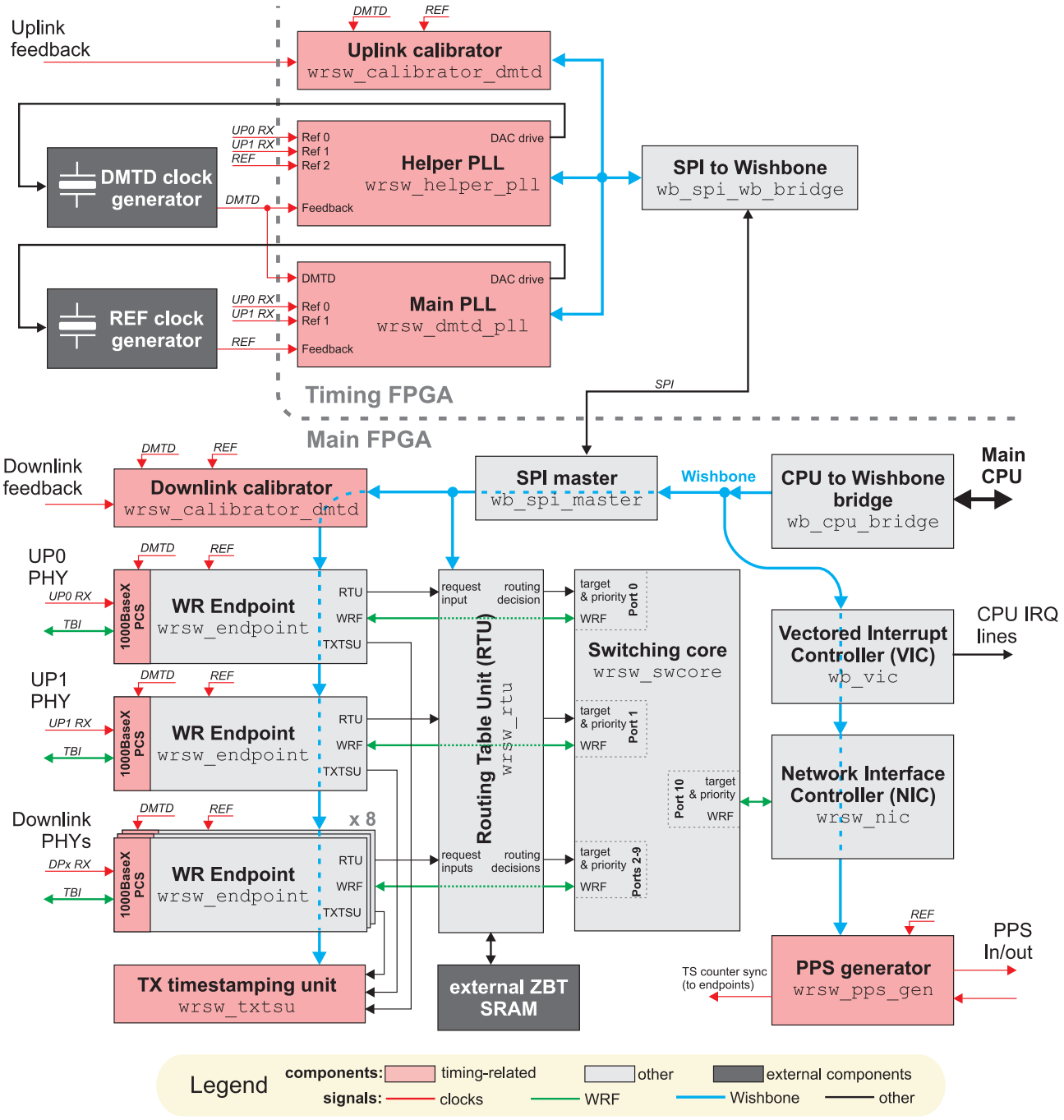

Figure 4.10. Block diagram of the FPGAs HDL design.

There are several clock domains in the design:

- REF clock - the timing reference clock, marshalling PHY data transmission and used to timestamp packets.
- RX clocks - the clocks recovered by the PHYs from incoming bitstreams.
- DMTD clock - used for DDMTD phase measurements.
- SYS clock - global system clock driving the entire packet processing pipeline and Wishbone bus. Its frequency is currently fixed to REF / 2 (62.5 MHz), but it needs not be synchronous to the other clocks as long as it is fast enough to sustain the minimum transfer rate on all WRF links.

The vast majority of the RTL (synthesizable) code has been written in VHDL'93 without any dependencies on third party libraries or IP cores. The Wishbone register layouts and bus slaves have been generated using the `wbgen2` tool [28]. Testbenches have been developed in SystemVerilog.

4.3.1. WR Endpoints

WR Endpoints (WRE, HDL entity: `wrsw_endpoint`) are responsible for the low-level Ethernet communication by adapting raw data streams coming from/to Ethernet PHYs into a form suitable for high-level packet processing. A simplified block diagram of the WRE is shown in Figure 4.11.

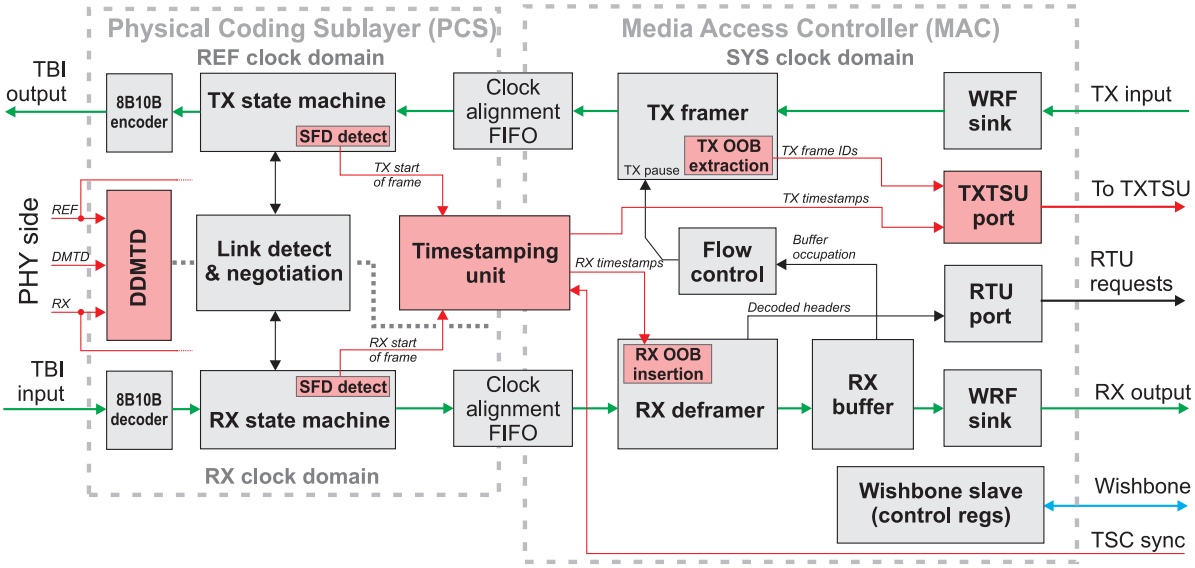

Figure 4.11. WR Endpoint block diagram.

The WRE comprises a gigabit Ethernet MAC optimized for low and deterministic packet latency integrated with a 1000Base-X PCS and blocks necessary to produce fine timestamps as described in section 3.5.5: a dual edge PTP timestamping unit and a DDMTD phase detector. The current version of WRE can communicate with the PHY either through an 8B10B-encoded Ten-Bit (TBI) Interface (used by the majority

of external serdes chips such as TLK1221) or an 8-bit parallel bus (available in most FPGA integrated transceivers, such as Xilinx's GTP).

On the host side, WRE provides several different interfaces:

- WR Fabric Interface (source port outputs received packets, sink port inputs the packets to be sent).
- Routing Table Unit (RTU) interface, delivering decoded packet headers (MAC addresses, priorities, VLAN IDs) for the filtering engine.
- TX Timestamping Unit (TXTSU), outputting the collected timestamps of outgoing packets.
- Wishbone slave, providing access to internal configuration and status registers.

The PCS part of the WRE is responsible for low-level framing, link detection, autonegotiation and 8B10B encoding and decoding. Its other important role is retiming the data from both TX and RX clock domains into a common, FPGA-wide SYS clock domain, done by means of two small asynchronous FIFO buffers. This allows multiple endpoints to work with asynchronous PHY clocks, while keeping the identical timing of the host-side interfaces – a feature essential for Ethernet switches. The PCS also contains the WR timestamper and DDMTD. Timestamps are passed to the MAC part for further processing, whereas the DDMTD phase measurements can be read by the host from one of Wishbone registers.

The MAC block implements the functionality of a typical Ethernet MAC extended with features improving the latency and facilitating operation inside an Ethernet switch, such as:

- VLAN tag insertion/deletion, allowing for trunking multiple VLANs in a single link and creating transparent VLAN access ports.
- Configurable buffering and flow control mechanism, which can selectively disable the sensitivity to PAUSE frames and RX buffering for a certain class of low-latency packets. This ensures the WRE will not block critical traffic even under a very high load.
- "Early" RTU output, providing the header data to the filtering engine as soon as possible, bypassing the RX buffer.

The WRE provides a number of VHDL *generics*, which allow the user to tailor the WRE features to match the particular requirements of a given application.

WR Fabric Interface

As previously mentioned, all inter-module packet transfers inside the switch are done through WR Fabric (WRF) interface [29]. WRF is a pipelined Wishbone-based unidirectional, point-to-point connection between a packet source (Wishbone master) and a packet sink (Wishbone slave). The data bus width is fixed to 16 bits as a compromise between the minimum system clock frequency (62.5 MHz for gigabit Ethernet) and the amount of required FPGA resources.

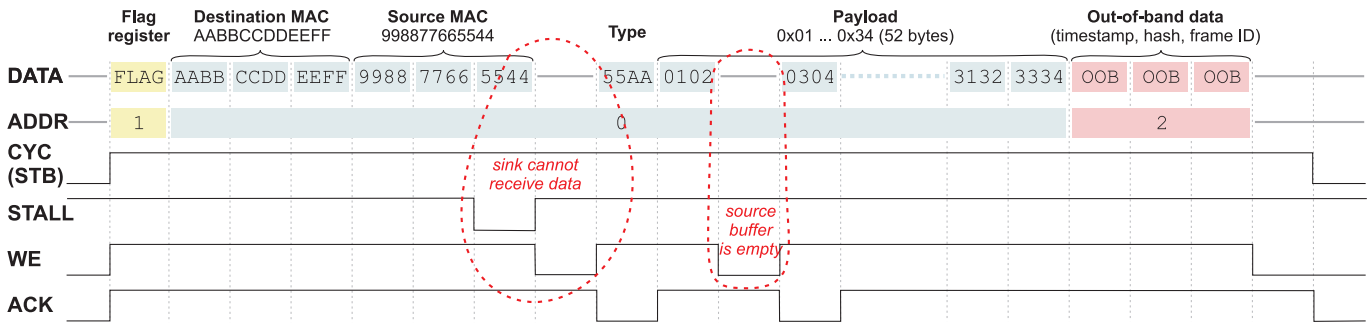

Figure 4.12. Transferring a packet using WRF interface.

An example of WRF packet transfer is depicted in Figure 4.12. Each transfer is contained within a single, locked, pipelined write cycle. Both sides of the link can throttle the traffic (the source via deasserting WE, the sink by asserting STALL). Every sink implements a standardized set of 3 registers:

- DATA – FIFO register to which the source sequentially writes the packet data.
- FLAGS – register for passing status information produced by the WRF source. For example, the WRE uses it to indicate the result of address filtering or the code of an error which caused the corruption of a received packet.
- OOB (Out Of Band data register) – another FIFO allowing to pass a user-defined data block associated individually with each packet.

Timestamp delivery

The OOB register has great importance for the PTP implementation, since it is used to transfer the packets' timestamps between the PTP engine running in the CPU and the hardware timestamping unit in all the endpoints. The way the timestamps are delivered to the CPU is illustrated in Figure 4.13.

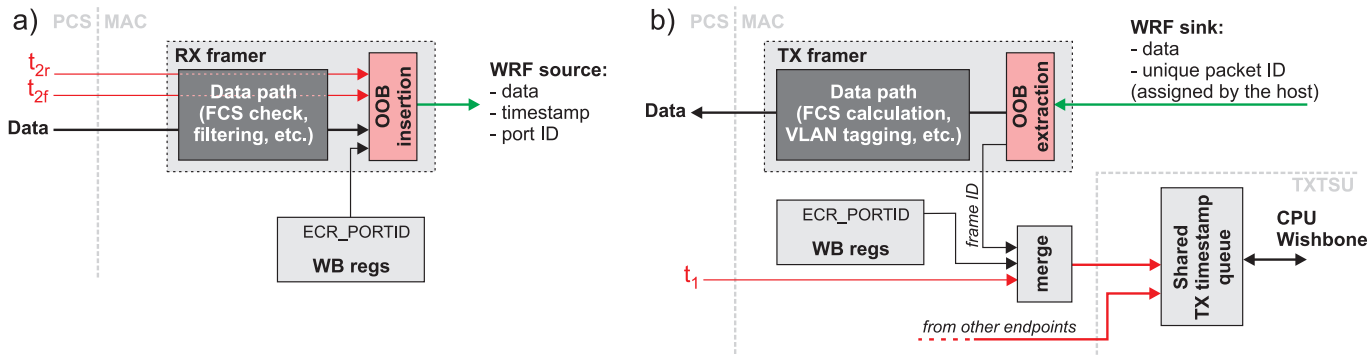

Figure 4.13. Passing TX and RX timestamps using WRF interface.

In the case of RX timestamps, the procedure is fairly simple (see 4.13a). The rising and falling edge timestamps produced by the PCS are associated with the unique number of the physical port which received the packet (set by the host via Wishbone).

Afterwards, the data is packed into 3 subsequent 16-bit words and transferred inside an OOB block. The CPU can then read it from the NIC RX buffer (4.3.4).

The process becomes more complicated for TX timestamps, as those cannot be delivered to the CPU using the WRF interface due to its unidirectionality. Moreover, PTP packets transmitted by the CPU are usually broadcast to all ports in the switch, thus a single PTP message may get a different TX timestamp on each port. This problem has been solved by assigning a unique identifier for each outgoing frame to be timestamped (see 4.13b). Endpoints can then associate the TX timestamp with the ID of the timestamped frame and the ID of the egress port. This information is passed to a shared TX timestamp queue (4.3.2) and read back by the CPU through Wishbone.

The presented method can easily produce precise timestamps for every single incoming and outgoing packet without any decrease of data throughput. That means no need for hardware detection of PTP frames, often met in commercial MACs. Furthermore, the WRE can be configured to embed a hash of each frame payload with both TX and RX timestamps, enabling the WR switch to automatically measure the residency time individually for each frame or turning it into a latency measurement device for testing other networking gear.

4.3.2. PPS Generator and TX Timestamping Unit

The PPS Generator (PPSG, `wrsw_pps _gen`), shown in Figure 4.14, serves as a local real-time clock, providing an absolute notion of time for all the blocks inside the WRS.

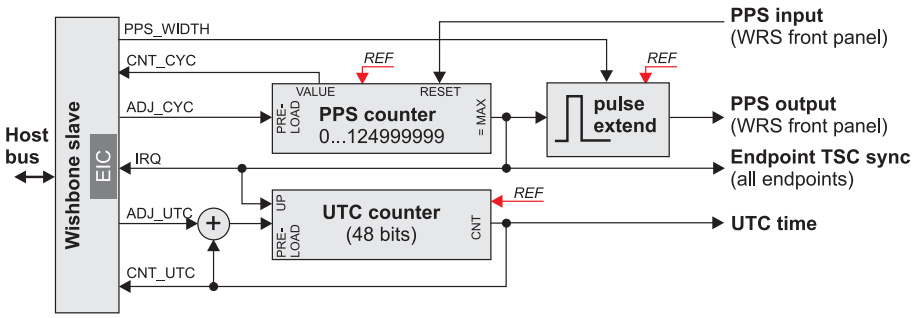

Figure 4.14. Block diagram of the PPS Generator.

The PPSG contains a set of Wishbone-controlled counters which produce the local PPS signal and implement a local real-time clock. There are two counters, which can be read and adjusted by the CPU via the Wishbone bus:

- PPS counter, counting from 0 to 124999999 at every REF clock cycle, thus giving a period of one second. When the maximum value is reached, the counter produces a pulse of programmable width on the PPS output. It's other purpose is synchronizing the timestamping units in all the endpoints, so all the timestamps in the systems are referenced to the same time scale.

- UTC counter, which counts full seconds and interrupts the CPU at the beginning of every second.

Both PPSG counters are adjusted by the PTP daemon (see 4.4.3) during coarse synchronization (3.5.7, stages 1 and 2). Once the initial synchronization is done, the counters are free running and the fine offset is corrected by phase shifting the REF clock.

The UTC counter is adjustable by simply adding an offset to its actual value. Adjustment of the PPS counter is done by setting it up to a given value right after it overflows, hence an initial value of x is equivalent to increasing (decreasing) the counter by $-x$. This simplifies the HDL, but the adjustment is not performed immediately (it can take up to 2 seconds).

When the WRS operates in PTP grandmaster mode, the beginning of each second is determined by an externally provided PPS signal (which simply resets the PPS counter). The UTC time is usually received from the serial port and set by pre-loading the UTC counter by the CPU.

The title of this subsection mentions another block – the TX Timestamping Unit (`wrsw_txtsu`). Despite a seriously sounding name, it's a simple shared FIFO register collecting all the TX timestamps (associated with originating port and frame IDs) from all the endpoints and delivering them to the CPU through Wishbone. TXTSU generates an interrupt when the FIFO is not empty to avoid polling in the CPU driver.

4.3.3. Routing Table Unit and Switching Core

These two very complex, tightly coupled modules implement the actual Ethernet switching engine. Since they do not participate in synchronization, their detailed description falls out of the scope of this thesis.

The RTU (`wrsw_rtu`) processes the header of an incoming packet and determines the destination port(s) to which the packet shall be forwarded. The input of the RTU are the decoded contents of the packet's header: source and destination MAC addresses, VLAN ID and PCP (Priority Code Point) field. The RTU searches for a hash table entry corresponding to the input data and decides whether the packet shall be forwarded or dropped. The hash table is stored in an external static RAM chip. If no matching entry has been found, the packet is broadcast and its header is forwarded to the RTU daemon running on the Main CPU. The software eventually updates the hash table (802.1D Learning Process).

The RTU outputs its decision in the form of a bit vector called Destination Port Mask (DPM), where each bit represents a certain port in the Swcore. If all the bits are set, the packet shall be broadcast to all the ports, if all the bits are zeroes, the packet shall be dropped. The RTU also evaluates the final priority of the packet, which can be assigned per-port, per-MAC, per-VLAN or simply extracted from the PCP field in the 802.1q header.

The Switching Core (`wrsw_swcore`, abbreviated as Swcore) is a generic, protocol-agnostic multiport packet queue. It receives packets from the 10 Endpoints through WRF interface and directs them to the ports determined by the RTU. There is also an 11th port, which enables the CPU to access the WR network via the NIC (4.3.4). The basic properties of the Swcore are listed below:

- Store-and-Forward architecture with shared, page-addressable packet buffer implemented using FPGA integrated RAM blocks.
- Non-blocking, achieves multi-gigabit bandwidth on each port.
- Configurable number of WRF ports.
- Up to 8 priority levels.

The Swcore is currently being upgraded with another module – Cut Through Switching Unit, which will route critical control traffic in cut-through mode, minimizing the routing latency.

4.3.4. CPU Network Interface Controller

The Network Interface Controller (NIC, `wrsw_nic`) connects one of the ports of the Swcore to the Main CPU, enabling the CPU to send and receive packets from/to the WR network. The NIC can be therefore seen as a network card integrated inside the WRS.

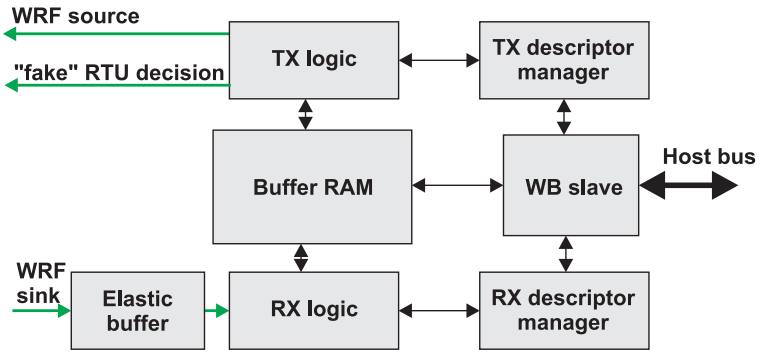

Figure 4.15. Block diagram of Network Interface Controller (NIC).

The NIC layout, shown in Figure 4.15 resembles a simple PC network card. Packets are stored in a shared buffer, which can be an independent RAM block or a part of the system memory (accessible via DMA). The buffer is governed by two descriptor tables (one for transmitted and one for received packets). Descriptors containing valid packets are marked as ready by setting an appropriate bit.

In order to send a packet, the CPU searches for an empty TX descriptor in the table and allocates a region in the RAM buffer big enough to store the packet's contents. Afterwards, it sets up the TX descriptor by setting:

- address of the beginning of the packet in the RAM buffer and length of the packet,
- destination port mask, indicating to which ports the Swcore shall send the packet,

- if the packet has to be timestamped, also an unique packet ID. The ID is embedded in the OOB block and associated with TX timestamp by Endpoint(s).

Once the descriptor is set up, it can be marked as ready. The NIC will then automatically start the transmission and indicate its completion (or an error) by producing an interrupt.

The reception path requires the CPU to initialize a number of empty RX descriptors pointing to free regions in the packet RAM prior to receiving any packets. When a packet comes to the WRF sink, the NIC takes first available free RX descriptor and writes the contents of the packet to the buffer, starting at the address specified in the descriptor. Once the reception is finished, the pending descriptor is updated with the length of freshly received packet, its RX timestamp and ingress port ID, extracted from the OOB block. Completion of reception (both successful and failed) also triggers an interrupt.

The operating system running on the Main CPU sees the NIC as 10 independent network interfaces, each representing a single physical port of the WRS. This is accomplished by generating "fake" DPM masks for TX packets and identification of ingress ports by checking the OOBs of RX packets.

4.3.5. DDMTD phase detector implementation

The theoretical model of the DDMTD phase detector presented in chapter 3 cannot be used in most practical implementations, since real-world clock signals always contain some jitter which results in glitches around transitions in the DDMTD output. These glitches can significantly hinder the accuracy of phase measurements (especially for poor quality clocks, produced by PHYs with oversampling CDRs). In order to minimize phase measurement error, a deglitching algorithm is required. Such algorithm takes the waveform of the glitchy transition $x(t)$ and estimates the time t_t (further called **phase tag**) at which a glitch-free transition should have occurred if both DDMTD clocks were ideal. Using SystemVerilog modelling techniques, the author tested 5 deglitching algorithms which were relatively simple to implement in FPGA. The comparison of the algorithms is presented in Figure 4.16.

Each of the algorithms was fed with a set of 100 glitchy transitions produced by a synthesizable DDMTD model from two clocks which were injected with a pre-defined amount of gaussian jitter. The measure of performance was the worst-case phase tag error (expressed as the difference between the deglitched and ideal phase tag, scaled to the full period of the signal). Surprisingly, the best results were produced by the easiest algorithm to implement (bit median). Its maximum phase error (collected for 100 phase tag samples) was at 0.15% of the full scale which corresponds to 12 ps at $T_{ref} = 8$ ns.

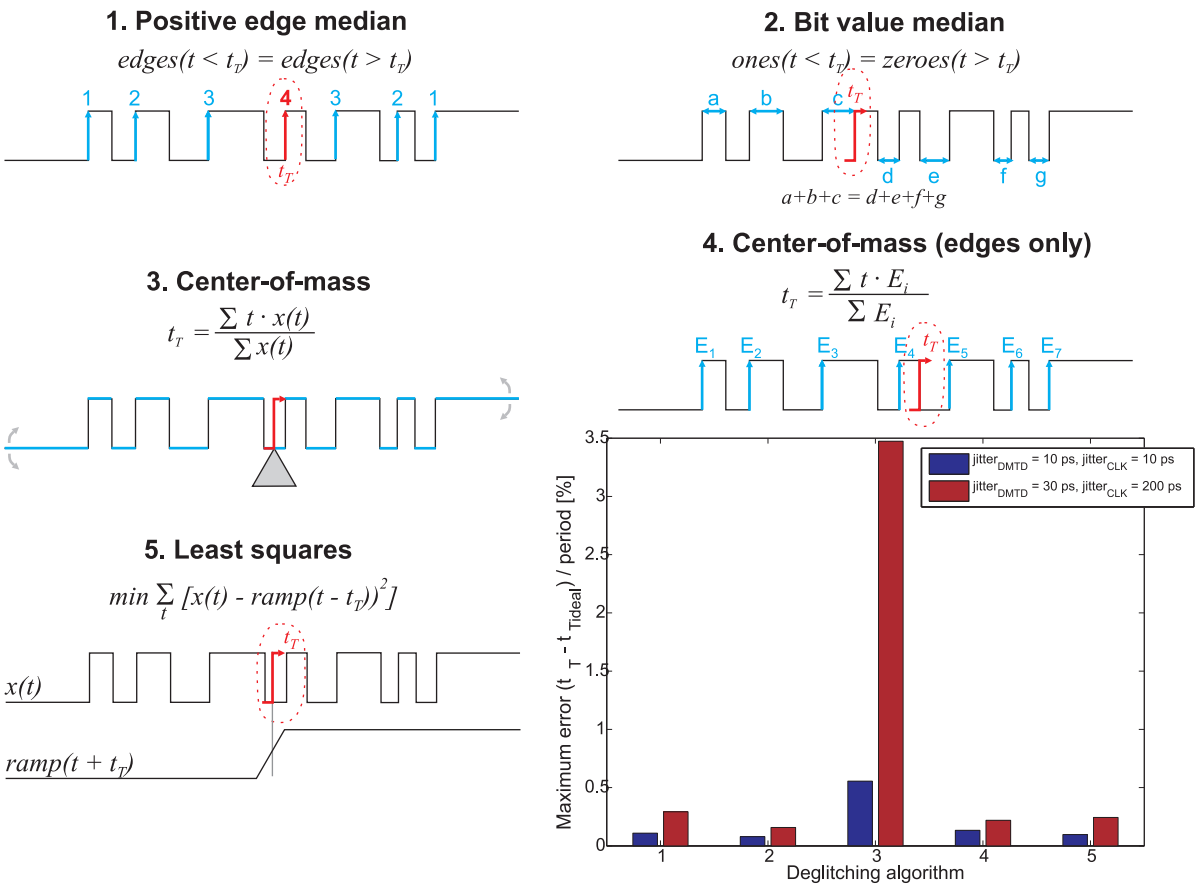

Figure 4.16. Comparison of DDMTD deglitching algorithms.

The actual HDL implementation of the deglitcher (HDL entity: `dmtd_with_deglitcher`) is based on a free-running counter and a state machine (FSM) shown in Figure 4.17. The FSM has three states:

1. `wait_stable_0` – initial state, waiting until the input signal stabilizes at logic 0. When the input has been stable for a predefined number of clock cycles (*stable 0 threshold*), the FSM changes its state to `wait_edge`.
2. `wait_edge`, which waits for the first glitch in the input, indicating a beginning of a transition to logic 1. Afterwards, the FSM stores the current value of the free-running counter in register t_t (phase tag counter) and jumps to `got_edge` state.
3. `got_edge`, which increments t_t register every time the input is at logic 0. After the input has been stable at a high logic level for a given time (*stable 1 threshold*), the FSM assumes that the transition area has ended. Subsequently, the value of t_t is outputted as the deglitched phase tag, the point at which $x(t)$ holds the same number of zeros to the right of t as the number of ones to the left (proof is left to reader). The FSM then goes back to `wait_stable_0` state and the whole cycle is repeated.

Since both the free running counter and the t_t counter have finite lengths, the actual implementation must properly handle their overflow. In the WRS, this has been achieved by fixing the value of N from equation 3.8 to a power of 2 causing the period of the

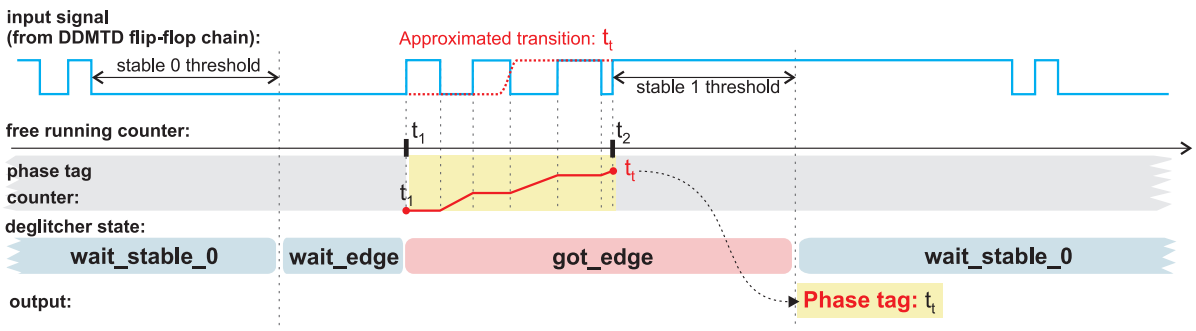

Figure 4.17. DDMTD deglitching state machine.

deglitcher input signal to be also a power of 2. In such case no special overflow handling is required (as long as the range of the counter is at least of the same size as the period of the DDMTD output), thanks to the properties of two's complement encoding of the counter value.

4.3.6. Helper PLL

The Helper PLL (HPLL, HDL entity: `wrsw_helper_pll`) produces the offset frequency (DMTD clock) used by all DDMTD detectors in the WRS. The block diagram of the HPLL is shown in Figure 4.18.

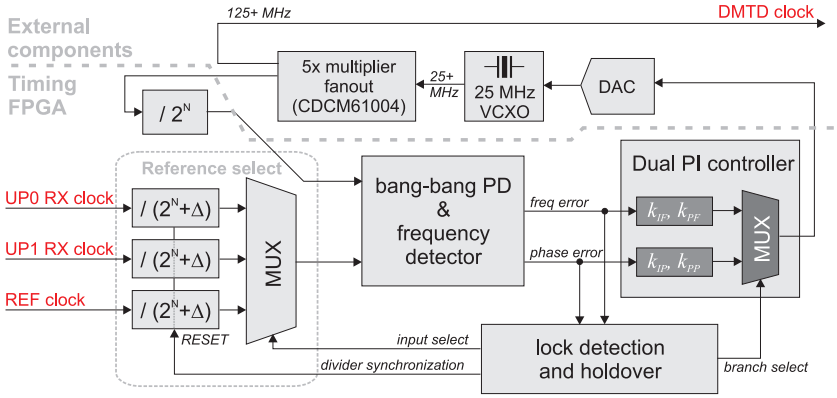

Figure 4.18. Helper PLL block diagram.

The offset clock is generated by an external VCXO oscillator tuned by a DAC. It can be locked either to the REF clock (in grandmaster mode) or to one of the RX clocks (boundary clock mode). The configuration of the reference and feedback clock dividers allows for obtaining offset frequencies expressed by (4.1):

$$f_{offset}\ [\text{ns}] = 125\ [\text{MHz}] \cdot \frac{2^N}{2^N + \Delta} \tag{4.1}$$

The upper boundary of f_{offset} is determined by the tuning range of the VCXO and the required phase measurement resolution. The lower boundary is constrained by the bandwidth of the Main PLL, since f_{offset} also defines the sampling rate of phase

measurements. In the default configuration, the WRS uses $N = 14$ and $\Delta = 1$, thus giving an offset of 7.628 kHz.

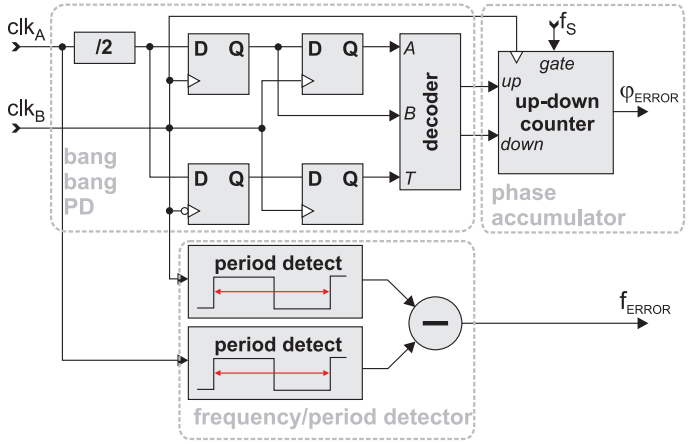

Figure 4.19. Structure of HPLL phase/frequency detector.

HPLL incorporates two different phase/frequency detectors, depicted in Figure 4.19:

- period detector, a low-gain linear detector which compares periods of the divided input clocks. Its sensitivity is limited to a single REF clock cycle.
- Asynchronous Accumulating Bang-Bang phase/frequency detector (BB PFD) – a high gain nonlinear detector with sensitivity better than single clock cycle.

The output of both detectors is fed to a dual channel Proportional-Integral controller. The first channel is used to quickly achieve the desired frequency offset, taking advantage of the linearity of the period detector. Its gain can be therefore quite high without the risk on instability or excessive overshoot.

When the frequency offset reaches the programmed value, the HPLL resets the divider counters (to minimize the phase shift between the divided clocks and thus speed up the locking even further) and switches to the second PI channel which processes the output of the BB detector. Since BB detectors are extremely nonlinear – they only provide a qualitative information about the phase shift (too small, correct, too big), the gain and bandwidth have to be much lower. For certain combinations of divider values, it is even impossible to lock the PLL using only the BB PFD.

The PI output is adjusted with a pre-defined bias to match the mid-scale of the oscillator's control voltage and outputted via an SPI port to the DAC which tunes the VCXO, thus closing the feedback loop.

The HPLL also incorporates a holdover unit which automatically switches the reference input when the currently selected reference source has failed. All HPLL parameters can be set by the Main CPU via Wishbone and updated at any moment. Wishbone is also used to check the lock status (the PLL assumes to be locked when the error value stays below a pre-programmed threshold for a certain number of samples). The CPU can also monitor all important signals inside the HPLL using a dedicated FIFO to obtain

an impulse/step response, a very useful feature for debugging the design and optimizing the PI parameters.

4.3.7. Main PLL

The Main DDMTD PLL (DMPLL, HDL entity: `wrsw_dmtd_pll`), depicted in Figure 4.20 is the main component of the WRS clock recovery system.

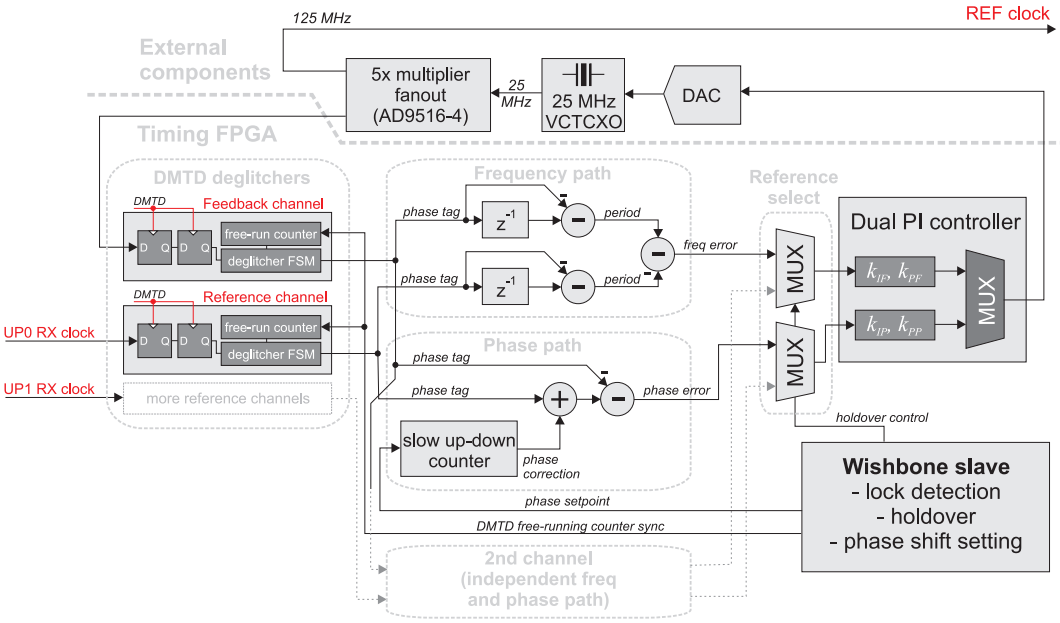

Figure 4.20. Block diagram of the Main PLL.

The DMPLL takes one of the PHY RX clocks as a reference, cleans it up and applies a programmable phase shift obtained with WR-PTP. The resulting REF clock is an in-phase copy of the PTP grandmaster clock (as long as the value of $offset_{ms}$ produced obtained with PTP is accurate). Each input channel (both feedback and reference) has an independent DDMTD with the deglitching state machine described in section 4.3.5. The free-running counters in all deglitchers are synchronized, so all the inputs share the same time base. Phase tags produced by the DDMTDs are the entry point for the control algorithm.

Similarly to the HPLL, the Main PLL also has two frequency and phase channels per reference input, so the phase lock can be achieved much faster. The frequency path subtracts subsequent phase tags generated for each input clock to obtain the periods of reference and feedback signals. Thanks to the properties of two's complement encoding and the power-of-two constraint on the DMTD offset divider, the frequency error can be calculated by simply subtracting the periods.

The phase path contains the actual phase shifter. The shifting is done by adjusting the value of reference tag with a setpoint provided by the Main CPU via Wishbone bus according to equation 4.2:

$$phase\_error = tag\_ref + phase\_correction - tag\_feedback. \quad (4.2)$$

Note that the phase setpoint is not applied directly – there is an up-down counter which slowly ramps the correction value until it reaches the programmed setting. This is to prevent the PI controller from "ringing" in response to a step in the phase error. In the current WRS design, the DMPLL achieves a phase shifting speed of 10 rad per second which is way above the needs of PTP phase tracking.

Each reference channel can have a different phase setpoint. The resulting phase and frequency errors are fed to a dual-branch PI controller of the same design as in the HPLL and similar bandwidth parameters. The phase branch bandwidth is reduced to filter out the noise in the phase tag values generated in the deglitchers. Since the DDMTD detectors are very linear, the overall PI gains can be much higher.

Both PI controllers (HPLL and DMPLL) are equipped with an "anti-windup" mechanism which disables the integrator when the DAC output is close to the minimum or maximum level. This is yet another measure for improving the locking time.

The reference input can be selected by multiplexing the values of phase and frequency error delivered to the PI controller. This is done by an integrated holdover unit, similar to the one used in the HPLL. All parameters of the Main PLL: PI gains, phase setpoints, lock detection thresholds and the choice of reference clock are also configurable through Wishbone.

4.3.8. PHY latency calibrator

The WRS also uses DDMTD phase detectors to calibrate the TX/RX latencies of both uplink and downlink ports. Due to the limitations of the hardware, two calibrators (HDL entity: `wrsw_calibrator_dmtd`) are required: one inside the timing FPGA, measuring the uplink PHY latencies and one residing in the main FPGA, taking care of the downlink ports. Both have the same structure, shown in Figure 4.21.

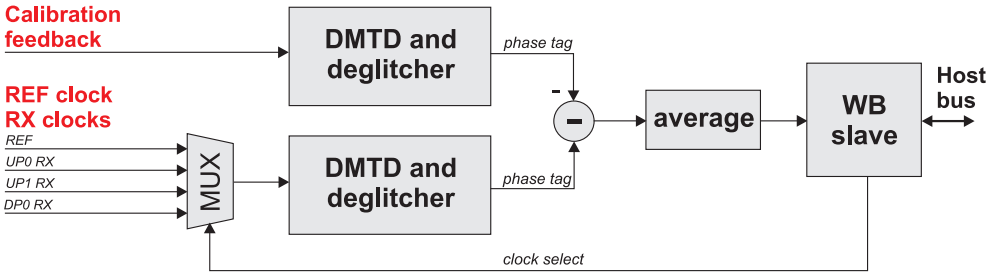

Figure 4.21. DDMTD-based PHY latency calibrator.

The uplink feedback signal is provided by the ADN4600 crosspoint switch, whereas the downlink feedback comes from a mux built with tri-state LVDS buffers. Measuring TX latencies is done by comparing the phase of the feedback signal with the REF clock.

The RX latencies are obtained by comparing the feedback with the RX clock of the PHY being calibrated.

Since the feedback signal involves much higher jitter level than the input clocks, the raw phase value has to be averaged. The CPU can read the latency value through Wishbone. Note that the calibrator does not select the source of the feedback signal – this has to be done via an I2C bus (controlled by the Timing FPGA or the Watchdog MCU). The calibration of one PHY takes around 200 milliseconds. Therefore, a single calibrator unit can be shared between all the PHYs to reduce the FPGA resource footprint.

4.4. Software

The last part of the WRS description covers the embedded software running on the Main CPU. The software works under the Linux operating system and manages all the high-level aspects of the operation of the WRS (including the actual WR-PTP protocol engine). It has been designed as a set of independent applications using a Remote Procedure Call (RPC) mechanism for communication. Among these applications are the Hardware Abstraction Layer (HAL), PTP and RTU daemons, which access the WRS hardware through custom developed drivers.

4.4.1. System architecture

The Main CPU runs a typical Embedded Linux-based system, built upon the Buildroot [10] Linux distribution. The architecture of the OS and its applications is shown in Figure 4.22.

The system is based on a 2.6.36 kernel compiled for Atmel AT91 architecture, with the board support files derived from the Atmel's AT91SAM9263-EK development kit. The kernel has been extended with WRS-specific board setup code which provides `platform_data` structures containing the configuration of FPGA peripherals, sets up the EBI1-FPGA bus and initializes the FPGA VIC.

Both the kernel and the root filesystem are loaded by the U-Boot boot monitor. Currently it is possible to boot up the device either from a microSD card or through the management Ethernet port using BOOTP/TFTP protocols. The bootloader also performs the very low-level CPU initialization (such as initializing the CPU PLL and the SDRAM controller).

The networking-related components of the WRS are managed by the two kernel modules:

- `wr-rtu`, which abstracts the RTU hardware for the RTU daemon.
- `wr-nic`, providing 10 network interfaces (`wruX` for the uplinks and `wrdX` for the downlinks) which are bound to the respective physical ports of the WRS. The NIC

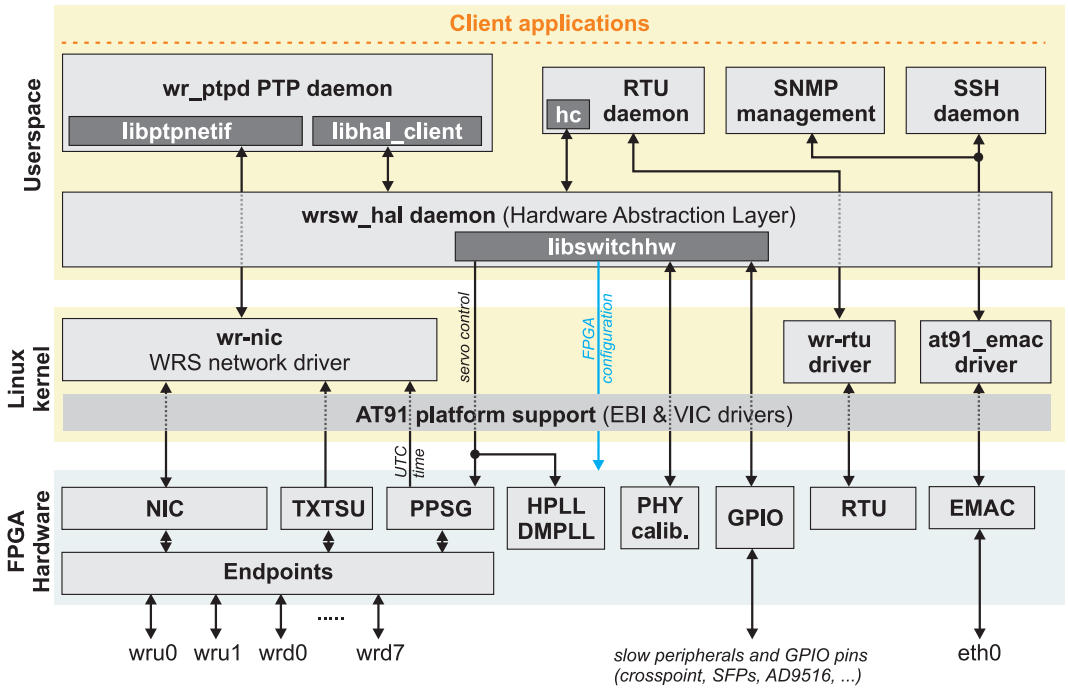

Figure 4.22. WRS software architecture.

driver also handles the coarse packet timestamping, polling of the DDMTD phase and the transmission/reception of the calibration pattern on each port.

The userland of the OS hosts several applications (mostly written in C language) which control various aspects of WRS operation. The applications communicate between each other through a simple Unix socket-based RPC (Remote Procedure Call) mechanism implemented by the `libwripc` library. The two programs which perform important synchronization tasks are the Hardware Abstraction Layer (HAL) daemon and the PTP daemon, both described in the following subsections. Another notable application is the RTU support daemon (`wrsw_rtud`), which handles MAC address learning, VLAN configuration and packet filtering by configuring the RTU. The WRS also runs a SNMP agent and SSH server, facilitating network management and diagnostics, as well as it provides few stand-alone diagnostic tools (for example a synchronization status monitor or an RTU diagnostic application), which are accessible through the SSH console.

4.4.2. Hardware Abstraction Layer

The Hardware Abstraction Layer daemon (`wrsw_hal`) manages all hardware peripherals of the WRS and acts as a bridge between the hardware modules and the high-level applications such as the PTP daemon. It performs the initialization of the WRS, provides numerous user-space device drivers and handles the low-level timing tasks, such as the PHY calibration and phase tracking.

The HAL is the first process launched during the start-up of the WRS and its first task is to initialize the WRS hardware. The initialization process begins with the setup of

the external components which must work prior to configuring the FPGAs (for example the AD9516 PLL). Afterwards, the HAL boots up both FPGAs with the firmware stored in the file system and sets up the interrupt-less user space peripheral drivers included in the `libswitchhw` library (i.e. the HPLL and DMPLL, calibrators, the PPS generator, etc.). In the following step, the HAL loads the kernel modules with drivers for the VIC, NIC and RTU. Finally, it sets up the physical ports of switch by assigning physical addresses and configuring the timing role of each port (master, slave, grandmaster, etc.).

Timing-wise, the HAL daemon acts as an intermediate layer between all the hardware participating in synchronization (PPS generator, calibration logic and feedback circuits, PLLs and endpoints) and the PTP software. The HAL API is a set of simple C functions, which can be called by client applications (for example, the PTP daemon) through `wripc` mechanism. The functions which are essential for synchronization are listed below:

1. `halexp_lock_cmd` – controls Sync-E clock recovery by programming the HPLL/DMPLL to lock to a given clock source. It can be also used to poll the lock status of the PLLs, configure gain parameters and clock source switchover.
2. `halexp_pps_cmd` – coordinates PLL phase shifting with PPSG time adjustments. The values of phase shift and offset corrections (provided by the client application in seconds) are converted to native hardware units and used to program the DMPLL and PPSG.
3. `halexp_calibration_cmd` – governs the entire PHY calibration process: controls transmission and detection of calibration pattern, configures the crosspoint and calibration buffers and measures PHY delays by polling the calibrator DDMTDs. Raw DDMTD phase shifts are converted to TX/RX delays, on the basis of the parameters of a particular PHY declared in the HAL's configuration file.
4. `halexp_query_port_state` – allows for polling the current state of a certain physical port, including syntonization status and timing parameters: TX and RX delays, $phase_{MM}$ and $phase_S$.

The HAL can also orchestrate the access to the timing hardware between multiple clients. For example, one can have an active PTP link working in the background and simultaneously tune the DMPLL parameters using `halexp_pll_cmd` function. This feature was intensively used during the optimization of the clock recovery PLL.

4.4.3. PTP Daemon

The PTP daemon encompasses a PTP protocol engine and WR protocol extensions, which are necessary for achieving sub-nanosecond accuracy. The application is a heavily modified fork of the `ptpv2d` project [25]. The most important modifications are:

- WR protocol stack, adding the calibration of the PHYs and physical layer syntonization.
- Multi-port operation.

- More precise internal timestamp format (UTC, nanoseconds, picoseconds), necessary for calculating the offset with enough accuracy.
- Modified servo, which uses a hardware PLL to maintain syntonization and only performs periodic offset adjustments.
- System-independent code: all system and networking functions are wrapped inside platform-dependent libraries which can be easily implemented on different embedded architectures (with and without an operating system).
- Fixed-point calculations (no floating point used).

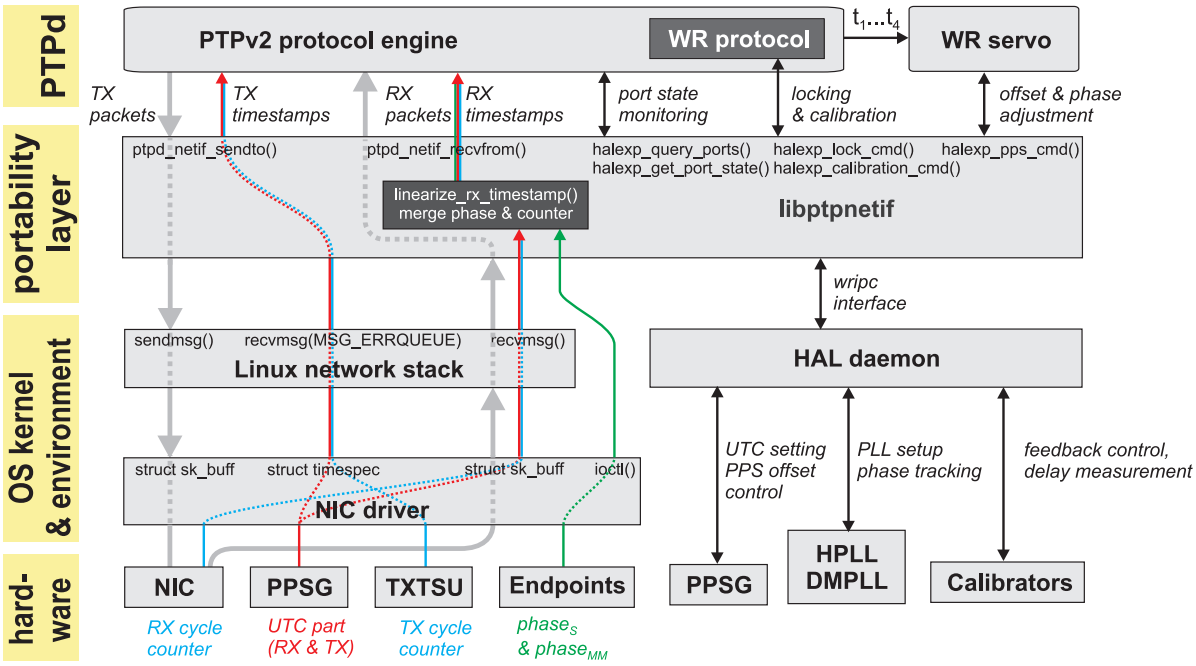

Figure 4.23. PTP daemon and synchronization software architecture.

The architecture of the PTPd and other software components participating in the synchronization process is shown in Figure 4.23. PTPd daemon lies on the top of the software stack and runs as an stand-alone application. WR protocol is handled by a separate state machine (defined in [11]), which is executed during UNCALIBRATED state of the main PTP FSM, therefore the support for WR has no significant impact on the standard PTP part of the daemon code. Gathered timestamps are fed to a detachable servo, which calculates the clock offset and master-slave delay. Due to the architecture of the servo, which relies on hardware-based syntonization, WRS cannot operate as a standard PTP slave. This limitation can be, however, overcome by dynamically switching between a WR and standard servo, depending on the mode of the remote master.

Another feature introduced for the WRS is the multiport mode, with a provision that all the ports are independent PTP clocks with fixed roles (master/slave). WRS PTPd implementation ensures that a failure on one of the ports port has no impact on the operation of the remaining ports by launching an independent instance of the PTP

engine per each port. The delays and offsets are calculated independently for each uplink and the master selection and switchover is performed outside PTPd by the hardware (DMPLL). This ensures that a failure of one of the uplinks will not harm the accuracy.

The presented model assumes that the PTPd never communicates directly with the OS and the hardware, but through a platform-dependent portability layer. The networking and timing APIs are provided by the libptpnetif library. In the WRS, libptpnetif also serves as a gateway to the HAL daemon `wripc` API by implementing wrappers for all the functions exported by the HAL. On OS-less platforms (such as the WR PTP Core [13]), the portability layer may also need to implement some basic POSIX functions.

The OS layer includes the Linux network stack and the NIC driver, which take care of transferring timestamped packets the HAL daemon, which controls the clock servo and the calibration process. Timestamping is distributed among several hardware and software components, mostly due to hardware constraints (shared UTC counter) and limitations of data structures used in the Linux kernel:

- TX timestamps are generated entirely by the NIC driver, since their fine (picosecond) part is always equal to 0. The nanosecond part is read from the TXTSU FIFO upon transmission of each packet, whilst the UTC time is obtained from the UTC counter in the PPS generator.
- Production of RX timestamps is done in two phases: the UTC part (from the PPSG) and the nanosecond part (from the RX descriptor header in the NIC) are stored within a `timespec` structure (resolution of which is limited to 1 ns) and passed with the received packet to libptpnetif. The information whether the rising edge counter is ahead of the falling edge counter (see 3.5.3) is encoded on the unused MSB of the 32-bit nanoseconds field. The actual merging algorithm (3.5.5) is implemented in libptpnetif, which obtains the missing phase information from the NIC driver through a custom `ioctl()` call.

The PTPd also incorporates a debugging mechanism, which can be used to monitor the status of synchronization through a dedicated `wripc` call.

4.5. Performance measurements

In order to evaluate the performance of the time and frequency transfer, the author built a measurement system shown in Figure 4.24.

The system consisted of two WR switches connected with 5 kilometers of bare single-mode G.652 fiber wound on a small plastic roll. The varying operating conditions of the link were simulated by heating the fiber with a hot air gun. The measurement procedure was split into two parts. The first part concentrated on the accuracy of slave's time by measuring the master-slave time offset. During the second part, the performance

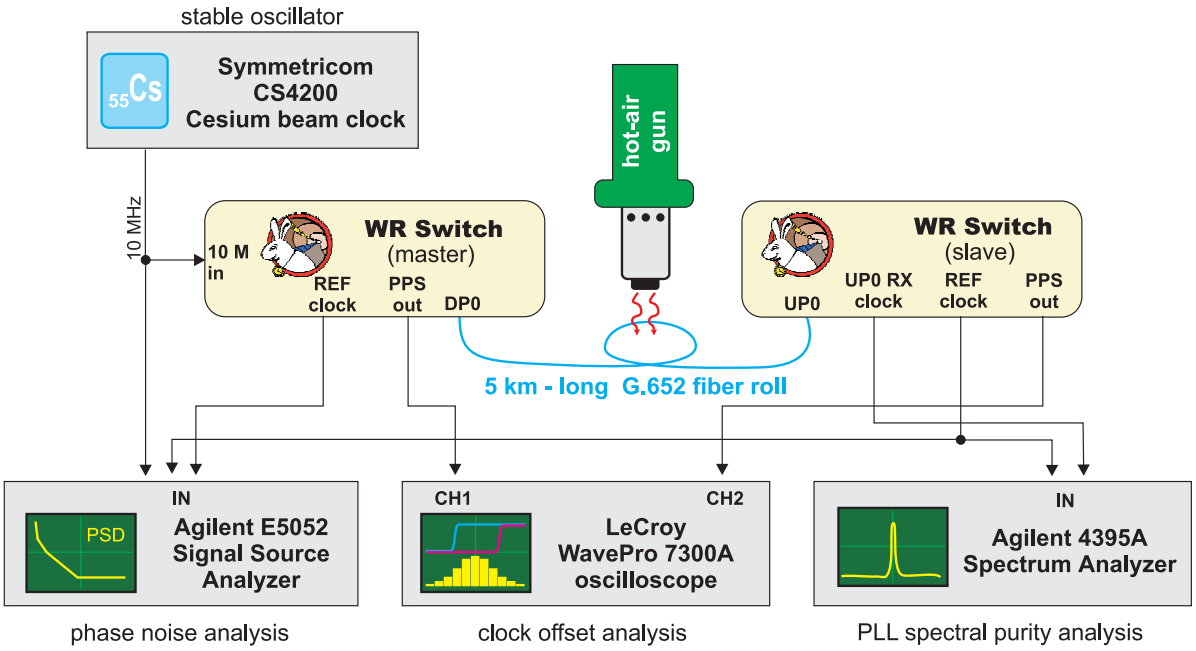

Figure 4.24. WR Switch test setup.

of the reference frequency transfer was characterized and the DMPLL frequency response has been optimized to minimize jitter.

4.5.1. Synchronization accuracy

The accuracy of synchronization was tested with a LeCroy WavePro 20 GSample/second oscilloscope by measuring the skew between master and slave PPS outputs over a period of several hours. The obtained samples of PPS signal skew were used to construct a histogram of master-slave offset, depicted in fig. 4.25.

Since both switches were operating in similar conditions (temperature, humidity, components from the same production batch) and the SFP transceivers were paired, the offset achieved an excellent mean value of 160 picoseconds with a standard deviation of 6.4 ps and a min-max span of 40 ps. These measurements, however, did not characterize the long-term PVT (Process-Voltage-Temperature) and aging effects in the active components of the WRS (FPGAs and clock distribution). It is expected that the variation of the offset over a long time span will not exceed 1 ns.

4.5.2. Syntonization performance

The second part of the measurement procedure focused on evaluating the syntonization performance by determining the frequency domain properties of the master and slave REF clocks: spectral purity and PSD of the phase noise.

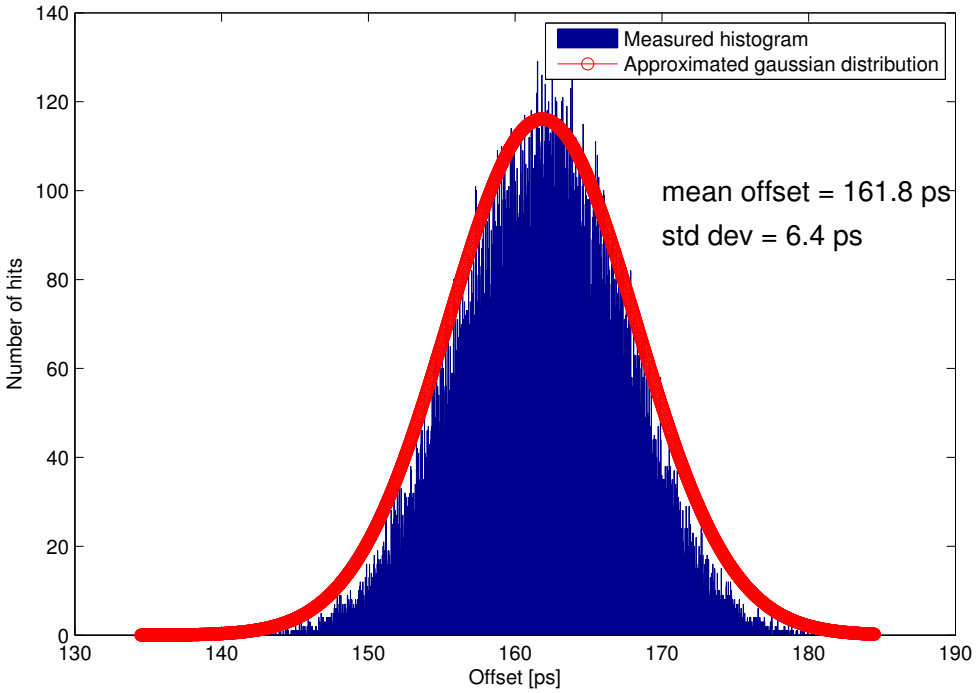

Figure 4.25. Measured histogram of master-slave PPS offset.

Recovered clock spectrum

An Agilent 4395A spectrum/network analyzer was used to check the spectrum of the slave's recovered REF clock. Figure 4.26 illustrates the difference between the spectra of the uplink PHY RX clock and the cleaned-up, phase-compensated output of the slave's Main PLL. Numerous spurs produced by the oversampling CDR in the uplink PHY (red trace) are completely removed in the REF clock (blue trace). The green trace shows the output clock of the PHY when the Main PLL is not yet locked to the master, illustrating the way oversampling CDRs work – when the reference clock is asynchronous with respect to the incoming serial stream, the CDR in the PHY must switch the interpolated clocks more often, which results in a larger number of spurs in the spectrum (green trace).

Spectral analysis also led to the discovery of a bug in PCB design. The switching mode power supply on the Timing PCB had an unshielded inductor causing an interference which manifested itself as two symmetrical spurs separated from the main 125 MHz peak by 520 kHz (switching frequency of the problematic supply).

Phase noise performance

The phase noise of the recovered clock was characterized with a state-of-the-art Agilent E5052 Signal Source Analyzer and a Symmetricom CS4000 cesium beam clock serving as a source of 10 MHz reference frequency for the master WR Switch. PSD

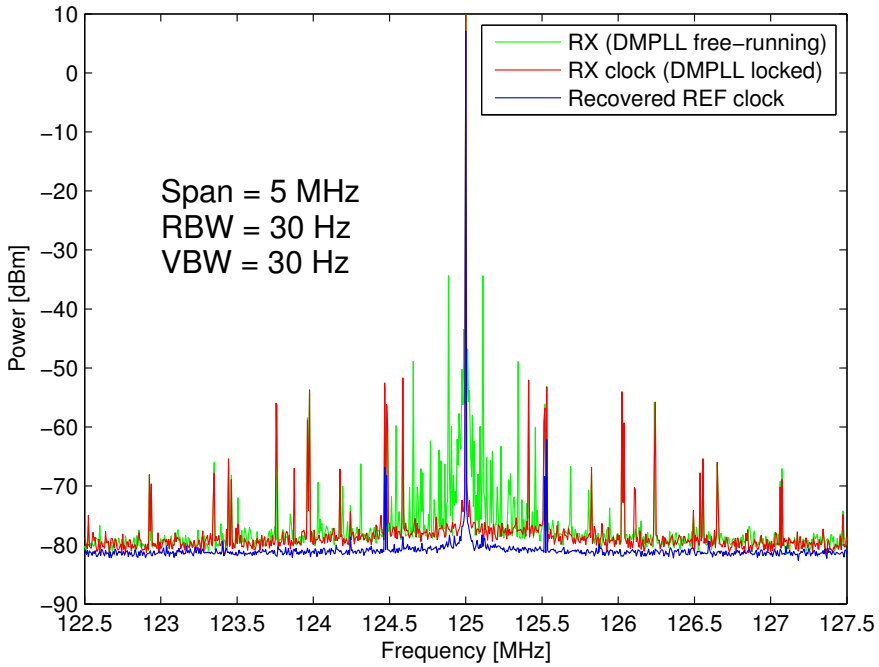

Figure 4.26. Spectrum of slave's PHY RX and REF clock.

measurements served as a base for the optimization of the DMPLL gain parameters and its implementation of DDMTD.

We began the characterization by obtaining the PSDs of the master clock locked to the external 10 MHz cesium reference, which is shown along with the 10 MHz cesium oscillator output in Figure 4.27.

The noise floor of the 125 MHz master clock lies at -110 dBc/Hz, which is much higher than the bare cesium output (-155 dBc/Hz). The difference is caused by the relatively high additive jitter of the external reference input shaping circuit and the AD9516 PLL reference inputs, when compared to the cleanliness of the cesium oscillator output. Spurs were caused by PCB signal integrity issues in the clock input circuit (the peak at 8 kHz is a result of intermodulation with the helper oscillator signal). Because of the low bandwidth of the slave's PLL, they do not affect the jitter of the slave's clock.

In the next step, the PSDs of the slave's free running oscillator and the slave's PHY RX clock have been measured (Figure 4.28). The traces are crossing at about 570 Hz. Below that frequency the master clock is better than the local oscillator, hence 570 Hz should be the theoretical cutoff frequency of the slave's PLL.

The transfer function of the DMPLL is determined by the proportional and integral gain of its PI controller. Using the PLL model described in section 2.5 where the PI transfer function is defined as $K_p + \frac{K_i}{s}$, we obtain the closed-loop gain 4.3:

$$H(s) = \frac{\eta\omega_n s + \omega_n^2}{s^2 + \eta\omega_n s + \omega_n^2} \tag{4.3}$$

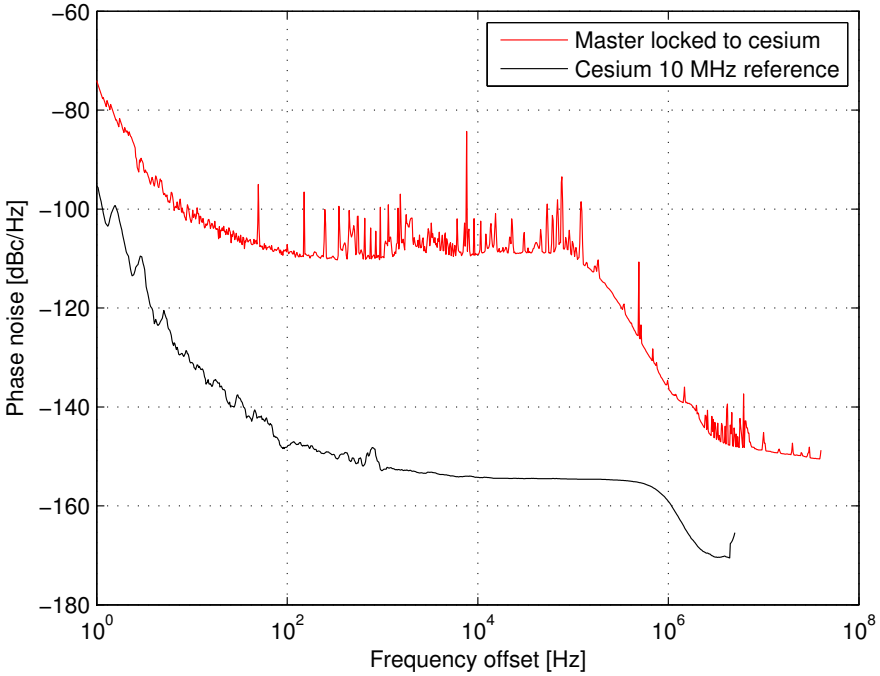

Figure 4.27. Master clock locked to cesium reference.

where ω_n is the so-called natural frequency of the loop and η is the damping factor. ω_n is proportional to the 3 dB cutoff bandwidth, while η defines the slope and peaking of the PLL transmittance (peaking decreases with increasing values of η at the cost of steepness of the slope of $H(s)$). Equation 4.4 expresses their relation with PI gains:

$$\begin{aligned} K_p &= \frac{2\eta\omega_n}{K_d K_{VCO}} \\ K_i &= \frac{f_s \omega_n^2}{K_d K_{VCO}} \end{aligned} \tag{4.4}$$

where f_s is the PI sampling rate, in the WRS equal to the HPLL frequency offset (7.62 kHz), and K_d and K_{VCO} are respectively, the gains of the phase detector and the VCTCXO.

The values of ω_n and η were chosen experimentally. The best results have been observed for $\omega_n = 15$ Hz and $\eta = 0.8$, which gave a 3 dB cutoff frequency of 200 Hz with 1.8 dB peaking (see Figure 4.29). Higher values of η were making the PLL unstable because the slope of $H(s)$ did not guarantee enough attenuation of the high-frequency noise produced by the phase detector. This can be improved by using a higher order PLL filter.

The PSD of the slave's recovered clock is depicted in Figure 4.30. The noise below the DMPLL cutoff frequency is at -80 dBc/Hz, resulting in an integrated rms jitter (5 Hz - 40 MHz) of about 2.3 ps. In order to reduce this value, the DMPLL's phase

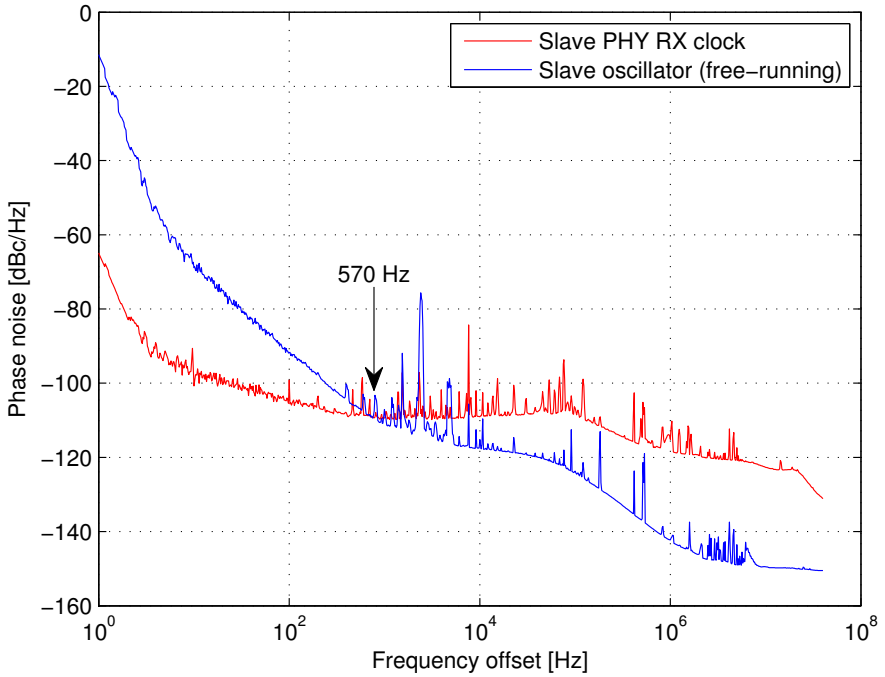

Figure 4.28. Slave's PHY RX clock compared to a free-running VCTCXO.

detector has been modified to average multiple phase measurements coming from 32 parallel DDMTDs sampling at slightly different phases. This optimization resulted in 10 dBc lower phase noise and decreased the overall rms jitter to 1.8 ps. The jitter integrated from 100 kHz to 40 MHz (a common measure of commercial oscillators) lies in the femtosecond range (680 fs rms).

These results surpass the 20 ps jitter requirement by an order of magnitude and are close to the theoretical resolution of the DDMTD (0.48 ps for a 125 MHz reference and N = 14). It is still possible to reduce the jitter even further, since the DMPLL parameters were obtained empirically and therefore may not be optimal.

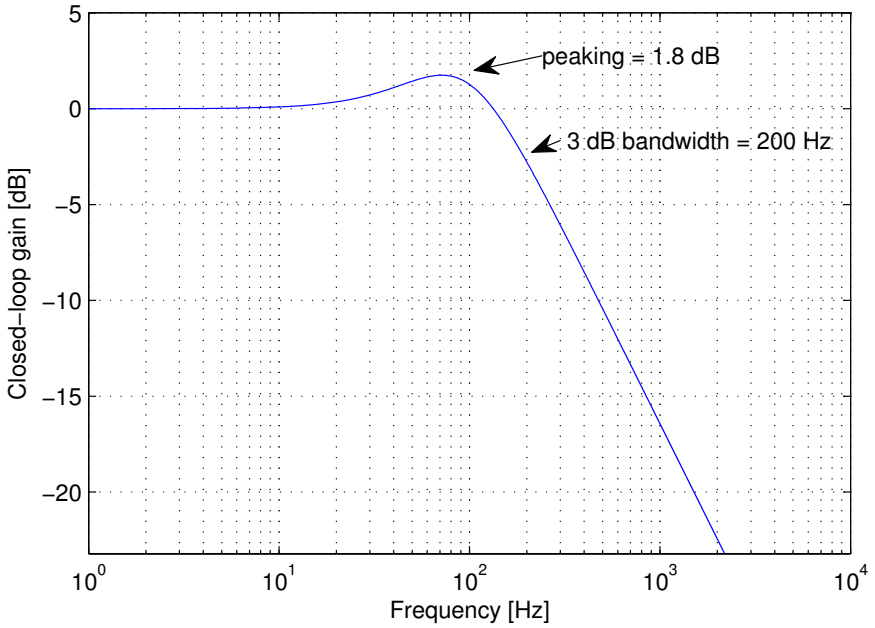

Figure 4.29. DMPLL closed loop gain.

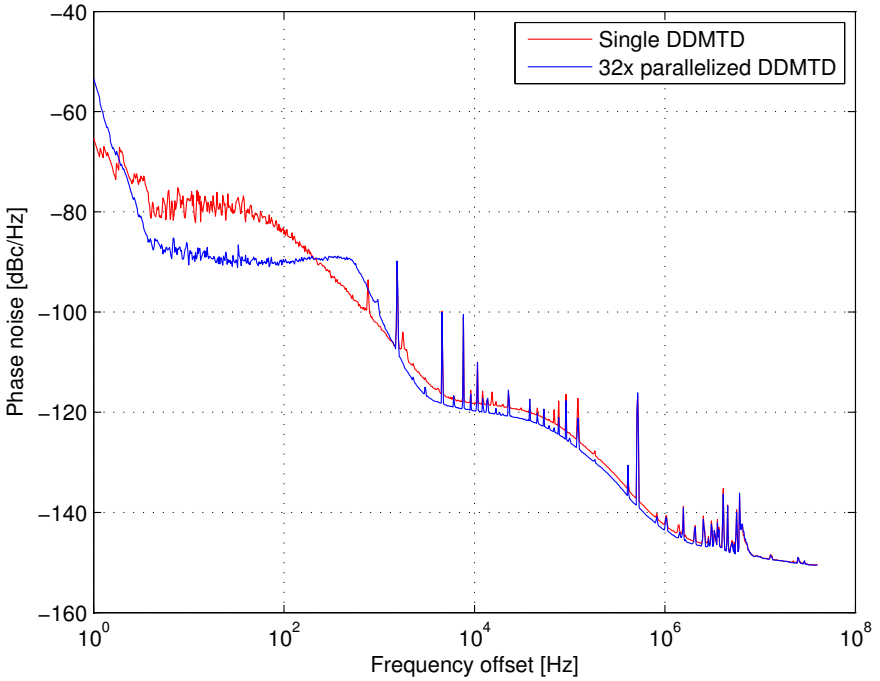

Figure 4.30. PSD of slave's recovered clock for different DDMTD implementations.

5. Applications of White Rabbit

The integration of Ethernet and fine timing in a White Rabbit network makes it a perfect choice for many systems, not only in the particle accelerator world. The list below presents some of the possible applications of WR:

- Phase-compensated radio frequency signal distribution, such as the RF clock or beam-synchronous timing in LHC and other accelerators.
- Synchronization of base stations for TDoA (Time Difference of Arrival) passive mobile device localization (for example, tracking a cellular phone in a dense, city environment with centimeter-level accuracy).
- Metrology and distribution of reference time.
- Large-scale signal acquisition systems.
- Astronomy/radioastronomy (synchronization of telescope arrays).
- Synchronization of radar and terrestrial precise approach instrumentation in airports.

This chapter presents two example systems: an RF clock distribution system using Direct Digital Synthesis and a distributed data acquisition system, which benefit from the features of the WR network, while keeping the costs at a level comparable to the traditional solutions.

5.1. Distributed Direct Digital Synthesis (D3S)

This application tackles the problem of distributing various RF clock signals over large distances in a big accelerator system. The LHC and the Experiments both require a number of clocks to be available around the whole machine. Among these clocks are: beam 1/2 bunch clock and 40.079 MHz timing reference. Currently, each of these signals has its own, independent (and custom-designed) distribution system, including separate cabling. Some of the clocks are transmitted in an analog form, hence requiring repeaters and expensive low-loss coaxial cables.

Distributed DDS facilitates multiplexing a number of clock signals in a single data link, provided that both sides of the link are synchronized with enough accuracy. The idea of the system is presented in fig. 5.1:

The transmitter module is a typical PLL loop with a DDS synthesizer chip fulfilling the role of a voltage-controlled oscillator. Instead of an analog control voltage, the loop delivers numeric values which control the phase and the frequency of the DDS output signal to keep it in-phase with the input clock.

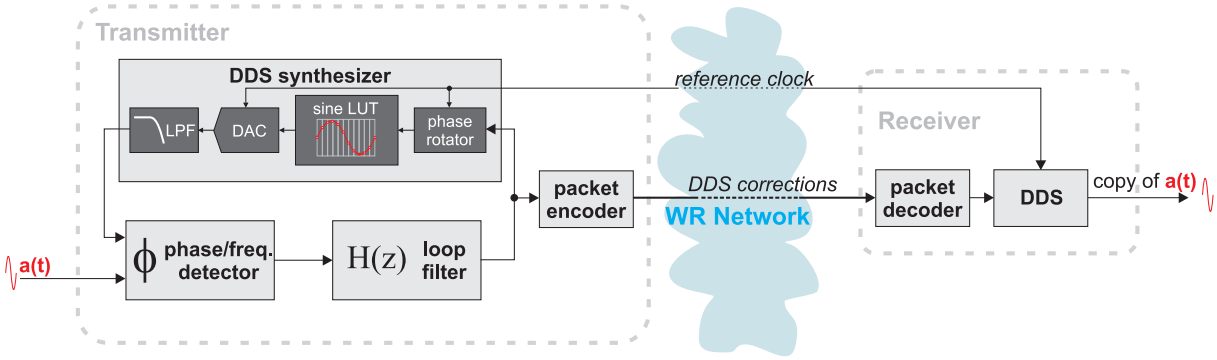

Figure 5.1. An example of a Distributed DDS system.

The control words are then encapsulated into Ethernet packets along with precise UTC times at which they had been generated and broadcast via the WR network to any number of receivers. An additional delta compression can be employed to reduce data throughput. The receiver simply decodes these packets and feeds the extracted control words into an identical DDS synthesizer at the same intervals calculated from the associated timestamps. Since both synthesizers are clocked at the same frequency and phase thanks to WR synchronization, the receiver's DDS produces an exact, slightly delayed copy of the original clock.

The presented method has many advantages over existing systems:

- Single system for multiple clocks. The number of signals is only limited by the network throughput and the bandwidth and stability of the signals.
- Signal quality does not degrade with distance.
- Very simple receivers with almost no analog components.
- Unlimited number of receivers.

The Distributed DDS scheme has also some downsides: slightly higher latency (due to encoding/decoding and network delays) and a tradeoff between the loop bandwidth (and thus, the ability to track quickly changing signals) and the required throughput. However, these can be neglected in the majority of clock distribution applications, where the phases of the signals are relatively stable.

5.2. Distributed Oscilloscope

Every particle accelerator installation contains distributed data acquisition (DAQ) systems, so the operators can have an insight into what's happening inside the machine. An example of a distributed DAQ system at CERN is the OASIS system [6]. Such systems can be seen as oscilloscopes with a huge number of signal and trigger inputs. They usually require:

- Configurable time base for all the input signals.
- Versatile trigger distribution.

► Flexible way of presenting the measured data to multiple users.

Advanced clock distribution features (distributed DDS) and low-latency message delivery make WR a perfect candidate for the backbone of large-scale DAQ systems. A concept of WR-based DAQ system is shown in fig. 5.2.

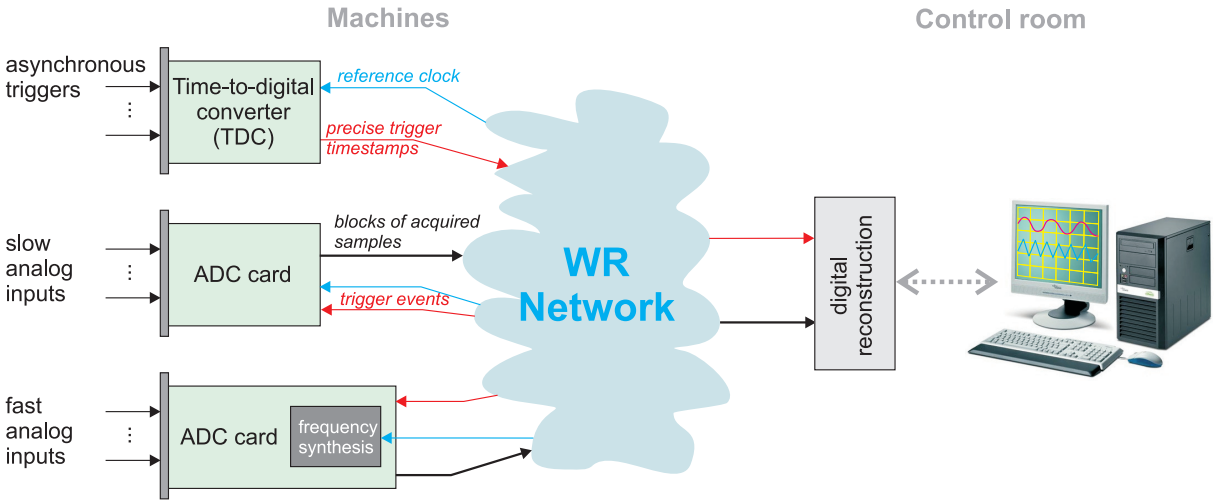

Figure 5.2. Distributed oscilloscope system using White Rabbit

Distributed DAQ works by sampling all inputs continuously (even when there are no triggers) and buffering the recorded samples for a pre-defined time. When a trigger event arrives (in form of a network message), the ADC card copies the interesting part of the signal from the buffer and sends it (along with the timestamp of the first sample in the data block) to the operator stations. The length of the ring buffer must be greater than the trigger event delivery time. Thanks to the low-latency features of WR, even in large networks this time should not exceed 100 – 200 us, hence a buffer of a few megabytes per channel is sufficient.

All the ADC cards in the system use the same clock for sampling the signals. It is also possible to use other frequencies, either through the Distributed DDS mechanism or by deriving them from the network reference clock using a PLL synthesizer.

A Distributed DAQ system can use either internal triggers (where the trigger is simply a UTC time at which the ADC is supposed to start the acquisition) and external triggers, produced by Time-to-digital converter cards from asynchronous trigger inputs. A reception of an external trigger pulse causes an automatic transmission of a trigger event to the appropriate ADCs.

As a result, we obtain blocks of acquired digital data with associated precise trigger timestamps. Both are transmitted through the WR network to the operators' workstations (usually standard PCs). There, the signals are digitally post-processed (to compensate for phase shifts and re-create the common time base) and displayed to the user in a convenient way.

6. Summary

The aim of this thesis was to develop an Ethernet-based system for precise clock synchronization. The realization of this goal was split into several stages and each of them required a significant amount of work to obtain optimal results.

The first stage focused on studying the existing timing solutions and the Ethernet and IEEE1588 standards. During that stage, the goals of the project were defined and an Ethernet-based network architecture was chosen, allowing for cost-optimal FPGA implementation of WR devices. In the second step, all the elements of an optical Ethernet link which have an impact on the accuracy of synchronization were thoroughly analyzed. Another notable achievement was the concept of a Digital DMTD phase detector, facilitating picosecond-level multichannel phase measurements in inexpensive FPGA devices. The result of this work is the WR synchronization algorithm presented in Chapter 3.

The correctness of the WR algorithm was confirmed by designing, manufacturing and characterizing the timing performance of the WR Switch, presented in Chapter 4. Timing measurements done on the WRS allowed for further optimization of the synchronization algorithm and confirmed that the goal of the thesis has been fully achieved.

The performance of the presented IEEE1588 and SyncE-based method exceeds the initial expectations both in terms of the accuracy of synchronization and cleanliness of the distributed clock signal. The WR Switch was compared with other PTP devices during ISPCS'2010 conference plug fest and is currently the most accurate publicly available PTP implementation. This gives confidence that the the presented method can be employed in larger networks to build a fully synchronous hierarchy which can be used for applications with demand for high-accuracy timing (e.g. accelerator sites and radiotelescopes).

WR has been already chosen as the timing system for the new FAIR (Facility for Antiproton and Ion Research) complex in GSI (Darmstadt, Germany) and for the renovated control system of the injector chain at CERN (Linacs, PSB, PS). It also gained recognition in the commercial world – companies are already designing WR-compatible cards (Elproma, National Instruments).

However, there are still many fields in which further research needs to be done: network reliability (fast switchover of broken links), optimization of the clock recovery PLL and development of a standardized method for calibrating WR devices. Also, a new version of WRS hardware (V3) is being currently designed. The full system,

incorporating a WR Master, WR Switches and WR Nodes (client cards) is expected to be commercially available at mid-2012.

The WR project is more than the synchronization algorithm and the WR switch, and it is not the work of one man. The author wishes to thank the people from CERN, GSI and all WR team members, without whom the goals of the project would have never been achieved.

Summarizing, the WR project has proven that sub-nanosecond synchronization can be done with an Ethernet-based network. I hope that the subject of this thesis will encourage others to conduct more successful research in the field of timing systems.

October 2009 - April 2011

Bibliography

[1] Alan V. Oppenheim and Alan S. Willsky and S. Hamid Nawab. *Signals and Systems*. Prentice Hall, 2nd edition, 1996.

[2] Alan V. Oppenheim and Ronald W. Schafer and John R. Buck. *Discrete-Time Signal Processing*. Prentice Hall, 2nd edition, 1999.

[3] Athanasios Papoulis and S. Unnikrishna Pillai. *Probability, Random Variables and Stochastic Processes*. McGraw Hill, 4th edition, 2002.

[4] Bahaa E.A. Saleh and Malvin Carl Teich. *Fundamentals of Photonics*. Wiley-Interscience, 2007.

[5] Lewis Caroll. *Alice in Wonderland*. 1865.

[6] CERN. *Open Analogue Signal Information System (OASIS)*. `http://project-oasis.web.cern.ch/`.

[7] Actel Corporation. *Implementing an 8b/10b Encoder/Decoder for Gigabit Ethernet in the Actel SX FPGA Family*. `http://www.actel.com/documents/SX_GB_Ethernet_AN.pdf`, 1998. Actel application note.

[8] D.A. Howe and T.N. Tasset. *Clock Jitter Estimation based on PM Noise Measurements*. `http://tf.nist.gov/timefreq/general/pdf/1895.pdf`, 2003. Proceedings of the 2003 IEEE International Frequency Control Symposium and PDA Exhibition.

[9] D.B Sullivan, D.W. Allan, D.A. Howe, F.L. Walls. *Characterization of Clocks and Oscillators*. `http://tf.nist.gov/general/pdf/868.pdf`, 1990. NIST Technical Note.

[10] E.Andersen, the Buildroot Developers. *Buildroot: making Embedded Linux easy*. `http://buildroot.uclibc.org/`.

[11] E.G. Cota, M. Lipiński, T. Włostowski, E.V.D. Bij, J. Serrano. *White Rabbit Specification: Draft for Comments*. `http://www.ohwr.org/documents/21`, 2010.

[12] Floyd M. Gardner. *Phaselock Techniques*. Wiley, 3rd edition, 2005.

[13] G. Daniluk. *White Rabbit PTP FPGA Core*. `http://ohwr.org/projects/wr-cores`.

[14] Gene F. Franklin and J. David Powell and Abbas Emami-Naeini. *Feedback Control of Dynamic Systems*. Addison Wesley, 3rd edition, 1994.

[15] IEEE. IEEE 1588: "Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. `http://standards.ieee.org/`, 2008. IEEE standard.

[16] IEEE. IEEE 802.3: LAN/MAN CSMA/CDE (Ethernet) Access Method. `http://standards.ieee.org/getieee802/802.3.html`, 2008. IEEE standard.

[17] Intel, Helwett-Packard, NEC, Dell. *Intelligent Platform Management Interface Specification*. `http://www.intel.com/design/servers/ipmi/spec.htm`, 2007.

[18] ITU. *Timing characteristics of synchronous ethernet equipment slave clock (EEC).* http://www.itu.int/rec/T-REC-G.8262, 2007.

[19] J. Serrano, P. Alvarez, M. Cattin, E. G. Cota, J. H. Lewis, P. Moreira, T. Wlostowski and others. *The White Rabbit Project.* In *ICALEPCS TUC004*, 2009.

[20] P.P.M. Jansweijer and H.Z. Peek. *Measuring propagation delay over a 1.25 Gbps bidirectional data link.* http://www.nikhef.nl/pub/services/biblio/technicalreports/ETR2010-01.pdf.

[21] John G. Proakis and Dimitris K. Manolakis. *Digital Signal Processing.* Prentice Hall, 4th edition, 2006.

[22] M.A. Lombardi, T.P. Heavner and S.R. Jefferts. *NIST Primary Frequency Standards and the Realization of the SI Second.* http://tf.nist.gov/general/pdf/2039.pdf, 2007.

[23] OpenCores/OrSOC. *Wishbone bus specification (rev. B4).* http://cdn.opencores.org/downloads/wbspec_b4.pdf.

[24] PICMG. *Micro Telecommunications Computing Architecture Base Specification.* https://www.picmg.org/v2internal/specorderformsec-member.htm, 2007.

[25] PTPv2d Developers. *ptpv2d Daemon (original project).* http://code.google.com/p/ptpv2d.

[26] Enrico Rubiola. *Phase Noise and Frequency Stability in Oscillators.* Cambridge University Press, 2009.

[27] Jeff Sonntag and John Stonick. *A Digital Clock and Data Recovery Architecture for Multi-Gigabit/s Binary Links.* IEEE Custom Integrated Circuits Conference, 2005.

[28] T. Włostowski. *wbgen2 - a Wishbone slave core generator.* http://ohwr.org/projects/wishbone-gen.

[29] T. Włostowski. *White Rabbit Fabric Interface (WRF) specification.* http://ohwr.org/projects/white-rabbit/documents/.

[30] R.C. Walker. *Designing Bang-Bang PLLs for Clock and Data Recovery in Serial Data Transmission Systems.* http://www.omnisterra.com/walker/pdfs.papers/BBPLL.pdf.

[31] WR Developers. *Open Hardware Repository (White Rabbit project home page).* http://ohwr.org/projects/white-rabbit.

List of Figures

Appendices

6.1. Thesis sources

The PDF version of this thesis has been included on the attached CD-ROM along with the LaTeX source code. All the drawings are included in Corel Draw, EPS, SVG and PDF formats.

6.2. WRS design and source code

The source code of the WRS is publicly available on the Open Hardware Repository website. It has not been included on the attached CD-ROM . The hardware, HDL and software design copyright belongs to CERN, and parts are copyright GSI Darmstadt. The sources are subject to one of the following licenses:

- Hardware design (schematics and PCB layouts): CERN Open Hardware License v. 1.0
- VHDL code: Lesser General Public License (LGPL) v. 2.1
- Software: GNU General Public License (GPL) v. 2 or LGPL v. 2.1