République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

Digital Phase Detectors in VLSI

Tagzout Samir

Thèse

Présentée pour l'obtention du diplôme de Doctorat en Électronique

Soutenue le 22 Février 2012 devant le jury composé de:

Président	MEHENNI Mohamed	Prof.	ENP
Rapporteur	BELOUCHRANI Adel	Prof.	ENP
Examinateur	HARAOUBIA Brahim	Prof.	USTHB
Examinateur	OULESIR BOUMGHAR Fatima	Prof.	USTHB
Examinateur	BOUSBIA-SALAH Hicham	MCA	ENP
Examinateur	FOUZAR Youcef	A.Prof.	Université du Québec à Trois-Rivières, Canada

ملخص:
تستخدم أجهزة الكشف عن الفرق الزمني في الأنظمة المتزامنة .هذه الرسالة تقترح مساهمتين تدخلان في مجال تصميم هذه الأجهزة . أولا تصميم الجهاز الخطي للكشف عن الفرق الزمني الذي يلبي مواصفات معايير الصناعة. في الأوساط الأكاديمية وضعت العديد من الأجهزة والبرامج المتكاملة أعطوا تحسينات تقنية. لكنها لا تلبي الاهتمامات التي يمكن استخدامها بفعالية في النظم الصناعية. في هذه الرسالة، من بين المقترحات الأخرى، فإننا نقدم تقنية جديدة للتعديل دقة الأرقام. ثانيا تصميم الجهاز التربيعي للكشف عن الفرق الزمني التي تطبق خوارزمية نشرت بنجاح و ثبت على صحتها نظريا .هذه الرسالة تقدم حلولا فعالة للتعقيدات التي وجد ت خلال ترجمة الخوارزمية إلى دارة مندمجة .وقد تم نشر بعض من عملنا في مقالتين في صحيفة و في مؤتمر بلجان تصحيح دولية .

كلمات مفتاحيه: الكشف عن الفرق الزمني , الأنظمة المتزامنة , دارة مندمجة , الخوارزمية

Résumé :
L'implémentation de composants (Propriétés Intellectuelles) conformes à des standards industriels et l'implémentation hardware d'algorithmes théoriquement validés présentent une multitude d'intérêts. Mais, elles nécessitent la compréhension et l'investissement de plusieurs niveaux d'abstractions. Dans cette optique, nous proposons deux Détecteurs de Phase génériques. L'un est configurable pour être conforme aux spécifications du standard "Clocks for the Synchronized Network: Common Generic Criteria" de Telcordia Technologies et l'autre implémente l'algorithme "Blind Carrier Phase Tracking with Guaranteed Global Convergence" publié et plusieurs fois référencé. Dans ces deux travaux nous mettons en valeurs des problèmes de conception et nous proposons des solutions qui les résolvent. Entre autres contributions, nous proposons la simplification de l'algorithme et des règles traités pour les adapter à des architectures VLSI. Nous montrons comment utiliser le moins possible de composants à grandes dimensions, comment éviter d'utiliser des diviseurs conventionnels et comment il est possible de multiplexer efficacement le temps d'exécution de multiplieurs. Nous proposons une nouvelle méthode d'implémentation de la fonction Arctangent et une nouvelle technique d'ajustement dynamique de la précision des nombres codées en complément à deux. Des simulations ainsi que des implémentations sur des FPGAs de Xilinx nous ont permis de valider nos propositions. Une partie de notre travail de Thèse est publiée dans deux articles, l'un dans un journal et l'autre dans un Proceeding à Jury international.

Mots Clés : Détecteur de Phase, VLSI, Arithmétique, Ajustement de la Précision, Arctangent

Abstract : Implementing components (Intellectual properties) compliant with industrial standards and implementing successful and theoretically validated algorithms is of utmost importance. It requires, though, working on several levels of abstraction. To that end, we propose in this Thesis two Phase Detectors; one configurable to be compliant with the Telcordia Technologies ”Clocks for the Synchronized Network: Common Generic Criteria” and the other implementing the ”Blind Carrier Phase Tracking with Guaranteed Global Convergence” algorithm which is published and several times referenced. We present the way we have dedicated and simplified the algorithm and the normalized rules for VLSI implementations. We highlight the related design issues and solutions. Among other contributions, we show how it is possible to efficiently time multiplex large multipliers, we present a novel Arctangent function implementation as well as a novel technique for a dynamic Accuracy adjustment of two’s complement numbers. Simulation results, FPGA implementations as well as two International publications have backed our work proposal.

Key Words: Phase Detector, VLSI, Arithmetic, Accuracy Adjustment, Arctangent.

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Professor Adel Belouchrani, for being the driving force behind the completion of this thesis. Without his support and ideas this work would not have been possible.
I also extend my gratitude to the Jury of my maintenance for giving their time and so valuable advices while evaluating this work.
I would like to express my appreciation to all my friends and colleagues. I am thinking particularly to Uncle Hussain Mir, Ant Naznee and Djamel Benamrouche for their great generosity and to Dr. Aziz Chihoub and Dr. Si Mahmoud Karabernou for supporting my early research work in CDTA and for sending to me a useful documentation helping in many achievements.
Special Thanks go to my wife and her friend A. Bouarroua for helping in the elaboration of this document.
Finally, I would like to deeply thank my lovely family; my wife, my parents, my sisters and my brothers for their warm encouragement and their understanding.

Dedicace

A notre petit Mohammed
A nos grands modèles de dévouement de la Compagnie de Région de l'ALN, de la Zone 2
– Wilaya III, ceux qui se motivaient en se répétant :

"Ad'naj Attarikh. Adhaghran Warrach"
Nous léguerons une leçon d'histoire
Nos enfants seront enseignés

Contents

List of Figures

List of Tables

Abbreviations

ASIC: Application Specific Integrated Circuits
BCPT: Blind Carrier Phase Tracking with Guaranteed Global Convergence
CPU: Central Processing Unit
DCO: Digitally Controlled Oscillator
DPLL: Digital Phase Locked Loop
DUV: Design Under Verification
***fb*:** Feedback output from a Synchronization System
FPGA: Field Programmable Gate Array
IC: Integrated Circuit
I/O: Input and Outputs
ISI: Inter-Symbol Interference
LPD: Linear Phase Detector
LSB: Least Significant Bit
LUT: Look Up Table
MSB: Most Significant Bit
NCO: Numerically Controlled Oscillator
QAM: Quadrature Amplitude Modulation
QPD: Quadrature Phase Detector
R&D : Research and Development
***ref*:** Input Reference to a synchronization system
RTL: Registre Transfer Logique
SNR: Signal to Noise Ratio
SoC: System on Chip
TDEV: Time Deviation criteria
VLSI: Very Large Scale Integration

Chapter 1

INTRODUCTION

The phase detectors are used in synchronization systems. They are widespread over the literature in two essential categories; Linear Phase Detectors (LPD) and Quadrature Phase Detectors (QPD) [1], [2].
The QPDs are widely provided in research publications. References [3] and [4] are among many other examples. In the other hand, LPDs are not presented with much of options. Usually, the same analogical and logical phase detector architectures are found in the description of synchronization systems. Actually, there are no details about specific functionalities needed for conformance to standards [2]. These functionalities are rather given in the data−sheets of industrial components without explicit specification about their belonging to phase detectors [5]. This Thesis gives detailed explanations on both of the phase detector categories by bringing two essential contributions:
1 - Design of a linear phase detector which is able to respond to normalized generic criteria. In the academic research teams, successful hardware and software IPs are developed. They respond to specific training and academic needs, as the implementation described in reference [6], and sometimes they even give technical improvements relatively to their states of the art as shown in references [7] and [8]. Nevertheless, they are not specified to be used in normalized industrial applications. So, we are describing a detailed design of an LPD configurable to respond to industrial generic criteria. In this proposal, among other solutions, we present a novel method to dynamically adjustment the accuracy of two's complement numbers particularly when they are used as operands to fixed width dedicated dividers.
2 - Design of a phase detector implementing the "Blind Carrier Phase Tracking with Guaranteed Global Convergence (BCPT)" algorithm [3] which is theoretically validated and cited several times [9]. This contribution presents efficient design techniques to solve some critical issues related to the needed arithmetic unit and storage components. We showed how it is possible to efficiently time multiplex large multipliers. We benchmarked our simulations against the theoretic validation approach published in [3] and the related design specification is published in the Proceedings of WOSSPA' 2011 [10]. We also improved the implementation of the Arctangent function to be run in three clock cycles while retrieving the angle resolution and precision as chosen for the application. The related novel tech-

Table 1.1: **Objective Framework including the phase detector designs**

National Program	Lab Program	Lab Project
Global Objective: High Tech entrepreneurship Promotion		
Specific Objective: IP Portfolio in Signal Processing	**Global Objective:** IP Portfolio in Signal Processing	
Results: Portfolio of VLSI IPs in Timing and Synchronization	**Specific Objective:** Portfolio of VLSI IPs in Timing and Synchronization	**Global Objective:** Portfolio of VLSI IPs in Timing and Synchronization
	Results: VLSI Phase Detectors	**Specific Objective:** VLSI Phase Detectors
		Results: - Phase Detector conformant to Industrial Standards - Phase Detector implementing complex signal processing algorithms

nique is published in the JCSC Journal [11].
These two contributions meet the need of Intellectual Properties (IPs) portfolios which could position our work and collaborators in the value chain of the VLSI industry. They can be considered in the logic of the useful research explicitly promoted in the National Research Program "Programme National de Recherche" [12] and tacitly defended in all the national R&D forums. Table 1.1 shows the Logical Framework including the need of Phase Detectors VLSI IPs.

Figure 1.1 gives an important motivation to work on VLSI IP creation. The related revenue estimation was done in 2004 [13]. It didn't take into account the present economic downturn. However, the actual growth trend, reported in reference [13], is well established as shown in Figure 1.2. Complex hardware IPs as well as complex theoretical algorithms are widely developed in universities and research centers. They constitute two competitive advantages worth to exploit.

The rest of this Thesis is organized as follows:
Chapter 2 presents the basic phase detector concepts to introduce the two following Chapters. It gives an overview about the role of phase detectors in simple synchronization systems and it briefly describes the usual analog phase detector and the basic elements of linear and quadrature phase detectors.
Chapter 3 describes standard criteria that should be addressed by our LPD. It gives the design specification to effectively make it configurable to be conformant to Telcordia Technologies GR−1244−CORE [16]. In this Chapter among other solutions, we propose a novel method to dynamically adjust the accuracy of two's complement numbers.
Chapter 4 describes quadrature phase detection when using the "Blind Carrier Phase Alignment with Guaranteed Global Convergence" algorithm. It gives a simplification of the algorithm in order to be realized with logical components and it addresses the related

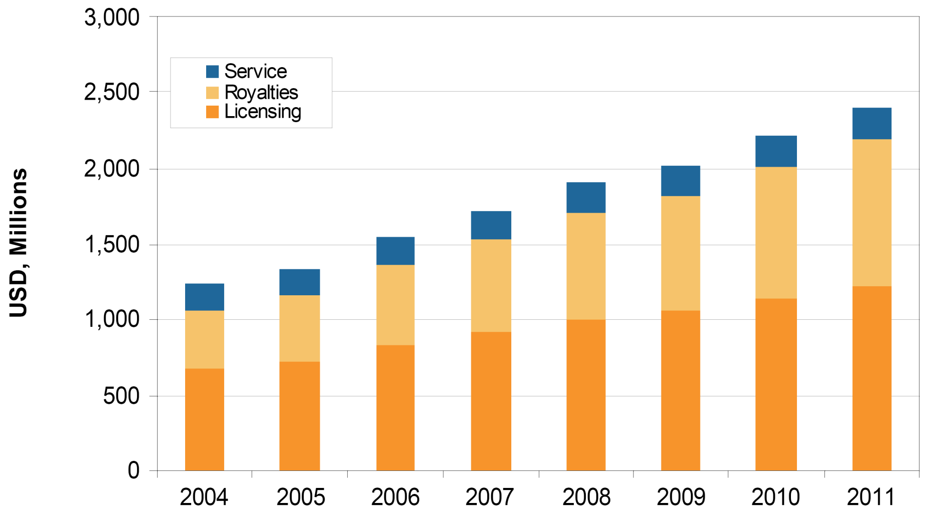


Figure 1.1: **Estimated Semi-conductors IP revenue. As presented in Reference** [13]

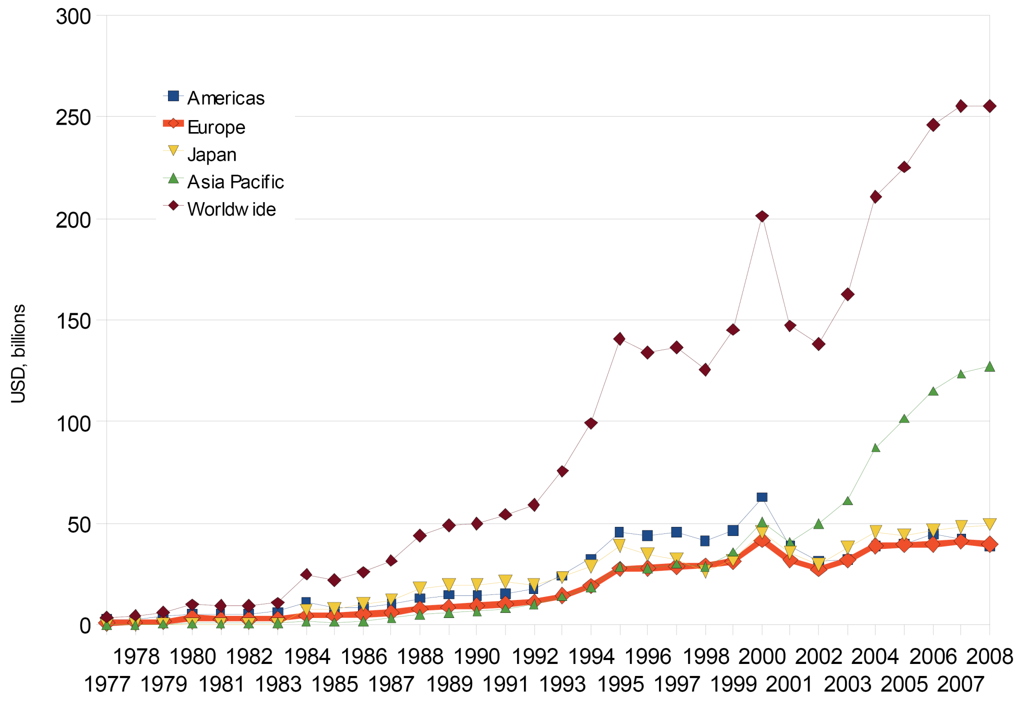


Figure 1.2: **Billing by Semiconductor firms. As presented in Reference** [13]

implementation difficulties including an Arctangent design requiring high speed and high precision data. This Chapter finishes by showing the way we benchmarked our simulations against the theoretic validation approach published in [3].
A Conclusion is given as an open synthesis. It summarizes our work and gives some of its future alternatives.

Chapter 2

OVERVIEW ON VLSI PHASE DETECTORS

It is possible to categorise VLSI phase detectors in many ways. In this Chapter we present them in two global categories; as Linear Phase Detectors and as Quadrature Phase Detectors.
Because the basics for all phase detectors derive from the concept of the simplest analog Phase Detector, we start our explanation by giving its basic expression. Then, we move on digital Phase Detectors with their two categorises the linear and the quadrature ones.

2.1 Introduction to Linear Phase Detector

Phase Detectors are used as entry components to PLLs. They start the phase tracking functionality by indicating the phase difference between a reference signal and a system feedback. Figure 2.1 gives a conceptual bloc diagram of a PLL producing a signal called *fb* and locking to an input reference called *ref*. It consists of three basic elements: a Phase Detector, a Loop Filter, and a Controlled Oscillator.
The Controlled Oscillator generates the desired output signal *fb* that is used for phase detection and correction. The Loop Filter processes the phase difference caught by the

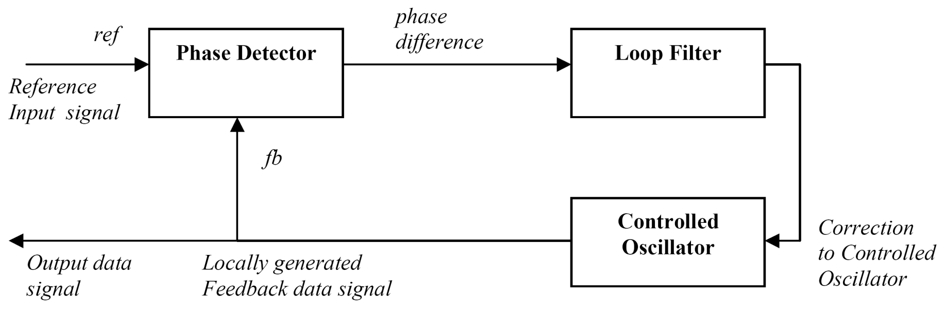


Figure 2.1: **PLL Basic Elements**

Phase Detector and outputs the needed correction so that *fb* locks to *ref*.
The topology in Figure 2.1 works in the analog as well as in the digital worlds. The two next sections give an introduction to basic phase detectors in those two worlds.

2.1.1 Introduction to Analog Phase Detector

In the analog world, the Phase Detector is a multiplier associated to a gain value K [15]. To simplify our explanation, let us consider that $ref(t)$ and $fb(t)$ are sinusoids with the same frequency and a phase shift of 90°. We also consider that the multiplier's gain K is 1.

$$ref(t) = A_{ref}\sin(\omega t + \phi_{ref}(t)) \tag{2.1}$$

$$fb(t) = A_{fb}\cos(\omega t + \phi_{fb}(t)) \tag{2.2}$$

If PD was the output of the Phase Detector:

$$PD = ref(t) * fb(t)$$

With some trigonometric manipulation, we can have

$$PD = [\frac{A_{ref}A_{fb}}{2}][\sin(\phi_{ref}(t) - \phi_{fb}(t)) + \sin(2\omega t + \phi_{ref}(t) + \phi_{fb}(t))] \tag{2.3}$$

Equation 2.3 shows that PD consists of two terms; a function of the phase difference of the two signals and a function running at twice the frequency of the signal $ref(t)$. When that second term is discarded by filtering, the remaining term constitutes the control signal CS for phase error correction.

$$CS = \frac{A_{ref}A_{fb}}{2}\sin(\phi_{ref}(t) - \phi_{fb}(t)) \tag{2.4}$$

For more analog phase detection details, reference [15] gives all the needed and related design aspects. That is done in the general context of ICs and PLLs.

2.1.2 Introduction to Digital and Linear Phase Detector

In the digital world, instead of extracting phase differences from an analog multiplier, logical components make it. The usual example for that is the exclusive OR (XOR) gate that may receive periodic ref and fb and provides a pulse width proportional to the phase difference (Figure 2.2).
Based on a XOR gate, other types of phase detectors can be elaborated. Usually, only one transition is considered for phase detection. That reduces error risks because of timing problems.
For more robustness, an all digital phase detector can be based on timers and edge detectors to extract the phase difference from its two input signals with respect to rising or falling edges (Figure 2.3). In these kinds of detectors, the precedence of the input transitions

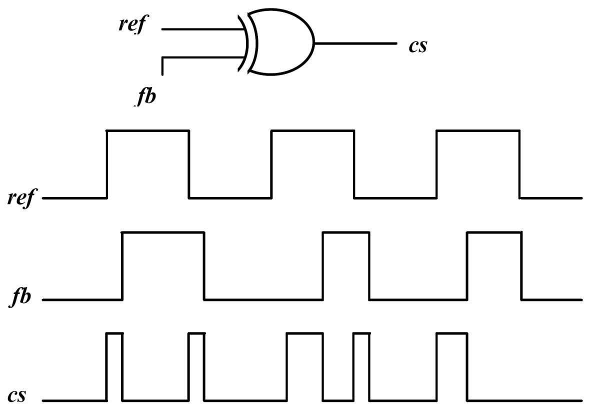


Figure 2.2: **XOR as a Phase Detector**

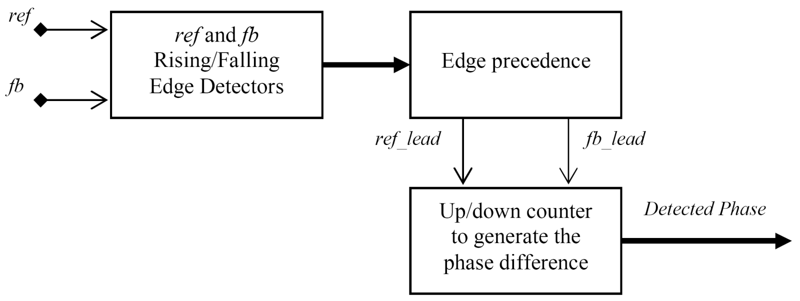

ref
fb
ref and fb Rising/Falling Edge Detectors
Edge precedence
ref_lead
fb_lead
Up/down counter to generate the phase difference
Detected Phase

Figure 2.3: **Digital and Linear Phase Detector**

is indicated to a counter that outputs the searched phase difference by incrementing or decrementing the phase steps. If *ref* was a sampled sinusoidal data, the same phase detector architecture as the one in Figure 2.3 could be used. In such case, to generate the rising or falling edge indicator, comparators would be used to monitor the signal's zero crossing. For instance, if n was a data sample index, the data rising edge is indicated when ($ref(n-1) < 0$ and $ref(n) \geq 0$) The basic phase detector functionalities cited in this section gives effectively the phase difference of fb relatively to ref. Many other processing operations can be efficiently performed at the phase detector's level. Limiting a range for that phase difference, considering phase transients or ignoring them and tracking the trend of phase changes are among capabilities that will be treated in Chapter 3.

2.2 Introduction to Quadrature Phase Detection

A modulated band pass signal $s(t)$ can be represented as:

$$s(t) = A(t)\cos(2\pi fc\; t + \phi(t)) \tag{2.5}$$

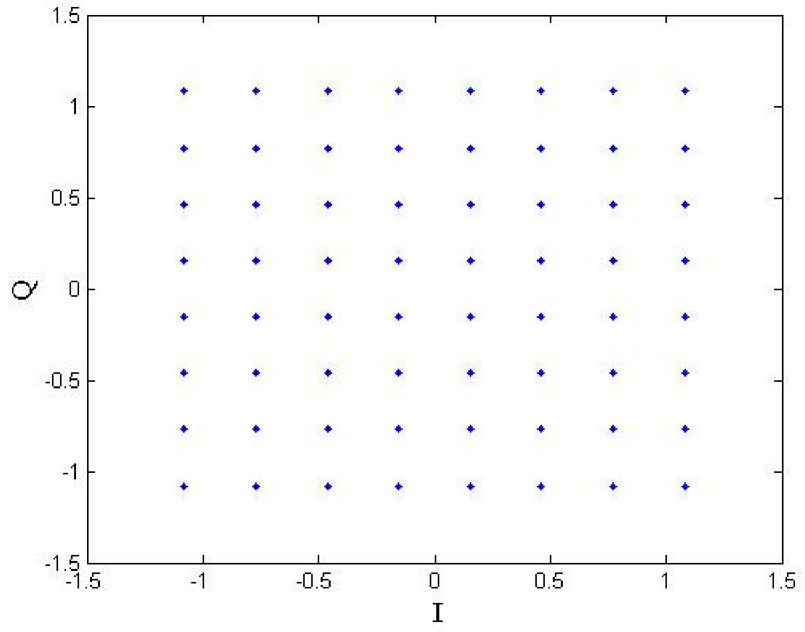


Figure 2.4: **Constellation example for a QAM signal**

where $A(t)$ is the amplitude modulation, $\phi(t)$ is the phase modulation, and fc is the frequency of the carrier [14].
Quadrature Amplitude Modulation (QAM) is used to generate modulated signals obtained by summing two signals with carriers that are $\pi/2$ out of phase. That phase difference is called quadrature phase offset and the two modulated signals are referred to as the in-phase (I) and the quadrature (Q) signals. The summation of those two quadrature signals can be shown to be mathematically equivalent to the amplitude and phase modulated signal in equation 2.5. Using trigonometric identities equation 2.5 becomes:

$$s(t) = A(t)[\cos(\phi(t))\cos(2\pi fct - \sin(\phi(t)))\sin(2\pi fct)] \tag{2.6}$$

Considering $AI(t) = A(t)\cos(\phi(t))$ and $AQ(t) = A(t)\sin(\phi(t))$

$$s(t) = AI(t)\cos(2\pi fct) - AQ(t)\sin(2\pi fct) \tag{2.7}$$

The I and Q amplitudes constitute a constellation diagram as presented in Figure 2.4.
The quadrature phase detector tracks the received signal to determine whether the demodulator's internal clock is sampling at the right time. Basically, it determines whether the receiver is sampling early or late to generate the needed phase corrections to the internal sampling clock. In Figure 2.5 the phase detector outputs an up-down signal to a counter in order to control a clock synthesizer. The synthesizer generates and offsets the needed sampling clock. Reference [14] explains in detail the QAM symbol timing recovery in which a quadrature Phase Detector is running. In summary, it is based on a predicted edge crossing symbol. As explained in that reference and considering that a received signal

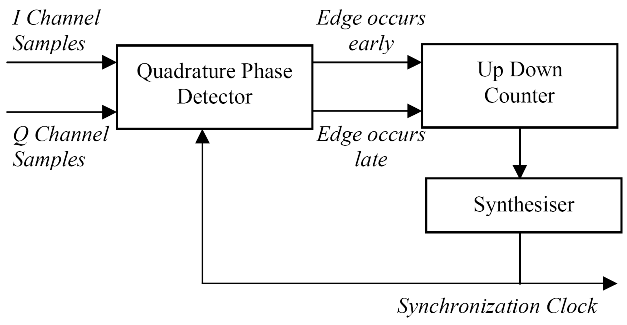


Figure 2.5: **Quadrature Phase Detector in a DPLL**

is sampled four times per symbol. $\mathrm{I}_{Current}$ and $\mathrm{Q}_{Current}$ are the I and Q channel samples for the current symbol while I_{Old} and Q_{Old} are the samples for the previous symbol. I_{Edge} and Q_{Edget} are the samples halfway between the current and the previous symbols. An edge occurs when the signal trajectories of the I or Q cross zero. When the symbols are alternately positive and negative and because they have equal amplitude, at an ideal sampling situation the amplitude of I_{Edge} and Q_{Edge} should be zero. Then, the conditions to check whether I and Q edges occurred early are:

$$(I_{Edge} > 0) \oplus (I_{Current} > I_{Old}) \tag{2.8}$$

$$(Q_{Edge} > 0) \oplus (Q_{Current} > Q_{Old}) \tag{2.9}$$

If the conditions in equation 2.8 and in equation 2.9 are true, a count−up is generated while a count−down is generated if those conditions are false (see Figure 2.6).
Because the receiver is not synchronized with the modulated signal, sent data cannot be recovered unless the related phase error is continually detected and used for the correction of data reception. Figure 2.7 shows the effect of a phase and frequency offset on a 64−QAM demodulated data. The literature presents a large number of carrier phase tracking algorithms [3]. Chapter 4 of this Thesis presents specifically a VLSI Design of a the blind carrier phase tracking with guaranteed global convergence that overcomes the conventional blind carrier recovery algorithm which show unstable behaviours for large constellation modulation schemes such as the 256−QAM.

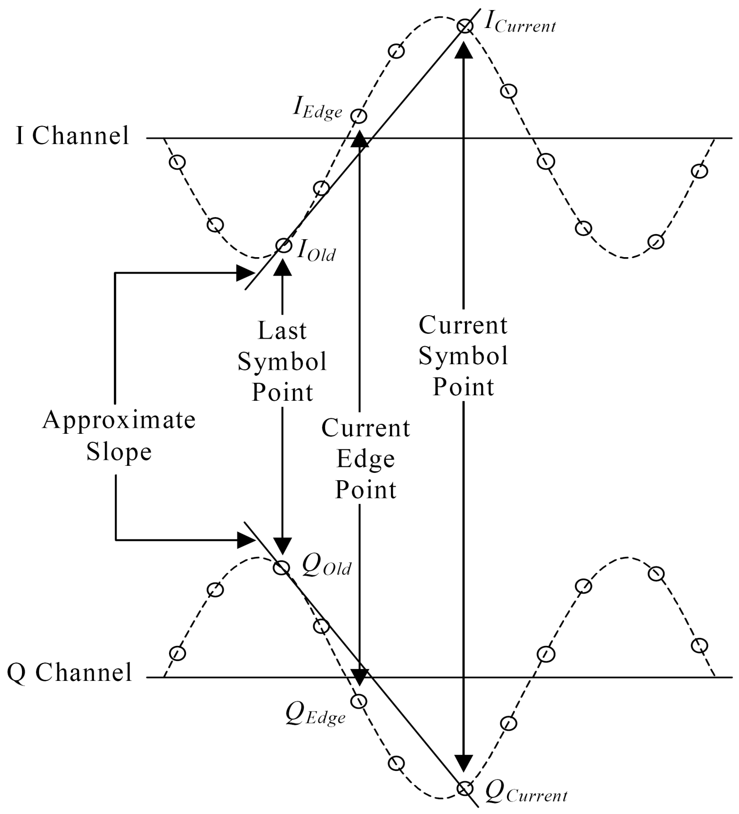


Figure 2.6: **Sampled Points for I and Q for a QAM−4 Signal as presented in reference** [14]

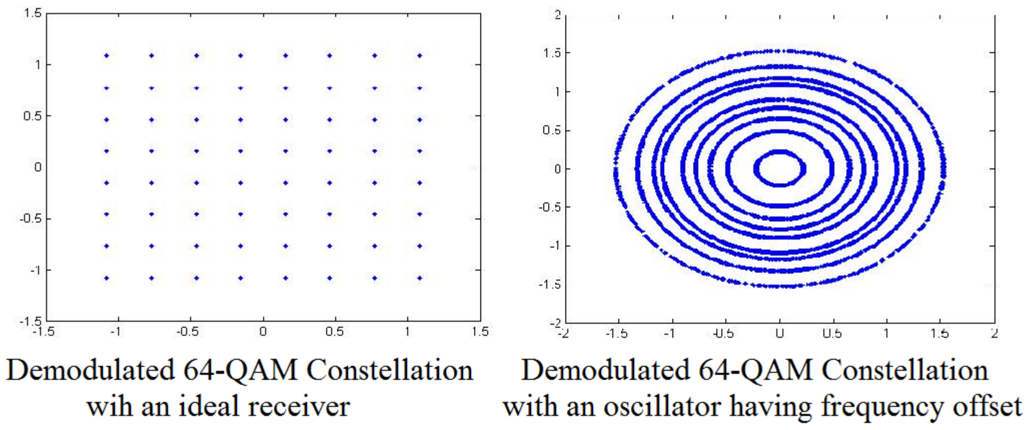


Figure 2.7: **Effect of phase and frequency offset on a 64−QAM Constellation**

Chapter 3

VLSI AND DIGITAL LINEAR PHASE DETECTOR

This Chapter describes the requirements to build an LPD. It specifies the design of the usual linear phase detector and goes through the main phase detection problems considered in synchronization common generic criteria [16]. It formulates those problems. Then, it gives the VLSI solutions to resolve them.

3.1 Problem Formulation Through Standardised Criteria

This section describes the global problems that should be addressed by the proposed solutions. We start by defining the timing concepts needed to understand the addressed problems. Those concepts are normalized in industrial standards like the GR−1244 [16].

3.1.1 Phase Alignment and Phase Offset

The phase alignment that we are considering in this section is a situation where the edges of *ref* and *fb* timely coincide. Their phase offset is actually their phase difference. At a system level, when a phase alignment or a phase offset is required, it can be an Output to Output alignment or an Input to Output alignment or a phase offset at the Input or at the output. To change the phase offset related criteria a multitude of modules may be evolved. The whole system's data paths as well as external offset settings determine the end alignments and offsets. In the phase detector, it is possible to set a kind of initial center frequency to propagate, through the system, controllable phase steps.

3.1.2 Phase Change during Pull−In

Pull-in is the process during which the output of a synchronization system is attempting to lock to the reference input. And the Pull-in range is a measure of the maximum output

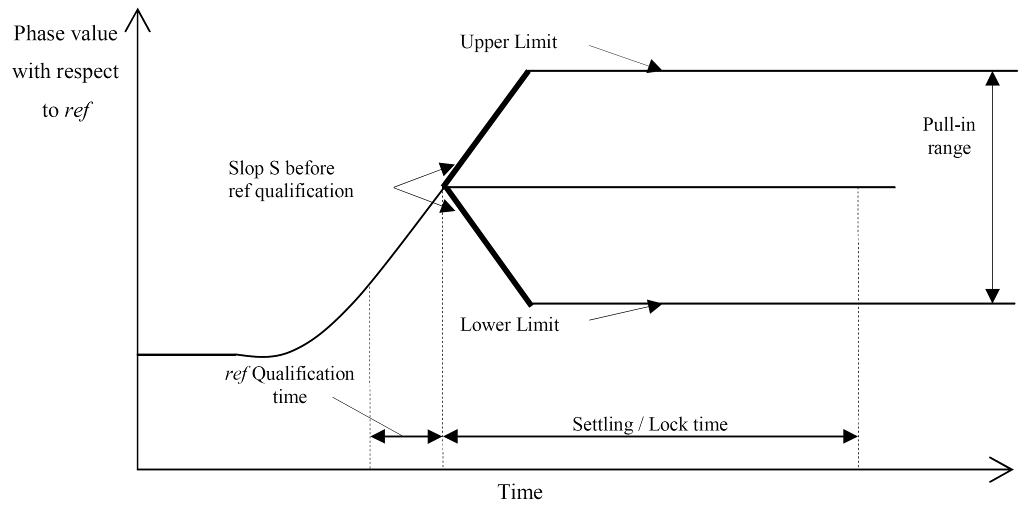


Figure 3.1: **Pull-in Range Mask Representation**

phase deviation respectively to the reference input phase. Figure 3.1 shows a timing representation of the Pull-in range process.

During pull-in the signal noise should be kept in the mask between the upper and the lower limits. The phase change caused by the locking process should not go over the slope limits S and $-S$ represented in Figure 3.1 .
Figure 3.2 shows an example of a valid phase pull-in. Note that the phase changes do not go neither beyond the required slopes nor beyond the required range. We can also note the settling time during which the signal tends to lock to the reference input phase.
Figure 3.1 shows a qualification time for the reference input. That is a time during which the reference is not yet valid; consequently, no phase detection is needed during that time. The mask, in that figure, defines the requirements for the phase pull−in range as well as the locking requirements.

3.1.3 Reference Input Qualification

The input references to synchronization systems (Figure 3.1) have qualification criteria as well as priority settings. Usually more than one reference is available to be used. And internal configuration sets the qualification criteria and a priority level for each of the references.
The qualification criteria determine the failure conditions for each reference and the priorities determine which reference to switch to when losing the used one.
For example reference [16] reports input failures according to:

- ⋄ Coarse Frequency Monitoring that invalidates the reference input when its corresponding frequency goes over 3% relatively to its nominal value

- ⋄ Single Cycle Monitoring that checks individual cycles and invalidates the reference for

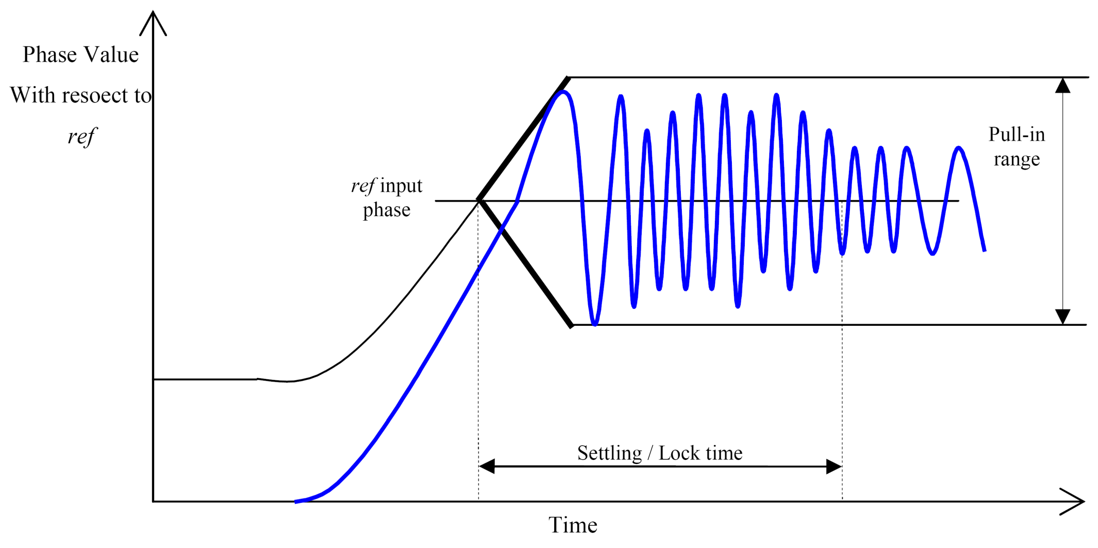


Figure 3.2: **Phase Pull-in Example**

phase hits.

3.1.4 Jitter and Wander Tolerance

This section gives the meaning of the Jitter and Wander tolerance. It starts by describing some concepts needed for Jitter and Wander measurement and for understanding the related tolerance masks.

Unit Interval (UI)

A UI is the reciprocal of data rate. In a synchronized system with one clock, a UI is the period of the system clock.

Time Interval Error (TIE) and Maximum Time Interval Error (MTIE)

TIE is the phase variation of a considered signal relative to an ideal reference input over a measurement period (called the observation time in [16]). Figure 3.3 shows an example of a TIE measurement. It shows also the MTIE calculation resulting from the TIE measurement. MTIE is the maximum pick to pick phase variation of the considered signal relative to the ideal reference over the observation time.

Jitter and Wander Descriptions

The Jitter is short term variations of the considered signal's significant instants (Figure 3.4). Those variations expressed as phase oscillation frequency are greater than 10 Hz. When those variations are long term ones (<10Hz) they are called Wander.

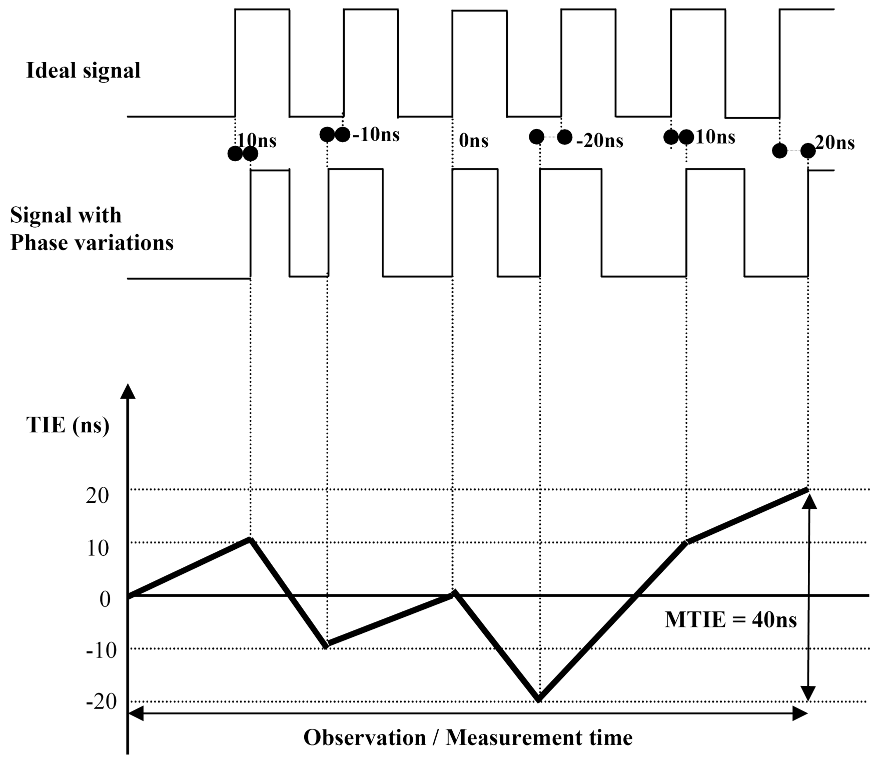


Figure 3.3: **TIE and MTIE Representation**

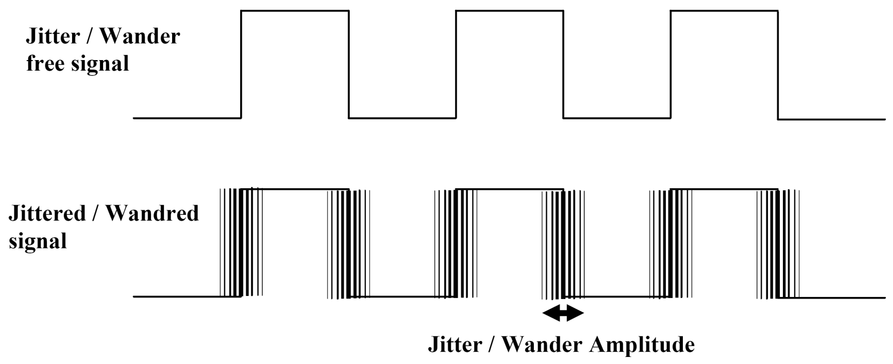

Jitter / Wander free signal
Jittered / Wandred signal
Jitter / Wander Amplitude

Figure 3.4: **Jitter representation**

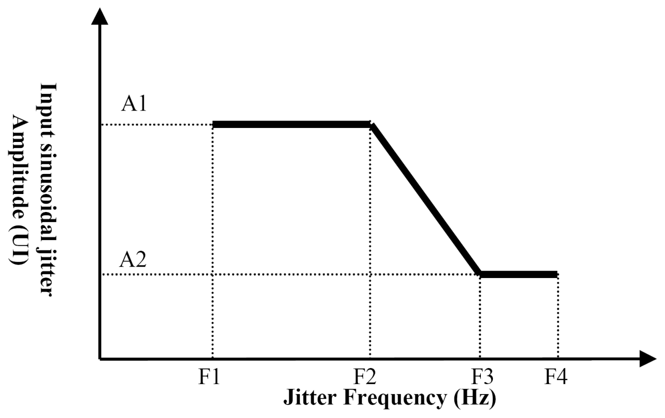


Figure 3.5: **Jitter tolerance Mask**

The Jitter is quantified in units of UIs or in terms of phase time while the Wander is usually measured using TIE, MTIE and TDEV measurement parameters.
The Jitter and Wander Tolerance is a measurement checking the higher levels of Jitter and Wander that are allowed to go through the system. A Jitter and Wander levels are characterized by their amplitudes and frequencies as illustrated in Figure 3.5 . A1 and A2 correspond to amplitude limits. F1, F2, F3 and F4 correspond to Frequency levels.

3.1.5 Phase Transient Tolerance

Synchronization rearrangements may cause unexpected short time phase step, called phase transient, causing the impairment of the system. So MTIE limits are set to characterize the transient tolerances. Figure 3.6 shows an example where different measurement times correspond to different transient tolerances.

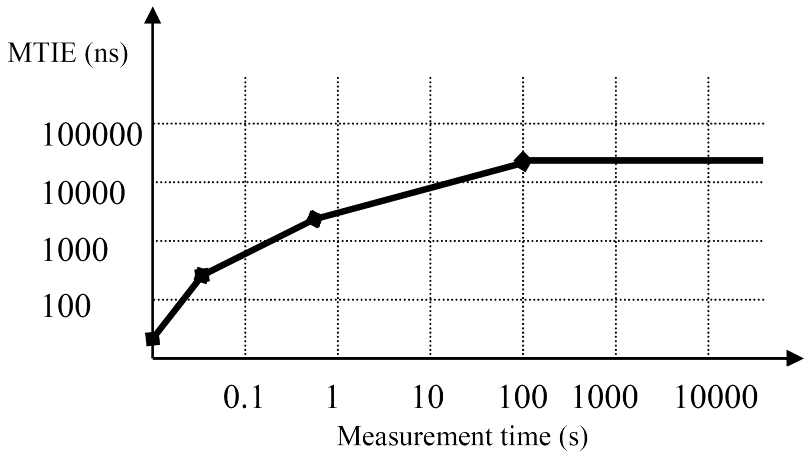


Figure 3.6: **Phase transient Tolerance Mask**

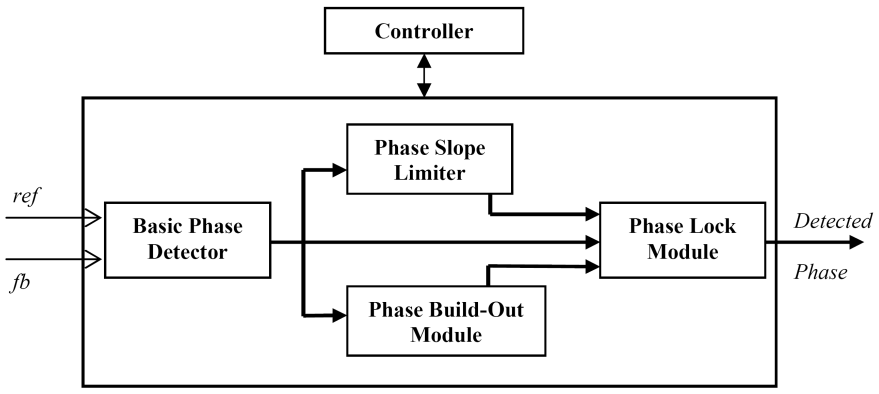


Figure 3.7: **Linear Phase Detector Top Level Bloc Diagram**

3.1.6 LPD Global Problem Description

The linear phase detector to design should resolve the global problems described in Table 3.1.

3.2 LPD Design Specification

The linear phase detector in Figure 3.7 is organized in five modules. (1) The basic phase detector giving the phase difference between the reference input (ref) and the feedback output (*fb*). (2) The phase slope detector and limiter. (3) The phase lock detector. (4) The phase Build-Out Controller. And (5) the Top level Controller that contains the Top phase detector's state machine as well as its register settings and status signals.

The Basic Phase detector tracks the *ref* and *fb* phase difference and sets offsets if they are needed. It also sets the limits to the *ref*-*fb* difference according to the needed application

Table 3.1: **Linear Phase Detector Global Problems**

GLOBAL PROBLEM	DESCRIPTION
Phase Difference between *ref* and *fb*	That is the basic phase difference which is tracked by usual phase detectors
Phase change during Pull-in	Phase movement when a synchronization system acquires lock
ref and *fb* not aligned or not offset as desired	Phase difference between *ref* and *fb* need alignment or offset control
Continuous Short Phase deviations	Continuous and short deviations difficult or impossible to detected instantaneously
Phase not Locking	Synchronization system does not lock or looses its lock
Phase Transient	Phase value goes out of the allowed phase range
Jitter Tolerance	The phase change should comply to a standardised mask
Transient occurs but should be ignored	Phase transient occurs but the related application requires to ignore it in defined conditions – requiring Phase Build-Out capability

Jitter Tolerance. The Phase Slope Limiter detects the slope of the phase difference and provides the ability of limiting it. The Phase Build Out module checks the evolution of the phase difference to determine whether the output phase should be tracking phase transients or ignoring them. After *ref* qualification (Section 3.1.3), the Phase Lock Module checks the deviations of the phase difference over a predetermined amount of time called a settling time in Figure 3.1 and Figure 3.2 . It indicates locking states when the phase difference keeps a deviation range within required limits. The top level controller contains the linear phase detector's top level state machine as well as the register map needed for the sub modules status and configuration.
Table 3.2 shows the causes of the global problems listed in Table 3.1 as well as their corresponding global objectives and design modules.

The following sections describe the design of the Top level modules.

3.3 Basic Phase detector Module

This input module (Figure 3.7) extracts the phase difference value from its two input signals, *ref* and *fb*, with respects to their rising edges. Figure 3.8 shows the basic phase detector block diagram.
The precedence of the *ref* or the *fb* rising edges is indicated respectively with ref_leads and fb_leads (Figure 3.8) and a counter outputs the searched phase difference by incrementing the phase steps between the two edges.

The rising edges of *ref* and **fb** are determined in two clock cycles as shown in Figure 3.9
.

In Figure 3.8 setting high ref_leads indicates that *ref* positive edge occurs before *fb* positive edge, and setting high fb_leads indicates that *fb* positive edge occurs before *ref* positive edge.

Table 3.2: **Global Problems vs. Global Objectives**

GLOBAL PROBLEM	CAUSE	GLOBAL OBJECTIVE	DESIGN IN
ref - *fb* phase difference	*ref* phase and *fb* phase not aligned and/or not synchronized	Find *ref*, *fb* phase precedence and difference	Basic Phase Detector
Phase Change during Pull-in	Sharpe Slope	Indicate Slope violation	Phase Slope limiter
		Correct Slope	
	Phase value out of range	Indicate range violation	Lock Detector
		Correct range violation	Basic Phase Detector
ref and *fb* not aligned or not offset as desired	Change at data path delay. Change at the system clock accuracy	Set alignment or set the desired offset	Basic Phase Detector
Continuous Short Phase deviations	Transferred or intrinsic Wander	Detect phase slop trend	Phase Slope limiter
		Correct the deviation	
Phase not Locking	During Lock timeframe, phase out of range	Indicate range violation	Lock Detector
		Indicate Lock	
Phase Transient	Short time phase out of range	Indicate range violation	Lock Detector
Jitter Tolerance	Phase value Out of the tolerated Jitter range	Indicate the range violation	Basic Phase Detector
		Limits the phase range to transfer	
Transient occurs but should be ignored	Phase hit	Phase build Out	Phase Build Out Module

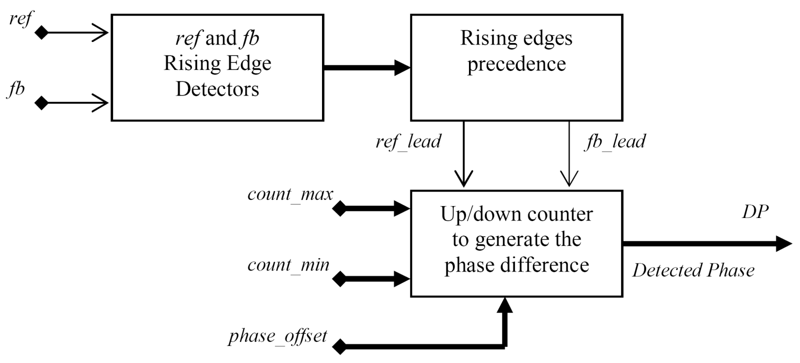


Figure 3.8: **Basic Phase Detector Bloc Diagram**

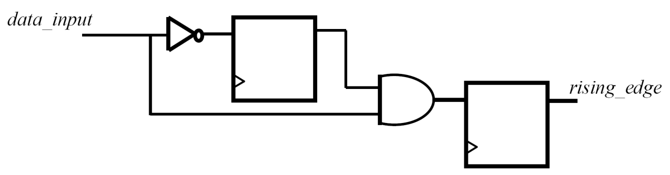


Figure 3.9: **Rising Edge Logic**

Table 3.3: **Basic Phase Detector I/O Description**

I/O Name	I/O Direction	I/O Description
clk	Input	System Clock
reset_b	Input	Synchronous reset
ref	Input	Reference input
fb	Input	Feadback signal
count_max	Input	Maximum allowed counter output (DP) value. Used for Jitter tolerance
count_min	Input	Minimum allowed counter output (DP) value. Used for Jitter tolerance
Phase_offset	Input	Phase offset value
DP	Output	Detected phase. Obtained from Ref, Fb phase difference

If ref_leads = 1 the up/down counter is incremented, and if fb_leads= 1 it is decremented. That way, the counter outputs the phase difference value in two's complement format.

Note that if ref was a sampled sinusoidal data, the same phase detector architecture, as in Figure 3.8 , might be used. In such case, to generate the rising edges indicators, comparators would be used to monitor the signal's zero crossing. i.e. if n was a data sample index, the data rising edge is indicated when (ref(n-1) < 0 and ref(n) ≥ 0).

3.3.1 Jitter Tolerance Capability

Considering that the system clock is running at F_{sys} frequency, it gives a phase step of ($1/F_{sys}$) for every increment for the absolute value of DP (|DP|).
If F_{data} was the *ref* and *fb* frequency, the largest |DP| value would be ((F_{sys} / F_{data})-1) = Max_{DP}. In any given application the allowed range of the Jitter amplitude is specified. So, if [-Max_{DP}, Max_{DP}] range goes over the Jitter tolerance range the count-up and count-down giving DP values are limited to max and min values (count-max and count-min in Figure 3.8) respectively preventing the intolerable Jitter to propagate through the system.

3.3.2 Initial Phase offset

As explained in Section 3.1.1, we can control the phase steps values (DP) by adding a desired constant value "phase_offset" to the counter increment. So, ideally when *ref* and *fb* are perfectly aligned the DP value should be equal to the wanted phase_offset to propagate.

3.4 Phase Slope Limiter

The speed of the phase change is formulated by the phase slope calculated by the differential fraction of a phase change value $(\phi_2 - \phi_1)$ over a delay $(t_2 - t_1)$ as presented in Figure 3.10 .
When $(t_2 - t_1)$ is pre-determined and constant the slope can be coded as the phase difference $(\phi_2 - \phi_1)$. While if the slope detection is related to events, the whole $(\phi_2 - \phi_1)/(t_2 - t_1)$ is implemented. For example, when a slope value is needed for a phase change between a measured value and a fixed threshold, ϕ_1, t_1 and t_2 are not known beforehand. So, the division operation becomes a must.
For both cases the phase slope monitoring can be achieved by successive comparisons of measured slopes against a maximum allowed phase slope as presented in Figure 3.11 .
The phase slope limit is a parameter that does not allow fast phase change to go through the system. It may be considered in the filtering part of a synchronization system. However, because all the required parameters are available at the phase detection part, it is worth implementing the slope limiter at this stage.
Obviously, the phase slope limiter module can be disabled if there are benefits to combine the slope limit parameters with the bandwidth settings in the used synchronization filter.
This section shows how the phase slope is monitored and limited for both of the cases; when $(t_2 - t_1)$ is a constant and when it is variable.
As mentioned above, Figure 3.11 shows how the phase slope calculation monitors continually the phase incline. Then, if a change is found sharper than an allowed slope, it is limited as presented in Figure 3.12 . When no phase alignment is required, the phase difference keeps being a fraction of the direct difference obtained from the *ref* and the *fb* phases. While, when the phase alignment is required, the phase, with the limited slope, jumps to be aligned to the direct difference between the *ref* and the *fb* phases.

3.4.1 Phase Slope Limiter Design Specification

The input phase slope is detected with a configurable timer, a phase and time interval detector and with a divider (Figure 3.13). A comparator monitors the input phase slope and generates an enable signal (called en_limit in Figure 3.13) to trigger the phase slope limitation when it is required.
The phase slope limit is characterized by two parameters the phase slope limit (slope_limit) that indicates violations and the phase slope fraction (PF) that is used to reduce the phase slope to a chosen level of phase incline. Then, if phase alignment is required, as soon as the input phase slope goes back to an allowed value, the generated phase with the limited

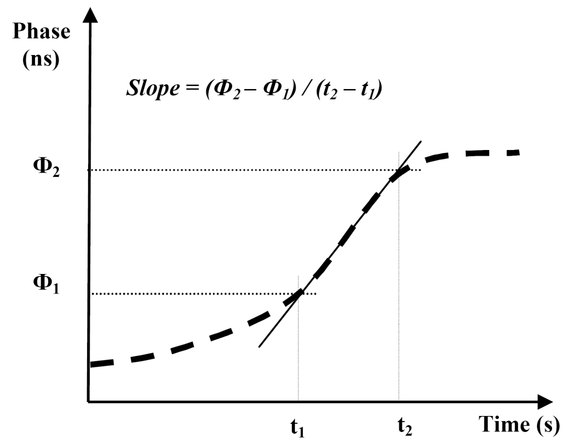


Figure 3.10: **Phase Slope Expression**

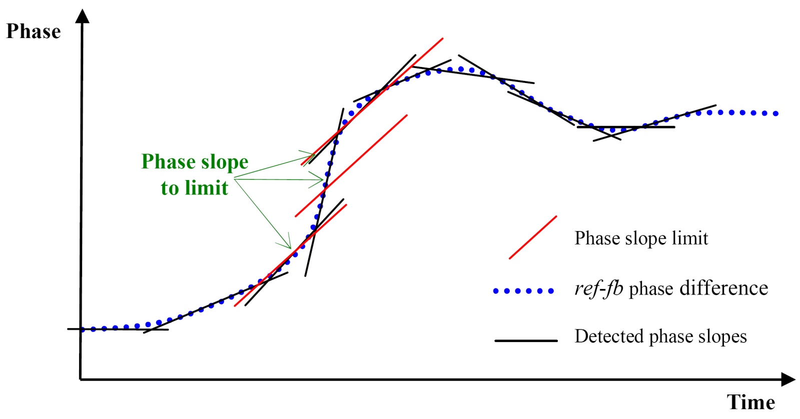


Figure 3.11: **Phase slope limit and monitoring**

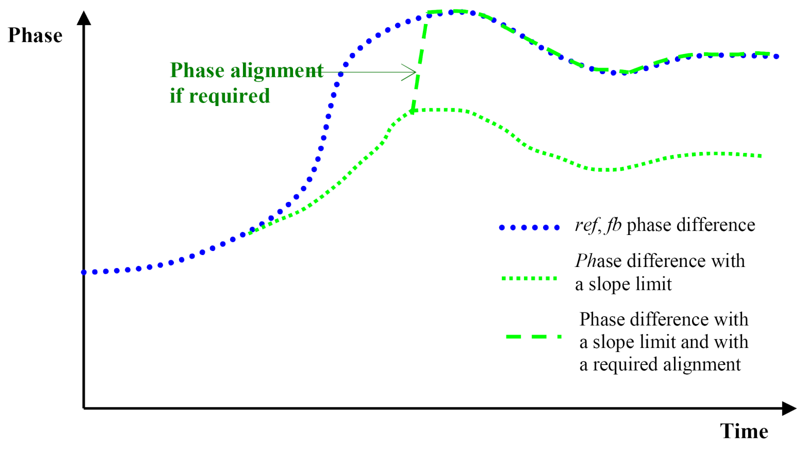

Phase
Phase alignment if required
ref, fb phase difference
Phase difference with a slope limit
Phase difference with a slope limit and with a required alignment
Time

Figure 3.12: **Limited phase slope representation**

Table 3.4: **Phase Slope Limiter I/O Description**

I/O Name	I/O Direction	I/O Description
clk	Input	System Clock
reset_b	Input	Synchronous reset
DP	Input	Detected phase. Obtained from *ref-fb* difference
en_slop_corr	Input	Enable phase slope correction
slop_t1	Input	Time reference to determine the time interval for phase slope measurement
slop_t2	Input	Time reference to determine the time interval for phase slope measurement
align	Input	Alignment commend
slope_limit	Input	Phase limit value
PF	Input	Phase fraction. *DP* * FP gives the limited phase offset. It is used when the *DP* phase slope goes over the allowed limit
slop_limit_vio	Output	Active high. Indication that the slope limit is violated
ph_s_corrected	Output	Active high. Indication that the phase slope has been corrected
DPS	Ouput	Detected Phase Slope value
phase_s_out	Output	Slope limiter output phase.

slope jumps to be aligned. If not (no alignment required) the phase with the limited slope does not jump; it tracks though the wanted and valid slope.
Table 3.4 shows the I/O of the phase slope limiter module. For the best possible phase accuracy, the phase value is coded with large bus-widths. However, dividers with any large bus-width are not available. In arithmetic implementation, dealing with large bus widths has been always problematic. For example, in programmable logic devices, the largest Core generated divider with Xilinx tools has 32bit width [18] and dividing multiplication results after using the dedicated 18bitMult [19] requires a reduction of the 36bit products to the only possible 32bit divider's operands. That leads to additional work to solve the induced problems of accuracy. In the case of our slope limiter, *ref* and *fb* may have low frequencies while the system clock used to count phase steps may run at a very high Frequency. So, the counted phase difference between *ref* and *fb* should be coded on a large bus width. For example, for a 1Hz *ref* and *fb* frequency and a system clock of 200Mhz, the phase difference between *ref* and *fb* may go up to 19999999 phase steps requiring 28 bitwidth.
Considering the absolute value of phase slopes, when a phase slope limit is required, we have no need to process larger values than the limit while the small slopes (numbers) are needed particularly to measure continuous and short phase deviations as mentioned in Table 3.1 and Table 3.2. In such designs it is beneficial to find a way of keeping the highest possible accuracy for relatively small phases even at the expense of relaxing the accuracy

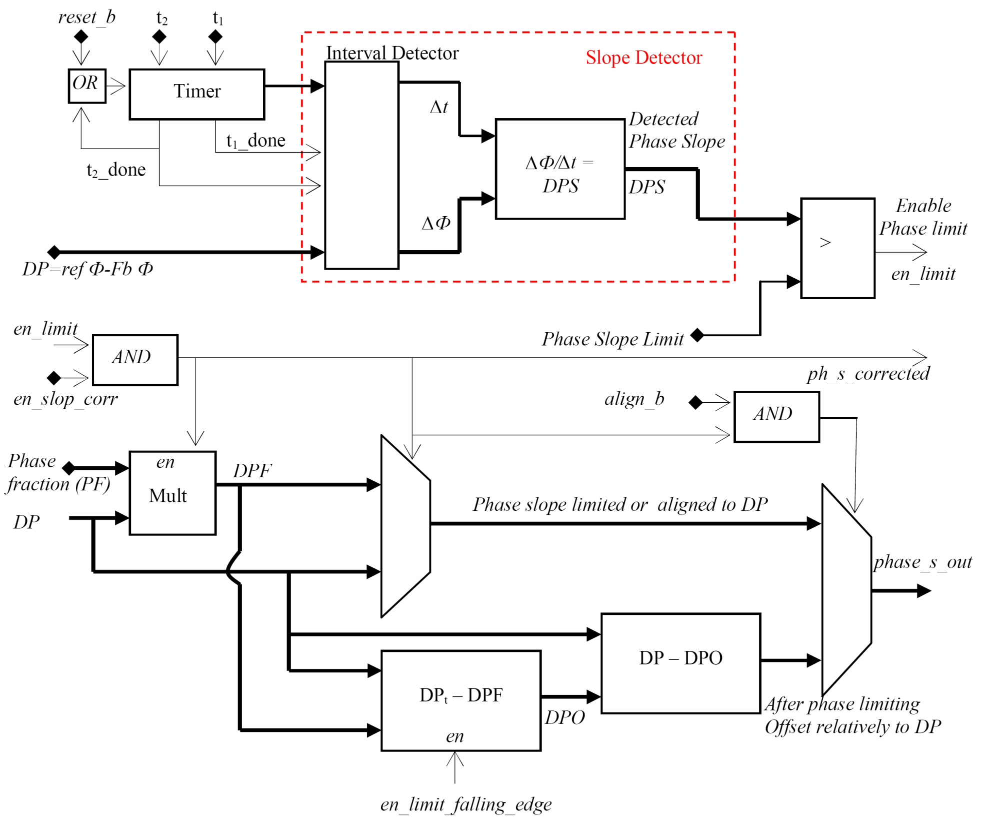


Figure 3.13: **Phase Slope Limiter Conceptual Block Diagram**

when dealing with large slopes. Actually, the next section is showing how to dynamically perform that accuracy adjustment. After determining the data thresholds corresponding to the needed accuracy levels, our technique can be used appropriately to reduce the data size and dynamically indicate which level of accuracy that is being processed.

3.4.2 Dynamic Accuracy Adjustment

Only two's complement integer numbers are considered. That means the accuracy has the same meaning as the resolution [20].
An N−bit two's complement fixed point integer can be expressed as follows:

$$x = -2^{N-1}x_{N-1} + \sum_{0}^{N-2} 2^i x_i \tag{3.1}$$

Where x_i represents the bit i of x.
When only the Most Significant Bit (MSB) represents the sign bit, $s = x_{N-1}$, x has the largest absolute value. Otherwise, all the redundant MSBs constitute the sign bits' extension.
Let us consider $x_M$the bit M of x representing the sign bit with the lowest index. If x does not have any sign extension, M would be the same as $N-1$. For $1 \leq M \leq N-2$

$$-2^{N-1}s + 2^{N-2}s + ... + 2^M s = 2^M s$$

Then, equation 3.1 can be written as:

$$x = -2^M s + \sum_{0}^{M-1} 2^i x_i \tag{3.2}$$

equation 3.2 shows that for every Nbit processed number if the x_M bit is detected, the Nbit number can be reduced to $M+1$ bits without altering the original Nbit accuracy.
Usually, to keep the size coherence of arithmetic operators, sign bit redundancy is not removed. Nevertheless, reference [8] shows that in some operations removing sign bit redundancy leads to important performance improvements. In the same way, when bit truncation is required, instead of reducing the LSBs, as usual, an option, would be to remove first, as much as possible, the redundancy of the sign bits, then proceed to the truncation of LSBs. That way, a combination of MSBs and LSBs is considered to give the best possible accuracy for a given amount of bits.
For example, 2 in 5bit two's complement is 11110. Truncating the Least Significant Bit (LSB) from that number would give 1111 (-1 in decimal). And truncating the MSB would give 1110 (2 in decimal). So, for a four bit operator truncating the MSB from -2 allows keeping the accuracy of the 5bit number.
This technique goes back though to the same accuracy as the usual LSB only truncation

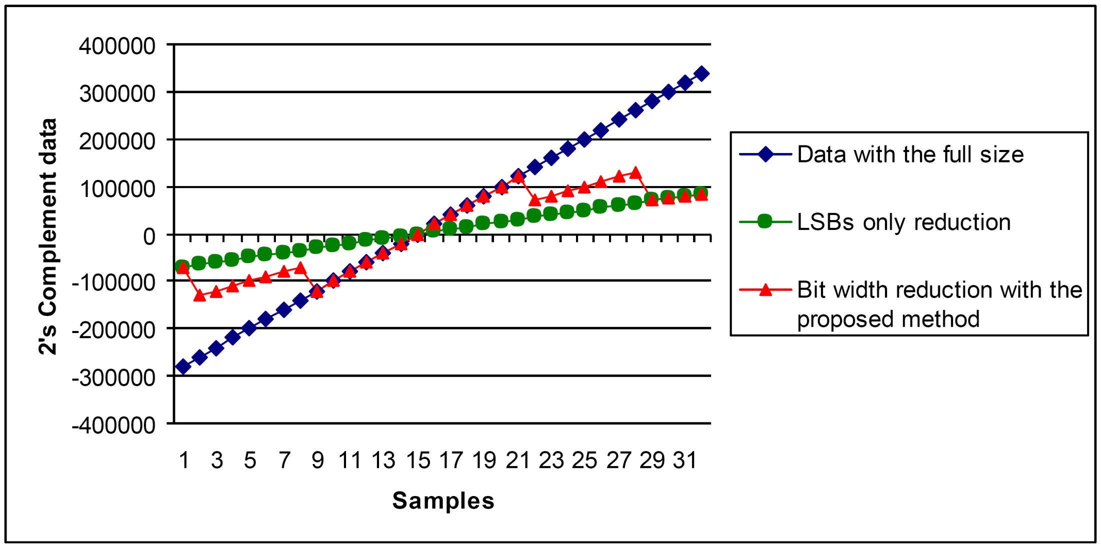


Figure 3.14: **2's Complement two bits reduction with the LSB only reduction and with the proposed method**

when operands have no sign extension. For instance, if we needed to truncate one bit from 01111 (that is 15), we would have no choice than getting 0111 (that is 7): in this situation no MSB can be removed.
Figure 3.14 shows 20bit two's complement numbers with the result of their two bits reduction; a reduction with the usual LSB only truncation technique and with the proposed technique. When the proposed technique is applied three levels of accuracy, corresponding to two bit reduction, are provided. If R was the number of bits to reduce, the technique would provide $R + 1$ levels of accuracy.

Accuracy Adjustment Design:

Based on the two's complement word reduction presented above, this section shows how to effectively reduce the word length while keeping its accuracy information.
Figure 3.15 (a) shows the logical circuit to implement the proposed method. The logic is completely generic. To simplify its explanation 2bit reduction is used for illustration.
In Figure 3.15 (a), the XORs dynamically find the sign extension and set (A1,A0) signals. The multiplexer MUX selects the reduced bus to output. A0 and A1 control the MUX and state the actual accuracy to be used forward in the system.

- ▷ (A1,A0) = (0,0) means that two MSBs are removed giving the accuracy of the numbers with the full size (Figure 3.14).
- ▷ (A1,A0) = (1,1) or (A1,A0) = (1,0) means that two LSBs are removed giving the accuracy of the numbers with the usual LSB reduction method

▷ (A1,A0) = (0,1) means that the MSB and the LSB are removed giving an intermediate level of accuracy.

$$A1 = \ln[33] \ \text{ xor } \ \ln[32] \tag{3.3}$$

$$A0 = \ln[32] \ \text{ xor } \ \ln[31] \tag{3.4}$$

When considering a divider, both of the related operands should be reduced to the same accuracy corresponding to the operand with the largest absolute value.
Because of equation 3.3 and equation 3.4, if the two operands were X and Y and their accuracy indicators were (X$_{A1}$,X$_{A0}$) and (Y$_{A1}$,Y$_{A0}$) respectively (Figure 3.15 .b):

$$X_{A1} = X[33] \ \text{ xor } \ X[32], \quad X_{A0} = X[32] \ \text{ xor } \ X[31]$$

$$Y_{A1} = Y[33] \ \text{ xor } \ Y[32], \quad Y_{A0} = Y[32] \ \text{ xor } \ Y[31].$$

If D$_{A0}$ and D$_{A1}$ were the accuracy indicators of the whole division (Figure 3.15 (b)), to make sure that the operand with the largest absolute value always takes the lead a combinatorial simplification gives

$$D_{A0} = X_{A0} \ \ or \ \ Y_{A0}$$

$$D_{A1} = X_{A1} \ \ or \ \ Y_{A1}$$

For N bit operands X and Y, and for a need of M bits reduction, with $1 \leq M \leq N-2$, individual accuracy indicators can be expressed as:

$X_{A(M-1)} = X[N-1] \quad \text{xor} \quad X[N-2]$

$X_{A(M-1)} = X[N-2] \quad \text{xor} \quad X[N-3]$

$\vdots$

$X_{A0} = X[N-M] \quad \text{xor} \quad X[N-(M+1)]$

Considering that $D_A = \{D_{A(M-1)}, D_{A(M-2)}, \cdots, D_{A0}\}$ as the operation's accuracy indicator (Figure 3.15 (c)), It would be expressed as:

$D_{A0} = X_{A0} \quad \text{or} \quad Y_{A0}$

$D_{A1} = X_{A1} \quad \text{or} \quad Y_{A1}$

$\vdots$

$D_{A(M-1)} = X_{A(M-1)} \quad \text{or} \quad Y_{A(M-1)}$

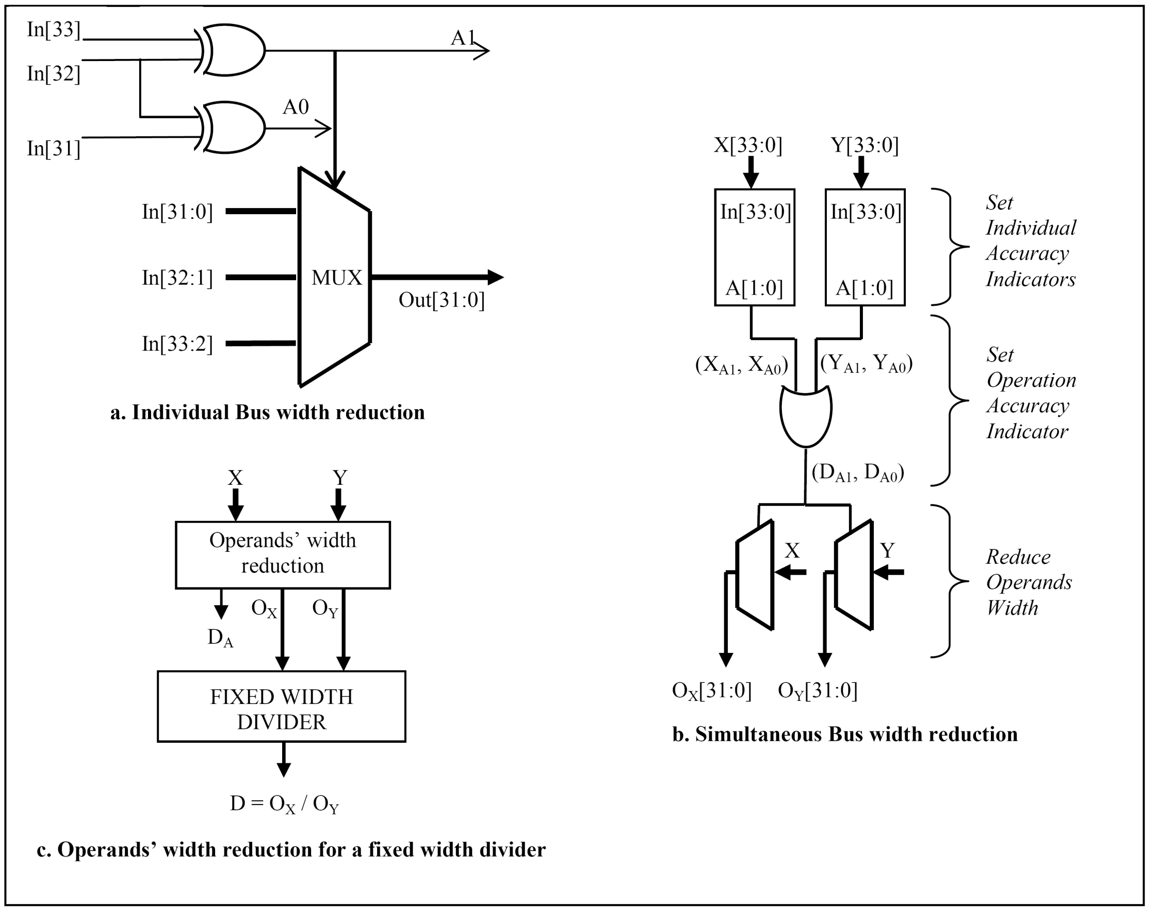


Figure 3.15: **Dynamic Accuracy Adjustment for a Fixed Width Divider**

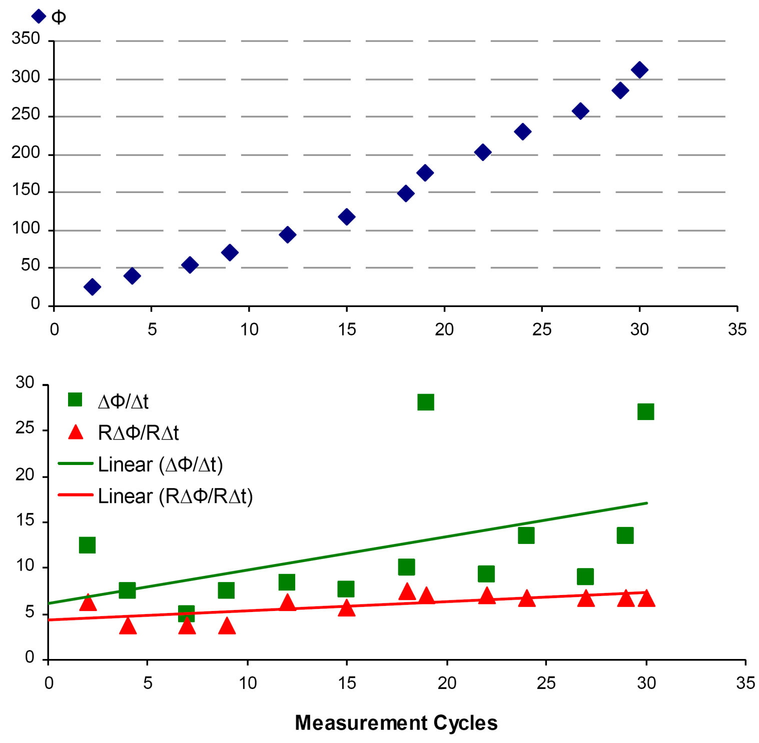


Figure 3.16: **Linear trends of the phase change detected by using the proposed method (squares) and by using the LSB only reduction (triangles)**

If the dynamic aspect of the proposed reduction (Figure 3.15 (c)) is not needed, it can be avoided by pre-setting an accuracy level (D_A) at power up and the related circuit runs with a fixed, but configurable, accuracy. When a bus width reduction is required, using the usual LSB reduction would present wrongly successive and continuous small changes as a phase without any deviation. For instance, when a 34bit phase is input to a 32bit generated divider [18], removing systematically two LSBs of the operands would lose, among other information, all the phase inclines that should be detected by successive divisions over 2 and over 3. Figure 3.16 shows a trend of a phase change captured by our proposed method and that would be lost if the phase and the coded time cycles were systematically reduced from LSBs.
In that figure, if two LSBs were removed from the corresponding data we would obtain a flattened trend instead of the significantly increasing trend. Without our proposed method, those small and continuous deviations would be detected tardily by measuring the wonder of the related signal [16]. Actually, Figure 3.11 shows that to comply with some standards [16] some phase slope limits are set. Then, the absolute values that are larger than the limits do not even need to be precisely known. In such cases, the smallest values give more sense to be known with the highest possible accuracy.

3.5 Phase Build-Out Module

In synchronized systems the output signal and its related reference input has an average fixed relationship. And the phase detector is the engine which regulates and pushes the system to that fixed relationship. In case of a phase hit the system does not always ignore it. It depends on that fixed relationship related to the considered application. Systems that are allowed to ignore the phase hit supports a standard concept called ”phase build−out” [16].
The steps to trigger a phase build out can be summarised as follows:

◇ the transient goes beyond the phase transient tolerance (Figure 3.6)

◇ the signal is not complying to the Jitter tolerance requirements (Figure 3.5)

◇ it goes beyond a phase build out triggering threshold and over a period of time called in Figure 3.17 ”allowed Transient period to not build−out”. The triggering threshold is not necessarily the transient or the Jitter tolerance range

◇ the phase slop of the transient is calculated

◇ the end of the transient is detected before a time out limit

◇ After the end of the transient, the phase is relatively settled (”settling time” in Figure 3.17); it does not violate neither the transient nor the jitter tolerance requirements for a determined amount of time

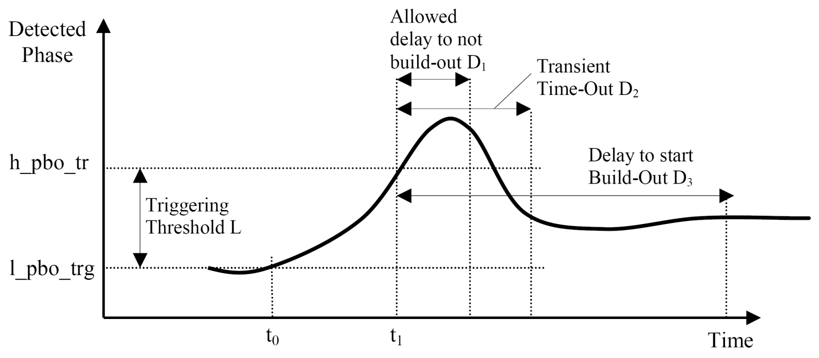


Figure 3.17: **Phase Build−out parameters**

When the phase build−out starts, an indication is sent to the filtering part of the synchronization system (Figure 3) to ignore the incoming phase and use the phase hold before the phase transient crosses h_pbo_trg (Figure 3.17).
Status signals are used to indicate that a build out is not required even if a transient occurred.

3.5.1 Phase Build-Out Design Specification

The Phase build-out module is enabled when en_pbo is set high and the phase slope limiter is disabled (Table 3.5). The state diagram in Figure 3.18 shows mainly the conditions that lead to start the phase build−out process. Basically, it checks that the trigger threshold is crossed, then it ensures that the transient delay is neither shorter than D1 nor larger than the time−out delay D2. It checks that the signal is settled for a required (D3 − D2) delay (Figure 3.17). Finally, it sets the start_pbo command to build−out the phase. Then, it uses the phase hold when the input DP phase was crossing the triggering threshold.

3.6 Phase Lock Detector

The Phase Lock Detector monitors the *ref−fb* phase difference (DP), or the output of the Phase Slope Limiter or the output of the Phase Build Out module. It checks that the related deviations do not go over pre−determined limits during the phase settling time (Figure 3.1). Table 3.6 shows the I/O description and Figure 3.19 shows the state diagram of the Phase Lock Detector.

Table 3.5: **Phase Build-Out I/O Description**

I/O Name	**I/O Direction**	**I/O Description**
clk	Input	System Clock
reset_b	Input	Synchronous reset
en_pbo	Input	Enable Phase Build Out functionality
en_slope_corr	Input	Enable slope correction. When high Phase is not build-out
DP	Input	Detected phase. Obtained from *ref-fb* difference
l_pbo_trg	Input	Lowest level of the phase build out trigger threshold
h_pbo_trg	Input	Highest level of the phase build out trigger threshold (h_pbo_trg – l_pbo_trg = L in Figure 3.17)
D_1	Input	Allowed delay to not build-out
D_2	Input	Phase Transient Time-Out delay
D_3	Input	Minimum Delay to start building out the phase
slop_limit_vio	Input	Indication that the slope limit have been violated
pbo_l_crsd	Output	Active high. Indicator that phase Build Out Threshold crossed
pbo_t_l	output	Time reference to start measuring the phase build-out triggering threshold
pbo_t_h	output	Time reference to finish measuring the phase build-out triggering threshold
t_time_out	Output	Active high. Indicator that the transient time-out is reached
pbo_start	Output	Active high. Phase Build Out started
no_pbo	Output	Active high. Indicate that a Phase Build-out process has not been started
phase_b_out	Output	Phase Build-Out output value: phase hold when the input *DP* phase was crossing the triggering threshold

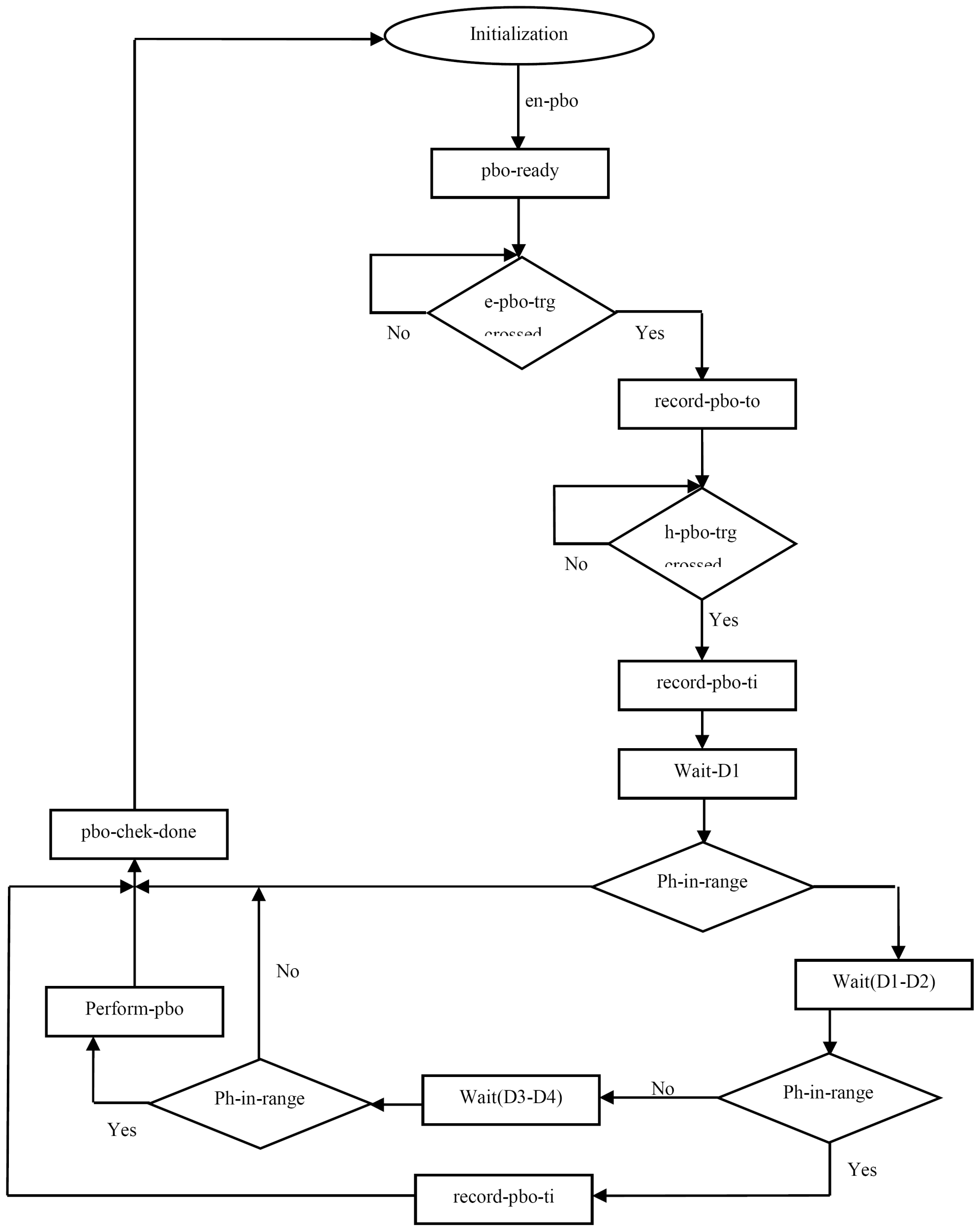


Figure 3.18: **Phase Build Out state diagram**

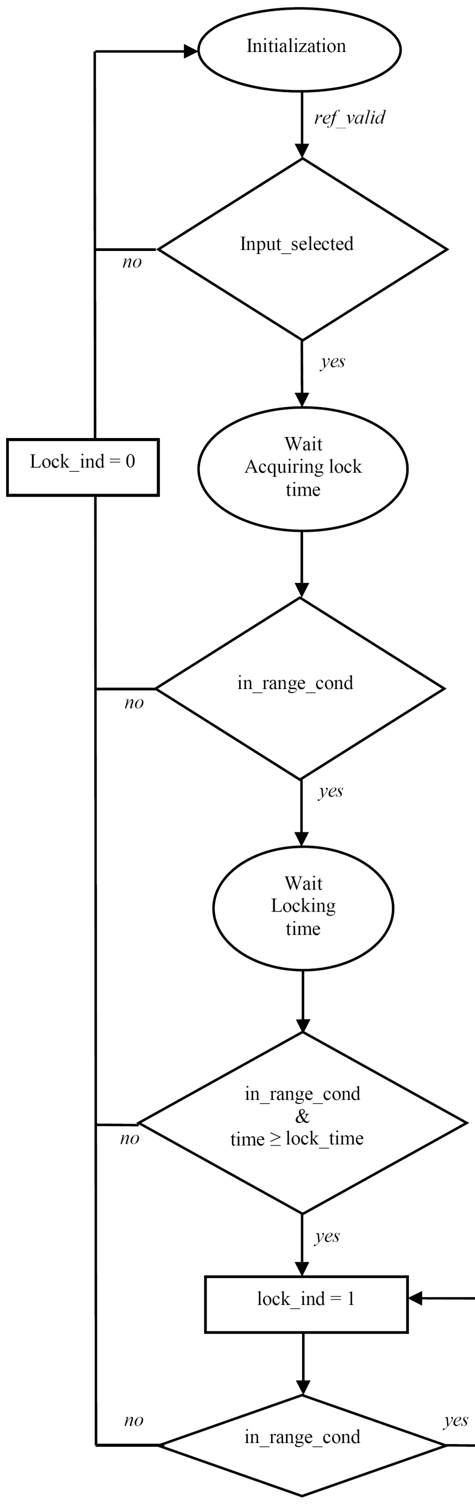


Figure 3.19: **Phase Lock Detector State Diagram**

Table 3.6: **Lock Detector I/O Description**

I/O Name	I/O Direction	I/O Description
clk	Input	System Clock
reset_b	Input	Synchronous reset
ref_valid	Input	Reference Input qualified
en_pbo	Input	Enable Phase Build Out Module
en_slop_corr	Input	Enable Slope Correction
slop_limit_vio	Input	Phase slope limit violated
lower_limit	Input	Lower phase limit to consider ref and Fb are locked
upper_limit	Input	Upper phase limit to considere ref and Fb locked. Allowed phase range = upper_limit – lower_limit
phase_b_out	Input	Phase value coming from phase build-out module
phase_s_out	Input	Phase value coming from the phase slope limiter
DP	Input	Phase value coming from the Basic Phase Detector
sel_phase_in	Input	Select one of the phase values 00: select *DP* 01: select *phase_s_out* 11: select *phase_b_out*
lock_ind	Output	Active high. Lock indicator

3.7 LPD Controller Module

This module is the top level controller of the Linear Phase Detector. As shown in Figure 3.20 a state machine controls the LPD data path and an Enhanced Parallel Port (EPP) sets the control parameters and records the status signals. The EPP details are given to provide a complete design reference. Any other interface works because no specific requirements are needed to configure our LPD.
The two next sections describe the LPD Top level control flow and the register map with its related interfaces.

3.7.1 LPD Top Level Control Flow

As presented in Figure 3.21 the user sets the Basic Phase Detector's parameters (Table 3.3) and records dynamically the phase difference PD that is used by the Phase Lock Detector to indicate the locking state. The Phase Slop Limiter and the Phase Build Out are configured and their outputs ph_s_out and ph_b_out may also be selected to be input to the Phase Lock Detector to monitor any related phase range violation. All the control and status signals of the Figure 3.21 are described as sub−module I/Os in Table 3.3, in Table 3.4, in Table 3.5 and in Table 3.6.

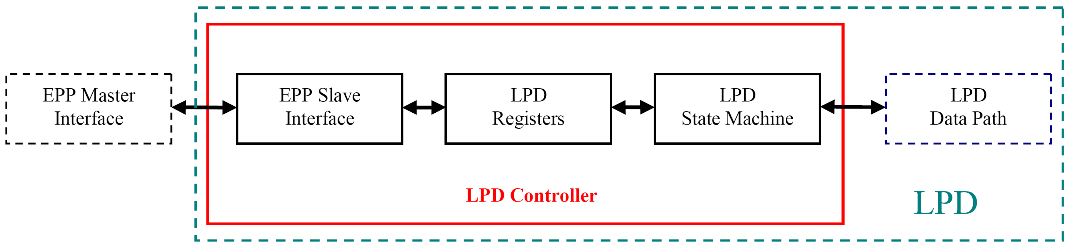


Figure 3.20: **LPD Controller between the external master interface and the LPD data path**

3.7.2 LPD Register Map and Interface

This section describes the followed methodology to build our register map and the details to implement the EPP protocol.

Register Addressing Mode

The addressing mode used for this project is a direct 8-bit addressing. However, larger and more complicated designs might require more elaborated addressing modes. An option would be to adopt indirect addressing mode where a base address is used as a page pointer and other set of addresses are used within each page.
Another option is to use chip select signals to select different parts of that elaborated circuit, and use the same set of addresses for the blocks that have exclusive chip select signals.
To build a register map, rules need to be followed as much as possible to ease the monitoring, the programmability, the design and the verification of the programmable registers. Here are some principals:

- ◇ Related registers are grouped in banks
- ◇ Register banks start from predictable address offsets
- ◇ Address offsets have to be multiples of a base power of two numbers
- ◇ Related bits in different registers and in different banks have to be in the same bit fields
- ◇ Address mapping leaves open addresses for future expansion
- ◇ Address mapping let the possibility to transform an implemented direct addressing mode to an indirect one. That way, if future expansion does not fit in an implemented register map, new pages can be created to contain expansions that go with existing register banks

Register Types

Read−Write Register can be updated in two situations: by the initialization condition and by the user when writing through the EPP interface.
Read Only Register records status signals.
Sticky Bit Register is a read only register but it cannot be updated dynamically. In this register, a bit set high does not go back to low even if the related status signal is low. It can be set low only after it is read through the controller.
Redundant Status Registers is a shift register recording continually changing status signals. At a read condition the output register keeps the last stable value constant to be transferred consistently to the Master interface (Figure 3.22).

Register Map Design Specification

As a detailed explanation of the Figure 3.20 , this section describes the Top level I/Os (Table 3.7), the content of the configurable registers (Table 3.8) and the registers configuration timing diagram (Figure 3.22 and Figure 3.23).

Figure 3.23 and Figure 3.24 present the Read/Write protocol of the configurable registers.

EPP Slave Design Specification

This section describes the EPP slave design as well as the related transfer protocol. Table 3.9 gives the corresponding I/Os.

The Address write cycle is shown in Figure 3.25 . Here is its protocol description:

◇ The peripheral data output is disabled and the epp_wait is low

◇ The host brings epp_rw_b low, which causes the byte to appear on epp_data, and brings epp_astrobe low

◇ The peripheral brings epp_wait high to signal that it is ready to latch the address

◇ The host brings epp_astrobe high to cause the peripheral to latch the address

◇ When the peripheral is ready for another byte, it brings epp_wait low

Data write cycle is identical to address write, except that the host uses epp_dstrobe instead of epp_astrobe.
The Address read cycle is shown in Figure 3.26 . Here is its protocol description:

◇ The peripheral brings epp_wait low. The host brings epp_rw_b high, and brings epp_astrobe low

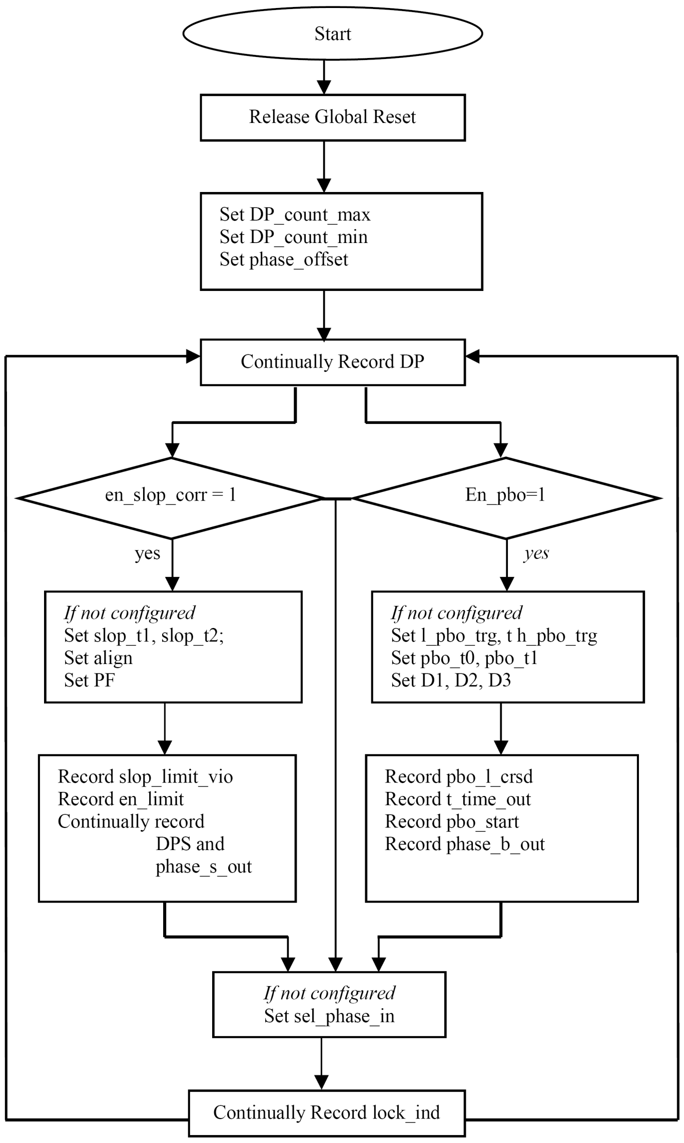


Figure 3.21: **Linear Phase Detector Controller State Diagram**

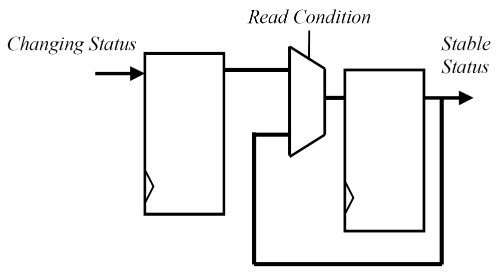


Figure 3.22: **Redundant Status Register**

Table 3.7: **I/O Description of the Configurable Registers' Bloc**

I/O Name	**I/O Direction**	**I/O Description**
clk	Input	System Clock
reset_b	Input	Synchronous reset
ref_valid	Input	Reference Input qualified
The following five signals are connected to the EPP Slave		
reg_address	Input	Address bus from EPP interface
data_to_reg	Input	Data bus from EPP interface
reg_enable	Input	Enables access to registers
r_w_b	Input	read write commend from EPP interface 0: write, 1: read
data_from_reg	Output	Data bus to EPP interface
The following four signals are connected to the Basic Phase detector		
count_max	Output	Maximum allowed counter output (*DP*) value. Used for Jitter tolerance
count_min	Output	Minimum allowed counter output (*DP*) value. Used for Jitter tolerance
Phase_offset	Output	Phase offset value
DP	Input	Detected phase. Obtained from r*ef*, *fb* phase difference

I/O Description of the Configurable Registers' Bloc. Continued

I/O Name	**I/O Direction**	**I/O Description**
The following ten signals are connected to the Phase Slop Limiter		
en_slop_corr	Output	Enable phase slope correction
slop_t1	Output	Time reference to determine the time interval for phase slope measurement
slop_t2	Output	Time reference to determine the time interval for phase slope measurement
align	Output	Alignment commend
slope_limit	Output	Phase limit value
PF	Input	Phase fraction. DP * FP gives the limited phase offset. It is used when the DP phase slope goes over the allowed limit
slop_limit_vio	Input	Active high. Indication that the slope limit is violated
ph_s_corrected	Input	Active high. Indication that the phase slope has been corrected
DPS	Input	Detected Phase Slope value
phase_s_out	Input	Slope limiter output phase.
The following thirten signals are connected to the Phase Build Out		
en_pbo	Output	Enable Phase Build Out functionality
l_pbo_trg	Output	Lowest level of the phase build out trigger threshold
h_pbo_trg	Output	Lowest level of the phase build out trigger threshold (h_pbo_trg – l_pbo_trg = L in Figure 26)
pbo_t_0	Output	Time reference to start measuring the phase build-out triggering threshold
pbo_t_1	Output	Time reference to finish measuring the phase build-out triggering threshold
D_1	Output	Allowed delay to not build-out
D_2	Output	Phase Transient Time-Out delay
D_3	Output	Minimum Delay to start building out the phase
pbo_l_crsd	Input	Active high. Indicator that phase Build Out Threshold crossed
t_time_out	Input	Active high. Indicator that the transient time-out is reached
pbo_start	Input	Active high. Phase Build Out started
no_pbo	Input	Active high. Indicate that a Phase Build-out process has not been aborted
phase_b_out	Output	Phase Build-Out output value

I/O Description of the Configurable Registers' Bloc. Continued

I/O Name	I/O Direction	I/O Description
The following six signals are connected to the Phase Lock detector		
acquiring_time	Output	Phase acquiring time. From Reference qualification to starting of settling time
settling_time	Output	Phase settling time. After the considered acquiring_time, the phase should be within its valid range for the duration of settling_time
lower_limit	Output	Lower phase limit to consider *ref* and *fb* are locked
upper_limit	Output	Upper phase limit to considere *ref* and *fb* locked. Allowed phase range = upper_limit – lower_limit
sel_phase_in	Output	Select one of the phase values 00: select *DP*. That is the default and normal setting to check the locking condition 01: select *phase_s_out* 11: select *phase_b_out*
lock_ind	Input	Active high. Lock indicator

Table 3.8: **LPD Rgister Map**

Signal Name	Register Type	I/O Description
reset_b	Read/Write	Synchronous reset
ref_valid	Read/Write	Reference Input qualified
ref_valid	Sticky Bit Register	Reference qualified status
The following registers are connected to the Basic Phase detector		
count_max	Read/Write	Maximum allowed counter output (DP) value. Used for Jitter tolerance
count_min	Read/Write	Minimum allowed counter output (DP) value. Used for Jitter tolerance
Phase_offset	Read/Write	Phase offset value
DP	Redundant Status Register	Detected phase. Obtained from r*ef*, *fb* phase difference

LPD Rgister Map. Continued

Signal Name	**Register Type**	**I/O Description**
The following registers are connected to the Phase Slop Limitter		
en_slop_corr	Read/Write	Enable phase slope correction
slop_t1	Read/Write	Time reference to determine the time interval for phase slope measurement
slop_t2	Read/Write	Time reference to determine the time interval for phase slope measurement
align	Read/Write	Alignment commend
slope_limit	Redundant Status Register	Phase limit value
PF	Redundant Status Register	Phase fraction. DP * FP gives the limited phase offset. It is used when the DP phase slope goes over the allowed limit
slop_limit_vio	Sticky Bit Register	Active high. Indication that the slope limit is violated
ph_s_corrected	Read Only Register	Active high. Indication that the phase slope has been corrected
DPS	Redundant Status Register	Detected Phase Slope value
phase_s_out	Redundant Status Register	Slope limiter output phase.

LPD Rgister Map. Continued

Signal Name	**Register Type**	**I/O Description**
		The following registers are connected to the Phase Build Out
en_pbo	Read/Write	Enable Phase Build Out functionality
l_pbo_trg	Read/Write	Lowest level of the phase build out trigger threshold
h_pbo_trg	Read/Write	Highest level of the phase build out trigger threshold (h_pbo_trg – l_pbo_trg = L in Figure 3.17)
pbo_t_0	Read/Write	Time reference to start measuring the phase build-out triggering threshold
pbo_t_1	Read/Write	Time reference to finish measuring the phase build-out triggering threshold
D_1	Read/Write	Allowed delay to not build-out
D_2	Read/Write	Phase Transient Time-Out delay
D_3	Read/Write	Minimum Delay to start building out the phase
pbo_l_crsd	Read Only Register	Active high. Indicator that phase Build Out Threshold crossed
t_time_out	Read Only Register	Active high. Indicator that the transient time-out is reached
no_pbo	Read Only Register	Active high. Indicate that a Phase Build-out process has not been started
phase_b_out	Redundant Status Register	Phase Build-Out output value

LPD Rgister Map. Continued

Signal Name	**Register Type**	**I/O Description**
The following registers are connected to the Phase Lock detector		
acquiring_time	Read/Write	Phase acquiring time. From Reference qualification to starting of settling time
settling_time	Read/Write	Phase settling time. After the considered acquiring_time, the phase should be within its valid range for the duration of settling_time
lower_limit	Read/Write	Lower phase limit to consider *ref* and *fb* are locked
upper_limit	Read/Write	Upper phase limit to considere *ref* and *fb* locked. Allowed phase range = upper_limit – lower_limit
sel_phase_in	Read/Write	Select one of the phase values 00: select *DP* 01: select *phase_s_out* 11: select *phase_b_out*
lock_ind	Sticky Bit Register	Active high. Lock indicator

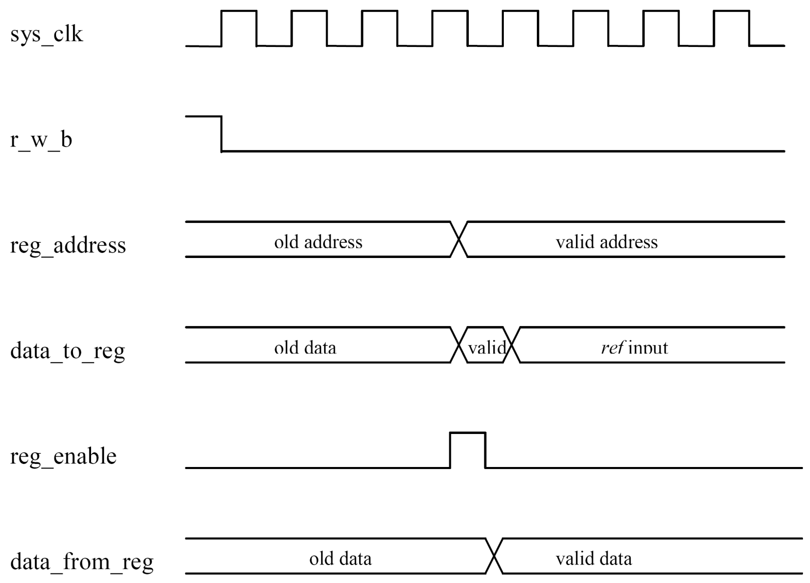


Figure 3.23: **Registers Write Timing Diagram**

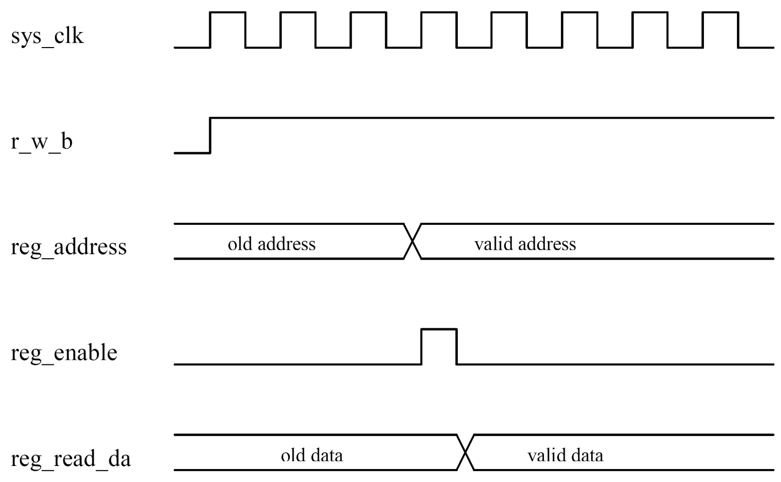


Figure 3.24: **Registers Read Timing Diagram**

Table 3.9: **EPP Slave I/O Description**

I/O Name	**I/O Direction**	**I/O Description**
clk	Input	System Clock
reset_b	Input	Synchronous reset
epp_rw_b	Input	EPP read write commend: 1: read operation 0: write operation
epp_astrobe	Input	EPP address strobe
epp_dstrobe	Input	EPP data strobe
epp_data	Input	EPP data bus. Transmits EPP data or EPP addresses
data_from_reg	Input	Data bus from registers
epp_wait	Output	EPP wait signal
epp_dir	Output	This signal indicates the direction of the transmitted data. May be used as a direction signal for an EPP data buffer on the board.
reg_address	Output	Address bus to registers
cs_signals	Output	Chip Select signals
data_to_reg	Output	Data bus to registers
reg_enable	Output	Enables access to registers
r_w_b	Output	Read/write commend to registers: 0: write 1: read

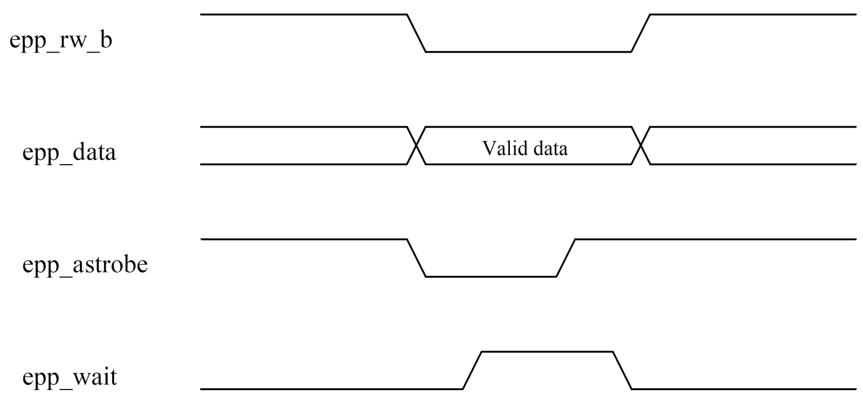


Figure 3.25: **EPP Write Cycle**

Table 3.10: **EPP Slave Internal Memory organisation**

Pointer	**Content**
1	The least significant byte of the address to write to the register block
2	The most significant byte of the address to write to the register block
3	The least significant byte of the data to write to the register block
4	The most significant byte of the data to write to the register block
5	The least significant byte of the data to read from the register block
6	The most significant byte of the data to read from the register block
7	Software reset

◇ The peripheral writes an address to epp_data, and brings epp_wait high to signal to the host that the address is available to be read

◇ The host reads epp_data and brings epp_astrobe high

◇ The peripheral disables the epp_data and brings epp_wait low

Data read cycle is identical to address read, except that the host uses epp_dstrobe instead of epp_astrobe.
Figure 3.27 gives the block diagram of the EPP interface. The EPP data input is synchronized and Write and Read State Machines manage to transmit data between an internal memory and the register bloc. The content of that internal memory is shown in Table 3.10.

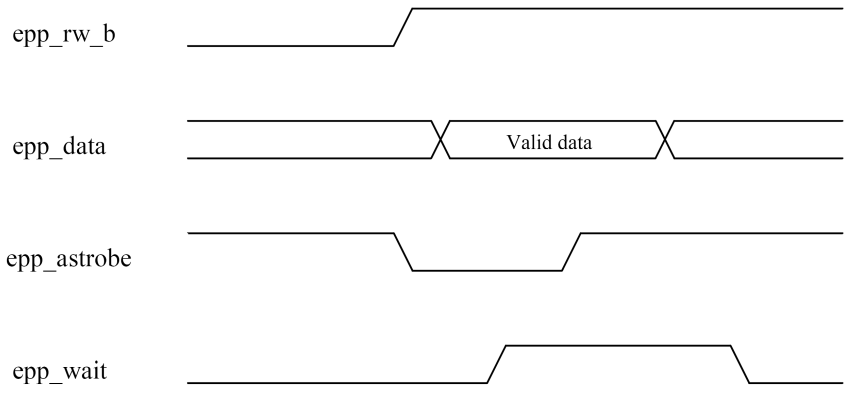


Figure 3.26: **EPP Read Cycle**

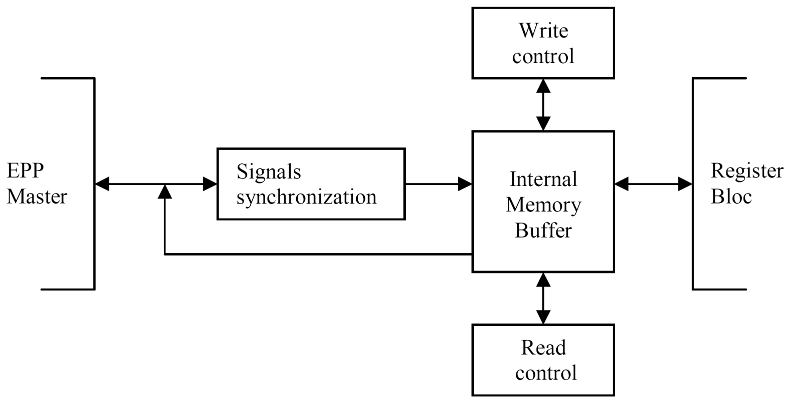


Figure 3.27: **EPP Interface Block Diagram**

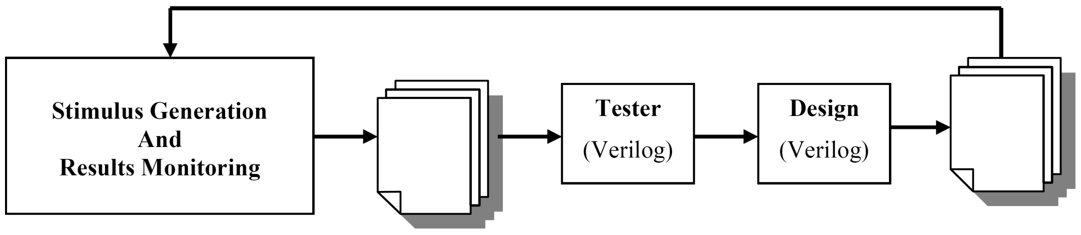


Figure 3.28: **LPD Verification General Approach**

3.8 LPD Verification

Our verification bench is illustrated in Figure 3.28 . Matlab files generate the needed stimulus and monitor the test results while a behavioral verilog tester is used as a wrapper for the design under test. Some of the test cases use only the verilog tester because they do not need to provide so elaborated signals.

Table 3.11 shows the test cases list. They can be described as follows:

◇ ref_fb_diff_detect detects the phase difference between *ref* and *fb*. It generates *ref* and *fb* with pre−determined phase differences and monitors the design's output to indicate failures in a text report

◇ ref_fb_offset sets offset values and monitors theirs effect on the *ref* and *fb* phase difference. Failures are reported in a text file

◇ ph_change_pull_in sets the criteria corresponding to phase pull−in as described in Section 3.1.2 and monitors the phase movement accordingly

◇ ph_lock_detect sets the criteria corresponding to locking conditions as described in Section 3.1.2 and in Section 3.1.4 and monitors the phase movement accordingly

◇ phase_transinet_detect sets the criteria describing a phase transient (Section 3.1.5) and monitors the phase movement accordingly. Because the normalized phase transient test is done after prototyping because of the required execution and monitoring time, this test shows only the possibility of setting the corresponding criteria and the possibility of catching individual transients

◇ jitter_tolerance_test sets the criteria fixing the allowed phase deviations (Section 3.3.1) and monitors the phase movement accordingly

◇ pbo_test sets the criteria described in Section 3.3.1, generates different scenarios of phase movements allowing to check whether the phase build out is performed appropriately

Table 3.11: **LPD Simulation Matrix**

Test case	**Test Case Description**	**Requirement**	**Abstraction**
ref _fb_diff_detect	It tracks the phase difference between *ref* and *fb*	Section 3.1	RTL
ref _fb_offset	Sets offset values to either align or offset *ref* and *fb*	Section 3.1	RTL
ph_change_pull-in	Sets criteria and tracks the phase movement when a synchronization system acquires lock. It indicates phase pull-in violations	Section 3.1	MATLAB and RTL
phase_lock_detect	Set locking criteria and detects phase lock. It also detects the non-locking states	Section 3.1	MATLAB and RTL
phase_transient_detect	Sets criteria and detects phase transients	Section 3.1	MATLAB and RTL
jitter_tolerance_test	Sets criteria and checks that phase movement does not go over the fixed limits	Section 3.1	RTL
pbo_test	Sets criteria and checks whether phase build out is performed appropriately	Section 3.1	MATLAB and RTL

Table 3.12: **LPD Implementation Results**

Phase Difference Precision (# of bits)	Worst Case Combinatorial Delay	Number of Fip-Flops (Available 19,200)	Number of used LUTs
32	9.658ns (27.3% logic, 72.7% route) 16 levels of logic	13,653	6,240
24	6.284 (29.6% logic, 70.4% route) 13 levels of logic	7,872	4,653

3.9 LPD Implementation

The Linear Phase Detector design effectiveness is demonstrated with Xlinx Virtex−5 LX devices dedicated for high performance general logic applications [17]. The bus width, of *ref*/*fb* phase difference (Figure 3.7), affect directly the FPGA implementation performances (Table 3.12).

Chapter 4

VLSI QUADRATURE PHASE DETECTOR : A BCPT DESIGN

To overcome the instability of conventional blind carrier recovery algorithms when applied to large constellation modulation, the ”blind carrier tracking with guaranteed global convergence (BCPT)” algorithm has been proposed, reused and several times cited in the literature. The need of its portable VLSI implementation requires its simplification to be based on a simple process with usual arithmetic operators. Actually, this Thesis presents that dedicated and simplified algorithm and its most important design issues and solutions. It highlights mainly some modules, such as some arithmetic and storage components showing, for instance, an Arctangent implementation that can be run only in three clock cycles while always retrieving the needed angle's resolution. Finally, simulation results are provided to show the effectiveness of the proposed design in a Quadrature Amplitude Demodulation environment.

4.1 BCPT Problem Formulation

Touching to the field of carrier tracking and synchronization drives inevitably to diverse phase tracking algorithm implementations. Those implementation options are widespread over the literature. For instance, reference [21] presents the conventional phase detection and tracking used in industrial DPLL products, [22] presents a VLSI phase tracking, using Hilbert transformation, placed behind an equalizer to reduce the carrier phase noise that attributes to the main SNR loss of a Virtual Base Station (VBS) receiver, [23] presents a microprocessor-based phase tracking system used in digital power metering.
The considered application, in this Thesis, focuses on the issue of blind carrier phase tracking, which is critical in blind adaptive receivers. It is known that in large constellation modulation schemes, such as 256 QAM, conventional carrier tracking schemes frequently do not converge and result in spinning [24] [25]. The blind Carrier Phase Tracking with Guaranteed Global Convergence algorithm, in reference [3], resolves that problem and shows that the carrier tracking is equivalent to a blind source separation problem involving the

separation of a linear unitary mixture of two independent real-valued components. These two components are the real and the imaginary parts of the emitted signal.
The theory in [3] has been cited several times in the literature [9] [26]. However, its straightforward VLSI implementation is not possible because of its computational complexity evolving mainly different matrix operators, large dividers and multipliers as well as a complex manipulation of trigonometric functions. That is not particular to the algorithm in [3]; one may go through any of the blind carrier phase estimators summarized in [27] to see that implementing physically their corresponding algorithms, as they are and with input data of more than 8 bits, would require a complex arithmetic manipulation of large buses with more than hundred bits. Actually, up to date, except our own paper [10], no blind carrier phase tracking with global convergence algorithm dedicated for VLSI implementation has been published. It is, then, necessary to overcome the difficulty of rewriting and simplifying such algorithms in order to express them with usual hardware operators while reducing the processed data size explosion. In that way, this work shows how to overcome that difficulty for the theoretical estimation presented in [5]. The proposal is general and does not target a specific technology. Hence, a special effort was done to minimize as much as possible the need of technology related components to ease the design portability and its FPGA validation. For instance, it is constrained to not have any large divider.
In this Chapter we give an overview on how a blind separation procedure is used to solve the carrier tracking problem. Then, we show the corresponding simplified expression allowing the hardware implementation of the algorithm. We present its overall hardware design specification emphasizing some critical details on modules like the arithmetic unit and the storage components. Finally, results from MATLAB and RTL simulations are presented to show the effectiveness of the proposed design specification.

4.2 The Blind Carrier Phase Tracking with Guaranteed Global Convergence

Consider a transmission system of QAM data. Assuming a flat and noise free transmission channel (or perfect equalization), the received signal at sample time k can be expressed as

$$x_k = \exp(j\phi_k)a_k \tag{4.1}$$

Where a_k is the transmitted data sequence which is complex, non-Gaussian and independent, identical distribution (i.i.d), and ϕ_k is a time varying phase shift due to frequency offset and phase jitter, referred to as the carrier phase.
To recover the transmitted data sequence , one needs to track the carrier phase ϕ_k in order to remove it. The aim of the blind carrier tracking is to recover the phase ϕ_k without the knowledge of the data sequence a_k . Equation 4.1 can be rewritten in terms of real and imaginary parts as

$$Re(x_k) = \cos(\phi_k)Re(a_k) - \sin(\phi_k)Im(a_k) \tag{4.2}$$

$$Im(x_k) = \sin(\phi_k)Re(a_k) + \cos(\phi_k)Im(a_k) \tag{4.3}$$

In matrix form, we have

$$\tilde{x} = Us_k \tag{4.4}$$

with

$$\tilde{x}_k = \begin{pmatrix} Re(x_k) \\ Im(x_k) \end{pmatrix}$$

$$s_k = \begin{pmatrix} Re(a_k) \\ Im(a_k) \end{pmatrix}$$

$$U = \begin{pmatrix} \cos(\phi_k) & -\sin(\phi_k) \\ \sin(\phi_k) & \cos(\phi_k) \end{pmatrix} \tag{4.5}$$

Under the assumption of the statistical independence of the real and imaginary parts of the data , equation 4.4 tells that the problem of carrier tracking is equivalent to the problem of the separation of a linear unitary mixture of two real-valued independent components [3].
Reference [3] develops a carrier tracking algorithm by utilizing the adaptive blind source separation algorithm proposed in [28]. Then, it ends up with the following updating rule for the estimation of the carrier phase shift:

$$\phi_{k+1} = \phi_k + \arctan(\lambda\gamma_k) \tag{4.6}$$

When ϕ_k is close to the convergence, equation 4.6 can be rewritten as:

$$\phi_{k+1} = \phi_k + \lambda\gamma_k \tag{4.7}$$

With some algebraic manipulation γ can be expressed as:

$$\gamma = xr_k xi_k(xi_k^2 - xr_k^2)\cos(4\phi_k) + (0.25(xi_k^2 - xr_k^2) - (xr_k xi_k)^2)\sin(4\phi_k) \tag{4.8}$$

xr_k and xi_k are the real and imaginary parts of the received signal of equation 4.1 at time instant k. λ is a step size that can be considered as a tuning constant.

4.3 Algorithm Expression for Hardware Implementation

We have obtained Equation 4.8 with the effort of minimizing, as much as possible, the number of large operators. No divider is used. It is translated below in a process to be executed with simple hardware operators constituted mainly by eight multipliers.
Let us call that process the Hardware-Dedicated-Procedure and consider the index k (the time instant) as implicit:
Hardware−Dedicated−Procedure

$in\_sq = (xi - xr) * (xi + xr);$
$in\_mult = xr * xi;$
$4\phi = 4 * \phi;$
$cos\_fac = cos(4\phi) * in\_sq;$
$cos\_term = cos\_fac * in\_mult;$
$sin\_term1 = in\_sq * in\_sq \ / \ 4;$
$sin\_term2 = in\_mult * in\_mult;$
$sin\_term = sin(4\phi) * (sin\_term1 - sin\_term2);$
$hyp\_fac = cos\_term + sin\_term;$
$\tan(\phi) = \lambda * hyp\_fac;$

$$\phi = \phi + a\tan(\tan(\phi)) \tag{4.9}$$

As shown in equation 4.7, when ϕ is close to the convergence, it can be rewritten as:

$$\phi = \phi + \tan(\phi) \tag{4.10}$$

At first glance, implementing the algorithm with equation 4.10 seems to be less area and time consuming. However, because equation 4.9 is, anyway, required for convergence acquisition, to use equation 4.10 one needs to determine the related application's convergence accuracy and determine accordingly when to switch from equation 4.9 to equation 4.10. So, the convenience of utilizing equation 4.10 is directly related to the used application where the convergence acquisition time and the convergence accuracy play the principal roles.

To keep our proposed architecture general, in this Thesis, only equation 4.9 is considered.

4.4 BCPT Design Specification

Figure 4.1 shows the phase tracking data path. The arithmetic unit receives at the same time λ, xr, xi, $\sin(\phi)$ and $\cos(\phi)$.
$\sin(\phi)$ and $\cos(\phi)$ come from a 'Sin and Cos' Look Up Table (LUT). The output of the Main Arithmetic Unit $\tan(\phi)$ is input to the Atan Unit to sort the updated ϕ. The obtained ϕ is a coded value with the same angle's resolution and precision as the one initially loaded from a controller to the 'Sin and Cos' LUT. It is summed with the previous ϕ value and fed back to be used with the next data samples xr and xi.

The three following sections give the main blocks and techniques that process data.

4.5 Arithmetic Unit

The Main Arithmetic Unit (Figure 4.1) contains the most critical part of the data path. It receives the input data from a synchronizer, λ from a CPU interface, $\sin(4\phi)$ and $\cos(4\phi)$

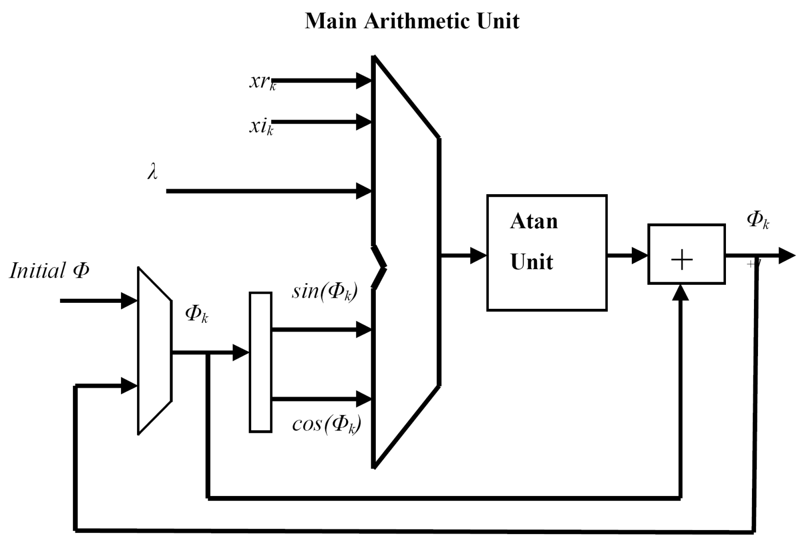


Figure 4.1: Quadrature Phase tracking block diagram

Table 4.1: **Time Multiplexing Multipliers**

	MULT1	**MULT2**	**MULT3**
Step1	In_sq	In_mult	
Step2	cos_fac	sin_term1	sin_term2
Step3	cos_term	sin_term	
Step4	tan(ϕ)		

from 'Cos and Sin' LUT and outputs tan(ϕ).
To perform the computation described in Hardware-Dedicated-Procedure a straightforward option would be to use eight large multipliers. Such implementation leads to large area consumption unless more elaborated design techniques are adopted. A time multiplexing technique is one of those alternatives. It allows reusing inactive operators, and then only three multipliers may be implemented. The multipliers' inputs are multiplexed sequentially to perform all the operations. Table 4.1 shows the followed steps to achieve all the Hardware-Dedicated-Procedure multiplications when applying all the time multiplexing possibilities. In Table 4.1, MULT1, MULT2 and MULT3 denote the three multipliers. Each multiplication operation is identified by its output signal name. The signal names are identical to the ones introduced in the Hardware-Dedicated-Procedure.

The considered time multiplexing does not show in detail the required data reformatting, the additions, the subtractions, and the multipliers' input multiplexing. Those operation delays are included adequately in the time multiplexing steps. Note that only MULT1 and MULT2 are intensively used. Under relaxed timing constraints, MULT3 can be removed and its corresponding operation added as a step to MULT1 or MULT2 sequences. Conversely, under tight timing constraints, the number of multipliers can be increased.

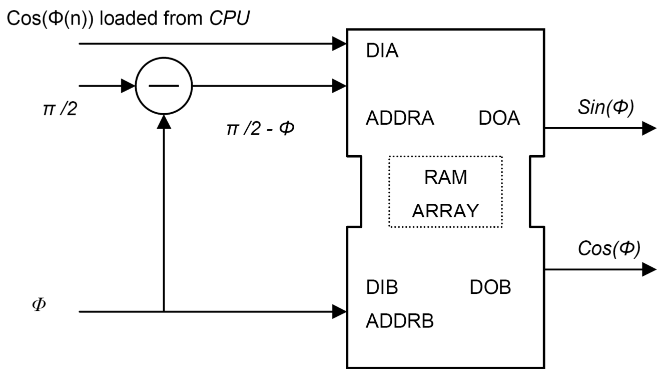


Figure 4.2: **Cos(ϕ) and Sin(ϕ) Look Up Table**

Note also that Step 4 can be removed when λ is coded as a power of 2, in which case tan(ϕ) is obtained by shifting accordingly hyp_fac (see Hardware-Dedicated-Procedure).

4.6 Sin And Cos Functions Implementation

The 'Sin and Cos' LUT needs to be built with a RAM loaded with cos(4ϕ) and sin(4ϕ) values. A dual−port memory can be used. For the same resolution of 4ϕ cos(4ϕ) and sin(4ϕ) values are the same considering $\sin(4\phi) = \cos(\pi\ /\ 2 - 4\phi)$. The memory is loaded by a CPU through one port with only cos(4ϕ) values. 4ϕ values coming from that CPU interface (initial 4ϕ) or from the feedback loop (Figure 4.2) are used as addresses to select the adequate cos(4ϕ) and sin(4ϕ) values.

4.7 Atan Function Implementation

The most critical storage element to implement is the one related to Atan function. To shorten as much as possible the pipeline of the data path, the Atan function needs to be performed in the least possible clock cycles while retrieving the ϕ initial resolution and precision. Hence, Atan approximations requiring necessarily several clock cycles or requiring only small angles as the one presented in [29] is not recommended for our implementation. The Arctangent function is widely used in Signal Processing Applications. Nevertheless, its difficult hardware implementation is always emphasized in the literature which tries to resolve the compromise between speed, accuracy and hardware resource consumption. Almost every paper presenting an Arctangent computing solution starts with a summary of the three usual Arctangent computing approaches: high order polynomial approximations, rational approximations and Look Up Table (LUT) based methods. In References [30], [31], [32], [33], the authors show that the usual approximation algorithms, like Taylor, Chebyshev and the Coordinate Rotation Digital Computer (CORDIC), are valid Arctangent implementations when high computational costs are unimportant. And the LUT based methods are simpler to implement but increasing the precision of the considered

input data may degrade unacceptably the corresponding memory performances. In References [34], [35] and [36], the authors acknowledge those very same statements and do more than proposing an alternate theoretical option, they present detailed hardware implementations as tradeoffs between speed, power and area consumption.
References [30], [31], [32], [33], [34] and [35] are the most recent published solutions proposing alternate options. More papers proposing arctangent solutions can be found in the literature, but those six references synthesis the most compiling solutions for Arctangent large scale integration.
R. Lyons proposes a method to estimate an angle ϕ of a complex value $(I + jQ)$ using a table that lists arctangent approximations based on the octant location of ϕ. The proposed algorithm can be used in applications where I^2 and Q^2 are pre-computed.
In the same way, S.Rajan et al. avoid using LUTs and usual approximation methods. They propose new approximations by using Lagrange interpolation and minimax optimization techniques. Every proposed approximation constitutes a trade off between accuracy and computational cost [31], [32].
M. Saber et al. present a piecewise linear approximation of the arctangent function to avoid high power consumption and long latency as an alternative to the other methods based on CORDIC algorithm, polynomial approximation or conventional LUT methods [34].
F. de Dinechin et al. use a unified view of most table−lookup−and−addition methods. They show that this option improves the speed/area tradeoffs mainly for precisions up to 16 bits [35].
R. Gutierrez et al present an architecture for the computation of the $a\tan(Y \ / \ X)$ operation using a logarithmic transformation to ease the division implementation and a combination of non-uniform segmentation and a multipartite LUT technique to achieve higher throughput than the approach based on CORDIC algorithm and lower area than conventional LUT-based approach [36].
References [35] and [36] show, with their own solutions and with their related references, that the trend to use LUTs in an unconventional way minimizes the impact of the tangent values (input data) on the needed memory size. They show working solutions for specific applications characterized mainly by the data throughput and the needed arctangent precision. In both references [35] and [36], the unconventional memory based solutions have accuracy price to pay; the related memory do not contain the exact needed output values as it is with usual LUT based solution. They contain approximations.
Actually, regarding arctangent hardware implementation, any proposed solution should be evaluated according to specific requirements; particular options should be followed for particular requirements. In this Chapter, we are presenting an Arctangent core that should comply with the following requirements:
Requirement-a: Input tangent values are coded as fixed point two's complement numbers. Our Arctangent is a single argument function.
Requirement-b: The performances of the Arctangent core should not be driven by the tangent input precision. Large word input data, like 32 bit words, should be accepted and then the use of the usual LUTs based methods should be avoided.

Requirement-c: the core should present the least possible pipeline stages to be used in the implementation of feedback algorithms. For positive tangent values, the Arctangent would be preferably run in one clock cycle. Hence, the use of large operators and the use of high order approximations should be avoided.

We show below that none of the Arctangent solutions described in the literature responds to all the above listed requirements. While there are applications which need Arctangent functions complying exactly with the listed three requirements. One of those applications is a blind carrier phase tracking described above. It is proposed by A. Belouchrani et al. in reference [3]. These applications need a new Arctangent hardware architecture. To that end, we are proposing a Content Addressable Memory (CAM) based solution to reduce as much as possible pipeline stages and to better control the output accuracy and then overcome the related drawbacks in the existing approximation implementations. And we are proposing an address decoding independent from the input tangent precision to overcome the related drawback in the usual LUTs based solutions. We are showing all the details resolving the challenges of such solution. We are making it easily reproducible and we are describing it at the logical gate level to allow its rapid FPGA validation. Its design effectiveness is demonstrated with Xilinx Virtex−5 LX devices dedicated for high performance general logic applications [17].

4.7.1 Arctangent Existing Solutions vs. our requirements

References [27], [28], and [29] do not show hardware implementation results. However, the related analytical formulations show the computational complexity and the needed operators to utilize for every method. They consider the input data to the arctangent function as a complex number − a two argument input I and Q to compute arctangent of $I \ / \ Q$. For those methods, input tangent values coded as fixed point real numbers do not fit. And compared to LUTs based solution, the proposed solutions suffer from inaccuracy. They are given as standalone solutions ignoring the unavoidable correlation of their intrinsic errors with the inaccuracy of the input tangent values. The problem of transferring the inaccuracy of the input through arithmetic operators concerns the solution presented in reference [34] as well. More, that reference that is based on a piecewise linear approximation does not show the intrinsic inaccuracy related to every linear piece. It does not show the possible non-monotonicities at the borders between intervals. A problem raised by F. de Dinechin et al in reference [35].

References [30], [31], [32] and [34] improve the usual polynomial and rational approximations. However, they do not compete against memory based solutions. The references [34] and [35] present clear improvement compared to previously published solutions. They attempted to extend the possibility of using the usual LUT based solutions. However, they still present accuracy and speed limitations because of data processing before obtaining the final and searched Arctangent values. The application example presented in Section 3 shows that data precision requirement may go to much more than 16 bits recommended in reference [35] and may go even more than the maximum of 22 bits precision presented in reference [36]. Furthermore, it shows that we need the exact Arctangent output values (not any approximated ones) to be fed back and re-used with the same precision and resolution as the initial input values.

In summary, our experience and F. Dinechin et al. in Reference [35] indicate that the LUT based solution becomes unusable when input data execeeds 10-12 bits. Below the 10 bit input data, that method is clearly a competitive candidate (in most cases that we know, it is the best candidate) because it generates Arctangent values with the desired format (precision and resolution); exact needed values are stored and decoded as exactly expected. Whereas, all other published solutions generate approximated Arctangent values for which some application does not work. Actually, for any algorithm feeding back approximated Arctangent values, as published in the existing literature; the evaluation of the overall computation error is specific and complicated.
The question is: can we keep using memories to store the expected and needed output values without using the input tangent value neither as a conventional address to let it be as large as needed, nor as a major factor evolved in an Arctangent approximation to minimize its inaccuracy propagation and impact.

The response is yes. The following sections show how to make that happen.

4.7.2 Proposed Arctangent Search Engine

The solution is to build the Arctangent function as a search engine where computed tangent values are compared to pre-calculated and stored tangent values. Because the stored tangent values correspond to the desired angle's resolution, and because tangent function is an increasing one in the range of $[-\pi\ /\ 2, \pi\ /\ 2]$, the comparisons can systematically indicate the searched angle's location. That moves the number of words to decode from depending on the tangent data precision to the number of angles relative to the angle's resolution and to the angle's range. For example, for a tangent value written with 18 bits, an angle resolution of $\pi\ /\ 180$ ($1°$) and a range going from $-\pi/2$ to $\pi/2$ using the proposed search engine reduces the number of cases to decode from $218 = 262144$ to 180.
Considering that $\tan(\Phi)$ is the input tangent value, tan_buffer is the buffer where pre−computed tangent values are stored and Φ the angle's value to sort, here is a simple pseudo code to formulate the search engine:

Arctangent Search Engine−Pseudo Code:

```
for each cycle do
  if new $\tan(\phi)$ is computed then
    for $0 < i \leq$ angle's number do
      if $\tan(\phi) \geq tan\_buffer[i]$ then
        $\phi = i$;
      end if
    end for
  end if
end for
```

For a simple illustration, let us consider that the searched values Φ is coded with 1° resolution and belongs to the range of [0 : π / 2[. The corresponding tan_buffer would be filled with 90 pre−computed $\tan(\phi)$ values. $\tan(0) = 0$ would be the smallest value and $\tan(89°) = 57.289$ the largest one. When the search engine is executed because a new $\tan(\phi) = 1.740$ is computed, it finds that $1.740 > 1.732$ that is the tan_buffer(60). Then, it stops searching to give an indication showing that the searched angle is 60° .
Obviously, if the angle's resolution was not 1° the index i (in the pseudo−code) would indicate the searched buffer's location that can be considered as a coded value of ϕ.

4.7.3 Proposed Arctangent Architecture

Figure 4.3 shows the conceptual block diagram of the search engine. During the initialization period, the tangent buffer is loaded, from the lowest to the highest tangent value. Then, every computed $\tan(\phi)$, obtained from an arithmetic unit, is compared to the buffer values to sort ϕ.

For better readability the used signals and parameters, in this Chapter, are labelled as follows:

◇ $pre - tan(\phi)$ as the pre−computed tangent values loaded in the tangent buffer

◇ $\tan(\phi)$ as the computed and input tangent value obtained from an arithmetic unit

◇ N as the angle's number

◇ Ind[i] as the i'th comparator's indicator. Considering that $0 \leq i < N$.

◇ ϕ_{min} is the smallest angle value corresponding to $i = 0$;

◇ ϕ_{max} is the largest angle's value corresponding to $i = N - 1$.

◇ ϕ[i] is the angle value corresponding to the i'th tan_buffer word and to the Ind[i]

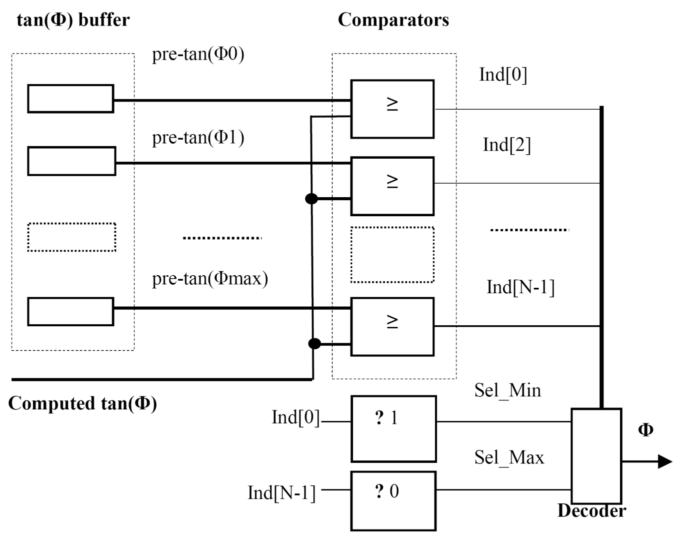


Figure 4.3: **Arctangent Conceptual Block Diagram**

When $\tan(\phi)$ is input, it is compared simultaneously to every buffer word $(pre - tan(\phi_i))$. By default, every comparator's indicator (Ind[i]) is set high. And for every comparator, if the computed $\tan(\phi)$ is larger than or equal to the considered $pre - \tan(\phi_i)$, the comparator's output (Ind[i]) is set low.
After every comparison, the indicators are decoded to output the searched value ϕ.
Because, in the interval [-$\pi/2$; $\pi/2$], the tangent function is increasing, the decoder has just to detect ϕ_{min} or ϕ_{max} or the two adjacent indicators presenting two different states state 1 and state 0 and showing the location of the searched ϕ.
So, to sort the searched value ϕ, we have to deal with three exclusive cases:

- ϕ_{min} is output when Ind[0] is kept high
- ϕ_{max} is output when Ind[N-1] is set low
- ϕ[i] is output when Ind[i] is different from Ind[i+1]

The selection cases to sort ϕ are illustrated in Figure 4.4

Because of the rotational symmetry of Arctangent, $\arctan(-x) = -\arctan(x)$ where x is a tangent value, and because sign information extraction and numbers' complementation are trivial, let us give the implementation details for a solution considering only positive tangent values.
The most challenging part of the proposed design is to execute timely the CAM and the decoding function. As shown in Figure 4.3, we consider that the CAM structure includes the tan_buffer and the comparators while the Decoder goes from the comparators indicators to the final output giving the searched ϕ.

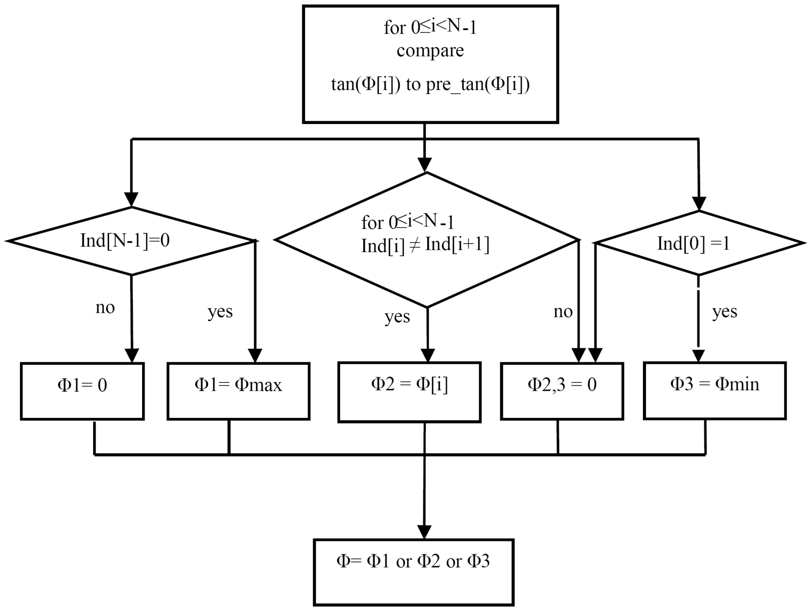


Figure 4.4: **Arctangent Functional Diagram**

The implementations of the CAM and the Decoder are given in the following sections respectively.

4.7.4 CAM Implementation

The CAM structure can be described − with a Hardware Description Language − in two processes representing the two corresponding blocs in Figure 4.3; the first one loads the pre−computed tangent values in the tan_buffer and the second one performs the comparisons and generates the indicators (Ind). The most critical part of the CAM is the comparator implementation that depends on the tan(ϕ) world length. In any kind of technology, to favour the speed, appropriate resources and algorithms need to be chosen to minimize that comparator's propagation delay from the Most Significant Bit to the Least Significant one. Regarding the Xilinx FPGA implementation, the option is to use the carry chain to cascade 'AND' gates. More specifically, the dedicated and fast component called MUXCY [37] are configured as a wide 'AND' gate, allowing the word length to be expendable and the most significant bytes to have the priority to trigger the comparison indicator. The global structure of our CAM and the use of the carry chain to speed up the comparisons make it resemble to the CAM described in the application note in reference [38]. We could not use the exact CAM described in reference [38] because the stored words in that application correspond exactly to the expected inputs. And to fill the used LUTs the authors, in reference [38], utilize a serial data setting specific to the Xilinx shift registers built in LUTs. Our inputs can have any tangent value; not only the ones we store in our tan_buffer. And for better portability, we use a usual buffer of registers that can be filled in a burst mode by incrementing the buffer's pointer and setting accordingly the pre−determined tangent values. The used comparator constitutes the slowest combinatorial part for the CAM. To give an idea about its physical implementation on the Virtex−5 LX 65 μm technology with 12−layers, a 32 bit comparator consumes 5 levels of logic, 16 LUTs and 3.99ns propagation delay where 33.7% is logic and 66.3% is routing.

4.7.5 Decoder Implementation

The Decoder receives the comparator indicators and outputs the searched value ϕ. Because tangent is an increasing function in $[0, \pi/2]$ range and the indicator's number is equal to the angle's number, one of the three exclusive cases (Figure 4.4) always gives the angle's location.
To reduce the amount of required combinatorial logic when reading the indicators two options are adopted:

◇ Make sure that no priority is set for exclusive cases. The exceptions indicated in Figure 4.3 and in Figure 4.4 (Ind[N-1] = 0 and Ind[0] = 1) are exclusive to each other and exclusive to the regular decoding ($Ind[i] \neq Ind[i+1]$). So, the outputs of the three cases are just ORed (Figure 4.4). After every decoding operation, those outputs are set back to the default '0' so that in the following operation, only the selected case goes through the OR gate

◇ All adjacent indicators, Ind[i] and Ind[i+1], are XORed to detect which pair of indicators are different (Figure 4.5). The XORs' outputs called "diff_detect" are one hot coded. The only diff_detect bit that is set high indicates the searched ϕ location

The searched ϕ location is an integer between 0 and N-1. Because it is unique for every ϕ, it can represent a coded value of itself. In applications like the one presented in section 4.4 in Figure 4.1 the sorted integer number corresponding to ϕ can be fed back to a cosines and sinus dual memory as an address. At that stage the integer number, corresponding to ϕ location, is all the needed information from the Arctangent function. That integer number is a proportional factor of ϕ. If a more accurate ϕ value is needed farther in the system, for the application itself or in a related test bench, the sorted integer number is multiplied by the appropriate sign and angle's step.
The large Multiplexer in Figure 4.5 is the slowest part of such Decoder module. It depends on the angle's range and resolution. Its straightforward implementation gives large data path delays because of the required combination and routing of the selection signals diff_detect and Sel−Max. On Virtex−5 LX device, 180 selections (N=180) of an 8−bit data bus give a Multiplexer of 228 LUTs, 34 levels of logic and 23.923ns propagation delay where 12.4% is logic and 87.6% is routing. To find faster ϕ value, either as an integer or as a more accurate real value, the option would be to pre-store in the needed format ϕ values and use the one hot diff_detect and Sel−Max signals to load out the searched ϕ. Figure 4.6

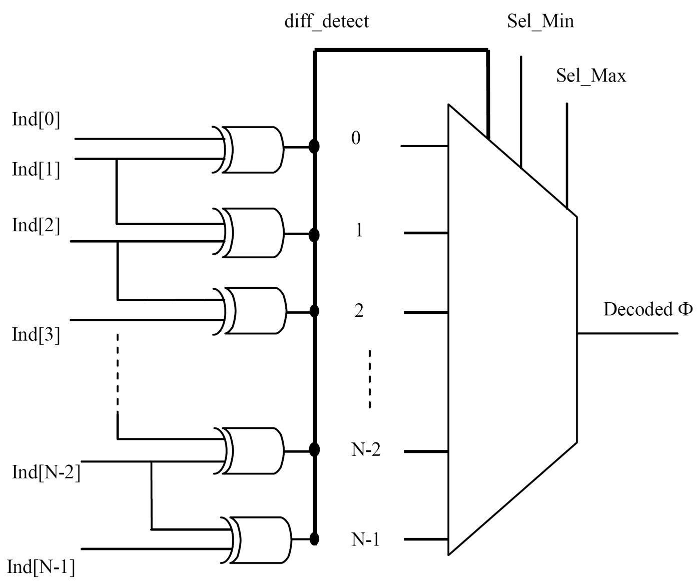


Figure 4.5: **Decoder Logical Circuit**

shows how to use D Flip Flops to eliminate any combination between diff_detect bits and Sel−Max.
As for the pre-tan(ϕ) buffer, the ϕ buffer is loaded at the system set up. Every predetermined ϕ value is input to Flip Flops that transmit it out when the corresponding enable signal ('en' in Figure 4.6) is set high. A wide OR gate outputs the selected ϕ and after every decoding operation, the outputs of the changed Flip Flops are set back to the default '0' unless their corresponding enable signal is kept high. The Flip Flop outputs are set back to '0' so that in the following operation only the selected case goes through the OR gate.
Note that for this implementation Sel−Min signal is not needed. If the searched ϕ is smaller than the stored $\phi[0]$ neither Sel−Max nor any of diff_detect bits is set high. In such case, ϕ keeps systematically the value '0'. For this Decoder, the wide OR gate constitutes the slowest combinatorial part. On Virtex−5 LX device, for every output bit, a stand alone 180−input OR gate consumes 32 LUTs and 5.663 ns propagation delay where 19.3% is logic and 80.7% is routing.

4.7.6 The Arctangent Core physical Implementation

This section gives the implementation results for resolutions of 1°, 0.5° and 0.25° and for values written with two different precisions. It presents a discussion that shows the compliance of the proposed core to the three requirements listed in the Introduction. It shows how to extend the proposed core functionality to the full range of [-$\pi/2$, $\pi/2$] and finishes with a discussion on the timing, area and power performances of the proposed core.

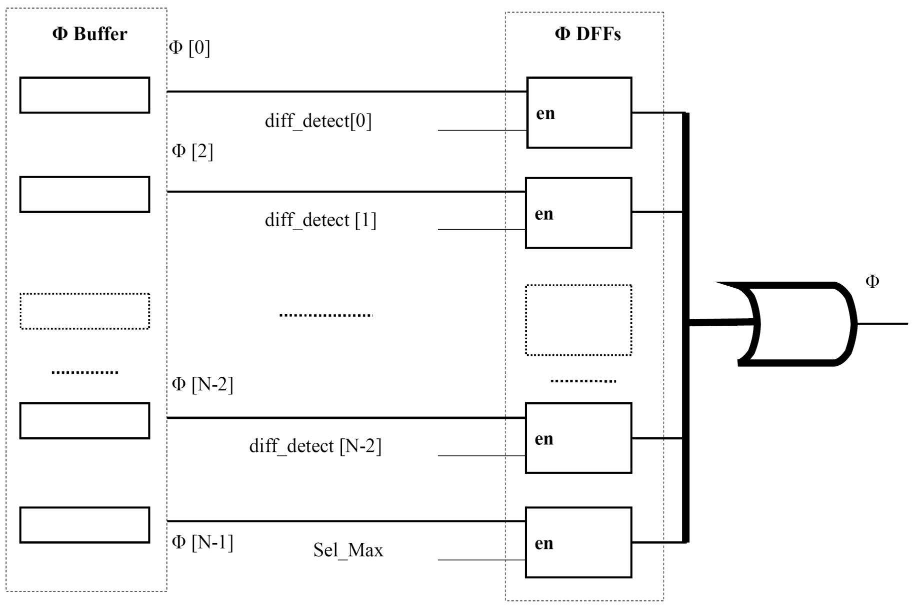


Figure 4.6: **Arctangent Output Selection**

Table 4.2 gives an example on the way ϕ and $\tan(\phi)$ are coded. It gives the hexadecimal coded values for the extreme numbers considering an angle resolution of 0.5° and a range of [0, 90°].

Table 4.3 shows the Arctangent implementation results on Virtex−5 LX devices. It corresponds to the architecture combining the illustrations in Figure 4.3, Figure 4.5 and Figure 4.6. Results related to 1° and 0.5° are obtained on an implementation on xc5vlx30 device. While the results related to 0.25° are obtained with an implementation on xc5vlx50 device. Description of the devices xc5vlx30 and xc5vlx50 can be found in reference [17].
For all the presented configurations, the worst case combinatorial delay corresponds to the slowest path of the wide OR gate outputting ϕ.
For xc5vlx30 the obtained clock path delay with 0.1 uncertainty consumes 2 levels of logic and a maximum of 2.64ns delay. For this device and for the examples shown in Table 4.3, the largest clock cycle to generate ϕ is 6.381 + 2.64 = 9.021 ns (110.8 MHz) and the smallest one is 3.823 + 2.64 = 6.463ns (154.7 MHz).
For xc5vlx50 the obtained clock path delay with 0.1 uncertainty consumes 2 levels of logic and a maximum of 2.9ns delay. For this case the largest clock cycle to generate ϕ is 7.743 + 2.9 = 10.643 ns (93.9 MHz).

4.7.7 Arctangent Implementation Extension to [-$\pi/2$, $\pi/2$] range

The proposed Arctangent core is generic and can be easily adapted to any sub-range of $[-\pi/2, \pi/2]$. To cover the full range of $[-\pi/2, \pi/2]$, the easiest option would be to apply

Table 4.2: **Example of Coding** ϕ **and** $\tan(\phi)$

Φ information	**tan(Φ) precision**	**tan(Φ[0])**	**Coded tan(Φ[0])**	**tan(Φ[N-2])**	**Coded tan(Φ[N-2])**
Φ precision is 12 bits Φ[0] = 0.5° Coded Φ[0] = 0x5 Φ[N-2] = 89.5° Coded Φ[N-2] = 0x37F Φmax = 90° and Coded Φmax = 0x384	32 bits	0.0087268	0x154E4	114.5886501	0x444CD725
	18 bits	0.008	0x8	114.588	0x1BF9C

Table 4.3: **Arctangent Implementation results**

Φ Resolution In [0,90°]	**Φ Precision (# of bits)**	**tan(Φ) Precision (# of bits)**	**Worst case combinatorial Delay**	**# of Used FF Available 19,200**	**# of used LUTs**
1° **90 angles**	7	17 bits	3.823ns (30.2% logic, 69.8.4% route) 3 levels of logic	2,767	1,742
	7	32 bits	4.947 (23.5% logic, 76.5% route) 3 levels of logic	4,141	2,372
0.5° **180 angles**	10	18	5.454ns (24.8% logic, 75.2% route) 6 levels of logic	6,837	4,128
	10	32	6.381 (22.6% logic, 77.4% route) 7 levels of logic	9,357	4,582
0.25° **360 angles**	15	19	6.177ns (27.8% logic, 72.2% route) 11 levels of logic	17,625	5,623
	15	32	7.743ns (25.6% logic, 74.4% route) 14 levels of logic	22,309	12,822

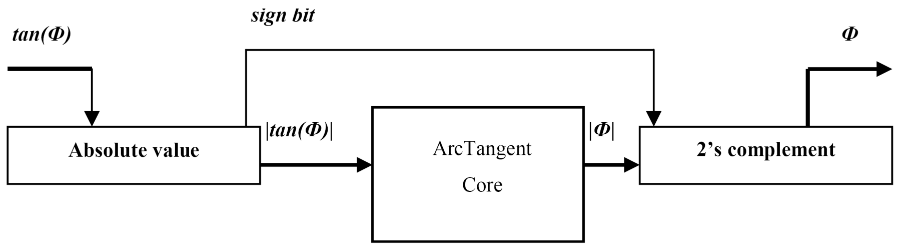


Figure 4.7: **Arctangent Core extension to cover the full range [-$\pi/2$, $\pi/2$]**

the rotational symmetry of the Arctangent function: Generate the absolute value of tan(ϕ) for processing. And after processing and obtaining the absolute value of ϕ, put back the right polarity as shown in Figure 4.7. The same configurable module is used backward and forward to generate a two's complement number and change the number's polarity as needed.
A clocked 32 bit two's complement generator consumes 8 levels of logic, 63 LUT Flip−Flop pairs with a 3.049ns maximum combinatorial path delay.

4.7.8 Conformance to the listed Requirements

Conformance to Requirement-a: tan(ϕ) is a fixed point two's complement number.
Conformance to Requirement-b: The most beneficial consequence of utilizing the proposed core is the low cost inferred by increasing the tan(ϕ) precision. Table 4.3 shows that going from the minimum possible precision to 32-bit precision is possible and does not decrease the hardware performances neither exponentially nor proportionally. Our solution has the advantage of accepting large tan(ϕ) precisions and generating the angle's value with the desired precision for cases absolutely impossible to implement with the usual LUT based solution that is the only theoretical candidate that would be able to comply with this requirement.
Conformance to Requirement-c: For applications where only the range $[0, \pi/2]$ is needed the proposed core can generate ϕ in one clock cycle. Only the Flip Flops connected to the wide OR gate are necessary (Figure 4.6). For an execution over $[-\pi/2, \pi/2]$ range as illustrated in Figure 4.7, ϕ can be generated through three clock cycles: one to generate the tan(ϕ) absolute value, one to input data to the wide OR gate and one to generate the final two's complement ϕ. Actually, with a flattened implementation, the module that generates the tan(ϕ) absolute value can be merged with the comparators. Then, only two clock cycles are needed to compute Arctangent over the whole range of $[-\pi/2, \pi/2]$. And the wide OR gate keeps giving the worst combinatorial path delay and infers directly the clock period to use.

4.7.9 The Arctangent Core allows easy re−pipelining and Re−timing

All the considered implementations favour speed rather than area consumption. Although, the implementation results are obtained with the free Xilinx ISE WebPack; no extra synthesis tools for optimisation are used, and the preponderant impact of routing on timing performances could be reduced by forcing a more compact localisation of the Arctangent components. However, that would prevent the place and rout tool from utilizing the unused resources located in the area of the Arctangent bloc.
Obviously, if utilizing more pipeline stages fits the used application requirements, more registers can be added to re-pipeline the proposed core and use higher frequencies. Examples of combinatorial delays of the most critical components are given with their number of logic levels to ease the re−pipelining analysis if it is needed

4.7.10 Saving Area Consumption

To save area consumption, one may use additional and professional synthesis tools, rather than just the free ISE WebPack. And depending on the needed modularity of the application, one may isolate the arctangent core and concentrate its implementation in the most compact possible area.
Another area saving can be made by replacing the two buffers of registers and their underlying routing by hard coded numbers distributed in shift registers located in LUTs already used for logic functions similarly as done in reference [38]. The drawback of such option is the lack of portability of the design to other technologies. For VLSI implementations other than FPGAs, this option would eliminate all the registers constituting $\tan(\phi)$ buffer and ϕ buffer in Figure 4.3 and Figure 4.6.

4.7.11 Power Saving

XPower, the used Xilinx ISE estimator shows a maximum power estimation of 348mW for the largest implementation of the core corresponding to an angle resolution of 0.25° and a 32−bit $\tan(\phi)$. That includes the Input Output Block consumption. In this Thesis, because we are presenting a generic solution portable on any VLSI technology and because our validation is done on FPGAs, our design specifications are not driven by power consumption. As a future work, our proposed core will be implemented at the transistor level to be more efficient on ASICs and on SoCs. The needed investigation for such implementations will consider saving power as a primer constraint. At that level of design, our proposed core will be completed with an investigation on the most balanced speed/power consumption; suitable transistor level components as well as power gating techniques will be brought to the proposed architecture.

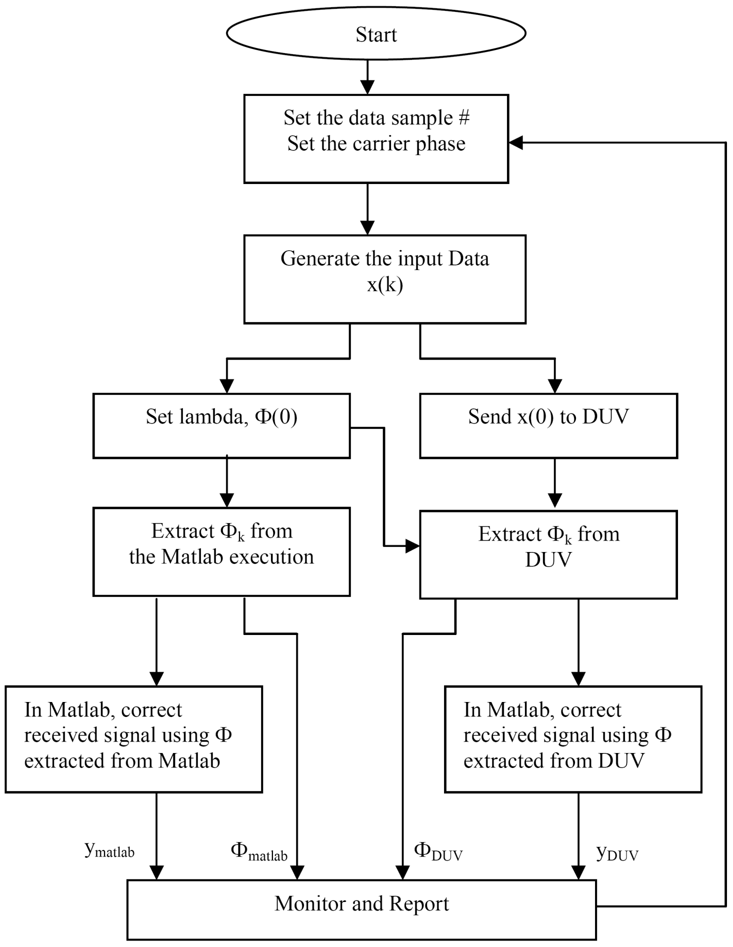
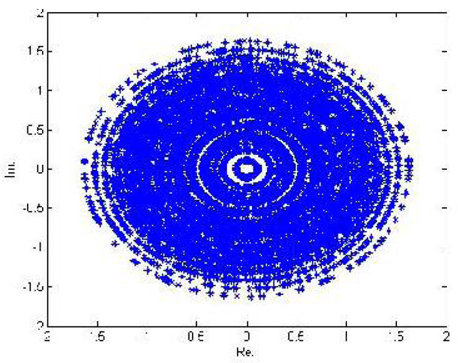


Figure 4.8: **Overall Phase Tracking Verification Flow**

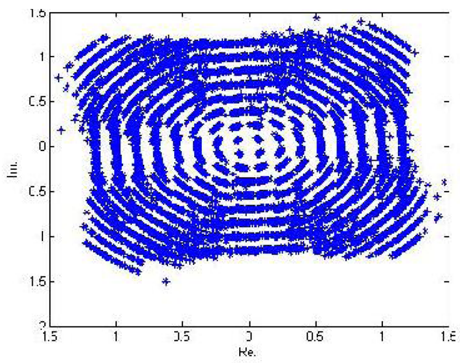
4.8 VLSI BCPT Validation

In this section, we show a simulation matching the validation published in [3].
So, as in reference [3], the carrier phase is assumed to be time varying and is given by $\phi_k = \phi_0 + 2\pi k \delta f_0$ Where ϕ_0 is the constant carrier phase error of 0.5 rad , and δf_0 is the carrier frequency offset of 0.001, and a signal-to-noise ration (SNR) level of 30db. λ is equal to 0.04 and the input data x(k) is computed using a 256-QAM.

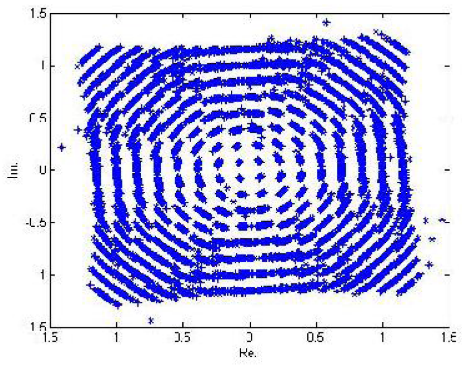
As shown in the verification flow in Figure 4.8, x(k) with the applied noise (Figure 4.9.a) is generated from MATLAB and input to the RTL netlist. The phase values, obtained with the RTL simulation, are adequately reformatted and sent to the MATLAB code where the received data are corrected and displayed as shown in Figure 4.9.b. The same functions; phase value extraction and data correction, are performed in MATLAB and corresponding demodulated data are displayed in Figure 4.9.c. The carrier phase tracking from both of the simulations is displayed in Figure 4.9.d. Comparing Figure 4.9.b and Figure 4.9.c



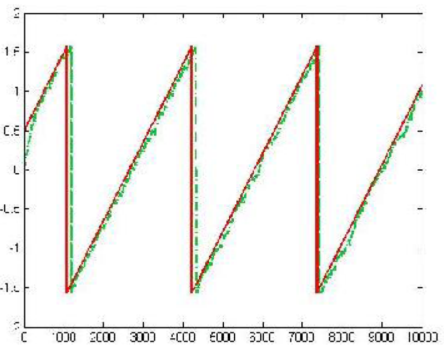
a. Input QAM256 Data with the applied Noise



b. Demodulated Data with RTL simulations



c. Demodulated Data with MATLAB computation



d. Phase Tracking:
— **computed in Matlab**
-·- **computed with the RTL simulation**

Figure 4.9: **Demodulated Data with MATLAB and RTL Simulations**

shows that the noise transferred by the proposed circuit is higher than the noise transferred by the MATLAB model. That is expected because of the data accuracy difference used in the two simulations. For any given application, our architecture's generated and transferred jitter and wander should be carefully evaluated to determine which noise can be tolerated, which filtering and/or which accuracy adjustment should be applied to satisfy the needed requirements. However, both Figure 4.9.b and Figure 4.9.d show clearly that the RTL netlist obtained from our Hardware-Dedicated-Procedure detects effectively the needed phase deviations and allows a closed carrier tracking.

Chapter 5

CONCLUSION

Developing theoretically working Signal Processing algorithms and Intellectual Properties compliant to industrial standards are two competitive advantages that put VLSI design researchers in the value chain of Semiconductor Industry [13], [39]. We used those two advantages to design two generic Phase Detectors. A linear Phase Detector and a Quadrature Phase Detector.
The Linear Phase Detector is designed to be generic and to be conformant to the Telcordia Technologies "Clocks for the Synchronized Network: Common Generic Criteria" [13]. Among other contributions, in the frame of Linear Phase detectors, we present a solution for implementing ratios with high precision data. When two's complement numbers with the smallest absolute values need the same as or even a better accuracy than the numbers with the largest absolute values, unlike the other published methods, we provide dynamic accuracy adjustment favouring meanly the removal of the sign bit redundancy while indicating which level of accuracy that is being processed. A phase slope monitoring application, utilized in synchronization systems, is given to show the appropriateness of the proposed method. The corresponding design is generic. FPGA implementation results are provided to show the performance effectiveness when adding our proposed bridge to dedicated and fixed size dividers.
For the Quadrature Phase Detector, we have proposed an optimization of the blind carrier phase tracking with guaranteed global convergence algorithm specifically suited for VLSI architecture. Based on statistical independence of the real and imaginary parts of the emitted signal, the new architecture uses a blind source separation procedure to achieve the carrier tracking with guaranteed global convergence. Efficient design techniques are presented to solve some critical issues related to the arithmetic unit and storage components. We showed how it is possible to efficiently time multiplex the large multipliers. We also presented an approach for implementing a fast Arctangent function using high precision data. A particular CAM architecture using directed inequalities rather than equalities and a Decoder using mostly parallel logic are employed to decrease the decoding complexity. That decoding complexity depends on the considered number of angles rather than the related memory address precision. The approach responds to the need of VLSI implementations of Arctangent functions running in few clock cycles, and receiving and

transmitting high precision data. An application example is given to show that need. The proposed solution enables designers to customise the output angle's precision and optimize the tangent input data to transfer the highest inaccuracy risk from Arctangent modules to other parts of the used systems. Different FPGA implementation results are provided to show our approach's appropriateness when neither the usual LUTs based solution nor the polynomial and rational approximations fit. Our proposal favours the speed and data precision rather than power and area consumption. To show the effectiveness of the whole algorithm design specification and to support the proof of concept, we benchmarked our RTL simulations against the theoretic validation approach published in [3]. Simulation results are satisfactory, as they match the results reported in [3].
We are engaged to port the proposed solutions to a transistor level design to increase circuit density and to utilize appropriate components for a more balanced speed and power trade off. Our work can be improved with further research and development in the field of bus-width optimisation. To that end more resources are planned for continuing this project and for adding-in more theoretical contributions and more engineers for conformance testing.

Bibliography

[1] John Howard and Hans-Peter Landgraf, "Quadrature sampling phase detection",©1994 American Institute of Physics, Rev. Sci. Instrum., Vol. 65, No. 6, June 1994

[2] Jiancheng Li, Tao Xu, Zhaowen Zhuang, Yongfeng Guan, "High Performance All Digital hase Locked Loop Mathematics Modeling And Design, "Proceedings of the 2008 IEEE International Conference on Information and Automation, June 20 -23, 2008, Zhangjiajie, China

[3] Adel Belouchrani, Wei Ren "Blind Carrier Phase Tracking with Guaranteed Global Convergence," IEEE Transaction on Signal Processing, VOL. 45. NO. 7, JULY 1997.

[4] N. Jablon, "Joint blind equalization, carrier recovery, and timing recovery for high order QAM signal constellations, "IEEE Trans. Signal Processing, vol 40, pp. 1383-1397, June 1992.

[5] "Combined IEEE1588 Timing over Packet and Synchronous Ethernet Device", Datasheet, http://www.zarlink.com/zarlink/hs/82_ZL30310.htm

[6] S. Tagzout, Y.I. El-Haffaf, A.K Oudjida, "A Reconfigurable Fast Walsh Transform Architecture for VLSI Implementation", International Conference on Signal Processing and Technology (ICSPAT'96), pp. 1964 - 1968, November 1996, Boston, USA

[7] S. Tagzout, K. Achour and O. Djekoune, "Hough Transform Algorithm for FPGA Implementation," Signal Processing Journal, Elsevier Science Publishers, 81 (2001) 1295-1301.

[8] S. Tagzout and L. Sahli, "Compact Parallel Multipliers Using the Sign-Generate Method in FPGA," , Microelectronics Journal, Elsevier Advanced Technology, Oxford, England, VOL 29, NÂ° 11, pp. 827-831, November 1998.

[9] Record from Web of Science in http://apps.isiknowledge.com

[10] S. Tagzout, A. Belouchrani and M. Ouali, "VLSI Architecture of the Blind Carrier Phase Tracking with Guaranteed Global Convergence," 7th International Workshop on Systems, Signal Processing and their Applications (WOSSPA 2011), Algiers, ©2011 IEEE, 191-194

[11] S. Tagzout, A. Belouchrani, "Arctangent Architecture For High Speed And High Precision Data," Journal of Circuits, Systems, and Computers (JCSC) ©World Scientific Publishing Company, Vol. 20, No. 7 (November 2011)

[12] Préambule du Programme national de Recherche, Direction Générale de la Recherche Scientifique et du développement Technologique, www.dprep.nasr-dz.org, Mai, 2010

[13] Iikk Tuomi, "The future of semiconductor Intellectual roperty Arcitectural Blocks in Europe", Instute for Prospertive Technological Studies, European Comission, EUR 23962 EN 2009

[14] Daniel Aspel, "Adaptive Multilevel Quadrature Amplitude Radio Implementation in Programmable Logic," Master of Science Thesis, April 2004 , Department of Electrical Engineering, University of Saskatchewan, Saskatoon, Saskatchewan, Canada

[15] Behzad Razavi, "Design of analog CMOS Integrated Circuits," MacGraw-Hill International Edition Inc., Electrical Engineering Series, 2001.

[16] Telcordia Technologies, "Clocks for the Synchronized Network: Common Generic Criteria," GR-1244-CORE, www.telcordia.com, USA, Issue 3, May 2005

[17] "Virtex-5 Family Overview", Product Specification, DS100 (v5.0), February 6, 2009

[18] Xilinx LogiCore, Product Specification, Divider v9.0, DS255 July 13, 2006.

[19] Xilinx LogiCore, Product Specification, Multiplier v9.0, DS255 July 13, 2006.

[20] Randy Yates, "Fixed-Point Arithmetic: An Introduction," Technical Reference, Digital Signal labs signal processing systems, August 23, 2007.

[21] Yuyu liu and al "A novel Clock and data Recovery Scheme Based on Sigma-Delta Quantization," ASICON 2005, 6th International Conference on Volume 1, Oct 205 pp. 436-440.

[22] Zhaoyang Zhang and al "Implementation Scheme of VSB Modulation for HDTV Terrestrial Broadcasting," ISCE'97 Procedings of 1997 IEEE international Symposium Consumer Electronics Dec 1997 pp. 67-70.

[23] J.K Wu and al, "Microprocessor-Based Phase Tracking System For Digital Power Metering," Electrical and Computer Engneering, 2004 Canadian Conference on Volume 1 May 2004, pp. 19-22.

[24] N. Jablon, "Joint blind equalization, carrier recovery, and timing recovery for high order QAM signal constellations, "IEEE Trans. Signal Processing, vol 40, pp. 1383-1397, June 1992.

[25] D. Godard, "Self recovering equalization systems," IEEE Trans, Commun, vol. Comm-28, pp. 1867-1875, Nov.1980.

[26] Retrieving documents in http://citeseer.ist.psu.edu/

[27] E. Serpedin, P. Ciblat, and G.B. Giannakis, "Performance Analysis of BlindCarrier Phase Estimators for General QAM Constellations", IEEE Trans Signal Processing, vol. 49, no. 8, pp. 1816-1823, August 2001

[28] J.-F. Cardoso, A. Belouchrani, and B. Laheld, "A new composite criterion for adaptive and iterative blind source separation." In Proc. ICAPSSP, 1994, pp. 273-276.

[29] Seughyeon Nahm and al, "A Fast Direction Sequence Generation Method For Cordic Processors," 1997 IEEE International Conference on Acoustic, Speech and Signal Processing, Volume 1, April 1997 pp. 65-638.

[30] Richard Lyons, "Another Contender in the Arctangent Race," IEEE Signal Processing Magazine, January 2004, pp. 109 110.

[31] Sreeraman Rajan, Sichun Wang, Robert Inkol, and Alain Joyal, "Efficient Approximations for the Arctangent Function," IEEE SIGNAL PROCESSING MAGAZINE, MAY 2006, pp. 108 111

[32] S. Rajan, S. Wang, R. Inkol, "Error reduction technique for four-quadrant arctangent approximations," IET Signal Process., 2008, Vol. 2, pp. 133138

[33] I. Koren and O. Zinaty, "Evaluating elementary functions in a numerical coprocessor based on rational approximations," IEEE Trans. Comput., vol. 39, no. 8, pp. 10301037, Aug. 1990

[34] M.Saber, Y.Jitsumatsu, T.Kohda, "A low-power implementation of arctangent function for communication applications using FPGA," Proceedings of IWSDA , 2009

[35] F. Dinechin, A. Tisserand, "Multipartite Table Methods" IEEE Transactions On Computers, VOL. 54, NO. 3, March 2005, 319330

[36] R. Gutierrez, V. Torres, "FPGA implementation of atan(Y/X) based on Logarithmic transformation and LUT-based techniques ", Journal of Systems Architecture, Vo 56, No. 11, (2010), 588596

[37] " Virtex-5 FPGA User Guide", UG190 (v5.3), pp.198, May 17, 2010

[38] J.-L. Brelet and B. New, "Designing Flexible, Fast CAMs with Virtex Family FPGAs", Application Note XAPP203, (Version 1.1), Sept2003.

[39] S. Tagzout, "IC IPs & Technology Migration A Business Opportunit", MBA 6262B, Faculty of Graduate and Postdoctoral Studies, Ottawa University, February 2009