Oversampled Multi-Phase Time-Domain Bit-Error Rate Processing for Transmitter Testing

By

Rozita Najafi Nejad Nasser

Department of Electrical Engineering

McGill University, Montreal

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Engineering.

October 2011

Abstract

High speed serial interfaces (HSSI) are continually pushed toward operating at higher speed to meet the demand for higher bandwidth. As a result, the timing constraints for HSSI devices get tighter. Consequently, HSSI devices experience issues such as timing jitter and bit-errors. This thesis investigates techniques to speed up bit-error rate (BER) and jitter testing of HSSI devices.

This work proposes an oversampling-based transmitter test scheme that accelerates transmitter jitter as well as eye diagram testing through the deployment of a multi-phase bit-error rate test circuit (BERT). The proposed scheme creates parallel BERT elements working in conjunction that are able to digitize the input signal jitter behavior in a multi-phase manner. The more phases we deploy the faster the test is completed.

We aim to accurately extract the transmitter jitter in time domain and finish the whole transmitter test within tens of milliseconds. This exceeds the performance of [2], which by itself was an improvement from seconds to 100 ms.

Résumé

Les interfaces sérielles à haute vitesse voient leur vitesse continuellement augmentée afin de satisfaire à des exigences de bande passante sans cesse croissantes. Ces interfaces sérielles doivent donc rencontrer des contraintes temporelles toujours plus serrées. Ceci a pour conséquence l'apparition de problèmes de vacillement et d'erreur sur les bits. Ce mémoire explore des techniques permettant l'accélération des tests de vacillement et de taux d'erreur sur les bits pour les interfaces sérielles à haute vitesse.

Nous proposons une méthode de test de transmetteur basée sur le sur échantillonnage qui accélère le test du vacillement et du diagramme de l'oeil par l'utilisation d'un circuit de test de taux d'erreur sur les bits (BERT) multiphase. La méthode proposée fait usage de plusieurs éléments de test en parallèle travaillant ensemble et permet de numériser le comportement du vacillement du signal d'entrée de façon multiphase. Plus le nombre de phases utilisé est élevé, plus rapide est le test.

La méthode proposée va au delà de nos résultats obtenus avec les interfaces de disque SATA [2], soit un temps de test passant de quelques secondes à 100 ms. Elle permet en effet d'extraire de façon précise le vacillement dans le domaine temporel et de compléter la totalité du test du transmetteur en quelques dizaines de millisecondes.

Acknowledgements

First, I would like to thank my supervisor, Professor Zeljko Zilic, for his guidance, encouragement, patience, and understanding during my master's studies; his support and insightful advice have always been of amazing value. This thesis would not have been possible without his support that allowed me to take part into the NSERC Engaged Program and his constant guidance that followed.

I would like to thank the crew at DFT Microsystems for the fruitful experience I had with them. In particular, I wish to thank Mohamed Hafed for giving me the chance of using research carried out at their company towards the completion of this thesis. I would also like to thank Carle Banville for providing me information about existing designs along with his ever useful suggestions and advice. I thank Nadine Kolment and Cameron Hayne for being willing to help and always finding time to provide me with their opinions.

I wish to thank Professor Andrwas Swidan who has always been a constant source of encouragement and support. His useful comments, good advice, and moral support have been invaluable.

I would like to express my appreciation to Gaetan Gauthier and Simon Bussieres for offering me internship opportunity at Matox Company for two semesters. This experience enlightened my first glance of video hardware. I thank Matrox video

hardware group for their friendship, support, and advice throughout my internship period.

I also thank the students in the MACS lab for their friendship, help, and motivation. Especially, I would like to thank Omid Sarbishei and Luca Montesi for taking an interest and having the time to help me in reviewing this thesis.

The most special thanks goes to my sister for her endless love and support; she was always there offering her unconditional help and assistance through all this time. My deepest gratitude also goes to my parents and my brother who have always believed in me and encouraged my pursuit of bigger and better things.

Table of Contents

List of Figures

List of Tables

Chapter 1 - **Introduction**

1.1 **Motivation**

Moore's law continues to drive the semiconductor technology roadmap to double the number of transistors on an integrated circuit (IC) approximately every two years; this constantly gives IC systems more functionality and higher performance. As the feature size of integrated circuits continues shrinking into submicron technologies, a complex and functionality-rich system needs to have fast enough input/output (I/O) to be efficient. To achieve better system capability and performance, I/O speed keeps increasing in advanced IC systems [11].

Decreasing feature size while increasing I/O speed leads to enhanced system capabilities and performance; however, such advancements also introduce many signal integrity challenges. Jitter, which is the timing deviation of a signal from its ideal behavior, is one of the most important challenges. In order to ensure a reasonable bit error rate (BER), the total available jitter budget must decrease as I/O speed increases. Other critical challenges include noise and power consumption. Low power constraints translate into noise being harder to control and a high signal to noise ratio (SNR) becomes a common issue. Frequency loss at high data rates is another signal integrity issue. When cost-effectiveness dictates keeping channel material unchanged, increasing I/O link data rate makes jitter, noise, and signal integrity issues very challenging [11].

1.1.1 BER Testing Perspective

A digital communication system consists of a transmitter, a communication channel, and a receiver. The channel, serves as a physical medium to send a signal from the transmitter to the receiver; it can be a pair of wires, an optical fiber, or any other communication medium. One problem associated with any communication system is that it may randomly corrupt the transmitted signal. As the transmitted signal propagates through the channel, it experiences interference and dispersion effects. Therefore, all communication systems experience some form of a transmission error [26].

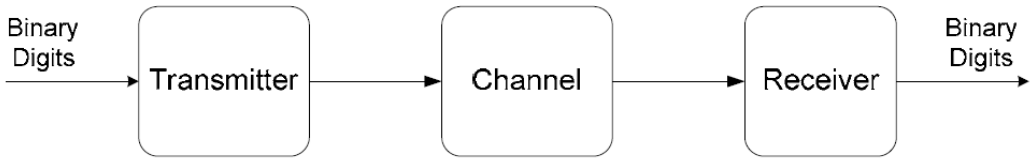

Figure 1-1 : Block Diagram of a Digital Communication System

Bit-error-rate (BER) is a measure of the performance of a communication interface [15]. Bit-error rate (BER) is the ratio between the number of incorrect bits received and the number of total bits received. It quantifies the probability that a data stream will reach the receiver with bit-errors introduced by the channel [2].

Bit-error rate (BER) is affected by factors such as the amount of noise in the communication channel, timing jitter, transmitter power, and the type of waveforms that are used to transmit information [2].

A bit-error rate tester (BERT) performs a bit-by-bit comparison between the data stream received from the design under test (DUT) and the data stream generated by a pattern generator. The DUT can be any communication interface which receives bit sequences and after certain signal processing it restores the sequences back [3].

1.1.2 Qualification challenges

High speed serial interfaces (HSSI) are a relatively recent means of high speed communication between blocks, electronic integrated circuits, circuit boards and systems. There are numerous HSSI standards addressing different applications; some of these are SATA, XAUI, and Fiber Channel [4]. High speed serial interfaces are also finding widespread use in consumer devices such as High-Definition Multimedia Interface (HDMI), PCIe, and USB [10].

Increasing demand for bandwidth is continuously pushing high speed serial interfaces (HSSI) toward higher data rates [23]. As silicon vendors are forced to provide HSSI with higher rates, timing budget is getting tighter; the traditional "Guaranteed by Design" cannot be applied anymore [4]. For instance, at 10 Gbps, the Unit Interval (UI) is 100ps, and at 40Gbps, the UI is only 25 ps. To maintain a reasonable BER (10^{-12}), random jitter should be in the order of sub pico-seconds or less. Such stringent requirements make development complex and expensive. As the data rate of HSSI devices keeps increasing, jitter, noise and signal integrity challenges are bound to become even harder to tackle and stricter silicon validation will become necessary [23].

Unfortunately, Automated Test Equipment (ATE) cannot always be used for HSSI testing and validation. Limitations appear on systematic solutions when data rates go above 6 Gbps [4]; this includes the testing of HSSIs on ATEs. The pace at which data communication rates increase is faster than the pace at which test equipment evolves [13][4]. For instance, with today's fastest ATE (AWG6000), only jitter tolerance testing of up to 3Gbps is performable [4].

Testing should be as inexpensive and fast as possible. Jitter and bit-error rate (BER) are just one of the hundreds of parameters to be tested on an average device. Unfortunately, on a traditional bench it might take hours or days to do BER or jitter testing [2][15]. In order to keep up with a competitive market, it is important to develop methodologies that to seek to speed up the BER testing process.

It is both challenging and expensive to perform transmitter/receiver jitter tolerance tests [4]. Most standards for serial links require a bit-error-rate (BER) level of 10^{-12}or less [1][2][15]. In order to guarantee a BER level of 10^{-12} at least 10^{13} bits should be run; obviously, this is extremely time-consuming, as even at 1.5 Gbps it takes almost 2 hours (111 minutes) to transfer all the required bits. The emergence of new applications requiring lower BER levels, for instance 10^{-14} make BER testing even more time-consuming [1]. Extraction techniques such as the dual-Dirac model are used to reduce the long test time [28][27].

1.2 **Contribution**

We seek techniques to address the urgent need in the semiconductor industry for cost-efficient solutions to qualify HSSI jitter and BER performance. Our aim is to improve the speed of bit error rate (BER) testing. Intuitively, this is achieved by creating parallel bit-error rate test (BERT) elements working concurrently. These elements must be able to digitize the input signal jitter behavior in a multi-phase manner. This requires a high bandwidth digitizer to capture the transmitter bandwidth. The more phases are deployed the faster the test will be. Depending on the target data rate, we deploy 16, 8, 4, or 2 phases and aim at obtaining BERT scan plots 16, 8, 4, or 2 times faster. The lower the target data rate, the more of a speed-up is obtained.

We can also develop accelerated transmitter jitter and eye diagram tests. This data can be accurately extracted from bit error rate (BER) testing results. Using the dual-Dirac model [28][27] we extract the jitter components from the BER-scan plots. Dual-Dirac was used since it enhances the speed of jitter testing by allowing extrapolation of bathtub curves at lower BER levels.

The overall innovative testing scheme has been verified at data rates up to 6.4 Gbps. The accuracy of the proposed scheme has been verified by comparison with the existing design. The comparisons were carried out on a DFT Microsystems DJ60HS Test Module.

1.3 **Thesis Outline**

In the remainder of the thesis, Chapter 2 presents the background of the research that was carried out. It first presents an introduction on HSSI technologies and briefly explains how BER and jitter are related to each other. The chapter continues with the discussion of transmitter jitter testing solutions and how jitter components can be extracted from BER-scan plots.

Chapter 3 covers the existing BERT testing scheme. This chapter mainly describes the implementation techniques used in a DJ60HS test module by DFT Microsystems Canada Inc. Chapter 4 starts by introducing the multi-phase BERT concept and explaining how the algorithm improves the performance of BER and jitter testing. The rest of the chapter covers major design considerations and their development and implementation.

In Chapter 5 results are presented. We perform different kinds of tests to ensure the performance and functionality of the multi-phase BERT scheme. Each test is done multiple times under certain conditions and results are evaluated using statistical methods. This chapter also includes results concerning the speed enhancement we have achieved in transmitter testing.

Conclusions and future investigation directions are provided in Chapter 6.

Chapter 2 - **Background**

High-speed digital interfaces have become a cornerstone of modern communication. Due to the challenges these interfaces pose, there have been works done on qualifying and testing timing specifications of high-speed digital interfaces and some books written on this topic [15][26][27]. While these books present more detailed description of high-speed interfaces testing, in this chapter it suffices to explain only the concepts that provide the required background information of the work done in this thesis. Much of what is state of art is actually "trade secret"; hence, there aren't many publications in open literature, including even the patents.

2.1 **HSSI Structure**

Many of today's electronic systems make use of high-speed serial links. Some of these links are: PCI-Express, XAUI, USB, etc. [26]. High speed serial interfaces (HSSI) make use of a serializer-deserializer transmission scheme called SerDes. As shown in Figure 2-1 the transmitter includes a synchronizer and a serializer. The transmitter (Tx) takes in parallel data and converts it into a serial stream which includes the clock [5]. The receiver (Rx) accepts high speed serial data from the transmission media makes use of a synchronizer, a deserializer, and a clock recovery unit to extract the clock and restore data to the original parallel format [15] [5].

Clock Data Recovery (CDR) circuits are used to extract clock information [25] in most high speed serial interfaces. The clock is encoded into the data signal in the transmitter and therefore recovered at the receiver side [23].

Phase-locked loop (PLL) circuitry in the transmitter provides an internal high clock. It locks the transmitter output rate to the reference clock.

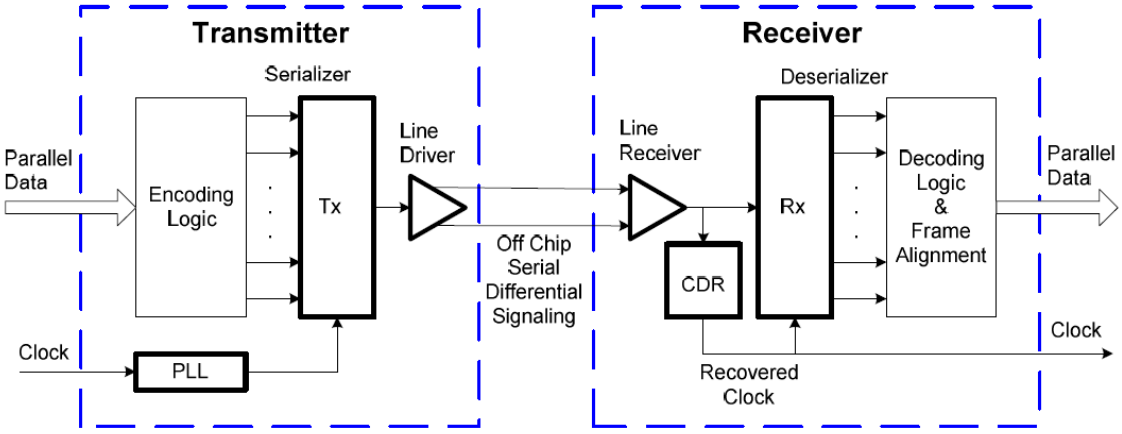

Figure 2-1 : Block diagram of an HSSI [13]

Bit errors in HSSIs are largely due to jitter. The jitter performance of a Serializer/Deserializer (SerDes) device can be characterized by the jitter found in the output of the transmitter and by the jitter tolerance of the receiver.

2.2 BER Mechanisms

In serial communication systems, the transmitter, the receiver, or the transmission media can cause distortion or bit-errors [15]. Many design choices such as chosen CDR mechanism, PLL bandwidth, encoding/decoding method, and transmitter

power affect the performance of communication interfaces [25][15]. As a measure of transmission quality of the overall communication system, BER is the probability of a bit error at the output of the receiver, compared to the input of the transmitter [15].

By definition, BER is derived by dividing the average number of transmission errors over the total number of the bits transmitted to the receiver in a specified time interval.

The test set up for measuring BER mainly consists of a block that compares the logic state of the data appearing at its two input port. One input contains ideal data (or transmitted data) and the other input contains received data. The compare block captures data over a specified amount of bits from this the amount of transmission errors is identified [26].

2.3 **Jitter Impact to BER**

An ideal waveform transmitted in a communication system consists of four components: a high level "1", a low level "0", a rising edge (0 to 1 transition) and a falling edge (1 to 0 transition).

When an ideal signal is transmitted over a communication system, it gets contaminated by a physical process called noise [15]. Noise deviates an ideal signal in two different manners; a timing deviation called jitter and an amplitude deviation referred to as amplitude noise [29].

At the receiver side, the Clock Data Recovery (CDR) samples the actual data signal at sampling instance t_s, and compares the sampled value with a threshold voltage V_t. If the value of the data at that instance is bigger than V_t , logic "1" is received; otherwise, logic "0" is received. The CDR of an ideal receiver samples data in the middle of each data bit ($t_s = \frac{UI}{2}$) [15].

As shown in Figure 2-2 , under the presence of jitter, the transition edge of the signal can fluctuate horizontally across the sampling point (along the time axis). The time deviation can cause a bit error – bit "0" is received as "1" or bit "1" is received as "0".

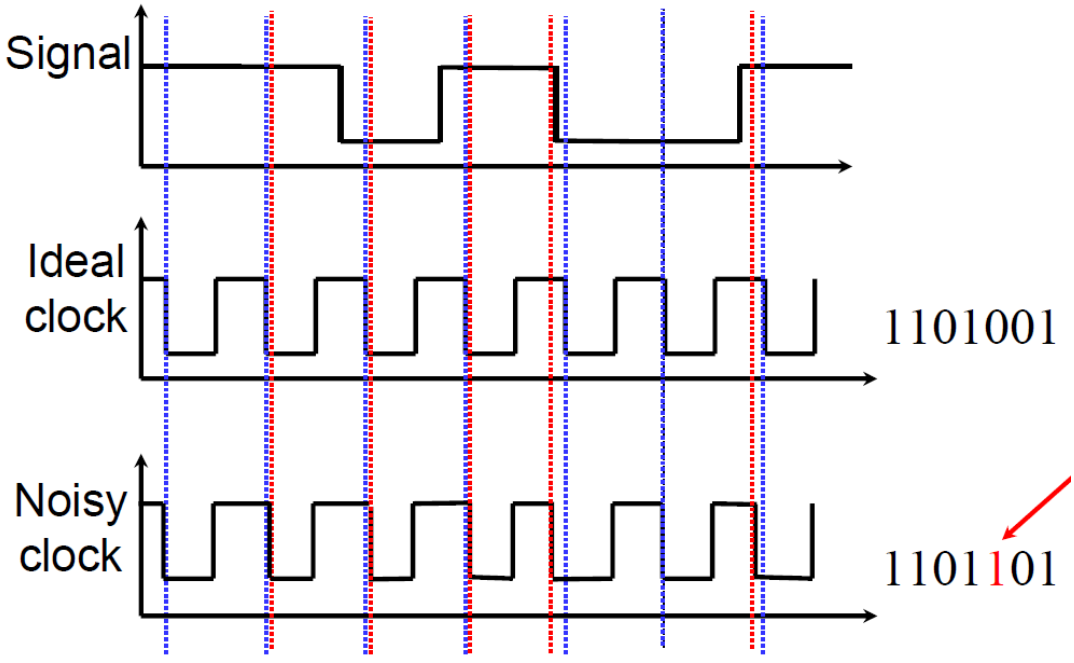

Figure 2-2 : Time deviation of the sampling clock causes bit-error [29]

If we ignore setup and hold time requirements, timing jitter can cause bit errors under two different conditions. The first case occurs when either the rising edge

lags behind the sampling instance or the falling edge is ahead of the sampling instance; here logic “1” is received as logic “0”. The second case occurs when either the falling edge lags behind the sampling instance or the rising edge is ahead of the sampling instance; here a logic “0” is received as a logic “1” [15].

Bit errors can also be caused by amplitude noise; if the momentary noise voltage exceeds the noise margin, a wrong value can be sampled. This can occur even though the sampling takes place at the correct moment [29].

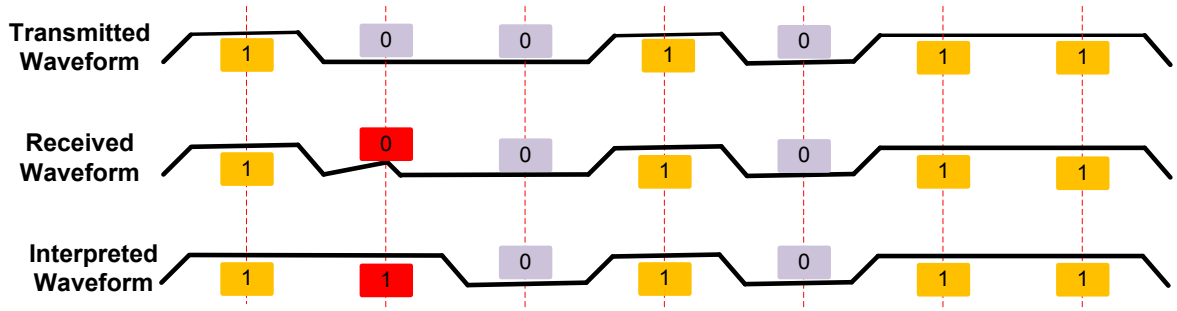

Figure 2-3: Signal voltage can fluctuate vertically across the sampling point and cause a bit-error [29]

2.4 Timing Jitter

Jitter is the time deviation of a periodic signal from its ideal timing with respect to a reference clock [24]. Jitter is a significant, usually, undesired factor in the design of almost all communications links such as PCI-e, USB, and SATA [15].

Jitter is composed of deterministic and random components [24]. Deterministic jitter (DJ) is a type of timing jitter or data signal jitter which is caused by predictable events. This type of jitter has a bounded peak-to-peak value. Random

jitter (RJ) is caused by random events and is usually statistically characterized by a Gaussian distribution quantified by a mean and a standard deviation [24]. Due to the fact that the peak-to-peak value of a Gaussian distribution approaches infinity, the distribution is unbounded and the peak-to-peak value is sample size dependent. Therefore, the peak-to-peak value of random jitter (RJ) is dependent on both the BER and total jitter which is defined at a certain BER level [15].

The probability density function (PDF) of total jitter (TJ) is obtained by performing the convolution of the PDF of the deterministic jitter (DJ) and the random jitter (RJ). Therefore the PDF of TJ is the convolution of the DJ and RJ PDFs [14].

Usually total jitter (TJ) and deterministic jitter (DJ) are specified separately in communication standards such as Serial ATA (SATA), XAUI, and fiber channel [2].

The amount of total jitter depends on the application and is associated with a certain bit-error rate (BER) [14]. For instance, if the transmitter (Tx) jitter specifications of an HSSI device requires a BER level of 10^{-12}, the amount of total jitter (TJ) should not exceed 0.3UI. Also, the DJ should not exceed 0.17UI. It is impossible to directly measure the total jitter (TJ) value at a BER level of 10^{-12} due to long testing time. This leads to the use of an RMS value of the random jitter (RJ). The RJ limit can be obtained by assuming all TJ contributes to RJ; RJ should not exceed 0.3 UI at a 10^{-12} BER level. The RMS value of the RJ limit

will be 0.021 UI which converts into about 7ps if the data rate is 3 Gbps and into about 3.5 ps at a 6Gbps data rate [2].

2.5 Related Work on Transmitter Jitter Testing

Jitter measurement and decomposition is important for accurately deriving the total jitter in a system and for identifying the root causes of jitter. Jitter decomposition has been investigated for years [18] [19], techniques proposed in [18][19] were mainly histogram-based analysis. The method proposed in [34] could extract sinusoid jitter using external ATE. This method was extended [35] to perform jitter spectral analysis. To ensure accuracy, both methods depend on external reference clocks and external equipment. However, the jitter characteristics of a signal may not be necessarily sinusoidal [37]. Assuming the jitter characteristics of a signal with a double-delta PDF leads to inaccurate estimating DJ and RJ, as well as the performance of the system [36]. As stated in [36], PDF of double-delta function would be the convolution of a random Gaussian-distribution with a sinusoidal function. The method reported in [37] , propose a spectral analysis scheme for extracting jitter without requiring an ideal external sampling clock.

Presently jitter analysis and characterization is performed using various products on the market. Some of these products perform their analysis on jitter measurements from time interval analyzers (TIA). Others use sampling oscilloscope jitter measurements or real-time oscilloscope jitter measurements [28]. Bit Error Rate Testers are also for jitter testing and validation on bench [2].

However, these solutions cannot be directly applied for in production at-speed testing because of their low throughput [15].

Today, there are not many systems that can perform jitter compliance testing for multi-gigabit devices in production [15]. Many jitter test solutions are based on extra on-chip circuitry or on add-on modules [10][15][17]. These solutions are limited by their low throughput, low accuracy, or high design complexity of the loadboard [15].

In [38] a technique for estimating total jitter is described, which pointing out the limitations of dual-Dirac [33][32] method, proposes a high-order polynomial tail fitting technique that can estimate jitter at low BER levels. Another approach of jitter decomposition is presented in [39] which can separate correlated and uncorrelated components of deterministic jitter. A novel decomposition algorithm is presented in [40] which investigates Gaussian tail behavior of measured distributions and determines random and deterministic components of jitter.

The transmitter jitter test solution proposed in [30] is based on the high-speed digital pins of the Agilent's 93000 ATE. This approach first builds a bathtub curve by shifting the compare strobes along the timing axis and level threshold axis. A jitter separation algorithm is then applied. However, even with a 2ps resolution, each BER measurement takes about 1 second. Both the test time and the accuracy need to be improved: the RJ here measured is approximately 1ps greater than the one measured by the bench [15]. The transmitter jitter testing scheme presented in [31] is not very accurate; the random and deterministic jitters

reported are respectively 1~2ps and 20ps greater than what measured by the bench equipment. Also, test parameters can be further optimized to achieve better test economy [15].

While transmitter testing usually takes seconds, the under-sampling based transmitter test scheme presented in [2] can accurately extract the transmitter jitter and finish the whole transmitter testing within 100ms.

Exceeding the current norm of 100 ms, the multi-phase BER-scan approach presented in this thesis accelerates building bathtub curves; consequently, jitter components can be extracted in tens of milliseconds. The extraction technique used in this work is based on the dual-Dirac model [32].

2.6 **BER Bathtub Curve**

BER-scan is an approach to the measurement of transmitter jitter bathtub-curve is usually used [28], in measuring transmitter jitter with a BER-scan approach. The bathtub curve is generated by sweeping the sampling position along the timing axis and then recording BER at each sampling position [6]. Each sampling position on the timing axis is referred to as a *phase* and the total number of bits transmitted is referred to as the *duration* of the test.

By sweeping the sampling position through the entire bit period (UI), and performing transmitter BER measurement (involving a certain number of bits for each phase) leads to a plot. This plot shows the BER on the vertical axis and the

time phase on the horizontal axis. The time phase is the one of a valid data of a serial signal.

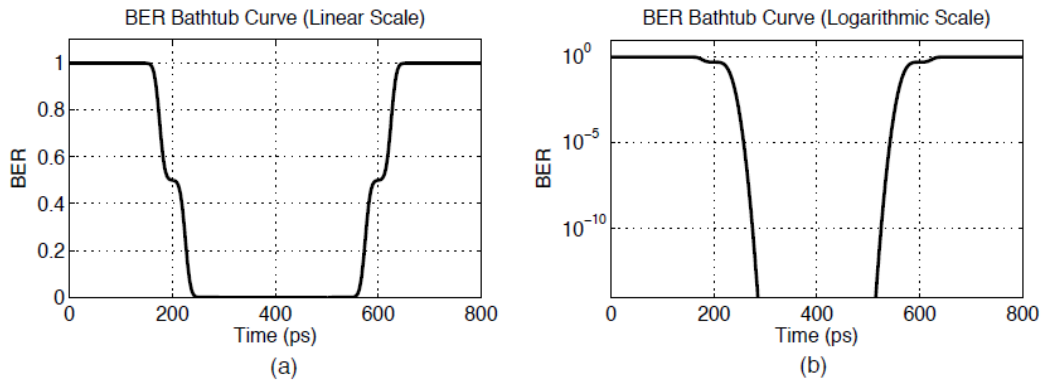

Figure 2-4 : BER bathtub curve in a (a) linear BER scale and a (b) logarithmic BER scale [27]

Usually, the two sides of the bathtub plot consist of a measured section and an extrapolated section. The top section (high BER) is calculated directly from the measured bit-error (TJ). The lower section (low BER) is calculated by extrapolation. In the transmitter BER-scan, the dual-Dirac model gives a better curve fit [28]. For instance, performing transmitter BER testing with a duration of 10^{7}bits yields the bathtub curve shown in Figure 2-5. The solid horizontal "lines" of the top section showing a BER higher than 10^{-7} are obtained by a direct bit-error measurement. On the other hand, the dotted "lines" below a BER level of 10^{-7} are calculated with extrapolation using the dual-Dirac model.

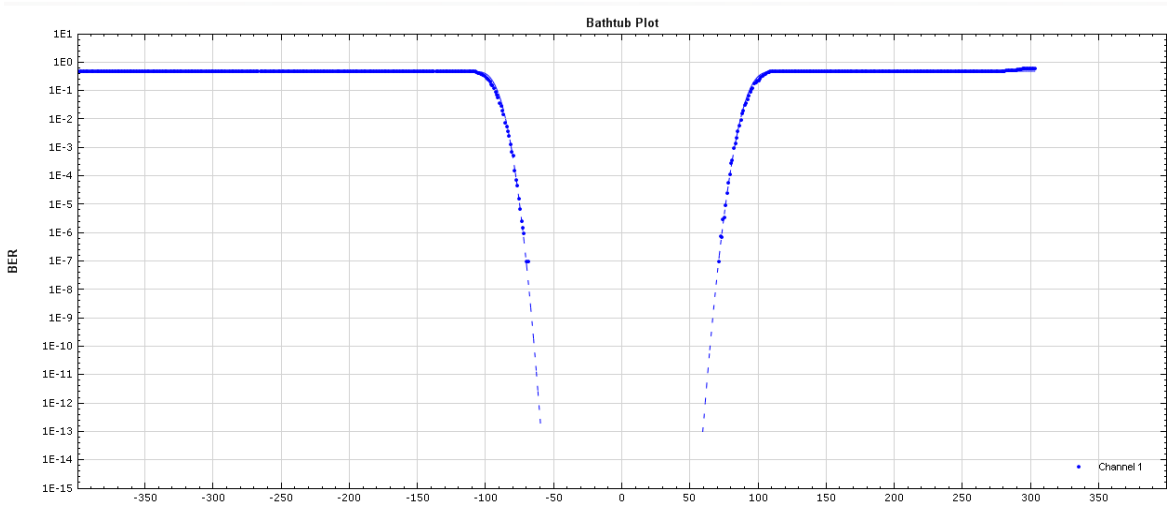

Figure 2-5 : The BER bathtub graph shows how the estimated BER (TJ) value is extrapolated from the measured BER (TJ) using the dual-Dirac jitter model.

BER bathtub graphs are useful in visualizing how a desired BER affects the data. Conversely, these graphs show what BER can be expected from a desired data. Examination of the intersection between the upper and lower sections of the plotted curves indicates how well the dual-Dirac model fits the measured jitter data [28].

2.7 The Dual Dirac Model and RJ/DJ Separation

In the bathtub-based transmitter jitter measurement, all we can see is the combined TJ profile; In other words, we see a convolution of the DJ and RJ. In order to derive the deterministic and random jitter (RJ) components, we need to work backwards [15].

RJ/DJ analysis separates a signal's aggregate total jitter into random jitter (RJ) and deterministic jitter (DJ) components [28]. Separating jitter components,

allows estimating peak-to-peak jitter values at very low BER levels that would otherwise take too much time to measure directly [15][27]. An additional benefit of this technique is that it helps diagnose and understand the underlying causes of jitter [28] . This is because BER gives little information about the mechanism that causes error but jitter does [29][15].

The dual Dirac jitter model was developed to model the RJ and DJ components of a digital signal. The reasoning comes from the fact that the deterministic component of the jitter is bounded [27]. This means that the simplest model for deterministic jitter is assuming that DJ is comprised of a pair of delta functions. In this case, the complicated convolution is reduced to a standard complementary error function [15]. The upper part of the bathtub curve is dominated by the DJ while the lower part is dominated by RJ [30].

The dual Dirac assumption serves as the model for modern bathtub curve fitting. The bathtub curve at lower BER levels is extrapolated using the higher BER values [28][27][15]. The extrapolation results can be verified by performing direct measurements at the lower BER levels. Therefore we borrow ideas from the transmitter BER scan for our jitter tolerance extrapolation.

Chapter 3 - **Existing BERT Implementation**

In this chapter we present BER test technology implemented by DFT Microsystems and provide a description of key implementation techniques that we are going to deal with in order to develop Multi-phase BERT scheme.

3.1 **Introduction**

Multiple data rates are being implemented on single High Speed Serial Interface (HSSI) devices due to a higher degree of integration. In order to accommodate multiple data rates on a single high speed serial interface (HSSI) device, the device needs to be capable of providing multiple rate clock signals. Multiple data rates can be implemented by changing divider ratios inside the phase locked loop (PLL) [7]. Phase locked loop is used for clock generation and has a key component: the voltage controlled oscillator (VCO). By having additional VCOs multiple data rates can be used within a single HSSI device. When the same PLL is used to provide multiple data rates, jitter performance at different speeds can vary. Jitter performance of the voltage controlled oscillator (VCO) of each PLL can only be optimized at one specific speed; at any speed other than the optimal one, jitter performance may decrease [7][4]. If multiple data rates within a single HSSI device are implemented by using different PLLs, the performance at one data rate does not relate to performance at another data rate [4]. Usually at high data rates jitter performance is better because the PLL has more dividers; the modulation clock frequency is higher [8].

The proposed SerDes test solution in [8] makes use of oversampling to cover a wide range of data rates and at the same time achieve a good performance in terms of jitter. As the PLL operating frequency range cannot cover the low data rates, the data is oversampled by a suitable factor, such that the sampling frequency lies within the acceptable range of the PLL. It should be noted that the factor by which the signal is oversampled affects the performance in terms of jitter. In fact, the sampling frequency of the oversampled signal, in spite of being achievable by the PLL, might become marginal with respect to the PLL operating frequency range. This could impact the performance, since more jitter exists closer to the ends of the range of the PLL operating frequency.

Table 3-1 below shows different data rate ranges with respect to their corresponding oversampling-ratio.

Target Data Rate	Oversampling-Ratio	Mode
6400 Mbps -3200 Mbps	1	1X
3199 Mbps - 1600 Mbps	2	2X
1599 Mbps - 800 Mbps	4	4X
799 Mbps - 400 Mbps	8	8X
399 Mbps- 250 Mbps	16	16X

Table 3-1 : Target Data Rate Vs. Oversampling-Ratio

What table 3-1 above indicates is that for the data rates in the range 6400-3200 Mbps no oversampling is done. For 3199-1600 Mbps, the oversampling factor is two. Likewise, data at rates within the range of 1599-800 Mbps are oversampled by a factor of four. Data in the range 799-400 Mbps range are oversampled by a

factor of eight, and data rates from 399 Mbps to 250 Mbps are oversampled by a factor of sixteen. As a result, lower data rates are also supported.

Oversampling is implemented by having the PLL output two clocks: one *pattern-clock* (the fast clock), and one *oversampling-pattern-clock* (the slow clock). Depending on the data rate, the slow clock is the fast clock divided by a factor of 2 (one, two, four, eight or sixteen).

$$ovrsmpl\_pat\_clk = \frac{pat\_clk}{ovrsmpl\_ratio}$$

3.2 Top Architecture (Tx-Pattern & Rx-Pattern)

The block diagram in figure 3-1 shows the top level architecture of the Tx and Rx-patterns. Rx-pattern which is actually the BERT receiver is the main block of concern, where the sampled bit stream is received from the SerDes and is compared to the reference pattern from *pattern-generator* or *pattern-memory*. We provide greater detail of each block in the following sections of this chapter.

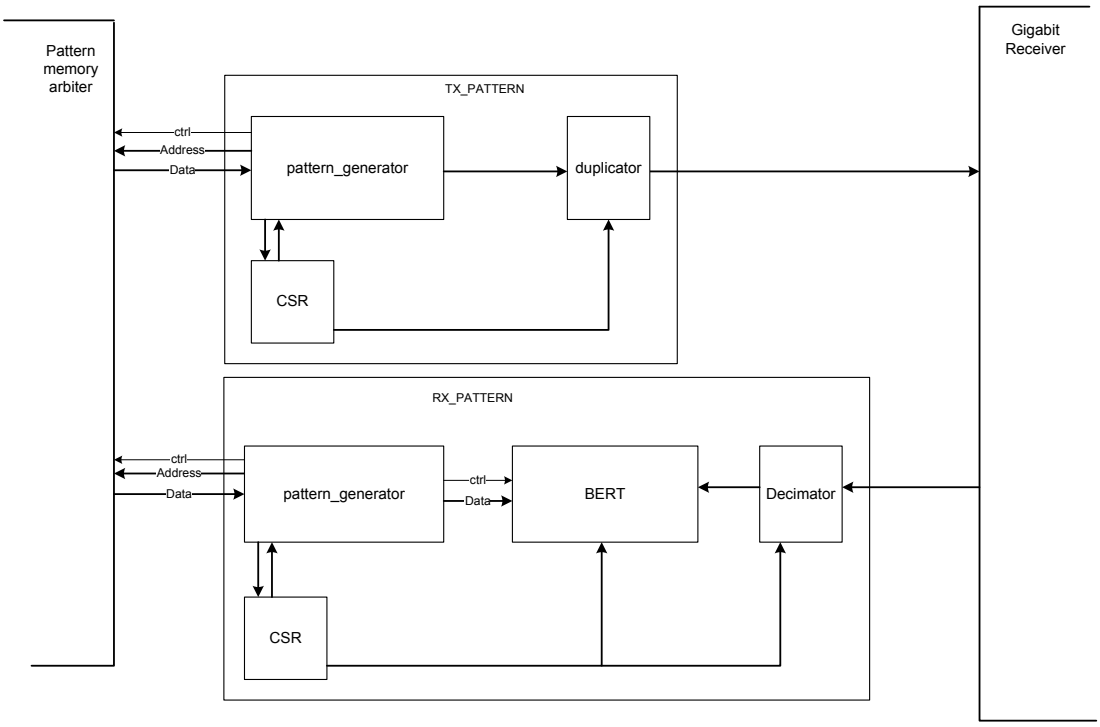

Figure 3-1 : Top Architecture (Tx-Pattern, Rx-Pattern)

3.3 **Pattern Transmitter**

According to what was mentioned in section 3.1, the gigabit transceiver samples at the highest speed range the HSSI supports. The *pattern-transmitter* block works by having its *patter-generator* receive a bit stream which it sends to the serializer (Ser) of the gigabit transmitter.

Bit streams coming out of the *pattern-transmitter* should be at the rate of the *pattern-clock* (fast clock) before entering the transmitter. However, the *pattern-generator* generates bit streams at the actual target data rate which is not the optimal rate except in the range of the oversampling-ratio of one.

Obviously, within the 6400 Mbps -3200 Mbps range no oversampling is done. So, the *oversampling-pattern-clock*, the *pattern-clock and* the clock of the gigabit transmitter are all the same. However, this is not true when the over-sampling is not of one. In this case, the data generated by the *pattern-generator* is based on the actual target data rate (slow clock) and is hence slower than the data rate at which the transmitter is tuned to receive from the *pattern-transmitter*. Data should be on the fast clock before leaving the *pattern-transmitter* block. This should be done in a reasonable way so that the data would not be altered due to speed differences and can be recovered correctly in the receiver.

We therefore need a clock-domain crossing block in between. This block is slowly written to (on the slow clock) and is read quickly by means of the fast clock (which is the *pattern-clock*). To achieve this, a *duplicator* is placed right after the *pattern-generator*.

3.3.1 The Duplicator

Figure 3-2 shows the timing diagram of the *duplicator*. *pat_clk* is the fast clock and *ovrsmpl_pat_clk* is the slow clock which here is equal to $\frac{pat\_clk}{2}$ since the oversampling-ratio is assumed to be two.

Starting from the most significant bit (MSB) the *duplicator* duplicates each bit an amount of time equivalent to the oversampling-ratio. For instance, in 2X mode the duplicator reads two times faster than the writing speed. This means it can start reading as soon as it has only received half of the data.

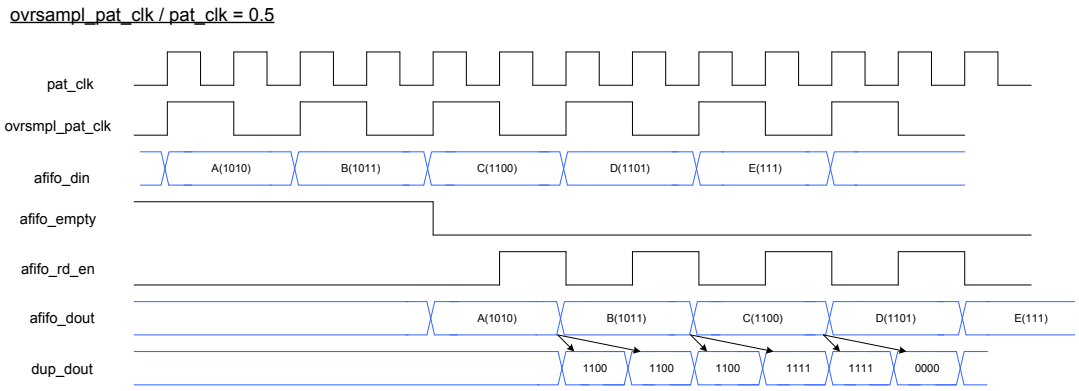

Figure 3-2 : Duplicator timing diagram (oversampling-ratio = 2)

When the data rate is in the range corresponding to the oversampling-ratio of 16 (399 Mbps - 250 Mbps), *ovrsmpl_pat_clk* is sixteen times slower than the pattern-clock *(pat_clk)*. Starting from the MSB each bit is duplicated sixteen times; in a 32-bit bus, duplicator will start reading after it has received only 2 of bits of data. Here, reading occurs 16 times faster than writing.

Thus, by adding a *duplicator* in between the *pattern-generator* and the gigabit transmitter, data is always sent from the *pattern_transmitter* to the receiver at the correct rate.

3.4 **Pattern Receiver**

The same reasoning is valid on the gigabit receiver side. The gigabit transceiver runs and samples data at the highest clock rate possible. Having a look at Figure 3-1 once more, one can see that when sampled data enters the *pattern_receiver* there exists a similar, but reversed, issue as with the *pattern_transmitter* and receiver.

Sampled data comes to the patter-receiver from the gigabit transceiver and is therefore on the fast clock (*pattern-clock*). On the other hand, the bit streams generated by the *pattern-generator* are on the *oversampled-pattern-clock*. Because the *pattern-generator* is outputting bit streams consistent with the real data rate, it is in fact running on the slow clock (*ovrsmpl_pat_clk).*

As stated before, when the data rate is in any range other than the range where the oversampling-ratio is one (Table 3-1), *ovrsmpl_pat_clk* is oversampling-ratio times slower than the pattern clock (*pat_clk).*

The *pattern-receiver* receives more samples from the gigabit receiver than the number of reference patterns generated by the *pattern-generator*. The Bit Error Rate Tester (*BERT*), which represents the main purpose of the design, will be comparing bit streams generated by the pattern-generator with bit streams received from the gigabit transceiver. In case of an oversampling regime, the rate at which the *BERT* is receiving samples from the *pattern-generator* is less than the rate at which it is receiving bits from gigabit transceiver. The *BERT* will not be able to accomplish a reasonable comparison between the two bit streams; it is not be possible to compare two different bit streams whose number of bits per unit of time (rate) is not the same. This is the motivation for adding a block called *decimator* between the gigabit receiver and the *BERT*.

3.4.1 The Decimator

Figure 3-3 shows how the *decimator* works. The *decimator* is needed to reverse what the *duplicator* does. The *decimator* slows down the speed at which sampled data from the gigabit transceiver is going to the BER tester; hence, it makes it comparable to the expected bit stream coming out of the *pattern-generator* or the pattern data memory.

As shown in Figure 3-3, data is written in the *decimator* at the speed of *pat_clk and the decimator* reads data at the speed of *ovrsmpl_pat_clk.* This second clock is slower than the first except when the sampling ratio is one. This explains why some of samples are dropped and data received from the gigabit transmitter gets matched with the expected pattern. This is fundamental for comparison of these two streams of bits.

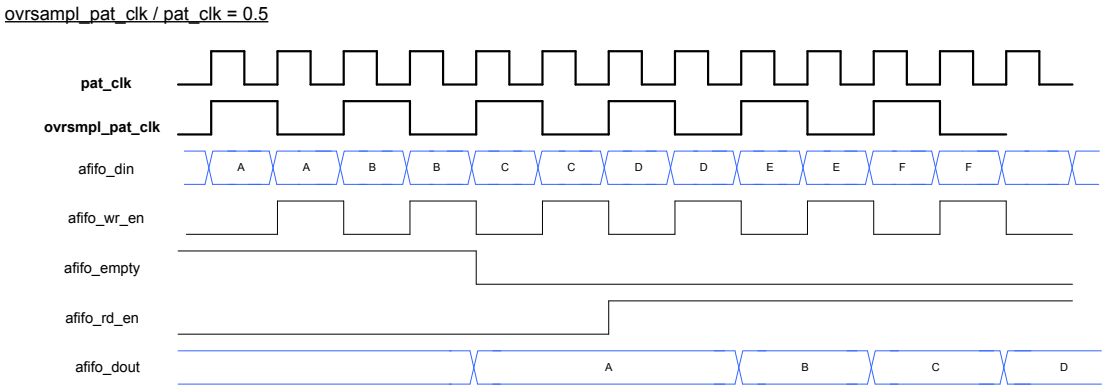

Figure 3-3 : Decimator timing Diagram (oversampling-ratio= 2)

For instance, with an oversampling-ratio of two, *ovrsmpl_pat_clk* is two times slower than the *pat_clk*. Thus, every other sample is dropped; if 32-bits of data are

received, every other sample will be dropped and only sixteen bits will remain. These sixteen bits are combined with the sixteen bits of the next 32-bits of data. Indeed, out of every two 32-bit samples received, the *decimator* reads only one 32-bit sample. Accordingly, it will drop the samples which were oversampled by the gigabit transceiver and it will pass samples to the BERT at the pace of the pattern clock.

When the oversampling-ratio is four (data rates in the range of 1599 Mbps - 800 Mbps), three out of four samples are dropped. This means that for every 32-bits of data, eight bits are kept. This is done on four 32-bit sets of data in a row to create a new 32-bit word. The time it takes *decimator* to read this is four times more the period of the pattern-clock.

3.4.2 BERT Engine

The main functionality of the Bit Error Rate Tester (BERT) is to compare a stream of bits received from the gigabit receiver against an expected bit pattern stored in a memory. The *BERT* reports the number of mismatches found as the result of the comparison.

Before the *BERT* starts its comparison process, the two bit streams must be aligned. To do so, the *BERT* performs a synchronization process in order to align the bit pattern coming from the Gigabit receiver to the bit pattern coming from an external pattern memory. A bit alignment process is required because the gigabit receiver deserializes the incoming bits with an arbitrary bit alignment.

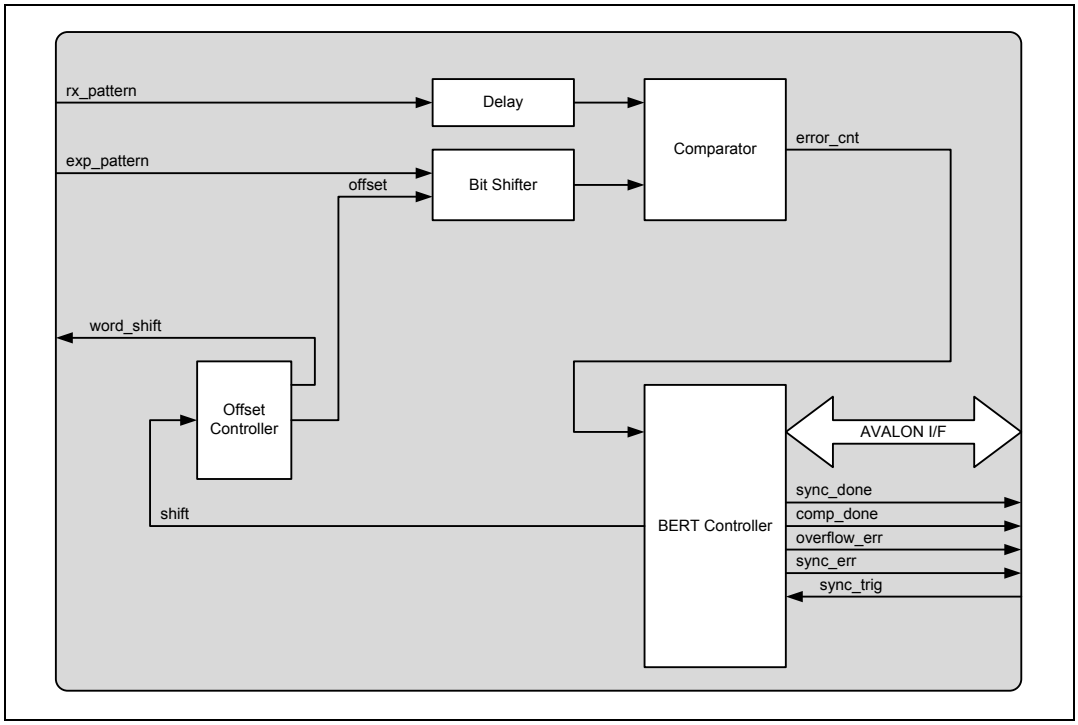

Figure 3-4 : High level block diagram of the BERT.

Synchronization is performed in an iterative process. If the number of mismatches between the two bit streams is greater than a certain threshold value (for instance 3), then the reference pattern coming from the pattern-memory is shifted by one. Once again, the pipelined *comparator* will find the number of bit errors. If the new amount of errors is still more than the threshold bit error value, the reference pattern is shifted by one bit once again. This iterative process continues until the two bit streams are synchronized together which means that the number of mismatch between them is less than the threshold bit error value.

After synchronization is done, the *BERT* starts the comparison process. It compares the two now aligned bit streams in the pipelined *comparator*. The pipelined *comparator* component works as shown in Figure 3-5. It performs a parallel comparison between 2^n bits of the two streams at once, where n is the

width of the reference pattern and the sampled pattern. For comparing each two bits together an XOR gate is used.

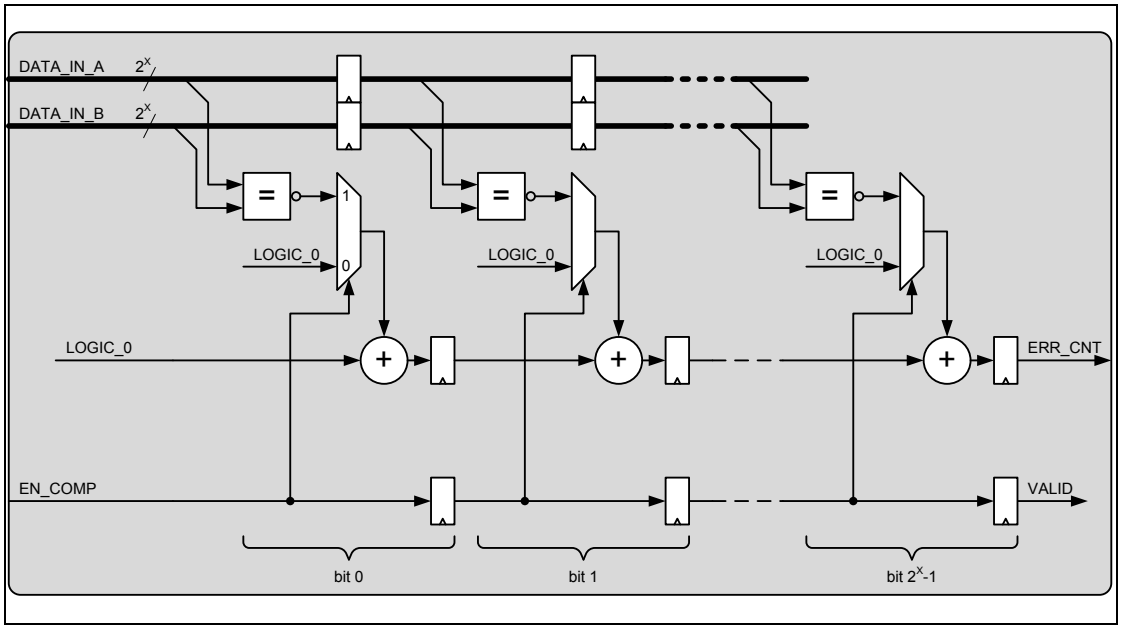

Figure 3-5 : Block diagram of the pipelined *Comparator*

The *BERT Controller* takes care of controlling the bit error counting process. It constantly reads the error count from the pipelined *comparator* adding readings together. The process stops once the number of the bits received from the two streams reaches a user defined amount. The number of recorded errors is then reported. The *BERT Controller* places the number of bit errors on the register interface so that it is software accessible software.

Chapter 4 - **Multi-Phase BERT**

We aim at improving the transmitter bit-error rate (BER) testing speed. This is achieved by having parallel bit-error rate testers (BERT) working in conjunction; each tester is to measure the BER value of a different phase. Depending on the data rate we sample at, we will have 1, 2, 4, 8, or 16 different phases and we will perform 1, 2, 4, 8, or 16 comparisons in parallel. By the end of the comparison process we have the bit-error count of multiple phases at the same time. This indeed accelerates generating BER-scan plots by the corresponding oversampling-ratio of the pattern data rate being tested. Using dual-Dirac extraction model we aim to calculate jitter components 1, 2, 4, 8, or 16 times faster.

This chapter mainly deals with the design of a BERT data receiver that enables reading measurements of multiple BERT engines at the same time. We seek methods to implement multi-phase BERT by adding the least possible circuitry to the existing design of the DJ60HS [8]. We would like to perform multiple comparisons in parallel. Clearly, adding multiple BERT engines enables performing the comparison of multiple phases all at once. This will require adding 15 other BERT engines; one is already present in the existing implementation. However, we are looking for the most optimized implementation in terms of circuit area and testing time. We try to achieve performing multiple and concurrent bit-error measurements without adding any extra BERT engines. We should also consider not to compromise on speed by adding extra delays to the implementation as the aim is to accelerate the bit-error rate testing process.

4.1 **Multi-Phase BERT Design vs. Existing Design**

The gigabit receiver is sampling more often than the actual data rate would imply; hence, SerDes is sending extra samples at phases other than the sampling phase delay. Consequently, the BERT receiver is receiving data at phases that are brought in by oversampling. For instance, if the gigabit transceiver is set to run at 4 Gbps and the target pattern data rate is 1Gbps, we expect to receive samples every 1ns but the gigabit receiver is sampling data every 250ps. This is a case similar to the range that requires an oversampling-ratio of four. The gigabit transmitter is sending information four times faster than the effective data rate; therefore, each time it is sending 3 extra samples to the *pattern-receiver* block.

Figure 4-1 and Figure 4-4 show the phase location of extra samples. These are brought in by the oversampling feature in different modes as the receiver phase delay is swept. Dashed lines are the equidistant extra samples we get at other phases. Solid lines show the actual phase we set the receiver phase delay at. As we shift the Rx-phase delay a new round of samples is obtained (shown by purple arrows).

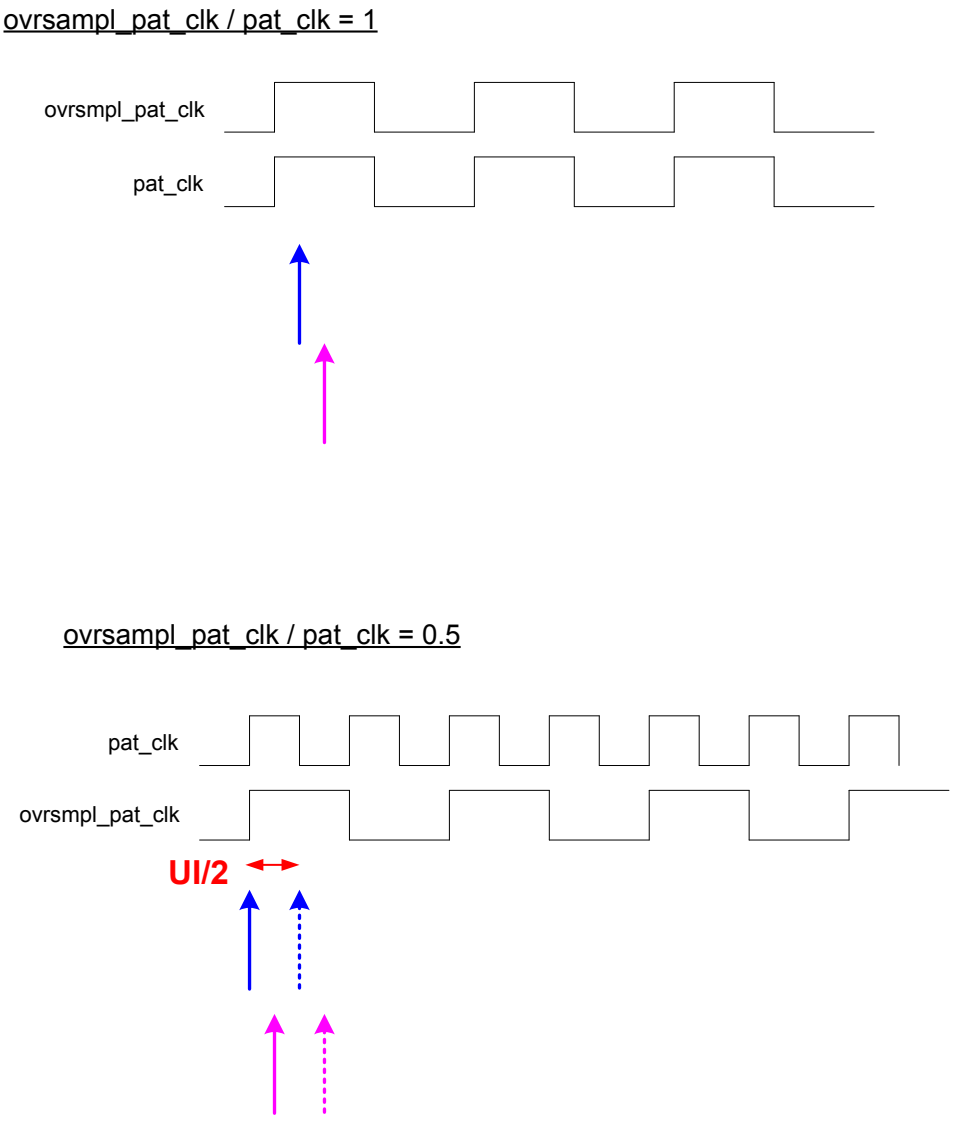

Figure 4-1 : Phase location of extra samples obtained due to oversampling (oversampling-ratio=1, oversampling-ratio=2)

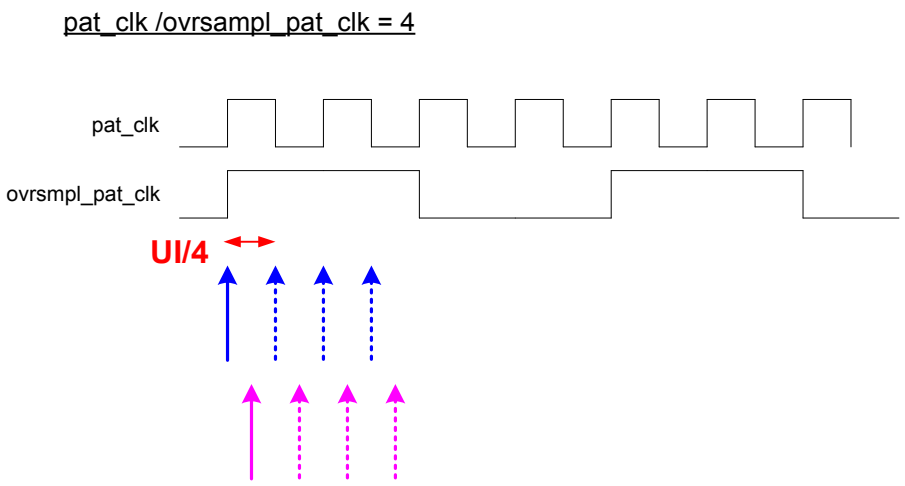

Figure 4-2 : Phase location of extra samples obtained due to oversampling (oversampling-ratio=4)

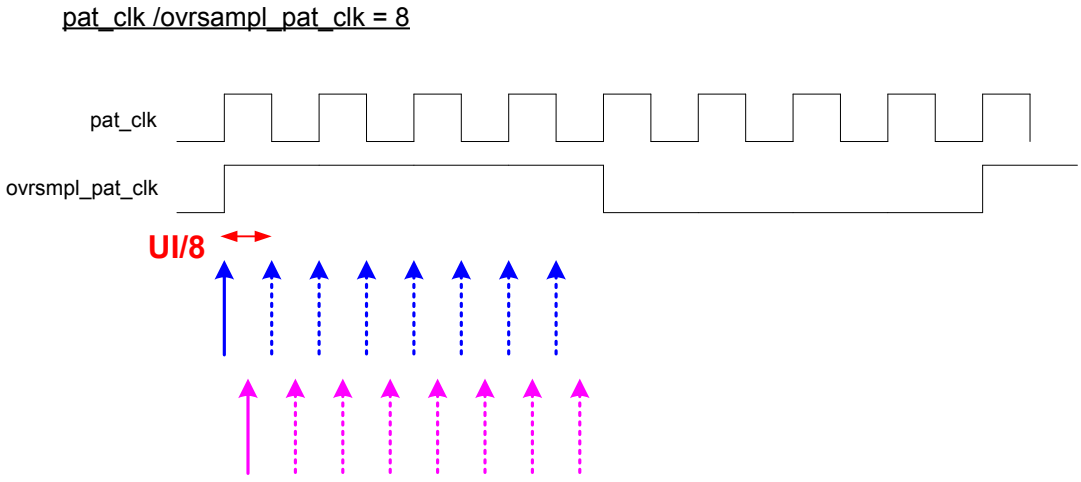

Figure 4-3 : Phase location of extra samples obtained due to oversampling (oversampling-ratio=8)

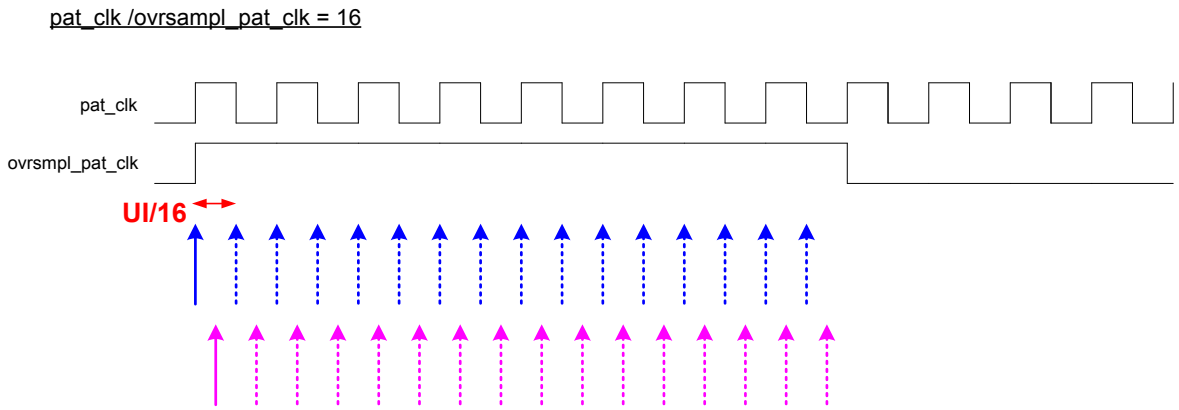

Figure 4-4 : Phase location of extra samples obtained due to oversampling (oversampling-ratio=16)

The *decimator* component in the bock level diagram of Figure 3-1 drops these extra samples found on the incoming bit stream. This is done to make the stream match the reference pattern.

Thanks to the oversampling feature of the design, we are able to digitize in a multi-phase manner. This translates into putting extra samples received from the gigabit transceiver to good use. This is indeed the main motivation for multi-phase bit-error rate testing (BERT).

It is possible to keep the extra samples and, instead of dropping them, use them to enable parallel BERT elements. The BERT must be upgraded to be able to perform multiple bit-error measurements at the same time. If the oversampling-ratio equals one, there is only one bit-error counter. In 2X mode, where the oversampling-ratio is two, there are two bit-error counters for two different phases. Similarly, in 4X, 8X, and 16X modes there are respectively 4, 8, and 16

bit-error counters, each measuring bit-error at different phases. Information from all bit-error counters can be read and stored at the same time.

In order to build bathtub curves, the sampling delay should be swept over 2UI with fine time resolution. Using the multi-phase bit-error counting technique, a bathtub can be generated by sweeping the sampling time delay over just a fraction of the unit interval (UI) and this is the main factor contributing to speed improvement with MPB. Regarding the required sweeping interval to obtain bathtubs, Table 4-1 gives a comparison of the former method with different ranges of the multi-phase method.

Mode of Operation	Former Method	MPB
1X	2UI	2UI
2X	2UI	UI
4X	2UI	UI/2
8X	2UI	UI/4
16X	2UI	UI/8

Table 4-1 : Required Shift Interval for a full BER-scan Test with MPB and Former Method

As Table 4-1 suggests, in the case of a target data rate within the range of an oversampling- ratio of one, the receiver phase delay should be swept over two unit intervals. On the other hand, we have an improvement with all other target data rates. Sweeping the sampling delay with a fraction of the UI is sufficient to obtain a bathtub BER-scan plot. For instance, in 2X mode, each time we are getting an extra sample at another phase which is at a $\frac{UI}{2}$ distance from the original sample. If the receiver phase delay is only swept over half of the unit interval (UI), ($\frac{UI}{2}$), the other half will have already been covered by extra samples from

the first pair. Hence there is no need to sweep the phase delay over the second half of the UI as we have already obtain samples at those other phases. Also, in order to have samples at the second unit interval (UI), the Rx-phase can only be delayed by half of a UI. Hence, a total delay of one unit interval (UI) is enough to cover the entire 2UI.

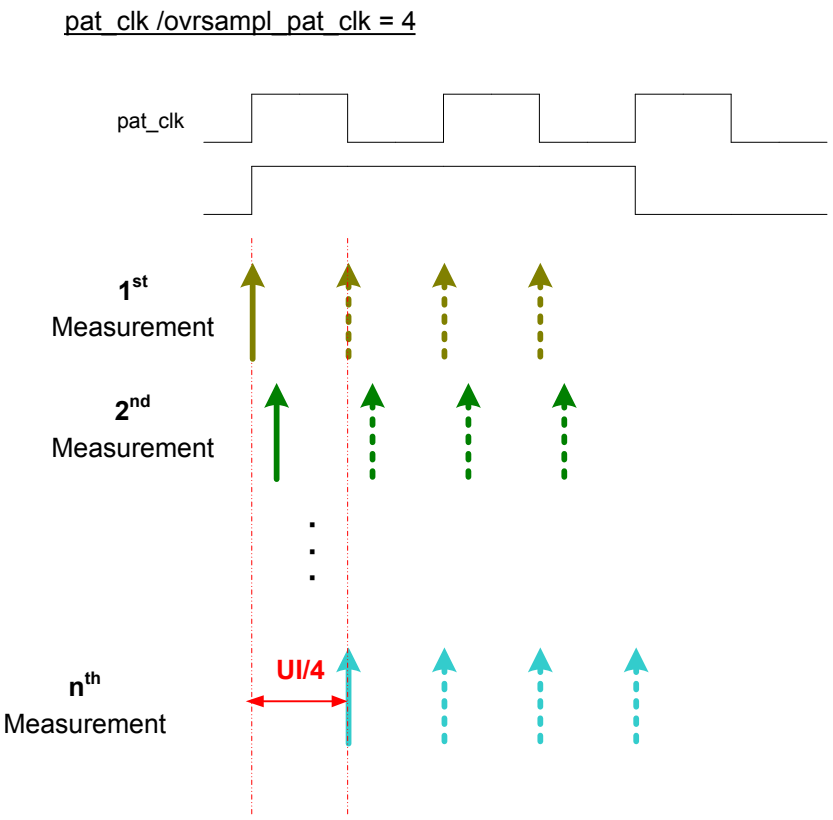

Figure 4-5 : Required interval Rx-phase should be swept over to cover one entire UI (oversampling-ratio=4)

Similarly, in 4X mode, covering a quarter of a (UI) will give samples covering the whole unit interval (UI). This time, a total delay of two quarters of a UI is sufficient in order to plot bathtubs.

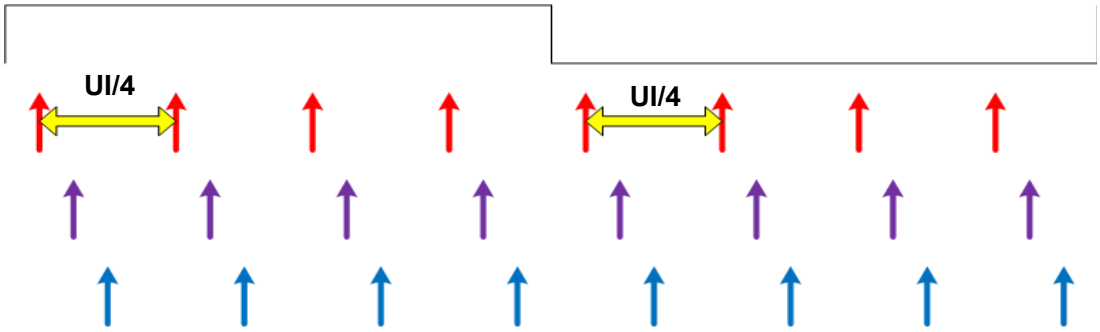

Figure 4-6 : Sampling phase interval MPB should be swept over for a full BER-scan test (oversampling-ratio=4)

The relative reduction of sweeping interval increases as the pattern data rate gets lower. In 8X mode a total delay of UI/4 is required; in 16X mode a total delay of UI/8 is required.

The Multi-phase BERT technique also reduces the overhead of communication between hardware and software. The elimination of significant overhead when testing multiple phases is another source of speed improvement in the multi-phase bit-error rate (BER) testing technique. There are at least three software commands required (though SPI) to get the bit- error count at one certain sampling phase. One command allows setting the sampling phase delay of a certain phase, one to start the bit error measurement, and one to obtain the bit-error count from the hardware. Each software (SPI) command takes much longer than hardware processing. Using the former method, three SPI commands are required for any single measured phase of the bathtub. With the new multi-phase

technique, this overhead of communication is reduced. As discussed before, there is no need to set the sampling delay at any single phase to get the bit-error count of a specific phase. We now need to set the phase for only a fraction of a UI and for the rest of the UI. The oversampling feature provides the rest of the samples at other phases of the UI. The BERT performs bit-error calculation of all the phases it is receiving at the same time and the results are ready for the software at once. Three SPI commands are hence required for a smaller amount of phases and using a fraction of UI instead of 2UI (see table 1). For instance, in 16X mode a full bathtub (along 2UI) needs us to pass the SPI the following commands: set up the receiver phase, carry out BERT and store the results of only 1/16 of all required phases. This, in fact, represents a speed improvement of 15/16*(delay of three SPI commands) seconds faster.

Another contributing factor to the acceleration of BER testing with the use of multi-phase BER is indeed performing the bit-error calculation of multiple phases at the same time. Due to the concurrent feature of the design, bit-error measurements of all phases received is done in parallel. Within the time the former design used to measure the BER of one phase, we can now measure the BER of multiple phases. In 1X, 2X, 4X, 8X, or 16X modes there are respectively 1, 2, 4, 8, or 16 error counters performing bit-error measurements simultaneously. This is achieved without adding multiple *BERT* blocks to the implementation which would requires a lot of circuitry. With minimum added cost we are going to achieve the measurement of multiple phases at once.

In addition, the *BERT* used to run on *oversampling-pattern-clock*, the slow clock. Now it will run on the fast clock and perform multiple bit-error measurements in parallel and in less time.

4.2 **Multi-Phase BERT Implementation**

The goal is to implement the multi-phase bit-error tester while modifying the existing BERT implementation as little as possible.

Figure 4-7 illustrates the *pattern-receiver* block. Based on what was said before, changes will have to be applied only on blocks that rely on data received from the gigabit receiver. Hence, our focus is on the *pattern-receiver* and the blocks in its neighborhood.

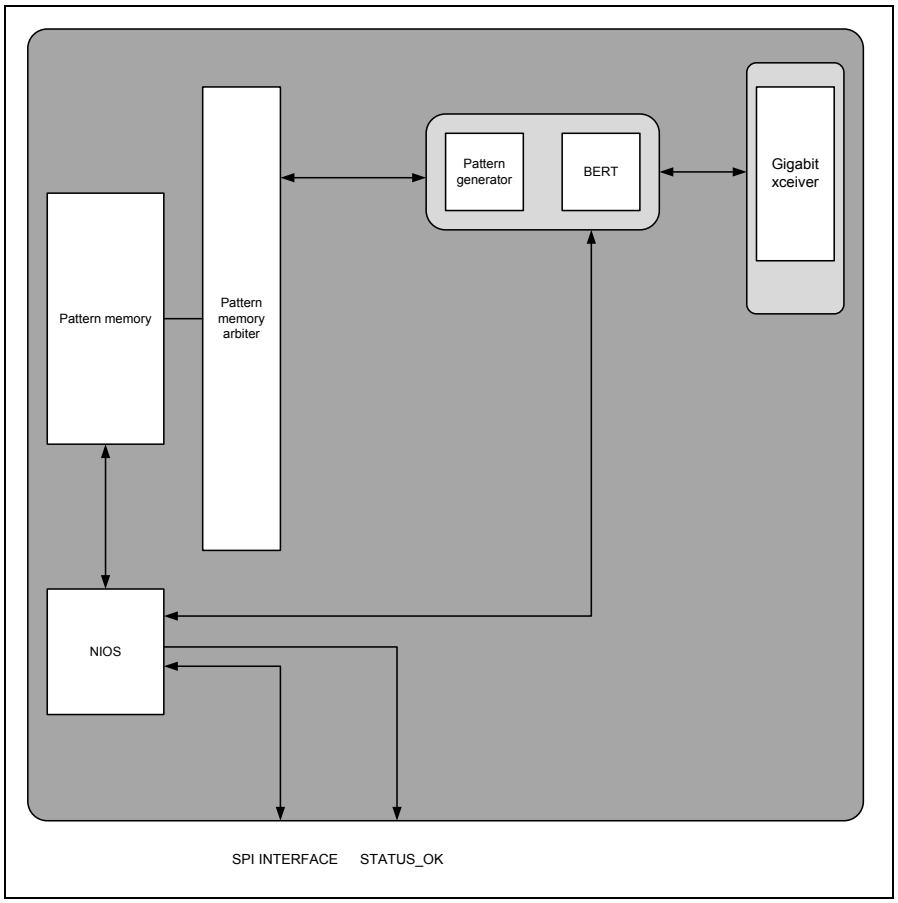

Figure 4-7 : Block level diagram of pattern-receiver

The objective is to also implement the enhancement with the least possible cost and circuitry added.

4.3 **MPB Pattern-Receiver**

Since we are going to perform a transmitter test, our main block of concern is Rx-pattern (BERT receiver). Here, we receive the sampled data from the gigabit receiver and evaluate the number of bit-errors.

Two important blocks inside the *pattern-receiver* are the *pattern-generator* and the *BERT*. The latter performs the comparison between the expected pattern coming from the *pattern-generator* (or *pattern-memory*) and the pattern received from the gigabit transmitter. Figure 3-1 shows the *pattern-receiver* in greater detail.

As discussed in the chapter 3, the *decimator* runs on the fast clock (*pattern-clk*) and it drops the extra samples received from the gigabit receiver. The *BERT* is running on the slow clock (oversampling-pattern-clock) since it is comparing two bit streams which both are both on the slow clock. Obviously, the *pattern-generator*, *pattern memory arbiter*, and *pattern memory* are all running on the *oversampling-pattern-clock* since the reference pattern should be generated at the actual pattern data rate.

However, with the multi-phase BERT, this scenario is going to change. We would like to keep all the samples received from the gigabit receiver, even the extra samples. The extra sample represent samples at other phases which we are going to use in order to speed up the bit-error rate (BER) testing process. Indeed, sampled data always enters the *pattern-receiver* on the fast clock (*pattern-clock*) and we do not want to drop the extra samples now. Hence, there is no need to keep the *decimator* at the interface of the gigabit transceiver and the *pattern-receiver*.

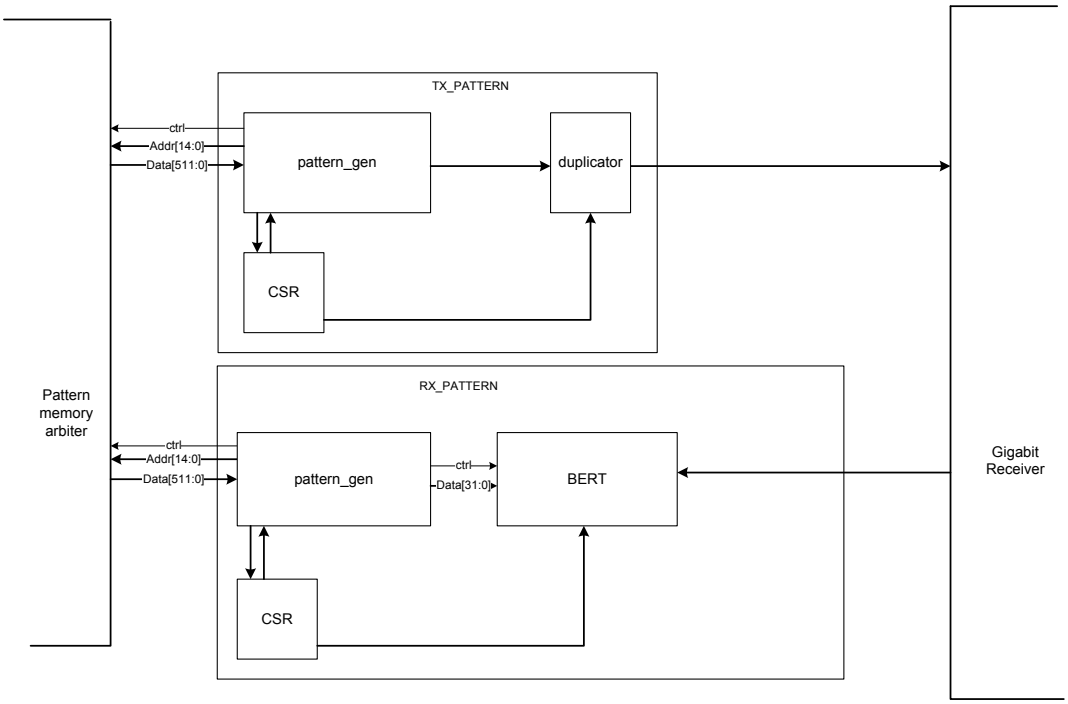

Figure 4-8 : Removing the *decimator* from pattern-receiver

Simply removing the *decimator* block from the *pattern-receiver* does not take complete care of the intended upgrade. Removing the *decimator* lets the *BERT* run on fast clock; then *pattern-generator* and *pattern memory arbiter* should also run on the fast clock. This will require modifying the *pattern-transmitter* as well, since it is using the same *pattern memory arbiter* as the *pattern-receiver* in Figure 3-1. Also, we should look for a way to make the *pattern-generator* generate reference patterns faster than the actual pattern data rate. However, the issue is not concerned with having the two bit streams at the same rate. Although we are receiving oversampling-ratio times more samples from the gigabit receiver, these samples are not measured at consecutive phases. In order to perform a valid comparison between the two bit streams, the reference bits should be arranged in the same special order at which sampled data is received. For instance, in 4X mode, *pattern-generator* should send the fifth reference bit after sending the first

reference bit. It should then send the 9th, 11th and 2nd bits. After the second reference bit, it should send the 6th, 10th, and then the 14th bits.

The *pattern-generator* frequency is: $\frac{f_{ref}}{32*oversampling-ratio}$ and is sending 32-bit reference data to the BERT. When switching the *pattern-generator* to the fast clock ($\frac{f}{32}$), is it necessary to ensured that the whole 32-bit word is concurrently latched at the *BERT*. When putting the *pattern-generator* clock on the fast clock, metastability issues may come up. These issues have to be considered when driving clock enable signals to control when the transaction starts and ends.

So far, the only obvious change needed is the removal of the *decimator*. For the rest of the implementation, because of the above explained reasons, we prefer not to make all the other blocks run on the fast clock. We are looking for the solution which requires the least possible amount of change to the existing implementation while minimizing the additional hardware cost. This is because we are going to evaluate our MPB by comparing its results with the results of the former implementation.

If we add a FIFO in between the *pattern-generator* and the *BERT*, in which data is written slowly and is read fast out of it, helps fixing the issues. Consider the case where oversampling-ratio is 4. Fast clock (*pattern-clock*) is 4 times faster than the slow clock (*oversampling-pattern-clock*). We want the 32-bits expected pattern to be written in the FIFO on the slow clock cycle, then FIFO reads bits 0-7 on first fast clock cycle, 8-15 on second fast clock cycle, 16-23 on third, and 24-31 on

forth fast clock cycle, illustrated in Figure 4-9. Just, we need to make each transaction in 32-bits busses; at each fast clock cycle we need to send 32-bits reference data to the BERT. We need to add reference data in between to make for the extra samples *BERT* is receiving from the gigabit receiver.

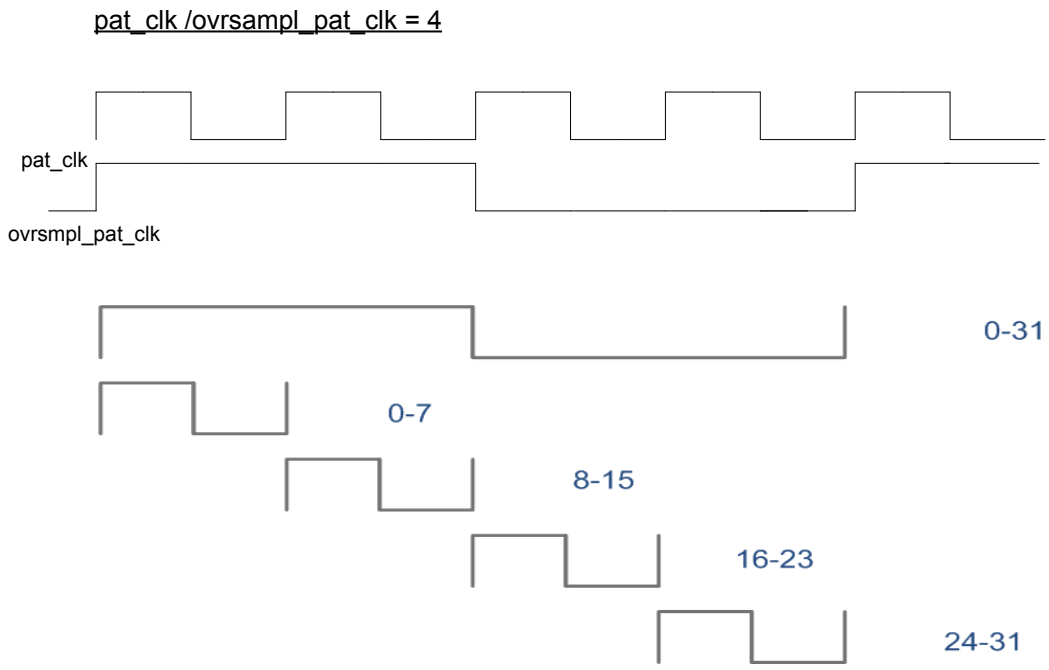

Figure 4-9 : Sending data from *pattern-generator* on fast clock rather than slow clock

This is very similar to what *duplicator* does in the *pattern-transmitter*. Bits 0-7 can be duplicated each four times to make the first 32-bit data. Similarly, duplicating each bit of the bits 8-15, 16-23, 24-31 four times will give the second, third, and fourth 32-bits expected data to the BERT.

One important advantage of putting a duplicator before the *BERT* is that it will make the expected data matched with the sampled data received from the gigabit receiver. For instance,

Figure 4-10 shows what is received for a data rate in the range of oversampling-ratio of 4. Each four bits coming in a row are not in fact measured at one particular phase; they belong to four different phases, each with a distance of $\frac{Unit\ Interval\ (UI)}{4}$ from the next one. Similarly, the second bits of the sampled data of each of the four phases come one after each other, after all the first bits are received.

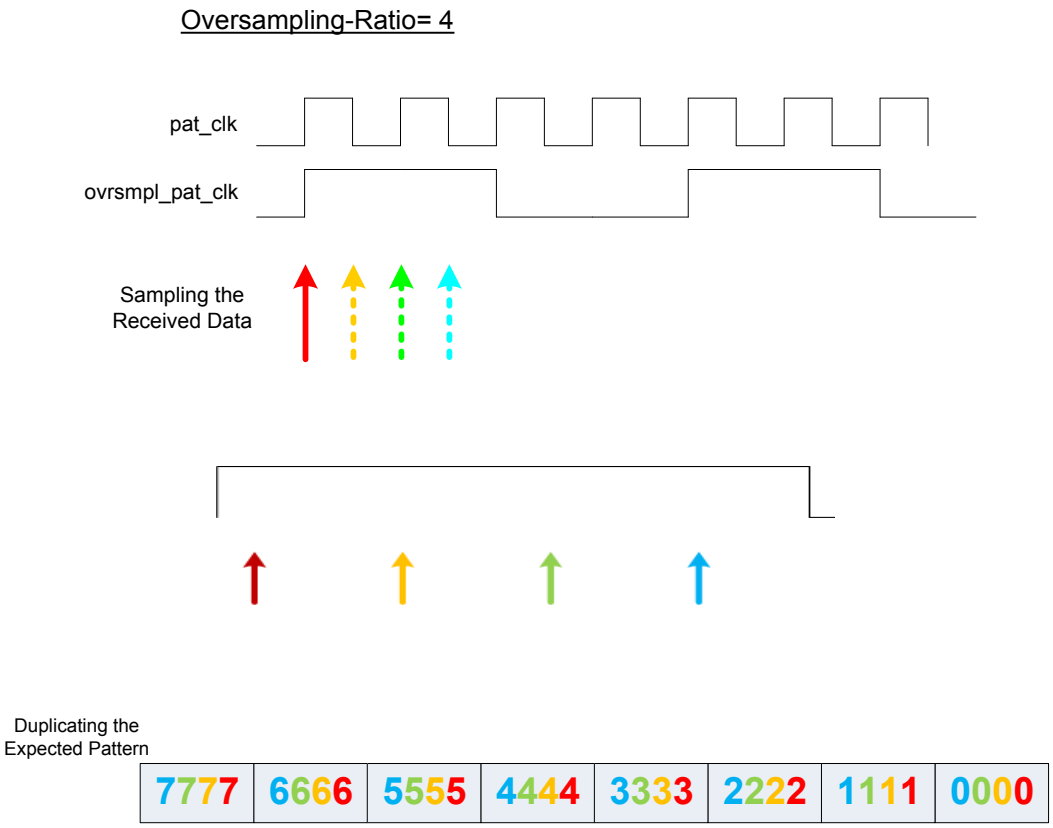

Figure 4-10 : Received sampled data at four different phases sent in a 32-bit bus. Duplication of each reference bit inside the duplicator makes the expected pattern bit streams in the same format as the bit stream coming from the gigabit receiver.

Putting a *duplicator* in between the *pattern-generator* and *BERT* will handle sending reference data from slow clock to fast clock. There are also some control signals from *pattern-generator* to *BERT* or from *BERT* to *pattern-generator*

which require clock domain changes. Signals from the fast clock to the slow clock domain are not recognized in the slow clock domain, since it is only a fraction of a bit in slow clock domain.

4.3.1 Multi-Phase *BERT* Engine

Considering the high-level block diagram of the BERT shown in Figure 3-4, we are looking for a way to implement a new BERT engine, multi-phase BERT engine, which is capable of taking care of all of the tasks currently the BERT engine takes care of, but for multiple phases.

4.3.2 Comparator

Pipeline comparator of the existing BERT can compare the two 32-bit bit streams together at once (See Figure 3-5).

In multi-phase BERT each 32-bits of expected pattern (out of the *duplicator*) and sampled data are like below (in case of oversampling-ratio of 4):

7777	6666	5555	4444	3333	2222	1111	0000

In 4X mode, bits 0, 4, 8, 12, 16, 20, 24, 28 of the two bit streams should be compared together since they are related to the first phase. Bits 1, 5, 13, 17, 21, 25, 29 are for the second Rx-phase. Likewise the rest of the bits are the data of the two other Rx-phases (See Table 4-2).

Since the *decimator* has been removed, any 32-bit contains the data for 4 different phases, which is 8 bits for each phase. However, 8 bits of each phase are not received in row. The second bit of each phase is received only after the first bit of all of the phases of the oversampling-ratio are received. For the comparison process, there should be multiple error counters which each track the number of mismatch of only one of the phases.

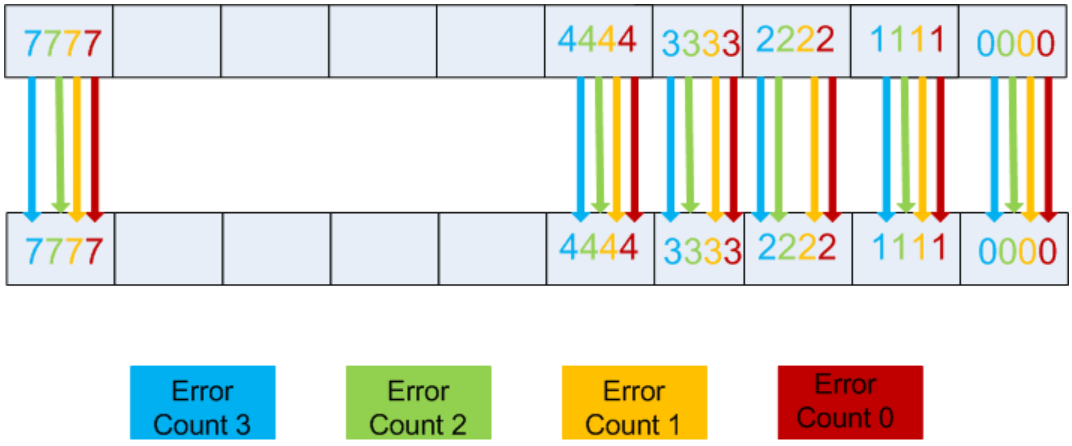

Figure 4-11 : Shows that as explained before, both the expected bit stream and the sampled bit stream coming from the gigabit receiver are in the same format and 4 error counters are each evaluating bit-error value of 4 different phases by comparing the bits of the same color.

At each mode, the number of different phases is equal to the oversampling-ratio of that range. The number of error counters needed is also equal to the corresponding oversampling-ratio. For this, we need at least four comparators if the oversampling ratio is four, eight if the oversampling ratio is eight, sixteen comparators for oversampling ratio of sixteen, etc. The maximum number of error counters required is 16, since there are 16 error counters needed in 16X mode, each counting the number of bit-errors of one of the sixteen phases.

In 2X mode we have two error-counters, one compares even bits, one compares odd bits. Even bits belong to the first phase, and odd bits belong to the second phase.

Table below shows the number of error counters and bit indices that should be compared together at each oversampling-ratio range.

Oversamling Ratio	1X	2X	4X	8X	16X
Number of Error Counters	1	2	4	8	16
Bit indices of the 1st phase	0-31	0,2,4,6,...,30	0,4,8,...,28	0,8,16,24	0,16
Bit indices of the 2nd phase	None	1,3,5,7,...,31	1,5,9,..,29	1,9,17,25	1,17
Bit indices of the 3rd phase	None	None	2,6,10,...,30	2,10,18,26	2,18
Bit indices of the 4th phase	None	None	3,7,11,...,31	3,11,19,27	3,19
Bit indices of the 5th phase	None	None	None	4,12,20,28	4,20
Bit indices of the 6th phase	None	None	None	5,13,21,29	5,21
Bit indices of the 7th phase	None	None	None	6,14,22,30	6,22
Bit indices of the 8th phase	None	None	None	7,15,23,31	7,23
Bit indices of the 9th phase	None	None	None	None	8,24
Bit indices of the 10th phase	None	None	None	None	9,25
Bit indices of the 11th phase	None	None	None	None	10,26
Bit indices of the 12th phase	None	None	None	None	11,27
Bit indices of the 13th phase	None	None	None	None	12,28
Bit indices of the 14th phase	None	None	None	None	13,29
Bit indices of the 15th phase	None	None	None	None	14,30
Bit indices of the 16th phase	None	None	None	None	15,31

Table 4-2: Error counters vs. oversampling-ratio

We implement multiple pipeline parallel comparators each counting bit-errors of one individual phase. We define a control input signal for each of the comparators. The control input determines which bits of the bit streams should be

compared together. It basically masks the bits which should not be considered in that comparator. Thus, we will have 16 pipeline comparators each as below.

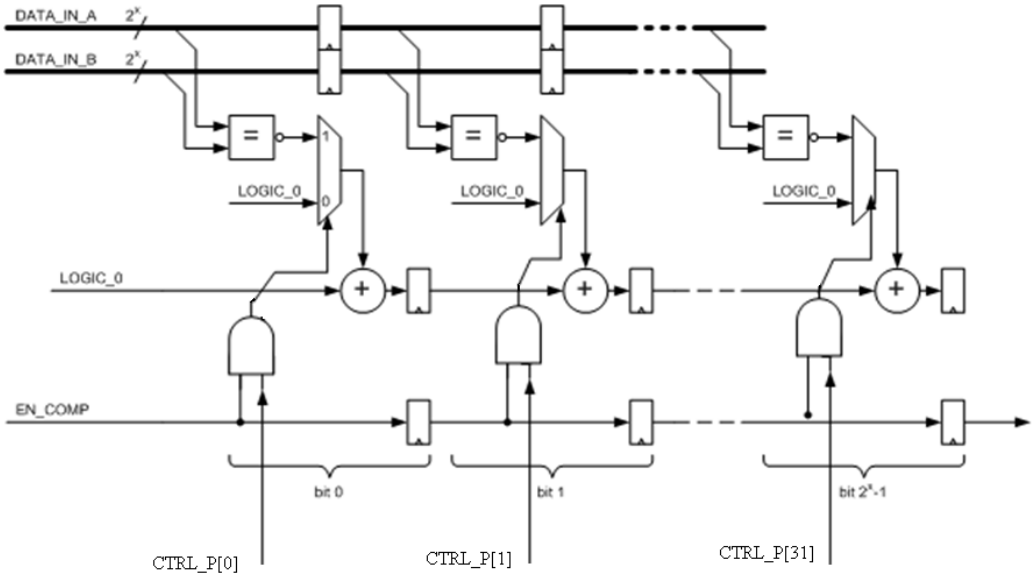

Figure 4-12 : MPB pipeline comparator

CTRL_P is to select which bits to mask and which ones to compare. The mask is defined differently depending on the oversampling-ratio and index of the current comparator (from 0 to 15). Table 4-2 shows which bits should be compared for each counter at a certain oversampling-ratio.

For instance, when oversampling-ratio is one, the control input of the first error-counter is all ones to enable comparison of all of the bits. The rest of the counters are not needed in 1X mode, so all the bits should be masked out.

When the oversampling-ratio is two, the control input of the first counter is "0101'0101'0101'0101'0101'0101'0101'0101" to specify bits 0, 2, 4, 6, 8, 10, 12, 14, 16, …, 30 bits.

Figure 4-13 shows the code for the control input of some of the comparators. The code is based on Table 4-2.

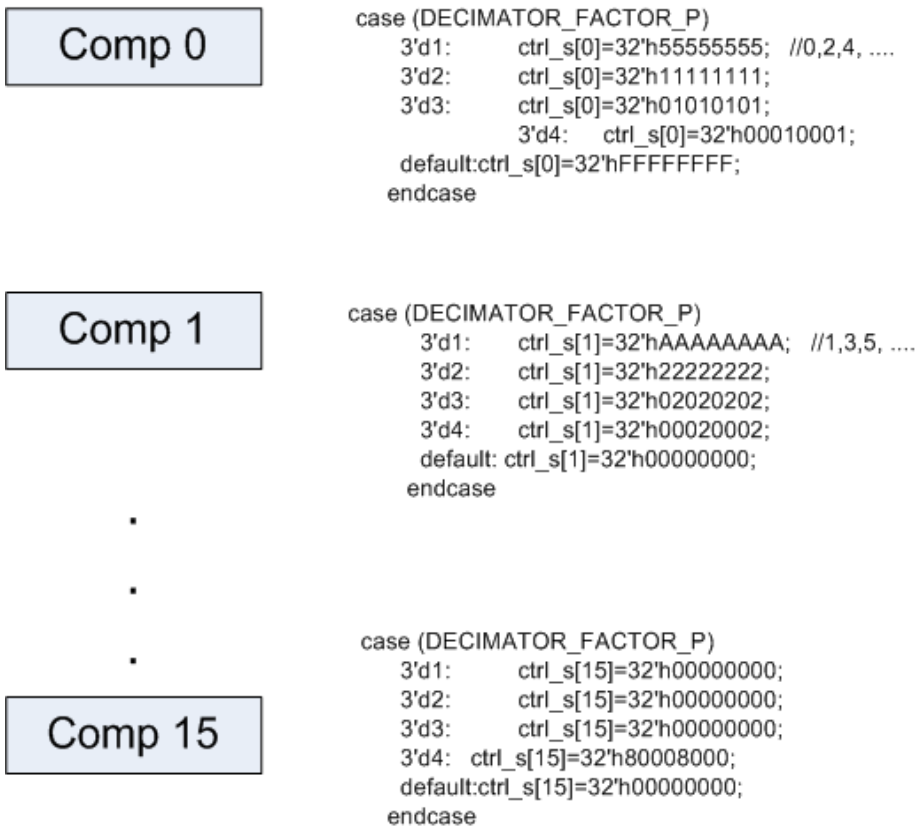

Figure 4-13 : Sixteen pipeline comparators in parallel with each other for MPB

DECIMATOR-FACTOR-P is indeed the oversampling-ratio. The control input of each comparator is calculated depending on the corresponding oversampling-

ratio. For example, regarding “comp15”, the control input for all the oversampling-ratios except 16 is all zeros. Because 2X modes needs 2 comparators, 4X mode requires 4, 8X needs 8, and only 16X mode needs 16 comparators since it is reporting mismatch errors of 16 different phases.

4.3.3 Sample Counter

It should be noted that the new parallel comparators are not reporting the number of mismatches in a 32bit comparison; depending on the oversampling-ratio in 16X, 8X, 4X, and 2X, each comparator is respectively measuring the number of mismatches in 2, 4, 8, and 16 bit comparisons. This is especially important since BER testing is normally done involving a specified duration (total number of bits), and somewhere we should count the number of the bits we compare.

The sample counter is used to count the number of samples the *BERT* has compared. As in the existing implementation, the pipeline comparator is used to measure the bit-errors within 32-bit of the input bit streams. Each time the *BERT Control* unit receives a mismatch result from the comparator, it adds 32 to the sample counter. In order to skip having adding more states to the state machine, which is inside the *BERT Control* and is deciding when to terminate the BER testing process, we add a multiplexer for the sample counter. This is simpler and requires less circuitry. Depending on the mode of operation (oversampling-ratio), the sample counter is respectively incrementing 32, 16, 8, 4 and 2 for oversampling-ratios of 1, 2, 4, 8, and 16.

Since all of the error counters are subject to change at the same time, only one sample counter is enough to count the number of bits compared at each moment. All the 16 total error counters, which continually sum up the bit-error count results of each of the pipeline comparators, are latched to the registry once sample counter reaches the user-defined duration of bits.

4.3.4 Synchronization

Bit-error testing process starts with synchronizing the two bit streams together. Synchronization is an iterative process. It performs bit-error measurement between the two streams till the number of mismatch is less than the certain threshold value. If the total number of bit-errors is more than the threshold value, the two bit streams are not aligned and reference pattern is shifted and bit-error is measured again. For this, total error counter is compared with the threshold bit-error value. In multi-phase BERT, there are 16 different total error counters. We only consider the first phase (0^{th} error-counter) for synchronization. We know different phases have fixed known (constant) distance between each other, so if we synchronize the first one all of them will be synchronized.

4.3.5 Register Interface

After the 16 total error-counters are measured for the specified duration, all of them are ready at the same time to be read with software. In order to make the 16 error-counters accessible to the software, they should be put on the register interface of software and hardware.

Compared to the former method that there was only one error-counter, we need 15 more registers. If there are not enough registers available at the register interface; even though, that is the most optimum way in terms of speed, other way out should be sought.

In order to keep the existing format, we send each error-counter out to the software through one door. While trying to put error-counters on the register interface one by one, it should be ensured that hardware does not write them into the register faster than software can read.

At each mode, there are oversampling-ratio error counters. Although, all of them are measured at exactly the same time we cannot pass them to software all at the same time, or even, with a clock cycle difference. First of all, software is slower than hardware, so even if hardware changes the content of the register, software will not recognize it. For that software should be let always checking the content of the register to detect the new value, but in that case software could not do anything else.

Adding a new register which addresses the phase index on the register interface will help solving this issue. Each time the BERT measurement is ready, software starts writing from 0 to (oversampling-ratio minus one) to this register. Anything software writes in the new phase-address register is the index of the error counter in hardware writes the corresponding error counter of that index on the error – counter register. In fact, instead of 16 writes in a row, hardware performs one read and one write 16 times.

4.3.6 Bit Shifter and Pattern-Memory

When the number of bit-errors measured is more than the threshold value a shift is requested from the *BERT Controller* unit to the *Bit Shifter*. The *Bit Shifter* as shown in Figure 4-14, is in fact a 16 line circular buffer where each line holds 32 bits. If the number of shifts requested is less than 32 bits, the shift will be done inside the internal *bit shifter*. When the number of the requested shifts is between 32 to 512, the address of the circular buffer is incremented by an integer value of: $\frac{number\ of\ requested\ shifts}{32}$; the remainder is shifted inside the internal *bit shifter*.

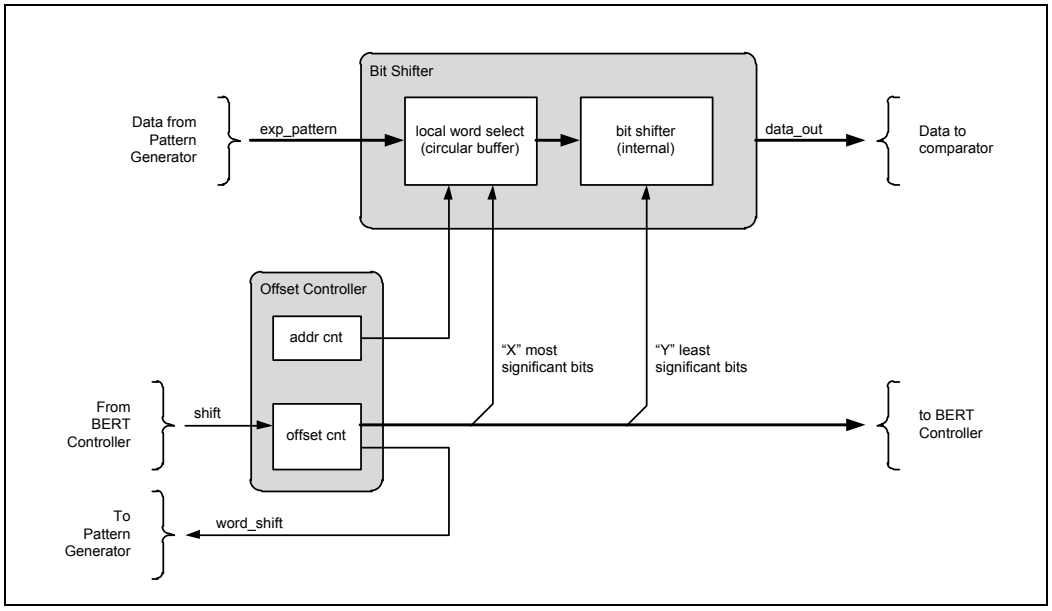

Figure 4-14 : Block level diagram of Bit Shifter

The longer the pattern is the more the synchronization process may take. Longer patterns generally require more shifts to get synchronized. The maximum number of shifts required for a pattern to synchronize is equal to the length of the pattern.

User defined patterns are can be stored in *pattern memory*. If a pattern is longer than 512 bit, it may need more than 512 shifts, and the circular buffer cannot handle that. In such cases, the *pattern memory* address will be incremented by the rounded value of $\frac{number\ of\ requested\ shifts}{512}$ and the rest of the shifts will be handled in the *Bit Shifter*. Incrementing the address of the *pattern memory* address is equal to a shift of 512 bits, since data in memory is stored in 512 bits blocks.

Regarding the multi-phase BERT this implementation seemed to cause problems. For instance, consider the case where the oversampling-ratio is 4. The *Bit Shifter* block which is apparently holding 512 bits of the expected pattern in its circular buffer is in fact holding the expected pattern data which is now passing through *duplicator*; therefore, it is actually holding 128 bits and each bit has been duplicated four times to make a total of 512 bits. If the *BERT Controller* block asks for a shift which is more than 512 bits, this cannot be handled inside the *Bit Shifter*. The *Bit Shifter* will ask the *pattern memory* to increment the memory address by one and the remainder of the shift is done inside *Bit Shifter*. The problem is that this is in fact skipping 512-128=384 bits of the expected pattern data stored in the *pattern memory*: these 384 bits are not passed to the circular buffer yet, and now memory address is being incremented by one and it is skipping the 384 bits!

The *Bit Shifter* is currently inside the *BERT*, after *Duplicator*. First thing that comes to mind is to change the hierarchy of the *Bit Shifter* and the *Duplicator*. Then, any shift request is handled first and the duplication is done after.

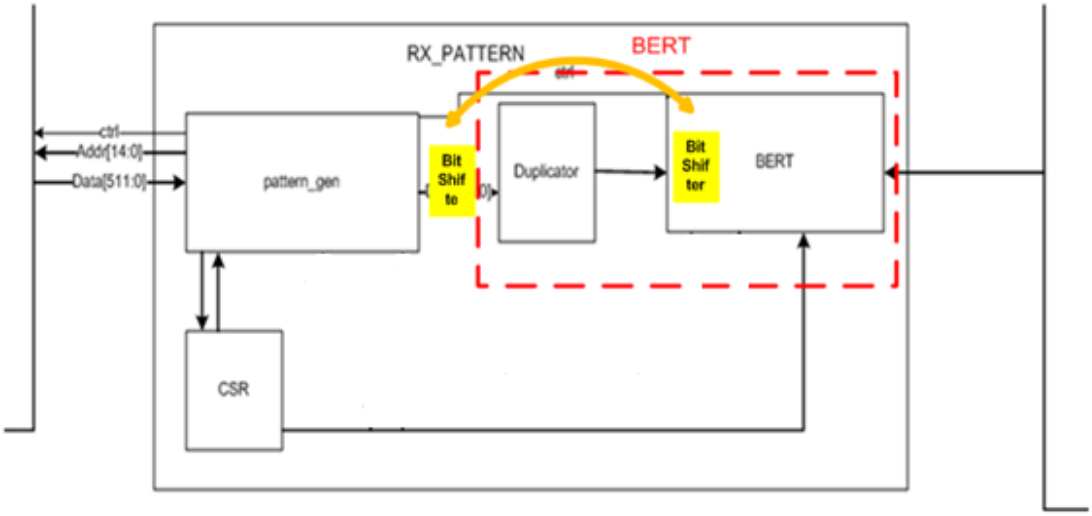

Figure 4-15 : Replacing Bit Shifter with Duplicator

However, this does not work, since shifting and duplicating are not commutative. The example below shows that to first duplicate and then shift or to first shift then duplicate, does not give the same result.

Example:

01010110:

- First shifting by one bit then duplicating each bit 4 times:
  1) Shifting by one: 00101011
  2) Duplicating: 0000 0000 1111 0000 1111 0000 1111 1111

- First duplicating each bit 4 times then shifting by one bit:
  1) Duplicating: 0000 1111 0000 1111 1111 0000
  2) Shiting: 0000 0111 1000 0111 1000 0111 1111 1000

An alternative solution may be to have multiple *Bit Shifter* blocks. In this case, in the range of the oversampling-ratio of 16 we need to have 16 *Bit Shifters*. Each of the *Bit Shifters* has a delay and a circular buffer, which all gets 16 times more.

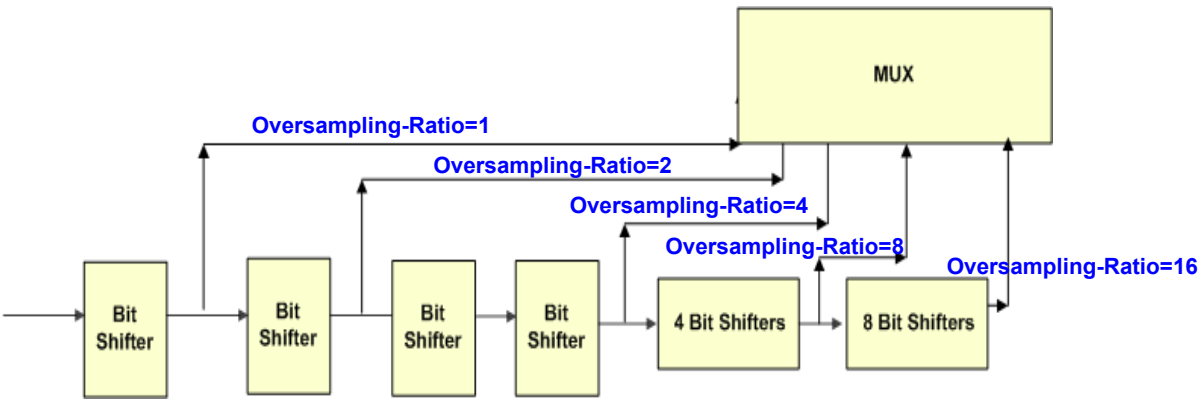

Figure 4-16 : Having multiple Bit Shifters to support different oversampling-ratios

Another option is to increase the number of entries in the circular buffer, so that it can have 15*16 lines more. This also requires adding considerable circuitry and delay to the existing one.

Assuming that the oversampling–ratio is 4, at a shift request of more than 512 bits, we can tell the *Pattern Memory Arbiter* to start sending expected pattern from 128^{th} bit instead of incrementing the memory address by one. Normally, incrementing the memory address by one is equal to sending the reference pattern from 512 bits after. Since now the oversampling-ratio is four, sending $\frac{512}{4} = 128$ bits from *Pattern Memory*, fills in the circular buffer. In other oversampling-ratio ranges, after there is a request of 512 bits shift from the *BERT,* instead of incrementing the memory address by one, we start sending the from the $\frac{512}{oversampling-ratio}$ bit of the current memory address.

Each memory address contains 512 bits of the reference pattern. 512 bits of the *Pattern Memory* is passed to the *BERT* in 32-bit blocks; that takes 16 clock cycles. This is achieved with two 4-to-1 multiplexers. Consider *DATA_IN_VALID* signal in gets asserted by *Pattern Memory Arbiter* every 16 clock cycle when a new 512-bit is ready to be sent to the *BERT*.

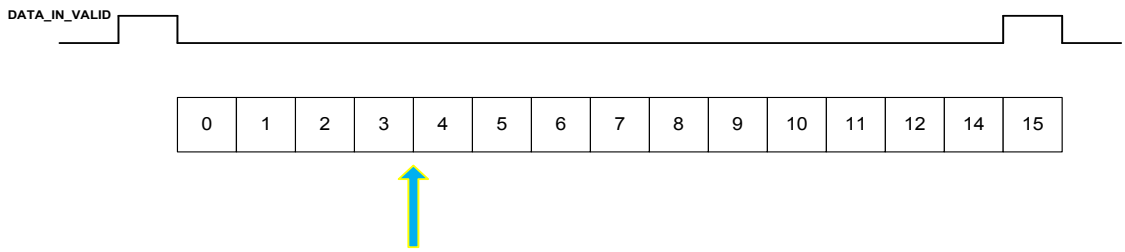

Figure 4-17 : Reference pattern is sent on the rising edge of DATA_IN_VALID from pattern memory. Pattern Memory Arbiter sets the DATA_IN_VALID. The 512 bits of the reference pattern is sent in sixteen 32-bit packets.

Therefore, starting to send from the 128th bit of a pattern memory address is in fact starting sending expected pattern from the 5th (index 4) 32-bit packet. When it reaches the 16th packet it should wait for four clock cycles till *DATA_IN_VALID* gets asserted once more and memory sends new 512 bits. The counter starts counting from the initial value we provide, to 15. When it reaches 15 and since *DATA_IN_VALID* is not set, the counter does not reset and keeps counting, it start re-counting from 0; this means that it sends the data which it had to skip sending.

Having an extra 512 flip flops to look into future takes 512 flip flops of the available register resources. Then when the index of the 32-bit packets of pattern memory reaches 15, from the next cycle on we can start sending from the next 512 flip flops which are holding the reference pattern of the next memory address.

However, this could only solve the problem of sending wrong data when it reaches the end of the first 512. Again, for getting a new 512-bit of *Pattern Memory* we have to wait for the next *VALID_DATA_IN*, which comes few clock cycles later.

When the oversampling-ratio is 16, 8, 4, or 2, after each request of 512-bit shift, instead of incrementing memory address we can start sending from respectively 32^{th}, 64^{th}, 128^{th} or 256^{th} bit. This is two wait respectively 1, 2, 4, or 8 clock cycles for the next 512-bit of the reference pattern to come.

Sending junk data while waiting for the next *VALID_DATA_IN*, may be an alternative to having extra flip flops to look into future. For example, in the case of an oversampling-ratio of 4, during the last 4 clock cycles that we are waiting for the new 512-bit of the reference pattern, we can send all junk data that is not relevant to the reference pattern. Considering that if it is going to synchronize, it will definitely skip the junk data and synchronize after, this makes the solution sound reasonable. Still, this does not solve the problem we had with other approaches.

Originally, we wanted to start sending from the $\frac{512}{oversampling-ratio}$ bit of the current memory address whenever there was a request of 512 bit shifts from the *BERT*. For instance, when the oversampling-ratio is four we would like to start from the 128^{th} bit, which is the 4^{th} 32-bit packet of the reference pattern. In fact we wanted to shift expected pattern 4*32 bits forward. Figure 4-18 shows the alignment which corresponds to a shift of 512 bits in the range of oversampling-

ratio of 4. It is actually shifting the reference pattern from *Pattern Memory* by $\frac{512}{4} = 128$ bits, since the reference bits will be duplicated four times before going to the *BERT*. 128 bits is sent within the first four 32-bit packets (0 to 3) sent in the first four clock cycles.

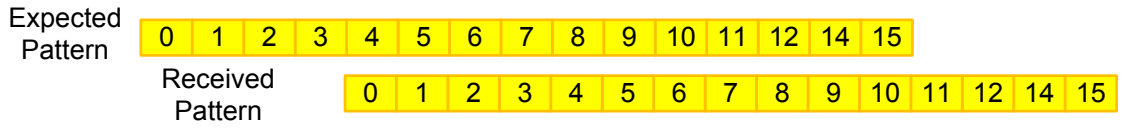

Figure 4-18 : When there is a shift request of 512 bits, considering each cube to be a 32-bit packet, we would like the get the above alignment between the expected pattern and the received pattern (oversampling-ratio is 4)

Instead of shifting expected pattern four clock cycles forward to get the alignment shown in Figure 4-18, we can delay the sampled pattern received from gigabit receiver by four clock cycles. The figure below shows that delaying the received pattern four clock cycles makes the same packet alignment as shown in Figure 4-18.

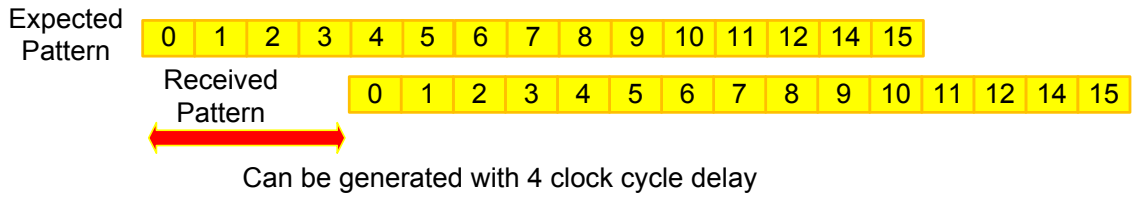

Figure 4-19 : Delaying the received pattern by four clock cycles to make for 512-bit shift request alignment (in case of oversampling-ratio of 4)

Figure 4-20 shows that having four 32-bit registers in a row delay the received pattern by four clock cycles. Originally, we wanted to compare the received

pattern with the reference pattern shifted 128 bit shifts forward. Keeping reference pattern constant and instead shifting the received pattern 128 bits backward gives the same alignment. Shifting the received pattern backward in fact can be achieved by delaying the received pattern coming from the gigabit transceiver.

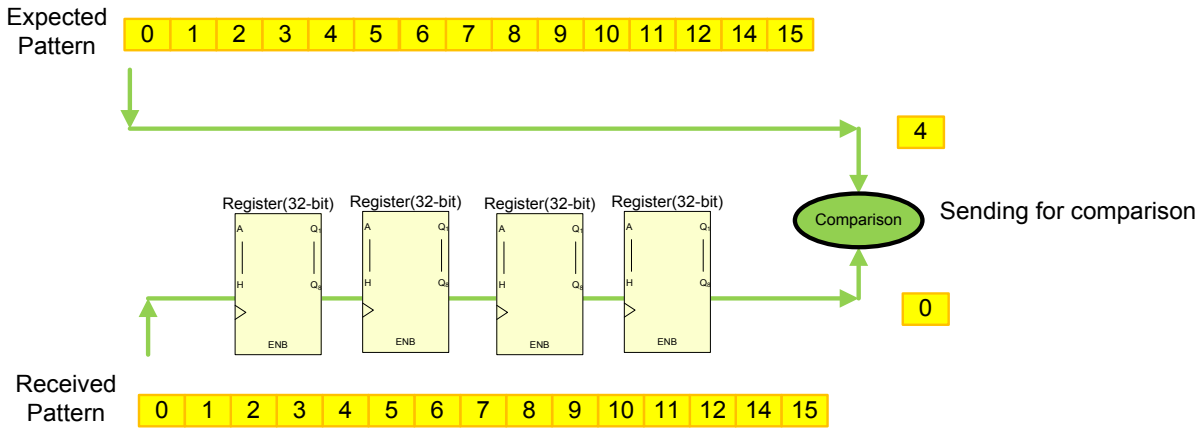

Figure 4-20 : Hardware implementation of delaying the received pattern by four clock cycles

In order to make the design compatible for all the oversampling-ratios (all possibilities), we use the implementation shown in Figure 4-21. In case of oversampling-ratio of 1 there is no need to delay the received pattern. When oversampling ratio is two and there is a request of 512-bit shift, the received pattern is delayed by 8 clock cycles. Similarly, for oversampling-ratios of 4, 8, or 16 respectively 4, 2, or 1 clock cycle delay of the received pattern is required at the request of 512 bits.

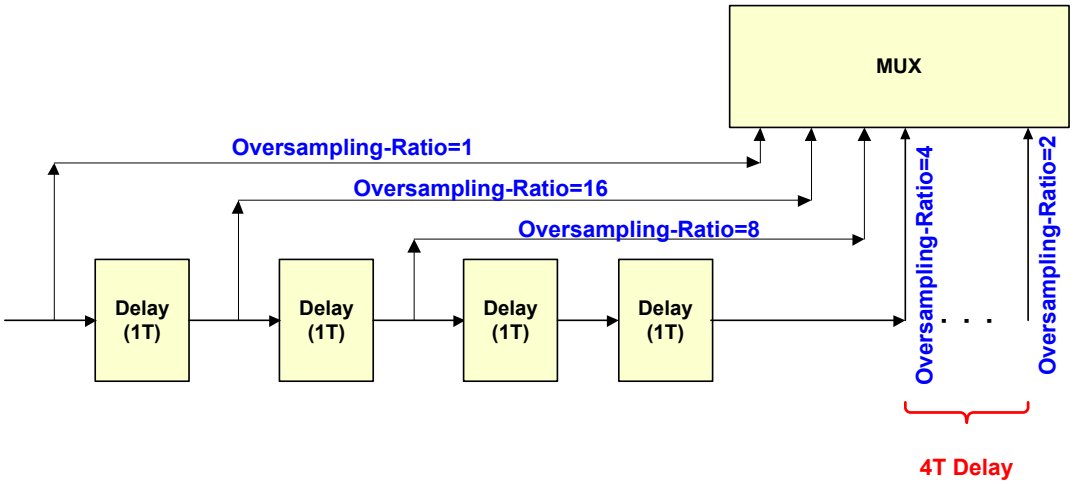

Figure 4-21 : Including all oversampling-ratios in the design

4.3.7 Arranging Bit-Error Counts and Calibrated Phases

With multi-phase BERT we are receiving multiple bit-error values, but they do not belong to one specific Rx-phase. We receive 16, 8, 4, 2, or 1 bit-error values which are corresponding to 16, 8, 4, 2, or 1 different phases which are not for successive points on the bathtub plot. Consider the bathtub plot shown in Figure 4-22, where oversampling-ratio is assumed to be 4. The red circles are the first results obtained from setting the Rx-phase delay on the first point. We get the bit-error values of all the phases shown in red circles on the bathtub. But as we see in Figure 4-22 they do not constitute consecutive points of the bathtub. The Rx phase is swept over the distance between the first and second red circle points, to get all the required points of one side of the bathtub. The distance between two successive phase delays is the phase-step or phase resolution.

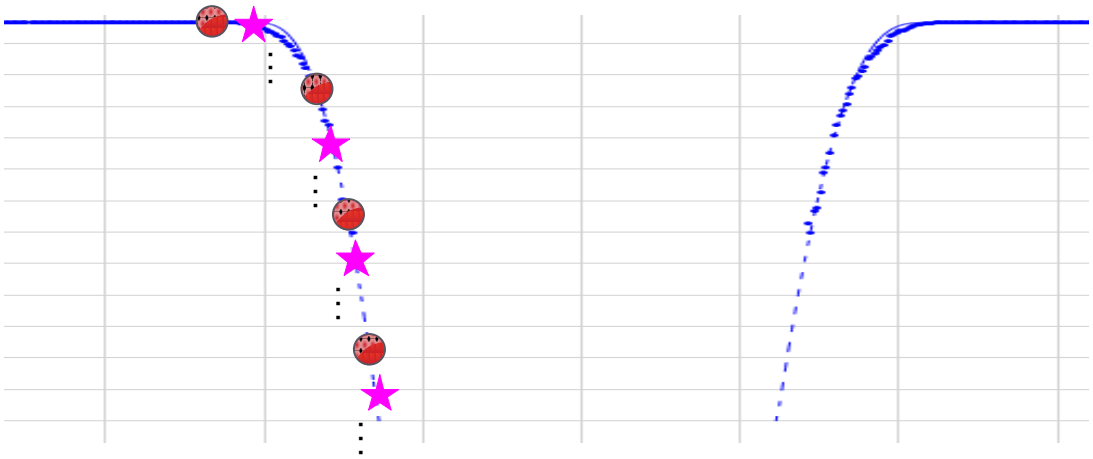

Figure 4-22 : Bathtub plot

With the former design, if in order to have a bathtub receiver phase delay is swept over the interval between a given start-phase and end-phase with a distance of certain step-phase, the total number of points BER measurement that should be measured is equal to:

$$Number\ of\ Elements = \frac{start_{phase} - end_{phase}}{step_{phase}} + 1$$

With the multi-phase BERT method, the number of elements required to have the BER value of the same points of the bathtub curve, is approximately $\frac{1}{oversampling-ratio}$ times less.

For any user defined start-phase, end-phase, and phase-step, first the number of number of UIs that can be fit within the interval between the start-phase and end-phase, is calculated. In order to cover each UI, starting from the start-phase, incrementing by the phase resolution (phase-step), we sweep the Rx-phase delay

to $\frac{UI_{period}(ps)}{oversampling-ratio}$. For each phase within this interval that Rx-phase delay is set at that point, the phase address register, which we used to address different phases within one MPB measurement, is written from 0 to (oversampling ratio-1) to get the bit-error value of the other phases.

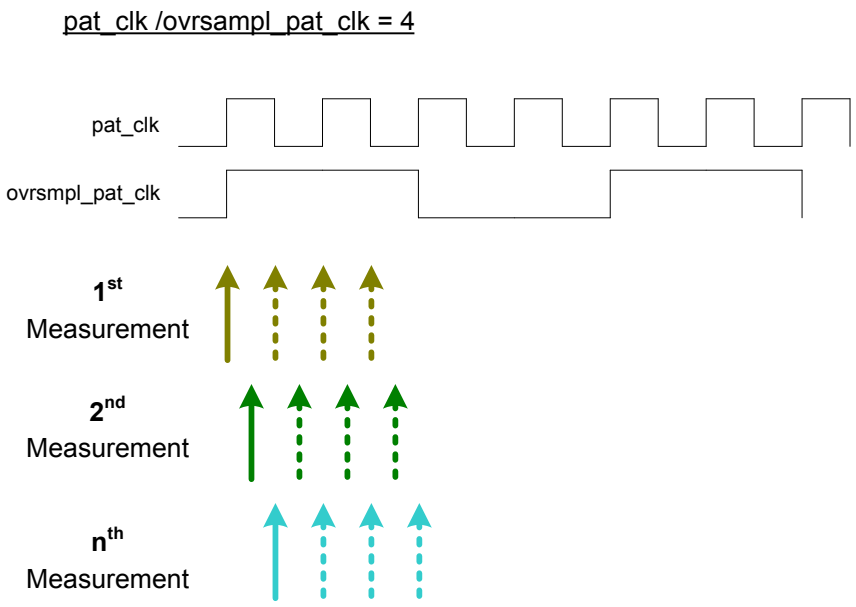

Figure 4-23 : Sweeping Rx-phase gradually to obtain BER-scan plot

The error count for each phase should be stored in an array which holds all the bit-error results; however, as discussed before, the error-counts we get at the same time are not for successive points. This should be considered while filling in the *Results* array; bit-error result of each phase should be stored in the right index of the Results array.

Each time receiver phase delay is set at a certain phase delay value, hardware gives the exact calibrated phase delay of the point where BER measurement was

actually done; i.e., the exact phase that receiver phase was eventually set at [21]. With multi-phase BERT method, we are not sweeping the receiver phase delay over entire phases; we cannot get the calibrated phase of all of the points back from hardware.

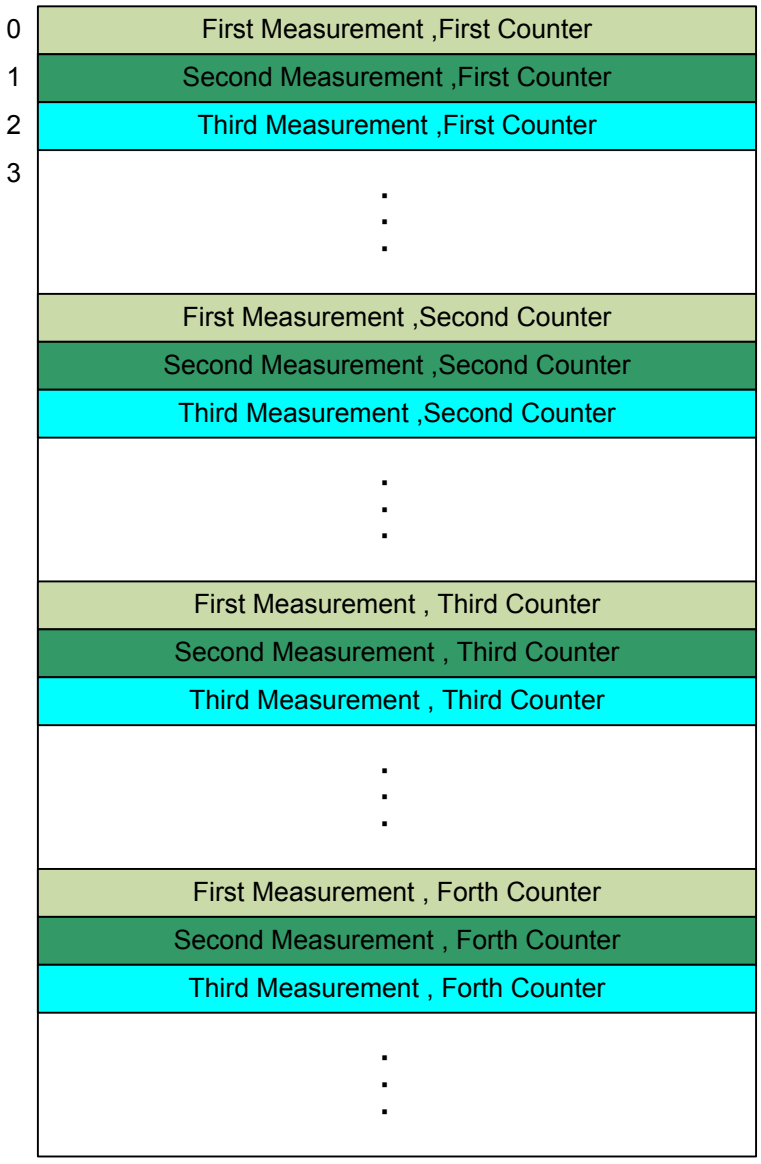

Figure 4-24 : Arranging bit-error values in order

We only have the calibrated phase delay of those phases at which we actually set the sampling phase delay. This is only $\frac{1}{oversampling-ratio}$ of the UI. For the rest of the phases, their calibrated phase delay is obtained with adding multiples of $\frac{UI\_period}{oversmapling-ratio}$ to the calibrated phase delay of the phases within the first $\frac{1}{oversmapling-ratio}$ part of the UI.

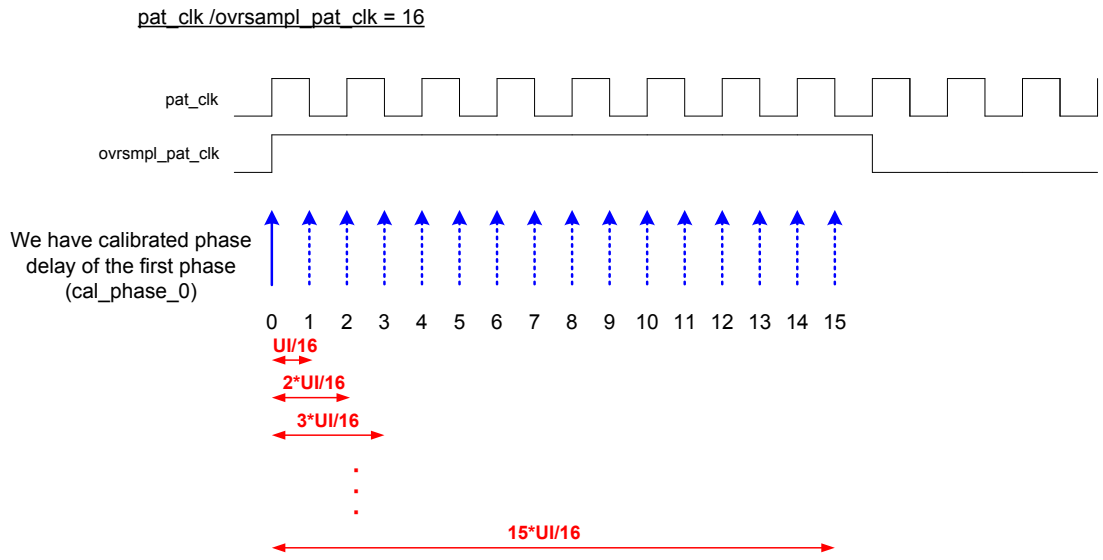

Figure 4-25 : Obtaining calibrated phase delays of phases on bathtub plot

Storing all the calibrated phase delays in another array has the same issue as the Results array. The calibrated phases obtained in a row are not for successive indices of the array. This should be considered as we calculate the calibrated phases to make sure we are storing them at the right indices of the array.

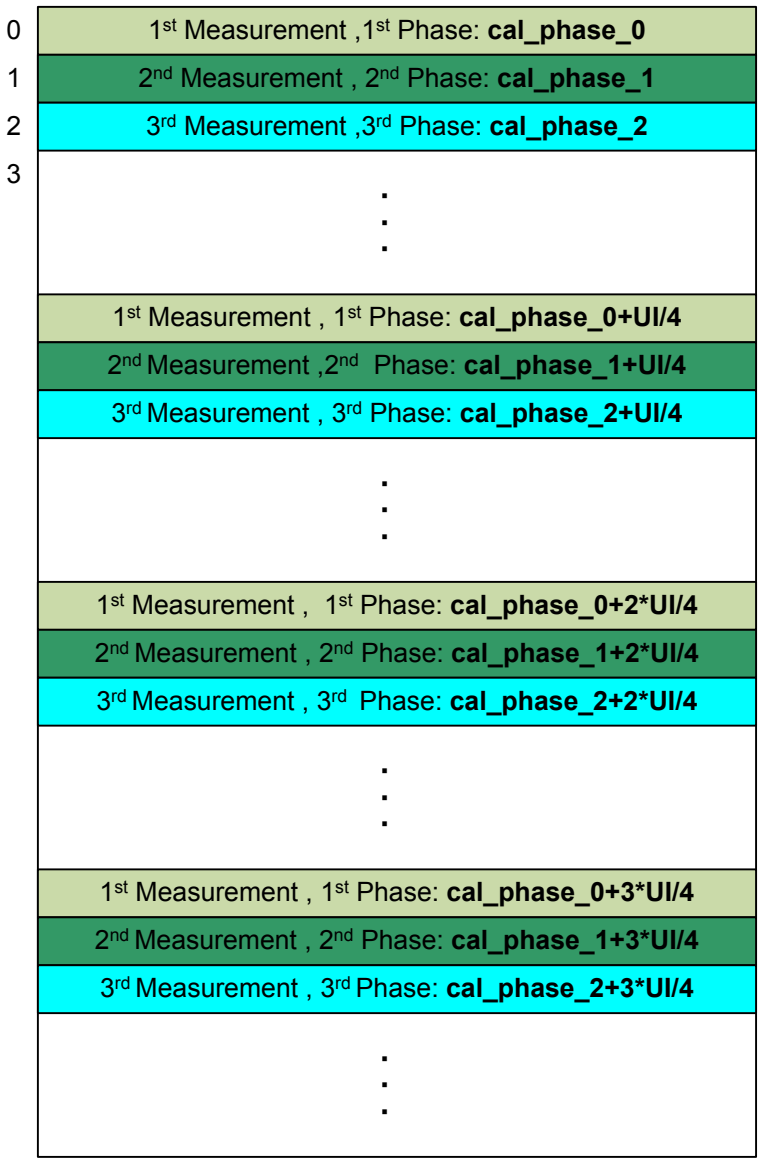

Figure 4-26 : Arranging calibrated phase delays in order. For each set of measurements we will only get the calibrated phase delays of the first phases. For the rest we will obtain their calibrated phase delay with calculation, by adding multiple fractions of UI to the calibrated phase delay of the first phase.

4.4 Multi-Phase BERT Possible Cases

Theoretically, multi-phase BERT will work when $\frac{UI\_period}{oversmapling-ratio}$ can fit an integer number of phase resolutions within itself; that is:

$$\frac{UI_{period}}{oversmapling - ratio} = n * step_{phase}$$

In this case, multi-phase BERT measures the BER at exactly the same phases as the ones the user had set. For instance, consider Figure 4-27. The red arrows are showing the phases the user wants to have BER measured on them. Assuming the oversampling-ratio is 8, starting to sweep the Rx-phase delay through the first $\frac{UI}{8}$ interval, we can measure the BER value at exactly the same phases the user had specified.

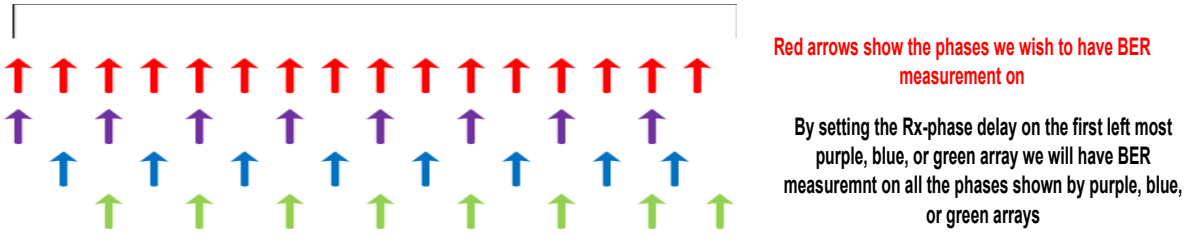

Figure 4-27 : MPB Case 1 – Method Successful

If the distance between different phases of an oversampled data is not a multiple of phase resolution (step), which is:

$$\frac{UI_{period}}{oversampling - ratio} \neq n * step_{phase}$$

All of the phases BER is evaluated, do not exactly match with the ones user has specified; there is always an offset between them. This is shown in Figure 4-28, where the oversampling-ratio is assumed to be 8. Red arrows (first row arrows) show the phases the user wants the BER measurement on. Purple arrows (second row arrows) show the phases where we measure BER value by setting the Rx-phase delay on the first phase. In order to get the BER measurement for rest of the phases, if Rx-phase is delayed by phase resolution, second round of BER measurements is obtained, which are shown with blue arrows (third row arrows). As shown they do not overlap with the remaining red arrows. We cannot shift more since this is 8X mode; the multi-phase BERT idea was to shift receiver phase delay within $\frac{UI}{8}$ (generally $\frac{UI\_period}{oversampling-ratio}$) not the entire UI.

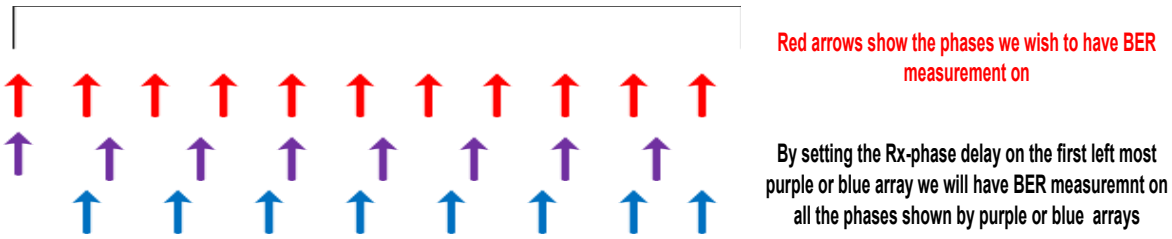

Figure 4-28 : MPB Case 2 – Although the samples might not exactly overlap with the ones user defines, MPB can handle this for the purpose of plotting BER-scan plots

Still, this does not cause a problem in generating bathtubs. What we do is that we take the phase resolution to sweep the Rx-phase delay between the starting of each UI to $\frac{UI\_period}{oversmapling-ratio}$,once the Rx-phase delay reaches there we have already all the BER values and calibrated phase delays calculated. We know that

these calibrated phase delays are not the actual ones user had requested but we report the actual calibrated phases to the user. Bathtub curves are plotted based on the BER-results array and the calibrated-phases array we obtain and are properly corresponding to each other.

The only other mathematical issue with the multi-phase BERT that might come into mind is when:

$$\frac{UI_{period}}{oversampling - ratio} \leq Phase\ Resolution$$

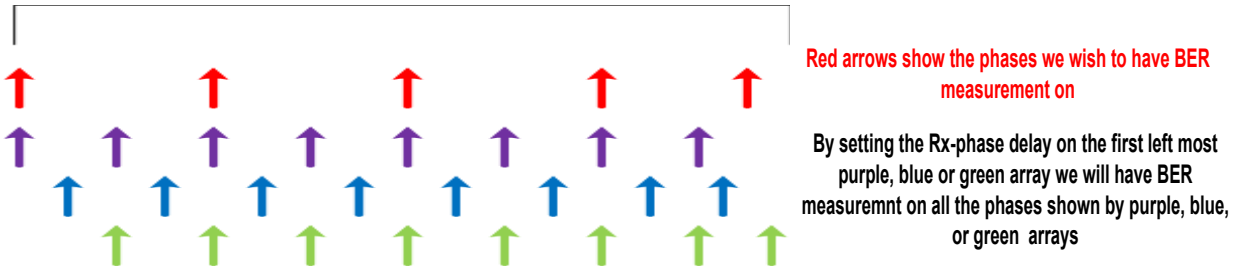

Figure 4-29 : MPB case 3 where $\frac{UI_{period}}{oversampling-ratio} \leq step_{phase}$ (oversampling-ratio= 8)

This case is shown in Figure 4-29. When phase resolution is bigger than the distance between oversampled phases ($\frac{UI_{period}}{oversampling-ratio}$), multi-phase BERT will not be able to handle this case.

However, this situation is almost impossible to happen. Below is the worst case considered when the data rate is really low and the UI length is relatively longer; hence, $\frac{UI_{period}}{oversampling-ratio}$ is maximum. For example, if the data rate is 250Mbps,

UI period is 4000 ps. This is within the range of oversampling-ratio of 16; $\frac{UI_{period}}{oversampling-ratio} = 250\,ps$. This means only when enters a phase resolution bigger than 250ps, multi-phase BERT is not applicable. Taking into account that the BER values are going to be used in plotting bathtubs where all the challenge comes up as the phase resolution gets smaller and is the desired result, we can assume that this condition never happens.

Chapter 5 - **Experimental Results**

All of the modifications mentioned in Chapter 4 were implemented in the firmware of the DJ60HS Test Module of DFT Microsystems Canada Inc. The goal was to validate the proposed multi-phase BERT and establish what actual speedup can be obtained.

This chapter first explains the test set up for the tests performs. The rest of the chapter includes the results of the experimental tests performed on multi-phase BERT in order to ensure its functionality and performance.

We start with the synchronization tests, since synchronization is the first and most important task of the BERT engine, without which the two bit streams will not get aligned and BERT will not report the right number of mismatch. We do the synchronization tests both with delaying transmitter phase delay and without delaying transmitter phase delay.

We then check the BER-scan test results to see whether by sweeping the receiver phase delay over 2UI we can have bathtub curve plots. To ensure the reliability of the bathtub curves generated we perform multiple BER-scan tests under the same conditions to track the edge displacement of the bathtub curves.

We also perform another set of BER-scan tests extract the jitter components of the bathtub-curves. We compare the results of the jitter components of each test with the results of the same test with the former design.

In the two last sections of this chapter, we investigate the speed up achieved with multi-phase BERT technique and we look into the cost of this enhancement.

5.1 **Test Setup**

We perform all the tests in serial loopback. The most popular solution for multi-gigabit Serializer/Deserializer (SerDes) testing in production, is to loop back the output of the transmitter to the input of the receiver. This loopback which may be done either internally or through the loadbord [15][16], compares the output of the receiver to the expected result. In many cases loopback test is the only commonly used test to cover a Serializer/Deserializer (SerDes) device. Also, loopback test is very popular because of its simplicity and high throughput [2].

DJ60HS gives us the option to do serial loopback either internally, by having the corresponding bit set in the FPGA implementation, or externally connecting the transmitter to the receiver with a cable. We perform the tests in internal serial loopback.

Figure 5-1 shows how external serial loopback test is performed. It basically connects the transmitter side to the receiver side. While we do not inject any extra jitter into the system, results of the loopback jitter test show the amount of noise existing within the system.

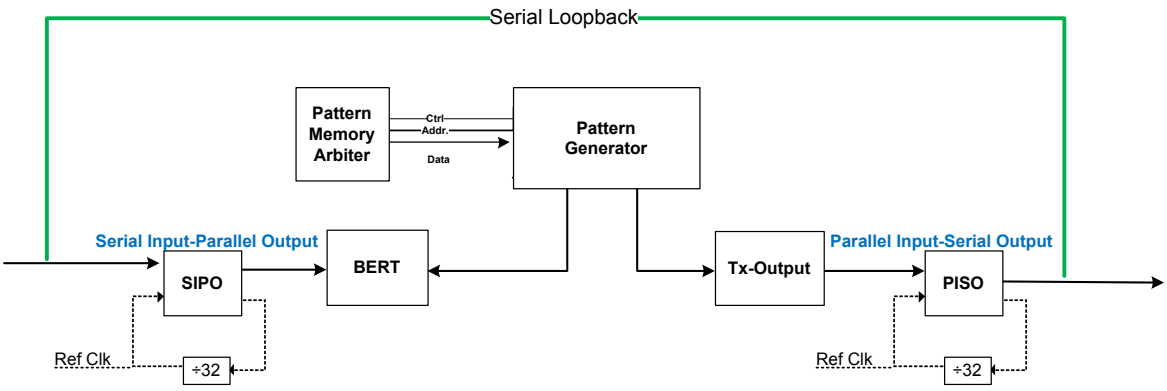

Figure 5-1 : Serial Loopback. A cable connects the transmitter to the receiver side.

BERT scan plots can be obtained by having the clock recovery unit (CDR) disabled and sweeping the sampling delay for the BER test through the whole unit interval (UI) with fine resolution. The phase resolution, which is also referred to as phase step, determines the accuracy of the test for extracting jitter components.

Before each BER measurement, we perform a synchronization test to align the two bit streams. Otherwise, the bit-error value reported by the BERT engine is not valid. Then, we sweep the receiver phase delay starting from the *start-phase* to the *end-phase*. In order to get a full bathtub curve we sweep the sampling delay over 2UI.

Using dual-Dirac model we separate a signal's aggregate total jitter into random jitter (RJ) and deterministic jitter (DJ) component. Below a certain BER value, the BER will be dominated by the random jitter which follows a Gaussian distribution. This allows extrapolating the bathtub curve at lower BER levels [28][27][15]. Extrapolation allows us performing the BER test involving less bits (duration), which helps us get around doing very long test times.

5.2 Synchronization Tests

Synchronization is the first and most important step of the bit-error rate testing process in our design. Before each BER measurement first the two bit streams should be synchronized together; otherwise, bit-error values are not reliable values.

Synchronizing longer patterns generally takes more time. For instance, if a pattern is 20 bits long, it might take up to 20 bit-shifts for synchronizing; whereas, a 800-bits pattern might require 800 bit-shifts. During the synchronizing process, the *BERT Controller* repeatedly sends different shift requests to the *Bit Shifter* block. *Bit Shifter* block by itself was one of the challenges of multi-phase BERT design, especially for longer patterns. If it is not implemented correctly and made compatible with the entire design, it will cause failure in the synchronization process. The following table summarizes the results of sync results of different patterns.

Pattern Name	Pattern Definition	Pattern Length	Sync Status
D21_5	01'	2	Pass
K28_5	10100000110101100000'	20	Pass
K28_7	00000111110000011111'	20	Pass
DIV16	1111111100000000'	16	Pass
DIV20	00000000001111111111'	20	Pass
DIV40	0000000000000000000011111111111111111111'	40	Pass
ALL_ZERO	0000000000000000000000000000000000000000'	1	Pass
PRBS-5		$2^5 - 1 = 31$	Pass
PRBS-7		127	Pass
PRBS-9		511	Pass
PRBS-11		2047	Pass
PRBS-13		8191	Pass
User Defined Pattern		720	Pass

Table 5-1 : BERT Sync Results for different patterns

It should be mentioned that for Sync tests, we enable clock data recovery (CDR). CDR skips the edges and moves around the centre of the UI where there is less bit-error probability [21].

5.2.1 Synchronization Tests with Sweeping Tx-Phase Delay

Since in the Bit Shifter block, after the bit shift request reaches 512 bits we delay the received bit stream from the gigabit transmitter and the functionality of the sync function highly depends on this block, we should make sure that sync is done successfully no matter what the transmitter phase delay is. With generating a loop which constantly increases the transmitter phase delay and runs a bathtub we verify sync performance of MPB in case of delaying the transmitter. For all of the settings in Table 5-1 we tried the above test and it could successfully sync for all values of transmitter phase delay with the length of one UI.

5.3 **BER-scan Tests**

Our final goal was to be able to do BER-scan tests and extract jitter information from the bathtub plots.

In order to plot a full bathtub curve, receiver phase delay needs to be swept over 2UI. For instance, at 5Gbps, if we set the *start-phase* to -400*ps* and *end-phase* to 400*ps*, with a phase resolution of 1ps we need to perform 801 BER measurements. In order to guarantee a BER level of 10^{-12}, more than 10^{12} bits should run for each test. Each BER measurement takes a minimum of 200 s. Also, there is about 0.1 ms overhead time for each BER measurement. This means, that

the total BER-scan test takes more than 801*(200+0.0001) seconds that is about 30 minutes. At lower data rates not only the number of phases we have to do the BER measurement on gets more, but also each BER test takes more time. For instance, at 250 Mbps, the total BER-scan test takes more than 8801*(4000+0.0001) seconds. In order to get around performing very long test time, we perform the BER-scan test involving less number of bits (duration) and extrapolate the curve at lower BER levels.

As mentioned in section 2.7, separating deterministic and random components of the total jitter (TJ), allows estimating peak-to-peak jitter values at very low BER levels that would otherwise take too much time to measure directly. The dual-Dirac model which we use for extraction, gives a better curve-fit [28][27].

The solid horizontal "lines" of the top section of all of bathtub curves we show in this chapter are obtained by a direct bit-error measurement and the dotted "lines" below a BER level of 10^{-7} are calculated with extrapolation using the dual-Dirac model.

While implementing multi-phase BERT we made several assumptions and there were different possible cases we discussed. This is where we can verify whether those assumptions were right and whether our design is robust to any condition or not. We will achieve this by observing whether we have bathtub curves without any bumps and discontinuities. Also, we will try to see whether we have discontinuities, glitches, etc.

First, we would like to verify whether the idea of the adding fractions of UI period to the calibrated phase of the first Rx-phase to get calibrated phase delays at other Rx-phases. We will observe whether each BER-scan test results can be plotted as bathtub shaped curves without any glitches and discontinuities.

Figure 5-2 is the bathtub result of one of the BER-scan tests done.

Data Rate: **5Gbps**
End Phase: **-400 ps**
Start Phase: **400 ps**
Phase Resolution: **1 ps**
Type of Pattern: **PRBS-5**
BERT Duration: **10^7Bits**

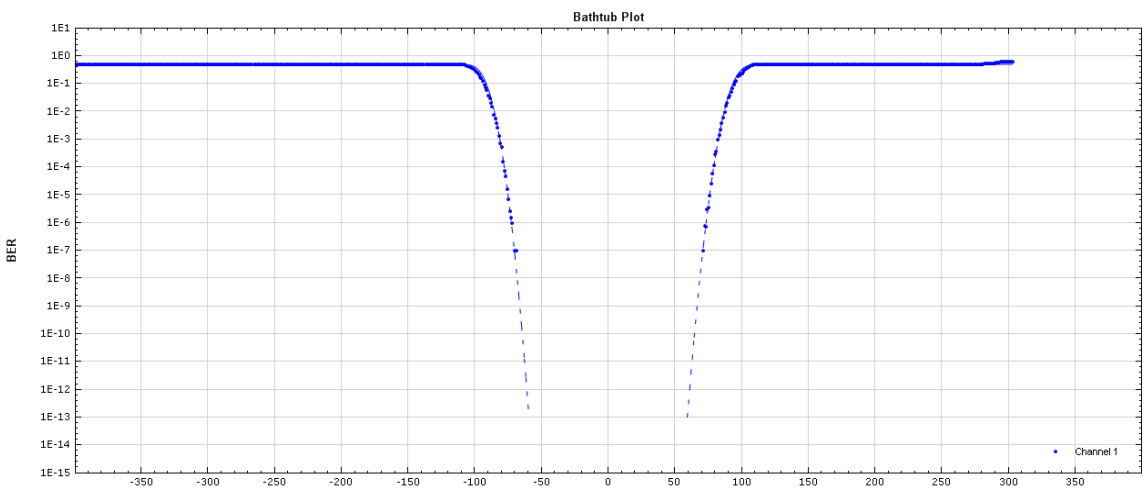

Figure 5-2 : BER-scan Result 1

We now look into the cases discussed in section 4.4. While Figure 5-3 shows an example of the first case, Figure 5-4 and Figure 5-5 demonstrate the successfulness of the method under the conditions of case 2.

- $\frac{End\_phase- Start_{phase}}{oversampling-ratio} = n * \frac{UI_{period}}{oversampling-ratio}$

Data Rate: **250Mbps**
End Phase: **-4400 ps**
Start Phase: **4400 ps**
Phase Resolution: **1 ps**
Type of Pattern: **PRBS-9**
BERT Duration: $\mathbf{10^7}$**Bits**

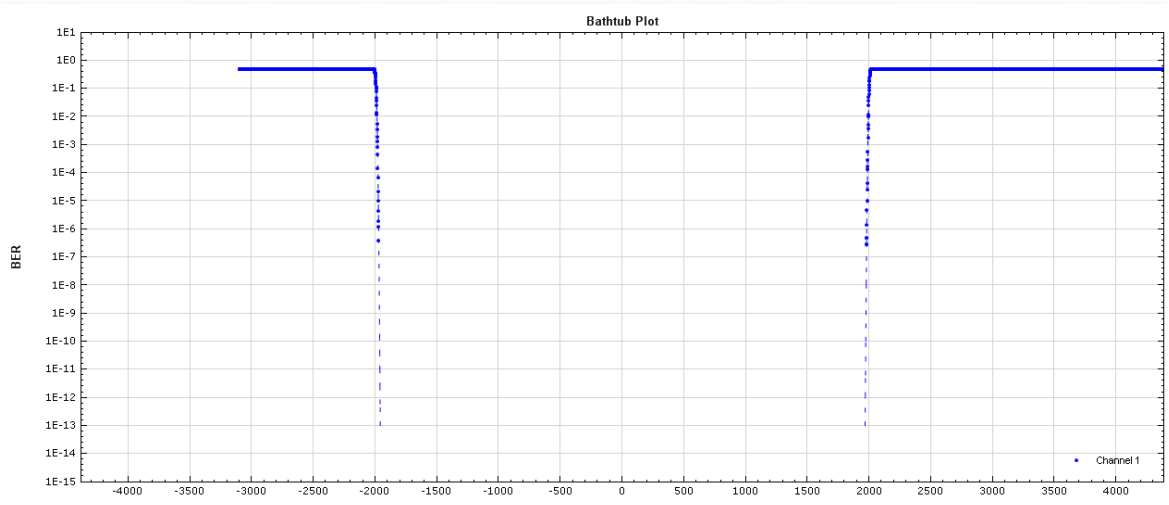

Figure 5-3 : BER-scan Result 2 - case 1. $\frac{UI}{oversampling-ratio} = \frac{UI}{16}$ **is an integer multiple of phase resolution (1 ps)**

- $\frac{End\_phase - Start_{phase}}{oversampling-ratio} \neq n * \frac{UI_{period}}{oversampling-ratio}$

Data Rate: **5Gbps**
End Phase: **-400 ps**
Start Phase: **400 ps**
Phase Resolution: **3 ps**
Type of Pattern: **PRBS-9**
BERT Duration: **10^7Bits**

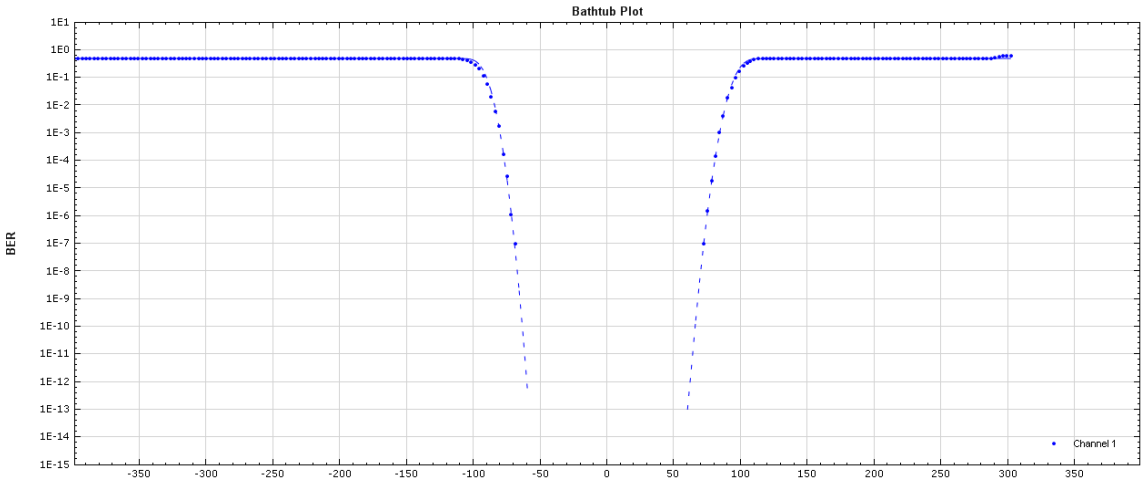

Figure 5-4: BER-scan Result 3 – Case 2. UI is not an integer multiple of the phase resolution (3ps), still BER-scan results obtained with MPB generate a fine bathtub curve

Data Rate: **5Gbps**
End Phase: **-400 ps**
Start Phase: **400 ps**
Phase Resolution: **11 ps**
Type of Pattern: **PRBS-9**
BERT Duration: **10^7Bits**

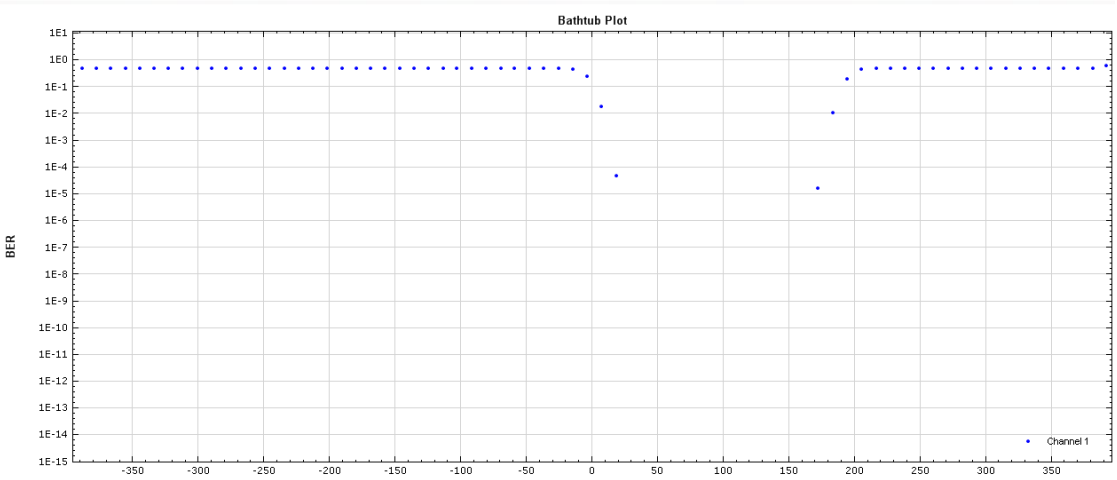

Figure 5-5 : BER-scan Result 4 (without extrapolation) – Case 2. UI is not an integer multiple of the phase resolution (11ps), still BER-scan results obtained with MPB generate a fine bathtub curve

This results show that multi phase BERT is able to generate perfect bathtub-shaped plots.

5.4 **Edge-Displacement Tests**

We will not bring a close to our evaluation and verification by just observing whether BER-scan tests will result to nice bathtub plots or not. We would like to verify whether bathtubs generated by MPB will completely overlap the bathtubs generated with the former method under exactly the same conditions and settings.

Comparing the average location of the bathtubs is a good estimate for verifying the reliability of MPB method. To achieve this we will compare the edge

locations of the two bathtubs. Edge of a bathtub is the halfway transition point which is usually more stable than the other points of the bathtubs [27].

A lot of BER-scan verification tests are done by monitoring the bathtub edge displacement [27]. For our case, we will monitor edge displacement of each implementation and we will compare the results together.

We know that bathtub curves are logarithmic functions of the number of errors versus phase. For now if we do not consider the logarithmic form and only consider the plot of number of bit-errors versus receiver phase, at the center of the UI the number of errors is always zero, then as we get closer to the edges the number of bit-errors per phase gets bigger till it reaches its maximum value at about the edge of the UI.

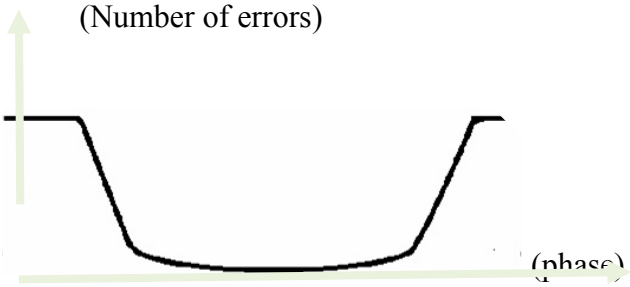

Figure 5-6 : Plot of the number of bit-errors Vs. phase

For non-clock patterns, the maximum number of errors can be approximated with $\frac{1}{2} * (Transition\ Density) * Duration$; where, duration is the number of bits the BER test is being run at each phase and transition density is $\frac{Number\ of\ Transitions}{Pattern\ Length}$. The illustration of above equation for approximation of maximum number of

errors is that at each transition between 0 and 1, if the signal is misshaped the sampled bit will be different from the expected pattern because it will take the adjacent bit as the sample of the current bit. In fact at each transition there is always $\frac{1}{2}$ probability to fall to the other bit (which is different) and sample the wrong value. Based on this for any pattern under test for a selected duration the maximum and minimum of bit-errors can be always determined. Edges of the bit-error curve are in fact half distance between the minimum and maximum. The edges of each bit-error curve are one good criteria of each bathtub which we would have to first of all make the edges stable for the bathtub.

For each setting we do the above test 100 times on both the former design and MPB. We calculate the average and standard deviation of calibrated phase of each transition and then evaluate the result as explained below with benefiting from confidence interval and confidence limit definition.

- Since we are trying to estimate the mean phase delay of each edge in the population, we choose the mean calibrated phase of the former design as the sample statistic.
- Select a confidence level. The confidence level describes the uncertainty of a sampling method. Often, researchers choose 90%, 95%, or 99% confidence levels; but any percentage can be used [22][12]. In our case we choose a 95% confidence level.
- Find the margin of error. Margin of error can be calculated by [22]:

$$Margin\ of\ error\ =\ Critical\ value\ *\ Standard\ deviation\ of\ statistic$$

- Specify the confidence interval. The uncertainty is denoted by the confidence level. And the range of the confidence interval is defined by the following equation [22].

$$Confidence\ interval\ =\ sample\ statistic\ \pm\ Margin\ of\ error$$

For instance consider the first set of tests.

- Find standard error. The standard error (SE) of the mean is:

$$SE = \sigma/\sqrt{n}\ =\ 0.686471443/10 = 0.0686471443$$

- Find critical value. The critical value is a factor used to compute the margin of error. To express the critical value as a t score (t*), follow these steps [22].
    - Compute α: $\alpha = 1 - \frac{Confidence\ Level}{100}\ =\ 1 - 0.95 =\ 0.05$
    - Find the critical probability (p*): $p* =\ 1\ -\ \frac{\alpha}{2}\ =\ 1\ -\ \frac{0.05}{2}\ =$ 0.975
    - Find the degrees of freedom (df): $d =\ n\ -\ 1 = 100 -\ 1 = 99$
    - The critical value is the t score having 99 degrees of freedom and a cumulative probability equal to 0.975. From the t Distribution Calculator, we find that the critical value is 1.96.
- Compute margin of error (ME) [12][22]:

$$ME = critical\ value *\ standard\ error = 1.96\ *\ 0.068\ =\ 0.133$$

- Specify the confidence interval. The range of the confidence interval is defined by the $sample\ statistic\ +\ margin\ of\ error$ [12] [22]. And the

uncertainty is denoted by the confidence level. The 95% confident interval is $218.2\ 15 \pm 0.133$.

We can conclude that if we would like our design to meet the 95% confidence level the population mean of the calibrated phase delays should fall within the 95% confident interval of the former design [12].

Table 5-2 summarizes the results of edge displacement tests performed on both the former design and MPB at different data rates. As shown in the figure at all of the data rates average of the edge displacement of bathtub plots obtained with MPB are all within the 95% confidence interval of the former design's.

Test Settings	Data Rate (bps)	Mode	Right Edge Phase AVERAGE (ps)	Right Edge Phase STD (ps)	Left Edge Phase AVERAGE (ps)	Left Edge Phase STD (ps)	% MPB within 95% confidence level (min of right and left result)
Start=-400 ps, End=400 ps	63750000000	1X	95.0625	0.98	-60.25	0.67	98% (Pass)
Start=-400 ps, End=400 ps	31875000001	1X	41.125	0.56	-275.9375	0.84	97% (Pass)
Start=-500 ps, End=500 ps	15625000001	2X	218.215	0.68	-421.92	0.677	96% (Pass)
Start=-500 ps, End=500 ps	31875000000	2X	22.8125	1.2	-282.9375	0.65	95% (Pass)
Start=-500 ps, End=500 ps	15625000000	4X	116.68	0.989	94.93	0.53	96% (Pass)
Start=-500 ps, End=500 ps	7812500001	4X	-59.8125	0.79	-95.25	0.59	95% (Pass)
Start=-500 ps, End=500 ps	7812500000	8X	119.875	0.34	101.125	0.987	96% (Pass)
Start=-400 ps, End=400 ps	3906250001	8X	-48.75	2.03	-86	1.7	97% (Pass)
Start=-400 ps, End=400 ps	3906250000	16X	-192.875	1.125	-212.0625	0.35	95% (Pass)

Table 5-2 : Edge-Displacement Test Results

5.5 Extracting Transmitter Jitter

BER-scan plots are used to extract transmitter jitter components, including random jitter (RJ), total jitter (TJ), and deterministic jitter (DJ). Jitter extraction is done using dual-Dirac model [15][27].

The dual-Dirac model separates total jitter into two subcomponents, random (RJ) and deterministic jitter (DJ) [28]. Random jitter is defined to have a Gaussian PDF and deterministic jitter PDF is defined to be composed of two Dirac delta functions [32][33]. When DJ and RJ are convolved together, they form a new PDF that closely matches the TJ's PDF at low BER levels [28][27].

In order to verify MPB implementation and determine the tradeoffs involved in terms of jitter, we will perform multiple (100 times) tests on both implementations and will compare the results.

Data Rate = 5000 Mbps	AVERAGE (Former)	STDEV (Former)	MIN (Former)	MAX (Former)	AVERAGE (MPB)	STDEV (MPB)	MIN (MPB)	MAX (MPB)
RJ Statistics	4.63	0.133	4.36	4.82	5.28	0.105	5.01	5.58
DJ Statistics	3.55	1.12	1.74	5.56	1.67	0.692	0.254	3.35
TJ Statistics	66.838	0.74579	65.266	68.692	73.854	0.88655	71.933	76.598
BER Level	1.00E-20	1.06E-35	1.00E-20	1.00E-20	1.00E-20	1.06E-35	1.00E-20	1.00E-20

Table 5-3 : Extracting Transmitter Jitter (Data Rate = 5000 Mbps)

Data Rate = 4000 Mbps	AVERAGE (Former)	STDEV (Former)	MIN (Former)	MAX (Former)	AVERAGE (MPB)	STDEV (MPB)	MIN (MPB)	MAX (MPB)
RJ Statistics	5.09	0.175	4.63	5.58	5.77	0.158	5.42	6.16
DJ Statistics	3.91	1.17	0.956	7.35	3.91	0.935	0.457	4.93
TJ Statistics	73.512	1.2778	70.617	79.326	81.387	1.385	78.762	85.274
BER Level	1.00E-20	1.06E-35	1.00E-20	1.00E-20	1.00E-20	1.06E-35	1.00E-20	1.00E-20

Table 5-4 : Extracting Transmitter Jitter (Data Rate = 4000 Mbps)

Data Rate = 4500 Mbps	AVERAGE (Former)	STDEV (Former)	MIN (Former)	MAX (Former)	AVERAGE (MPB)	STDEV (MPB)	MIN (MPB)	MAX (MPB)
RJ Statistics	5.42	0.323	4.32	5.96	5.23	0.104	4.99	5.54
DJ Statistics	3.46	1.89	0.979	8.59	4.2	0.743	2.25	5.76
TJ Statistics	77.563	3.259	65.543	84.698	75.79	0.76015	74.057	78.607
BER Level	1.00E-20	1.06E-35	1.00E-20	1.00E-20	1.00E-20	1.06E-35	1.00E-20	1.00E-20

Table 5-5 : Extracting Transmitter Jitter (Data Rate = 4500 Mbps)

Data Rate = 3500 Mbps	AVERAGE (Former)	STDEV (Former)	MIN (Former)	MAX (Former)	AVERAGE (MPB)	STDEV (MPB)	MIN (MPB)	MAX (MPB)
RJ Statistics	7.49	16.8	4.88	174	6.11	0.114	5.87	6.39
DJ Statistics	16.7	95.6	0.39	962	4.86	0.857	2.79	6.56
TJ Statistics	119.14	325.98	77.253	3341.4	88.414	0.75765	86.829	90.307
BER Level	1.00E-20	1.06E-35	1.00E-20	1.00E-20	1.00E-20	1.06E-35	1.00E-20	1.00E-20

Table 5-6 : Extracting Transmitter Jitter (Data Rate = 3500 Mbps)

As shown in the tables above, the total amount of jitter related to the multi-phase BERT is greater than for the former design. The reason is that it takes a while for power supply to stabilize, and since MPB is running on a faster clock it waits less for this state and it results to more jitter.

5.6 **Multi-Phase BERT Speed Up**

We are going to verify the actual speed up factor in running a full BER-scan test at different data rates. For this we run each test under certain conditions (pattern, data rate, start-phase, end-phase, phase-resolution, duration, etc.) 100 times different data rates and record the time it takes the tool to plot the bathtubs, compare with the previous method. Table 5-7 summarizes the results obtained.

Average run time required to run a BER-scan test in order to get a full bathtub plot is mainly dependent on factors such as, the number of phases BER

measurement is performed on which is actually phase-resolution. The more accuracy we are looking for the smaller phase-resolution (phase-step) is set, hence there are more phases that BER should be tested on. The tests we performed are with the minimum phase-resolution supported which is 1 ps. BER test also depends on the duration of the tests. BERT duration may also be called BERT length; it determines the number of bits tested at each phase. Following tests have been done with BERT duration of 10000000 (10 million) per phase.

Data Rate	Mode	Run Time (Former)	Run Time (MPB)	Speed Up Factor
6400	1X	4.27	4.3	0.993
6000	1X	4.816	4.84	0.995
5000	1X	4.4116	4.4382	0.994
4500	1X	4.9847	5.02	0.992
4000	1X	5.3019	5.3496	0.991
3500	1X	6.2039	6.2597	0.991
3000	2X	15.15	8.851	1.712
2500	2X	19.782	11.601	1.705
2000	2X	28.021	16.58	1.69
1562	4X	41.123	11.1143	3.7
1550	4X	42.008	11.451	3.6658
800	4X	132.61	36.2322	3.66
750	8X	149.22	20.437	7.301
400	8X	476.74	65.66	7.26
370	16X	386.26	29.914	12.912
250	16X	821.73	63.7	12.9

Table 5-7 : Speed-Up Obtained at Different Data Rates

BER-scan test run time is mostly dependent on the target data rate. Observing the average run time with former design will confirm. At lower data rates the UI is longer. If we would like to keep the accuracy of the test by continuing with the same phase resolution (1ps), that will require performing BERT on many more

phases. For instance at 5000 Mbps, sampling phase delay is swept over -400 ps to 400 s interval with a phase resolution of 1 ps. At 250 Mbps, the minimum interval sampling delay should be swept over in order to get a full bathtub plot is -4400 ps to 4400 ps. Setting phase resolution to 1 ps, this requires performing BER evauation on 8801 phases, this is 11 times more compared to 801 phases required at 5000 Mbps.

Furthermore, considering that each time Rx-phase is set by software and then hardware passes the bit-error value to software, will give an idea of how much the problem of overhead of communication will get worse for too many phases.

We were expecting speed up factor of respectively 1, 2, 4, 8, or 16 in 1X, 2X, 4X, 8X, 16X mode. The speed enhancement actually achieved (Table 5-7) shown is less than what we expected. Since we did not add more registry for the extra error counters in our implementation we are not reading all the 16 error-counters at the same time. This causes a difference between the actual speed up achieved and the one we expected theoretically. Instead of reading all the bit-error values at once, we are reading them one by one each time by first writing the phase index and reading the value. Also, this overhead is the reason that the tests are slower than before in 1X mode. In 1X mode the overhead of communication has been increased, because instead of getting the bit-error value with one read command, we now require one read and one write command to get each bit-error value from hardware.

5.7 MPB Enhancement Cost

This section investigates the tradeoffs that are involved in accelerating BERT with multi-phase technique.

The relative cost of speed enhancement is verified by probing- how many extra logic cells, flip-flops, embedded FPGA memory, etc. [9] have been used to implement multi-phase BERT.

	Former Design	MPB Design
Family	Stratix II GX	Stratix II GX
Device	EP2SGX90FF1508C3	EP2SGX90FF1508C3
Logic Utilization	48%	53%
Combinational ALUTs	20,947 / 72,768 (29%)	23,804 / 72,768 (33%)
Dedicated Logic Registers	23,144 / 72,768 (32%)	25,256 / 72,768 (35%)
Total Registers	23273	25385
Total Pins	466 / 739 (63%)	458 / 739 (62%)
Total Virtual Pins	2	2
Total Block Memory bits	2,520,760 / 4,520,488 (56%)	2,521,265 / 4,520,488 (56%)
DSP Block 9-Bit Elemens	8 / 384 (2%)	8 / 384 (2%)
Total GXB Receiver Channels	16 / 16 (100%)	16 / 16 (100%)
Total GXB Transmitter Channels	16 / 16 (100%)	16 / 16 (100%)
Total PLLs	5 / 8 (63%)	5 / 8 (63%)
Total DLLs	0 / 2 (0%)	0 / 2 (0%)

Table 5-8 : Relative Cost of MPB Method vs. Former Method

As shown in Table 5-8, relative cost of multi-phase BERT is very close to the former design. The only visible difference is in logic utilization which MPB uses 5% more logic. 5% increase in logic utilization by MPB is comprised of 4% increase in combinational ALUTs and 3% increase in dedicated logic registers. MPB implementation takes about 2000 (2112) more flip flops. That's about 8% of

total number of registers required for the entire design. The total number of pins in MPB has been decreased by 1%.

Also, it does not take much longer executing multi-phase BERT firmware.

Chapter 6 - **Conclusions**

6.1 **Conclusions**

This thesis has presented an oversampled multi-phase time-domain BER processing for transmitter testing. With improving BER-scan testing speed, jitter testing has been accelerated; jitter components can be extracted from the BER-scan plots which are now built faster. Eye diagram tests which normally take hours can also be generated faster since they are obtained with performing multiple BER-scan tests. Also, the speed enhancement obtained allows us to perform the tests with the best accuracy which is 1ps phase resolution. Accuracy is an important issue in jitter testing, but normally because of long test times smaller phase resolutions are skipped.

Using time domain oversampling we were expecting respectively 1, 2, 4, 8, or 16 times speed up in bit-error rate processing in 1X, 2X, 4X, 8X, and 16 X modes. Table below summarizes the data rates defined in each mode.

Target Data Rate	Oversampling-Ratio	Mode
6400 Mbps -3187.5 Mbps	1	1X
3187 Mbps - 1562.5 Mbps	2	2X
1562 Mbps - 781.25 Mbps	4	4X
781 Mbps - 390.625 Mbps	8	8X
390.625 Mbps- 250 Mbps	16	16X

Table 6-1 : Actual Oversampling-Ratio Ranges

The speed up we actually obtained is a bit different from what we expected. This is because of the way we implemented the method. It causes overhead of

communication by not providing sufficient registers for all the bit-errors evaluated by hardware. Although, we are computing the bit-error values of multiple phases in parallel at once, we are not passing them to the software all simultaneously. It is just because we wanted to first make sure that the method actually improves the speed and works fine before changing the registry interface. For better speed performance the best is to benefit from the parallelism of multiple error-counters while passing the bit-error values to the software. This can be done simply by adding 15 registers to the register interface.

The developed scheme has been tested at different data rates with different patterns, duration, resolution, etc. each multiple times. All known worst case scenarios have been considered.

The reliability of the developed scheme has been verified by evaluating the statistical results of several tests such as BER-scan, edge-displacement, jitter decomposition, and eye diagram tests. We have verified whether the scheme can successfully build bathtub plots and eye diagrams and have confirmed the results by comparing them with the results of the tests done with the former scheme under the same conditions. Consistency of MPB scheme results is verified by performing multiple tests and extracting jitter components of all of them and evaluating the statistical values such as average, and standard deviation.

Finally, we have investigated the tradeoffs that are involved in accelerating BERT with multi-phase technique. The relative cost of speed enhancement in terms of extra logic cells, flip-flops, embedded FPGA memory, etc. has been verified. The

speed enhancement is achieved with little increase logic utilization and 2000 registers added.

6.2 Future work

In future, this work can be expanded for incorporation with multiprocessor design [41][42][43], for enhancing debug features [44], in reliable networks on chip [45][46], as well as reversible [47], embedded high-density memory [48] design, and the extension is possible with transform-based techniques under lack of available data [49]. Finally, the low-power [50] and the sequential design test [51] can be applied.

Moreover, in future, Bit-error rate (BER), BER-scan, jitter, and eye diagram tests can be further improved in terms of speed. Bypassing the overhead of communication by providing enough registers for all of the parallel error-counters is the first step to achieve this. Having sufficient registers to make all the calculated bit-error values accessible to software at the same time, instead of passing them all through one port one by one, will make the speed up closer to what we had expected. That is to improve the speed respectively by about 16, 8, 4, 2, and 1 times faster in 16X, 8X, 4X, 2X, and 1X mode.

Also, observing the bit-error values of all of the phases within the interval of 2UI, which is required to build a full bathtub curve, will give another hint to improve the speed performance of the BER-scan test.

We know a bathtub curve is indeed the logarithmic plot of the total number of bit-errors at each phase versus phase. At the center of the UI the number of errors is always zero. That's why bathtub plot goes to minus infinity in the middle of UI. As we get closer to the edges the number of bit-errors per phase gets more till it reaches its maximum value at about the edges where the bathtub plot gets flat (constant). The transition usually happens faster; i.e., compared to the number of phases their bit-error value is either zero or maximum; the transition takes very few phases to rise up from 0 to the maximum value. This is while jitter components are extracted mostly based on the bit-error values of the transition.

On the other hand, to achieve better accuracy we normally select the smallest phase resolution (1 ps). For instance, at 3000 Mbps to have a full BER-scan plot with good accuracy, start-phase = -899 ps, end-phase = 899 ps, and phase-resolution = 1 ps. This requires performing 1799 BERTs. There are 1799 phases BERT is performed on, but a transition from 0 to maximum is done in only 40 phases (40 ps). Considering the other transition which is from the maximum bit-error value to 0 also takeing 40 ps, we are doing about 1719 extra measurements. They are extra because we know that as bit-error value reaches 0 or its maximum value it will keep constant for a while. Therefore, there is no need to measure the same value over and over. At 250 Mbps the sampling delay is swept over the interval between -4400 ps and 4400 ps, with a time resolution of 1 ps, BER-scan requires 8801 BER measurements. As each transition takes about 60 ps, we need the accurate bit-error value at only 120 phases (integer phases).

We can have a scheme that goes through the center of UI with bigger phase resolution then slows down for better accuracy only when it is close to the transitions. This will certainly skip a lot of unnecessary measurements and speeds up BERT more than we have achieved. The speed-up is particularly useful as the data rate gets lower, since as target data rate gets slower, the UI keeps increasing. The ratio of the phases, at which or close to which the transitions occur, is much smaller than the total number of phases of the BER measurement. This involves many unnecessary measurements. This is the reason why BER-scan tests gets really slower at lower data rates.

References

[1] Y. Fan, Y. Cai, L. Fang, A. Verma, B. Burcanowski, Z. Zilic and S. Kumar, “An Accelerated Jitter Tolerance Test Technique on ATE for 1.5GG/s and 3GB/s Serial-ATA,” *Proceedings of IEEE International Test Conference*, Oct. 2006

[2] Y. Fan, Y. Cai and Z. Zilic, "A High Accuracy, High Throughput Jitter Test Solution on ATE for 3 Gbps and 6 Gbps Serial-ATA," *Proceedings of IEEE International Test Conference*, Oct. 2007

[3] Y. Fan and Z. Zilic, “Bit Error Rate Testing of Communication Interfaces,” *IEEE Transactions on Instrumentation and Measurements*, Vol. 57, No. 5, pp. 897-906,May 2008

[4] Y. Fan and Z. Zilic, "A Versatile Scheme for Validation, Testing and Debugging of High Speed Serial Interfaces," *Proceedings of IEEE High Level Design Validation and Test Workshop,* HLDVT’09, Nov. 2009

[5] Y. Fan and Z. Zilic "Accelerating Jitter Tolerance Qualification for High Speed Serial Interfaces," *Proceedings of the 10th International Symposium on Quality Electronic Design*, ISQED'09, March. 2009

[6] Tektronix, "Jitter Measurement and Timing Analysis," Product guideline, http://www.tek.com/applications/serial_data/jitter.html

[7] T. Miyazaki, M. Hashimoto and H. Onodera, “A Performance Prediction of Clock Generation PLLs: A Ring Oscillator Based PLL and an LC

Oscillator Based PLL,"*IEICE Transactions on Electronics* 2005 E88-C (3): 437-444

[8] DFT Microsystems Canada Inc., DJ60HS Test Module/ Solutions

[9] Altera Corporation. Section I. Stratix II Device Family Data Sheet, May 2007 http://www.altera.com/literature/hb/stx2/stx2_sii5v1_01.pdf

[10] M. Hafed, D. Watkins, C. Tam, and B. Pishdad, "Massively Parallel Validation of High-speed Serial Interfaces Using Compact Instrument Modules," *Proceedings of IEEE International Test Conference*, 2006

[11] M. P. Li, *Jitter, Noise, and Signal Integrity at High-Speed*, Prentice Hall, 2007

[12] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, New York: McGraw-Hill, 1984

[13] ITRS. *The International Technology Roadmap for Semiconductors*, 2007 Edition

[14] Y. Fan and Z. Zilic, "Qualifying Serial Interface Jitter Rapidly and Cost-effectively," *Springer Journal of Electronic Testing: Theory and Applications,* Volume 26, 2010,17 pages, DOI: 10.1007/s10836-009-51315

[15] Y. Fan and Z. Zilic, "Accelerating Test, Validation and Debug of High Speed Serial Interfaces", Springer, 2011. ISBN: 978-90-481-9397-4

[16] T. Yamaguchi, "Loopback or Not," *Proceedings of IEEE International Test Conference*, 2004, p. 1434

[17] Y. Cai, B. Laquai, and K. Luehman, "Jitter Testing for Gigabit SerialCommunication Transceivers," *IEEE Design & Test of Computers*, Vol. 19, Issue 1, Jan, 2002

[18] W. Dalal and D. Rosenthal, "Measuring Jitter of High Speed Data Channels Using Undersampling Techniques," *Proceedings of IEEE International Test Conference,*1998

[19] M. Li, J.Wilstrup, R. Ressen and D. Petrich, "A New Method for Jitter Decomposition through Its Distribution Tail Fitting," *Proceedings of IEEE International Test Conference*, 1999

[20] Altera Corporation. *Mercury Programmable Logic Device Family Data Sheet*, San Jose, California, January 2003

[21] M. Hafed, N. Abaskharoun and G. W. Roberts, "A Stand-Alone Integrated Test Core for Time and Frequency Domain Measurements," *Proceedings of IEEE International Test Conference*, 2000

[22] Maxim Integrated Products, Inc. *HFTA-05.0: Statistical Confidence Levels for Estimating Error Probability*, Application Notes, 2007

[23] P. Noel, F. Zarkeshvari and T. Kwasniewski, "Recent Advances in High-Speed Serial I/O Trends, Standards and Techniques," *Proceedings of 18th Canadian Conference on Electrical and Computer Engineering*, 2005

[24] Analyzing Jitter Using a Spectrum Approach, Tektronix Application note

[25] Altera Corporation, *The Evolution of High-Speed Transceiver Technology*, White Paper, San Jose, California, 2002

[26] G. W. Roberts, F. Taenzler and M. Burns, An Introduction to Mixed-Signal IC test and Measurement, 2nd Edition, New York: Oxford University Press, 2011

[27] Jose Moreira, Hubert Wekmann, An Engineer's guide to Automated Testing of High-Speed Interfaces, Artech House, 2010

[28] Analyzing Jitter Using Agilent EZJIT Plus Software, Application Note 1563, 2008

[29] Jianmin Zhang, David J. Pommerenke, Jitter, EMC Symposium, 2006

[30] G. Hansel, K. Stieglbauer, K. Schulze and J. Moreira, "Implementation of an Economic Jitter Compliance Test for a Multi-Gigabit Device on ATE", *Proceedings of IEEE International Test Conference*, 2004

[31] Y. Cai, A. Bhattacharyya, J. Martone, A. Verma, and W. Burchanowski, "A Comprehensive Production Test Solution for 1.5GB/S and 3GB/S Serial-ATA," *Proceedings of IEEE International Test Conference*, 2005

[32] Agilent Techonologies, "Jitter Analysis:The Dual-Dirac Model, RJ/DJ, and Q-Scale", 2005. White Paper

[33] R. Stephens, "What the Dual-Dirac Model is and What It is not" Tektronix 360 Knowledge Series, 2006

[34] T. Yamaguchi, et al, "Extraction of Peak-to-Peak and RMS Sinusoidal Jitter Using an Analytic Signal Method," In Proc. of VLSI Test Symposium, 2000, pp 395 – 402.8

[35] T. Yamaguchi, et al, "Timing Jitter Measurement of 10 Gbps Bit Clock Signals using Frequency Division," In Proc. of VLSI Test Symposium, 2002, pp 207 – 212

[36] M. Li, and J. Wilstrup, “On the Accuracy of Jitter Separation From Bit Error Rate Function,” In Proc. of International Test Conference, 2002, pp 710 – 6

[37] C.-K. Ong, D. Hong, K.-T. Cheng, and L.-C. Wang, “Jitter spectral extraction for multi-gigahertz signal,” ASP Design Automation Conference, 2004

[38] D. Hong and K.-T. Cheng, “An accurate jitter estimation technique for efficient high speed i/o testing,” IEEE ATS, 2007

[39] Q. Dou and J. A. Abraham, “Jitter decomposition in high-speed communication systems,” IEEE European Test Symposium, pp. 157–162, 2008

[40] S. Erb and W. Pribyl, “An accurate and efficient method for ber analysis in high-speed communication systems,” IEEE ECCTD, 2009

[41] Grbic, S. Brown, S. Caranci, R. Grindley, M. Gusat, G. Lemieux, K. Loveless, N. Manjikian, S. Srbljic, M. Stumm, Z. Vranesic and Z. Zilic, " Design and Implementation of the NUMAchine Multiprocessor ", *Proceedings of 35th ACM/IEEE Design Automation Conference DAC `98*, pp.66-69, San Francisco, Jun. 98

[42] S. Brown, N. Manjikian, Z. Vranesic, S. Caranci, A. Grbic, R. Grindley, M. Gusat, K. Loveless, Z. Zilic and S. Srbljic, "Experience in Designing a Large-Scale Multiprocessor using Field-Programmable Devices and Advanced CAD Tools", *Proceedings of 33rd ACM/IEEE Design Automation Conference DAC `96*, pp. 24-29, Las Vegas, June 1996

[43] R. Grindley, T. Abdelrahman, S. Brown, S. Caranci, D. DeVries, B. Gamsa, A. Grbic, M. Gusat, R. Ho, G. Lemieux, K. Loveless, N. Manjikian, P. McHardy, S. Srbljic, M. Stumm, Z. Vranesic and Z. Zilic, "The NUMAchine Multiprocessor", *Proceedings of IEEE International Conference on Parallel Processing, ICPP 2000*, pp. 487-496, Toronto, ON, Aug. 2000

[44] M. Boulé, J-S. Chenard and Z. Zilic, "Debug Enhancements in Assertion-Checker Generation", *IET Computers and Digital Techniques,* Vol. 1, No. 6, pp. 669-677, Nov. 2007

[45] MH. Neishaburi and Z. Zilic, "Enhanced Reliability Aware NoC Router", *Proc. Intl. Symposium on Quality Electronic Design*, ISQED 2011, Mar. 2011, 6 pages

[46] M. Neishabouri and Z. Zilic, "Reliability Aware NoC Router Architecture Using Input Buffer Sharing", *Proceedings of Great Lakes Symposium on VLSI*, pp. 511-516, May 2009

[47] Z. Zilic, K. Radecka and A. Khazamipour, "Reversible Logic Synthesis from Non-reversible Specifications", *Proceedings of IEEE/ACM Design Automation and Test in Europe, DATE'07*, pp. 558-563, Apr. 2007

[48] B. Polianskikh and Z. Zilic, "Induced Error-Correcting Code for 2bit-per-cell Multi-Level DRAM", *Proceedings of IEEE Midwest Symposium on Circuits and Systems*, pp. 352-355, Dayton, OH, Aug. 2001

[49] Z. Zilic and Z. Vranesic, "A Multiple-Valued Reed-Muller Transform for Incompletely Specified Functions", *IEEE Transactions on Computers*, vol. 44, No. 8, pp. 1012-1020, August 1995

[50] R. Zhang, Z. Zilic and K. Radecka, “Energy-Efficient Software-Based Self-Test of Wireless Sensor Network Nodes", *Proceedings of IEEE VLSI Test Symposium*, *VTS06* , pp. 191-196, Apr. 2006

[51] K. Radecka and Z. Zilic, "Arithmetic Transforms for Verification of Sequential Datapaths", *Proceedings of IEEE International Conference on Computer Design, ICCD*, pp. 348-353, Austin, TX, Sep. 2001