A Versatile FPGA-based High Speed Bit Error Rate Testing Scheme

Yongquan Fan

Department of Electrical and Computer Engineering
McGill Univerity, Montreal

August 2003

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering

Abstract

FPGAs have witnessed an increased use of dedicated communication interfaces. With their increased use, it is becoming critical to test and properly characterize all such interfaces. Bit error rate (BER) characteristic is one of the basic measures of the performance of any digital communication system. Traditionally, BER is evaluated using Monte-Carlo simulations, which are very time-consuming. Though there are some BER test products, none of them includes channel emulator. To overcome these problems, this thesis presents a scheme for BER testing in FPGAs, with a few orders of magnitude speedup compared to Monte-Carlo method. This scheme consists of two intellectual property (IP) cores: the BER tester (BERT) core and the additive white Gaussian noise (AWGN) generator core. Two challenging testing cases are successfully conducted using the testing scheme. We demonstrate through case studies that the proposed BER testing solution exhibits advantages in speed and cost compared with the existing solutions.

Résumé

Au cours des dernières années, les architectures FPGA ont connus une augmentation importante de l'utilisation d'interfaces dédiées pour la communication. Il devient donc crucial de tester et de caractériser convenablement ces types d'interface. La performance d'un système de communication numérique est généralement mesurée par le Taux d'erreurs sur les bits (TEB). Traditionnellement, le TEB est évalué à l'aide de simulations logicielles par la méthode de Monte-Carlo. Malheureusement, cette méthode est reconnue pour être très coûteuse en temps. Bien qu'il existe quelques appareils de test mesurant le TEB, aucun de ceux-ci n'offre l'émulation de canal. Pour palier à ces problèmes, ce mémoire propose une technique pour effectuer le test du TEB (à l'aide d'un) FPGA. Cette dernière permet de réduire considérablement le temps d'exécution comparativement à la méthode utilisant les simulations de Monte-Carlo. La solution proposée est composée de deux blocs de propriété intellectuelle (PI): le bloc de test TEB (TEBT) ainsi que le bloc générateur de bruit blanc Gaussien additif (BBGA). Deux exemples de test ont été exécutés avec succès, en utilisant notre technique de test. Nous avons aussi démontré, à l'aide d'études de cas, que la solution proposée de test de TEB offre des avantages en temps et en coûts comparativement aux solutions déjà existantes.

Acknowledgments

I would, first and foremost, like to thank my supervisor Professor Zeljko Zilic. His guidance and support throughout my Master's studies have been greatly appreciated. He is very insightful in directing research topics and is also willing to listen to students' opinions. Through the thesis work, he always guided me at critical points and provided me lots of up-to-date hardware and software resources that greatly facilitated the work. What I have learned under his direction will greatly benefit me in my future studies and career development. I am also very grateful for his financial support that allowed me to focus on my studies during my two years at McGill.

Maher Hamdi of Arrow Electronics is acknowledged for providing us a Mercury board and demonstrating us how to use the CDR functions of Mercury devices. The final designs of the thesis work are implemented in this board. I acknowledge Altera Corporation for providing us an 8B10B IP core for our CDR testing experiments.

I also wish to thank my wife Ji Lei for her love, encouragement and support. She always cooked the best meal in the world for me. She took almost all the housework so I could spend more time than others on my studies. My thanks also go to my parents, sisters and brothers for their patience to endure my second scholastic stay.

Besides the research experience I have obtained at McGill, the experience as a teaching assistant is also invaluable. I would like to thank Prof. Katarzyna Radecka, Prof. Miguel Marin, Prof. James J. Clark and Prof. Mourad El-Gamal for their guidance and collaboration to my TA work.

The MACS lab is a breeding ground for quality students and research. I would like to express my thanks to Prof. Gordon Roberts, Prof. Nick Rumin and Prof. Radu Negulescu

for their excellent course instruction. Finally, my thanks go to all the students in MACS lab who provided me help and enjoyment. For this thesis, I especially thank Jean-François Boland for translating the abstract into French, and Kahn-Li Lim for proofreading.

Table of Contents

Abstract ..... **i**

Résumé ..... **ii**

Acknowledgments ..... **iii**

Table of Contents ..... **v**

List of Figures ..... **viii**

List of Tables ..... **x**

Chapter 1 - Introduction **1**

1.1 Motivation 1

1.1.1 FPGA Perspective 1

1.1.2 BER Testing Perspective 2

1.2 Thesis Outline 4

Chapter 2 - Background **7**

2.1 BER Background 7

2.1.1 Factors Affecting BER 7

2.1.2 BER and SNR 9

2.1.2.1 BER Performance of Digital Baseband 9

2.1.2.2 BER Performance of Modulated Transmission 15

2.2 BER Testing 18

2.2.1 AWGN Channel Model 18

2.2.2 AWGN Theoretical Properties 19

2.2.3 Software Simulation 23
2.2.4 Hardware Emulation 25
2.2.5 BER Confidence Level 27
2.3 Proposed BER Testing Scheme 30

Chapter 3 - BERT Core Design 32

3.1 Overview 32
3.2 Serial BERT Design 33
3.2.1 PRBS generation 34
3.2.2 Testing Synchronization 35
3.2.3 Bit Slip Detection 37
3.2.4 State Control and BER Calculation 43
3.2.5 Output Display 44
3.3 Parallel BERT Structure 44
3.3.1 Design Strategies 44
3.3.2 System Architecture 45
3.3.3 Function Verification 47
3.4 Synthesis Results 49

Chapter 4 -AWGN Core Design 50

4.1 AWGN Generation Method Overview 50
4.1.1 Existing Methods 50
4.1.1.1 CLT Method 50
4.1.1.2 Box-Muller Method 51
4.1.1.3 Mixed Method 52
4.1.1.4 Cellular Automata Based Method 53
4.1.2 Our Method 54
4.2 Generating Random Variables 55
4.2.1 One Bit Random Number Generator 55
4.2.2 Multiple-Bit Random Number Generator 58
4.3 Gaussian Variable Generation 59
4.3.1 Implementing a Single Generator 60
4.3.1.1 Generating V_1 and S 60
4.3.1.2 FIFO Implementation 61
4.3.1.3 Generating W 63

4.3.1.4 Generating Outputs 64
4.3.2 Implementing Two Generators 64
4.3.3 Accuracy Improvement 65
4.4 Experimental Results 66
4.4.1 Experimental Statistical Properties 66
4.4.2 Synthesis Results 69
4.4.3 Comparison 69

Chapter 5 – Case Studies 71

5.1 CDR Circuitry Testing 71
5.1.1 Significance of the Serial Communication Interface 71
5.1.2 Structure of the Serial Communication Interface 74
5.1.3 Mercury Gigabit Transceiver 76
5.1.4 Testing Setup 79
5.2 Baseband Transmission Testing 84
5.2.1 Baseband Signal Formats 84
5.2.2 SNR Setting 87
5.2.3 Testing Setup and Results 88

Chapter 6 - Conclusions 93

6.1 Conclusions 93
6.2 Future Work 93

References 95

List of Figures

Figure 1-1: Block Diagram of a Digital Communication System 3
Figure 2-1: Basic Elements of a Digital Communication System 7
Figure 2-2: Binary Matched Filter Receiver 10
Figure 2-3: Probability Densities of y 12
Figure 2-4: BER vs SNR for Baseband Transmission 15
Figure 2-5: Probability of a Symbol Error for PSK Signals 17
Figure 2-6: AWGN Communication Channel Model 19
Figure 2-7: The PDF of a Gaussian Variable 21
Figure 2-8: The CDF of a Gaussian Variable 22
Figure 2-9: Software Model of a BPSK Communication System [16] 24
Figure 2-10: Dialog Box of AWGN Channel Simulation Block [16] 25
Figure 2-11: Graph of the Binomial Distribution ($n = 10^8$, $p = 10^{-7}$) 28
Figure 2-12: Test Time vs Confidence Level (*CL*) 30
Figure 2-13: Proposed BER Testing Scheme 31
Figure 3-1: The Structure of the Serial BERT 33
Figure 3-2: The Circuit for PRBS Generation 34
Figure 3-3: Synchronization Circuits (n =3, DUT delay = 3) 35
Figure 3-4: Simulation Waveforms of the Synchronization Circuit 37
Figure 3-5: Circuit for Bit Slip Detection 39
Figure 3-6: Simulation Waveforms of Bit Slip Detection 42
Figure 3-7: Waveforms of the Serial BERT (Error Burst and Bit Slip) 42
Figure 3-8: The Structure of the Parallel BERT 45
Figure 3-9: Sub-module Circuit of the Parallel BERT 46
Figure 3-10: Waveforms of the Parallel BERT (Load → Measure) 48
Figure 3-11: Waveforms of the Parallel BERT (Error Burst and Word Slip) 49

Figure 4-1: Block Diagram of Mixed Method [17] 52
Figure 4-2: Transformation from Random Variables to Gaussian Variables 53
Figure 4-3: A True 1-bit Random Variable Generator 56
Figure 4-4: LFSR-based Pseudo Random Number Generator 58
Figure 4-5: Block Diagram of a Single AWGN Generator 60
Figure 4-6: Structure of the Synchronizing FIFO 62
Figure 4-7: Plot of Function $W(S)$ 63
Figure 4-8: Block Diagram of Two AWGN Generators 65
Figure 4-9: New CLT Method 66
Figure 5-1: CDR Transmission Mechanism 72
Figure 5-2: Applications of Multiple Transceivers 72
Figure 5-3: Current-Mode LVDS Driver [52] 73
Figure 5-4: Block Diagram of a Transceiver 74
Figure 5-5: HSDI Circuitry Block Diagram 77
Figure 5-6: HSDI PLL Block Diagram 78
Figure 5-7: CRU Block Diagram 79
Figure 5-8: Testing Setup for the Mercury HSDI Transceiver 80
Figure 5-9: 8B10B Coding Process 82
Figure 5-10: Block Diagram of Word Alignment 83
Figure 5-11: Baseband Signal Formats 85
Figure 5-12: The Structure of a NRZI Encoder and Decoder 86
Figure 5-13: BER Testing Setup for NRZ Digital Baseband 88
Figure 5-14: BER Testing Setup for NRZI Digital Baseband 90
Figure 5-15: Plot of Measured BER and Theoretical BER for Digital Baseband 91

List of Tables

Table 1-1: High Frequency Serial Communications Test Requirements [4] 2
Table 2-1: An Example of BER Estimation (CL=99% and $p = 10^{-10}$) [27] 29
Table 3-1: The Outputs of the First 16 Clock Cycles in Figure 3-3 36
Table 3-2: The Output of the Comparator 40
Table 3-3: Synthesis Results of the BERT 49
Table 4-1: Sample Recurrence Equations 57
Table 4-2: Performance of our AWGN Generator 68
Table 4-3: Q(x) Relative Error of our AWGN Generator 69
Table 4-4: Synthesis Results of the AWGN Generator 69
Table 4-5: Comparison of our Method with Mixed Method 70
Table 5-1: Mercury Device Family 76
Table 5-2: BER Measurements for Digital Baseband 89

Chapter 1 - Introduction

1.1 Motivation

1.1.1 FPGA Perspective

As FPGAs and the associated design software have evolved to include multimillion gates, specialized communication interfaces, such as clock data recovery (CDR) circuitry and enhanced phase-locked loops (PLLs), are increasingly being included in FPGAs for high-speed communication applications. Additionally, the performance and capacity improvements of FPGAs give their users sufficient processing power to implement a wide range of communication interfaces, including various wireline and wireless modulation schemes, modern turbo error correcting codes and spread spectrum schemes. In consequence, FPGA-based designs are more and more widely used in digital communication systems to replace ASIC implementations.

Among these FPGA new features, high-frequency serial communication interfaces are probably the most important. They are mostly realized using CDR circuits to extract the clock from a data stream. Specialized CDR circuitry in Altera Mercury devices provides data rates of up to 1.25 gigabits per second (Gbps) per channel, and total CDR bandwidth of 45 Gbps [1]; the rate increases to 3.125 Gbps per channel in Altera Stratix GX devices. Lattice Semiconductor's Field Programmable System Chip (FPSC) includes 10 Gbps line interfaces in ORLI10G and 1.5625 Gbps per channel CDR subsystems in ORT82G5 [2]. Xilinx's Virtex-II Pro FPGAs provide up to twenty-four 3.125 Gbps full duplex Rocket I/O transceivers, with an aggregate baud rate of up to 75 Gbps [3]. In consequence, FPGA-based serial communication interfaces are being widely adopted into backplane applications, short and long-haul communications, mass storage access networking, and computer peripherals. However, the testing of gigabit-rate serializer and deserializer (SerDes) devices is still challenging. According to International Technology Roadmap for Semiconductors (ITRS), the technology requirements for high-frequency serial

communication test are continuously putting pressure on the test industry. Table 1-1 shows the device requirements for history and projections [4].

Table 1-1: High Frequency Serial Communications Test Requirements [4]

Year of Production		2001	2002	2003	2004	2005	2006	2007
High-performance-level serial transceivers								
Serial data rate (Gbits/s)		10	10	40	40	40	40	40
Max reference clock speed (MHz)		667	667	2500	2500	2500	2500	2500
High-integration-level backplane and computer I/O								
Serial data rate (Gbits/s)	Production	2.5	3.125	3.125	10	10	40	40
	Introduction	3.125	--	10	--	40	--	--
Max reference clock speed (MHz)	Production	166	166	166	667	667	2500	2500
	Introduction	--	--	667	--	2500	--	--

Currently, testing functionality of high-performance serial interfaces must be done by using expensive, stand-alone pattern-generators and bit-error-rate detectors. This approach is very time consuming and the cost is hence high. There is an urgent need to develop testing equipment to characterize the performances of the serial communication interfaces, along with other communication interfaces. This thesis proposes a low cost scheme that uses existing FPGA resources to test the functionality of serial interfaces.

1.1.2 BER Testing Perspective

Bit error rate (BER) is the ratio of the number of incorrect to the total number of received bits. For qualifying the reliability of an entire digital communication system from “bits in” to “bits out”, BER characteristic is the fundamental measure of the performance of a digital communication system.

As shown in Figure 1-1, a digital communication system consists of a transmitter, a channel, and a receiver. The transmitter changes the raw information (sequences of binary digits) into a format that is matched to the characteristics of the channel. Depending on applications, the transmitter may consist of a source encoder, an encryptor, a channel encoder, a carrier modulator or a spread-spectrum modulator.

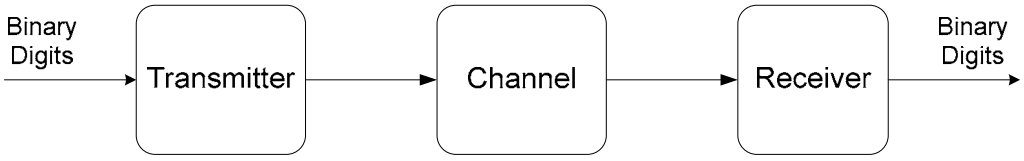


Figure 1-1: Block Diagram of a Digital Communication System

The receiver accepts the signal from the channel and recovers the transmitted binary digits. The recovered digits are usually processed to permit interfacing with the final destination, such as a computer monitor or the human ear.

The channel is the physical medium used to send the signal from the transmitter to the receiver. The medium may be the air, wire lines, optical fiber and so on. One essential feature of the communication channel is that the transmitted signal is corrupted in a random manner by a variety of possible mechanisms, such as additive thermal noise generated by electronic devices; man-made noise, e.g., automobile ignition; and atmospheric noise, e.g., electrical lightning discharges during thunderstorms.

In a digital communication system, either the channel or the communicating devices (sending and/or receiving end) can introduce distortion or cause errors. As modern communication interfaces are quite complex, besides inherent device and timing imperfections, the correctness and performance of communication interfaces depend on many design choices, such as types of waveforms used to transmit the information over the channel, the transmitter power, the characteristics of the channel (i.e., the amount of noise, the nature of the interference), and the method of demodulation and decoding. At macroscopic level, BER is a fundamental measure of the communication system performance, whose importance has been widely recognized [5]. As a measure of how well the overall communication system performs, BER is the probability of a bit-error at the output of the receiver, compared with the input of the transmitter.

In the development and manufacturing of a digital communication system, it is critical to quickly and precisely test the BER performance at the receiving end. In many cases, a self-test or an integrated test is necessary. In practice, the complexity and nonlinearity of the communication system prohibit us from obtaining the closed-form expression for BER. Traditionally, BER has been evaluated using software simulations, where the real communication system (transmitter, channel and receiver) is emulated by its software model and its statistical behavior is estimated by transmitting thousands of bits in the software model. Although software simulations are easy to set up, they are time consuming to conduct. Hence, the simulation time needed to obtain reliable estimation of the BER greatly limits the exploration of the solution space for optimizing the design of digital communication interfaces.

Hardware-based solution is commonly 100,000 to 1 million times faster than the best simulation software at the same abstraction level [6]. While there exist hardware-based products available for BER testing [7], [8], [9], [10], none of them includes communication channel emulators, which are necessary in testing the BER performance of a digital communication system. Such testers are not convenient to evaluate the BER performance of a digital communication system.

To overcome this problem, this thesis proposes a novel scheme for BER testing in FPGAs. The test scheme mainly consists of two IP cores: a BERT core and an AWGN core. The BERT core is used for BER testing, while the AWGN core is used for communication channel emulation. The proposed scheme can be used to test the performance of a wide range of communication systems, including native CDR interfaces, as well as various user-defined modulation/demodulation, spread spectrum and error correcting code cores. The proposed BER testing scheme is easy to set up and is a few orders of magnitude faster than software simulations.

1.2 Thesis Outline

In Chapter 2, the background of BER performance and the methods of testing BER under the presence of noise are first introduced. Hardware emulation exhibits speed advantages

over software simulations. After examining currently available hardware-based BER testing solutions for a digital communication system and their drawbacks, we present a low-cost, high-speed BER testing scheme at the end of Chapter 2 [12]. The whole testing solution can be implemented in a single FPGA device.

Chapter 3 presents the function and detailed implementation of the BERT core. The core can be configured for either a serial or parallel interface, depending on the interface requirements of a design under test (DUT). The structure of the serial BER tester is given first. Then, the structure of the parallel BERT is derived from the serial one. The BERT core is capable of sending pseudo random bit sequences (PRBSs) for a serial BERT or pseudo random word sequences (PRWSs) for a parallel BERT to a DUT, and then giving the BER performance of the DUT by comparing the output from the DUT with the PRBSs or PRWSs. This core can automatically keep synchronization with the DUT regardless of the delay of the DUT. Simulation and synthesis results targeting Altera Mecury FPGAs are presented in the last part of this chapter. The core is verified by simulations and by running real tests.

In Chapter 4, an overview of methods for AWGN generation is first given. To overcome the disadvantages of existing methods, a novel method is presented for AWGN generation [11]. Then, we reveal the detailed implementation of the proposed AWGN core and its performance compared with existing implementations. This core is based on Polar method and a novel Central Limit Theorem (CLT) method implementation. With Polar method, it is convenient to build a single or two independent AWGN generators with high speed and high precision. The novel CLT method can further smoothen the variance of a Gaussian distribution without speed penalty, while the traditional CLT method exhibits speed penalty. The experimental results show that the proposed AWGN core is suitable for channel emulation.

As examples, Chapter 5 presents two applications of the proposed BERT core and AWGN core: one is testing a gigabit transceiver included in Altera Mercury FPGAs; the other is testing the BER performance of a baseband transmission system under different

SNR conditions. All the work is done on an Altera Mercury board. The two cases further validate the two cores. They also demonstrate that the proposed BER testing scheme has advantages in speed and cost compared with traditional solutions.

Chapter 6 summarizes the work done in this thesis. The importance of the thesis and some possible future research directions are also presented at the end of Chapter 6.

Chapter 2 - Background

2.1 BER Background

2.1.1 Factors Affecting BER

Figure 2-1 shows the basic elements of a digital communication system. Almost every element of the system can affect the BER performance. We now briefly describe the function of each component and how the components may affect the BER performance of the system.

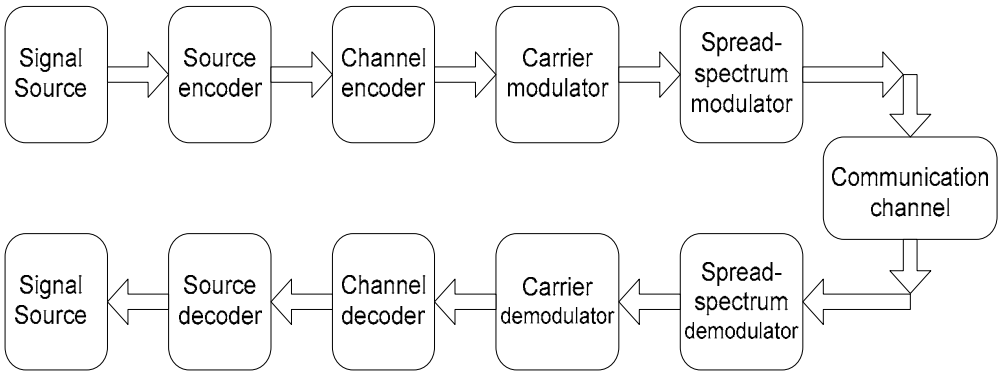


Figure 2-1: Basic Elements of a Digital Communication System

The s*ignal source* refers to the original information that we intend to transmit in the communication system. The information may be either an analog signal or a digital signal.

The *source encoder* efficiently converts the signal source into a sequence of binary digits (message). One of the important objectives of source encoder is to represent the message by as few binary digits as possible, subject to the need to reconstruct the input adequately at the output. Hence, source encoding is data compression procedure.

The *channel encoder* is used to introduce some redundancy in the binary information sequence in a controlled manner. The redundancy is then used at the receiver to overcome the effects of noise and interference encountered in the transmission of the signal through the channel. Repeating the messages is the simplest form of redundancy. However, it is not efficient. In a digital communication system, forward error correction (FEC) is often used, in which encoding permits error correction without the necessity of the receiver asking the transmitter for additional information.

The *carrier modulator* modifies the channel encoder output signal in a manner that matches the characteristics of the channel. It usually produces an analog waveform that permits multiple use of the channel by several transmitters and that is transmitted efficiently through the channel. In a digital communication system, there are two kinds of modulations: binary modulation, where each bit from the channel encoder is transmitted separately, and *M*-ary modulation ($M > 2$), where *b* coded information bits are transmitted at a time by using $M = 2^b$ distinct waveforms $s_i(t)$, $i = 0, 1, \ldots, M\text{-}1$, one waveform for each of the 2^b possible *b*-bit sequence. The modulated waveform consists of signal segments corresponding to the discrete symbols at its input.

Spread spectrum is a technique for providing some immunity to frequency-selective effects such as interference and fading. A signal is spread over a wide range of frequencies so that single-tone interference affects only a small portion of the signal. Spread spectrum also has other advantages, related to simplified methods of sharing a channel among multiple users.

Communication channel is the main source of errors. Its function and characteristics have been discussed in Chapter 1.1.2

In Figure 2-1, the blocks before the communication channel compose the transmitter; the blocks after the communication channel compose the receiver. The function of each block in the receiver is simply a mirror image of the function of the corresponding block in the transmitter. The receiver must undo each operation that is performed at the transmitter.

As can be seen from the above discussion, the communication channel can introduce noise and cause some errors, while channel encoding mechanisms are needed to eliminate or reduce the error. It can also be seen that various modulation schemes also provide a mechanism to improve the performance of a digital communication system. A measure of how well the demodulator and decoder perform is the rate with which bit errors occur in the decoded sequence. In general, BER performance is determined by the code characteristics, the types of waveforms used to transmit the information over the channel, the transmitter power, the characteristics of the channel (e.g., the amount of noise, the nature of the interference), and the method of demodulation and decoding.

2.1.2 BER and SNR

Among the factors discussed in Chapter 2.1.1, noise is the main enemy of BER performance. The noise introduced by a communication system is usually described by a Gaussian probability density function. Representing the function mathematically makes it possible to predict the BER performance of the system.

As the ratio of the number of incorrect and the total number of received bits, BER is related both theoretically and practically (by measurements) to the signal-to-noise ratio (SNR). SNR is the fundamental input quantity that determines the channel capacity *C* for a given bandwidth *B,* according to the fundamental Shannon law:

$$C = B\log_2(1+SNR)$$

In practice, communication system designers balance between bandwidth and SNR to maximize the channel capacity for an acceptable BER performance. There are several types of communication systems in which this balancing act is played in different ways.

2.1.2.1 BER Performance of Digital Baseband

In baseband transmission, the data and clock are transmitted as digital waveforms, and different waveforms are used to transmit 1’s and 0’s. Baseband schemes, such as commonly used non-return-to-zero (NRZ) CDR encoding, combine clock and data signals on the transmitting side and decouple them at the receiver. Careful timing

extraction leads to a reduction in the number of transmission errors, which is equivalent to an increase in the system SNR.

In a baseband transmission system, the receiver in the system has two tasks: one is to keep synchronization, sampling the received bit stream at an appropriate time point and speed; the other is to decide whether the sampled value represents a binary one or zero. Assuming that synchronization is always kept, the following discussion introduces the "one or zero" decision principle and evaluates the BER performance of a binary matched filter receiver in digital baseband communications.

In order to recover the signal from the background noise in a receiver, we may wish to maximize the output SNR without regard to preserving the shape of the signal waveform. A matched filter is a linear system that significantly alters the shape of both the signal and the noise in a way that increases the SNR. Figure 2-2 shows the structure of a binary matched filter receiver [13], [14].

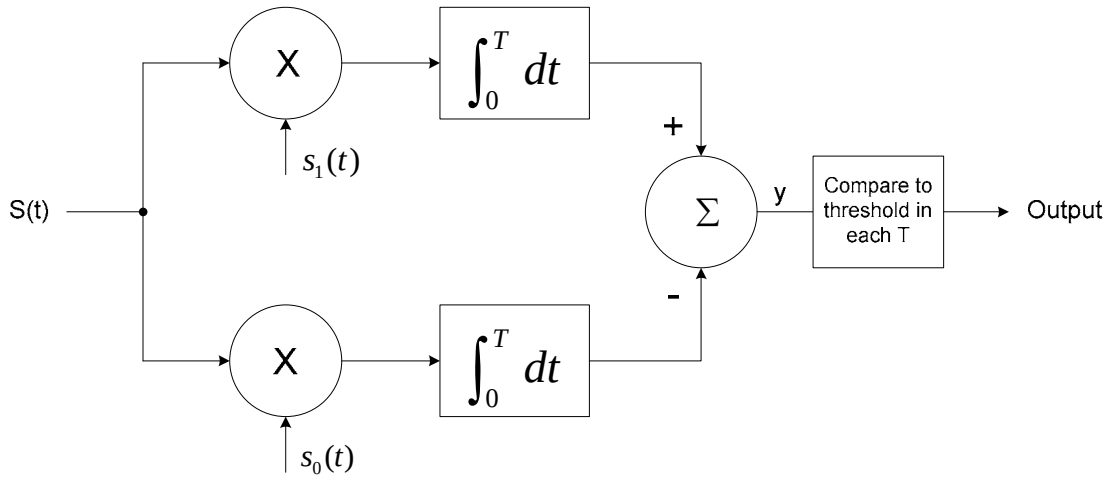


Figure 2-2: Binary Matched Filter Receiver

The receiver includes two filters, and each of the filters consists of a multiplier and an integrator. The receiver compares the output of the two filters, one matched to $s_0(t)$ and the other matched to $s_1(t)$. The difference of these two filters is then compared to a threshold value. In the case the threshold value is zero, the receiver is detecting which

matched filter output is larger. If the top filter output is larger than the bottom, *y* is positive; if the bottom filter output is larger, *y* is negative.

The output of each integrator is a number composed of a deterministic part (due to the signal) and a random part (due to the additive noise). The additive noise is assumed to be zero-mean Gaussian and its frequency spectrum is flat. Suppose the input signal to the receiver is $s_i(t) + n(t)$, where *i* is either zero or unity, depending on which signal is being transmitted. The input to the comparator is then given by

$$y = \int_0^{T_s} s_i(t)[s_1(t) - s_0(t)]dt + \int_0^{T_s} n(t)[s_1(t) - s_0(t)]dt$$

The average value of *y* is obtained by adding together the average values of the two integrals, where the average of the second integral is zero since the noise is zero mean. Therefore, the mean value of y is

$$m_y = \int_0^{T_b} s_i(t)[s_1(t) - s_0(t)]dt$$

The mean value depends upon which signal is being transmitted.

The variance of y is the expected value of the square of the difference between y and its mean. The variance is

$$\begin{aligned}\sigma_y^2 &= E\{[y - m_y]^2\} \\ &= E\{[\int_0^{T_b} n(t)[s_1(t) - s_0(t)]dt]^2\} \\ &= E\{\int_0^{T_b}\int_0^{T_b} n(t)n(v)[s_1(t) - s_0(t)][s_1(v) - s_0(v)]dtdv\}\end{aligned}$$

The expected value of a sum is the sum of the expected value, so the expected value symbol can be moved within the range of the integral signs. The only random part of the integral is that containing noise *n*. As the noise is assumed to be white with power spectral density $G_n(f) = N_o/2$, the autocorrelation of the noise is the inverse Fourier transform of the power spectrum, or $R_n(t) = N_o\,\delta(t)/2$. Taking this and $E\{n(t)n(v)\} = R_n(t - v)$ into account, we have

$$\sigma_y^2 = \{\int_0^{T_b}\int_0^{T_b} \frac{N_o}{2}\delta(t-v)[s_1(t)-s_0(t)][s_1(v)-s_0(v)]dtdv\}$$

By the sampling property of the impulse, the above equation is equal to

$$\sigma_y^2 = \frac{N_0}{2}\int_0^{T_b}[s_1(t)-s_0(t)]^2\,dt$$

As can be seen, this result is independent of which signal is sent.

Based on the mean and variance of *y*, the probability density of *y*, under the assumption that a 0 (m_0) or 1 (m_1) is being transmitted, is drawn and shown in Figure 2-3. The probability density fits into one of the two probability density functions labeled with $p_0(y)$ and $p_1(y)$, depending on what is being transmitted. The two functions have the same variances but different mean values.

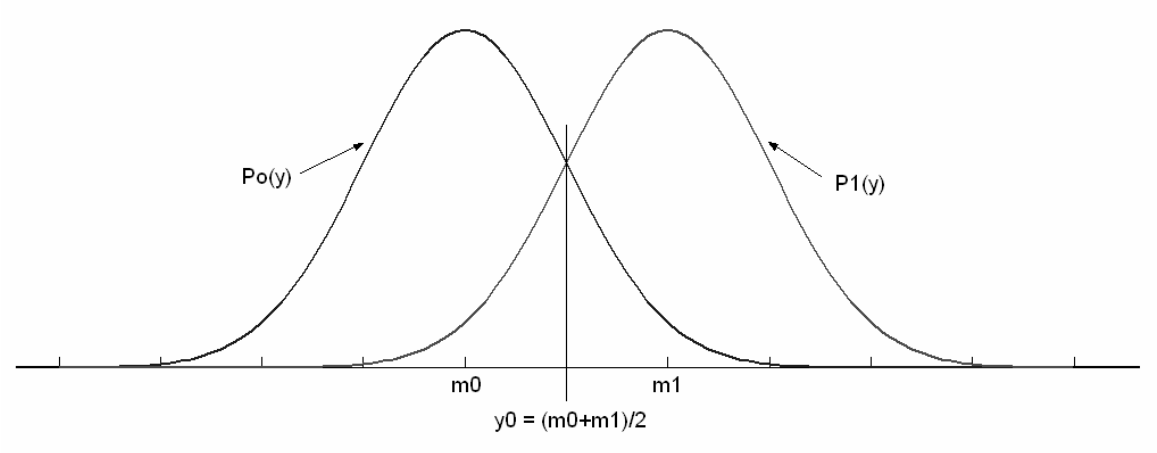


Figure 2-3: Probability Densities of y

In the comparator, the threshold is chosen as the point at which the two probability density functions cross. It is labeled as y_0 on the diagram. If *y* is greater than y_0, we assume that $s_1(t)$ is being sent; if *y* is less than y_0, we assume $s_0(t)$ is being sent. Due to the symmetry, y_0 is the midpoint between the mean values of the two probability density functions. Thus,

$$y_0 = \frac{m_1 + m_0}{2}$$

$$= \frac{1}{2}\int_0^{T^b} [s_1(t) + s_0(t)][s_1(t) - s_0(t)]dt$$

$$= \frac{1}{2}\int_0^{T_b} [s_1{}^2(t) - s_0{}^2(t)]dt$$

As the integral of the square of the signal is the signal energy over the bit period, the threshold is

$$y_0 = \frac{E_1 - E_0}{2}$$

where E_1 and E_0 are denoted as the energy for the two signals $s_0(t)$ and $s_1(t)$, respectively.

Next, the probability of error for the binary matched filter receiver is evaluated. The probability density of the comparator input follows one of the two Gaussian curves shown in Figure 2-3. It follows the curve labeled $p_0(y)$ if a 0 is being transmitted, and follows the curve labeled $p_1(y)$ if a 1 is being sent. The probability of mistaking a transmitted 1 for a 0 is the integral of $p_1(y)$ between $[-\infty, y_0]$, and the probability of mistaking a transmitted 0 for a 1 is the integral of $p_0(y)$ between $[y_0, \infty]$. Hence, the error probability is given by the area under the tail of the Gaussian density function.

Assuming that the two signals, $s_0(t)$ and $s_1(t)$, have equal energy, the probability of an error is

$$P_e = \frac{1}{\sqrt{2\pi}\sigma}\int_0^{\infty} \exp[\frac{(y - m_0)^2}{2\sigma^2}]dy$$

$$= Q(\sqrt{\frac{E(1-\rho)}{N_o}})$$

where Q( ) is the Q-function (its definition will be discussed in Chapter 2.2.2), E is the average energy of the two signals, ρ is the correlation coefficient of the two signals, and N_0 is the noise power per Hz . E, N_0 and ρ are defined as

$$E = \frac{E_0 + E_1}{2}$$

$$N_o = \frac{\sigma^2}{f_m} \quad (f_m \text{ is the bandwidth})$$

$$\rho = \frac{\int_0^{T_b} s_0(t)s_1(t)dt}{E}$$

As can be seen from the equation about P_e, the BER is determined by three factors: the average energy per bit E, the correlation of the two signals ρ, and the noise power per hertz N_0. The BER decreases when either ρ decreases, or E/N_o increases. The above principle applies to the general case of unequal energies.

In order to investigate the relationship between BER and E/N_o, we consider three cases where each correlation coefficient ρ is different. The first case assumes that $s_0(t) = s_1(t)$. The correlation coefficient is then $\rho = 1$, and BER becomes

$$P_e = Q(0) = 1/2$$

This result is easily explained, since the same signal is used to transmit both 0 and 1, which means no information is supplying and the receiver can only randomly guess the information.

The second case assumes that $s_0(t) = -s_1(t)$. The correlation is then $\rho = 1$, and BER is a minimum at

$$P_e = Q\left(\sqrt{\frac{2E}{N_o}}\right)$$

The third case assumes that $s_0(t) = 0$ and $s_1(t) = 1$. The correlation is then $\rho = 0$, and BER becomes

$$P_e = Q\left(\sqrt{\frac{E}{N_o}}\right)$$

Figure 2-4 shows the relationship between the BER and the signal to noise ratio, E/N_o, for the above three values of correlation: -1, 0 and 1. Note that the abscissa is in dB, which is 10 times the logarithm of E/N_o.

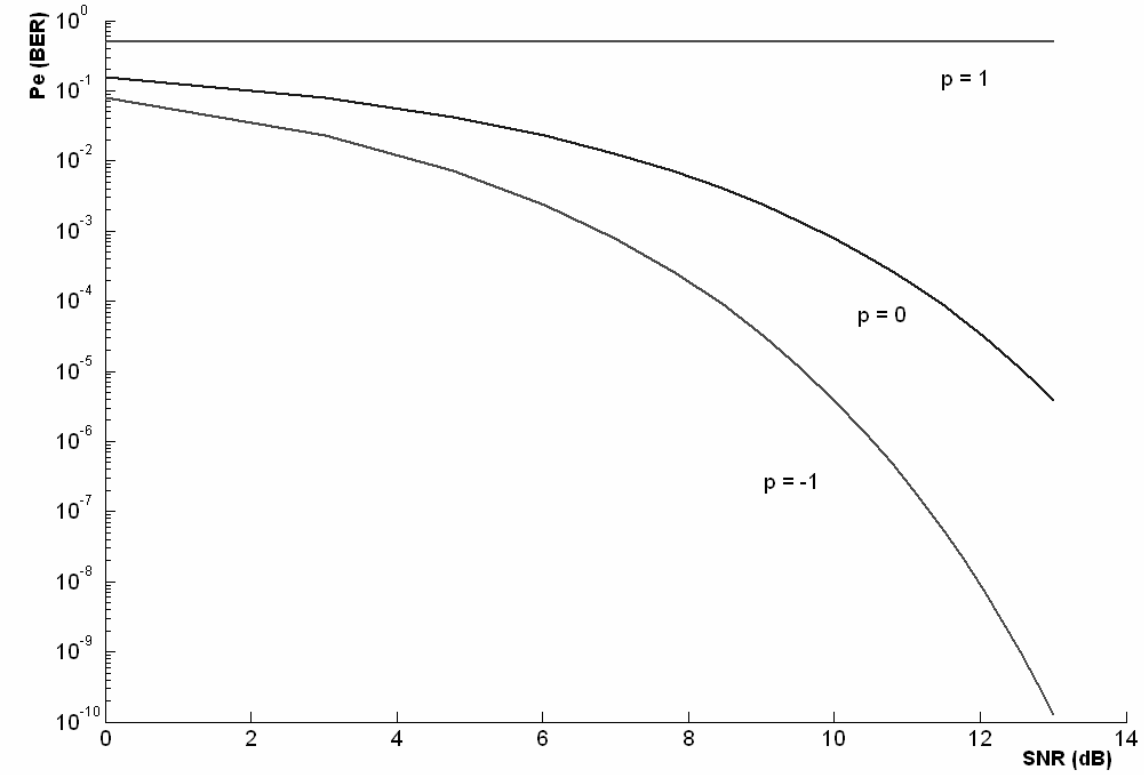


Figure 2-4: BER vs SNR for Baseband Transmission

2.1.2.2 BER Performance of Modulated Transmission

Another class of communication systems employs modulation schemes for communication over a given portion of spectrum. The modulator at the transmitter performs the function of mapping the digital sequence into sinusoidal signal waveforms. The BER performance of receivers varies widely, depending on the modulation schemes. As an example of the relationship between BER and SNR for different modulation schemes, we briefly discuss the probability of an error for *M*-ary Phase Shift Keying (PSK) modulation.

In communication systems where carrier phase tracking is possible (coherent demodulation), PSK is often used. Although many other modulation systems are in use,

PSK systems are very common. In *M*-ary PSK modulation scheme, the frequency of the carrier stays constant while the phase shift takes on one of *M* constant values. Digital phase-modulated signal waveforms may be expressed as

$$s_m = g(t)\cos[2\pi f_c t + \frac{2\pi}{M}(m-1)], \qquad 1 \le m \le M, \qquad 0 \le t \le T$$

For binary phase shift keying (BPSK) modulation (M = 2), the two signals $s_1(t)$ and $s_2(t)$ are antipodal, hence the error probability is

$$P_2 = Q(\sqrt{\frac{2\varepsilon_b}{N_o}})$$

When M = 4, M-ary PSK becomes Quadrature Phase Shift Keying (QPSK). In such modulation schemes, as the baseband digital signal is modulated by a complex exponential (sine and cosine waves), two real-valued data streams appear and have to be processed separately. They are referred to as *I* channel (In phase) and *Q* channel (Quadrature). The symbol error probability for QPSK is

$$P_4 = 2Q(\sqrt{\frac{2\varepsilon_b}{N_o}})[1-\frac{1}{2}Q(\sqrt{\frac{2\varepsilon_b}{N_o}})]$$

For M > 4, the symbol error probability does not reduce to a closed-form equation and must be evaluated numerically. The detailed analysis of the probability of an error for M-ary PSK can be found at [13], where the probability of a symbol error for PSK signal is approximated by

$$P_M \approx 2Q(\sqrt{2k\frac{\varepsilon_s}{N_o}}\sin\frac{\pi}{M})$$

with $k = \log_2 M$. This approximation is good for all values of *M*.

Figure 2-5 illustrates the symbol error probability as a function of the SNR per bit for M = 2, 4, 8, 16, and 32. This figure clearly illustrate the penalty in SNR per bit as M increases beyond M = 4. For example, at $P_M = 10^{-5}$, the difference between M = 4 and M = 8 is

approximately 4 dB, and the difference between $M = 8$ and $M = 16$ is approximately 5 dB. For large values of M, doubling the number of phases requires an additional 6 dB/bit to achieve the same performance [13].

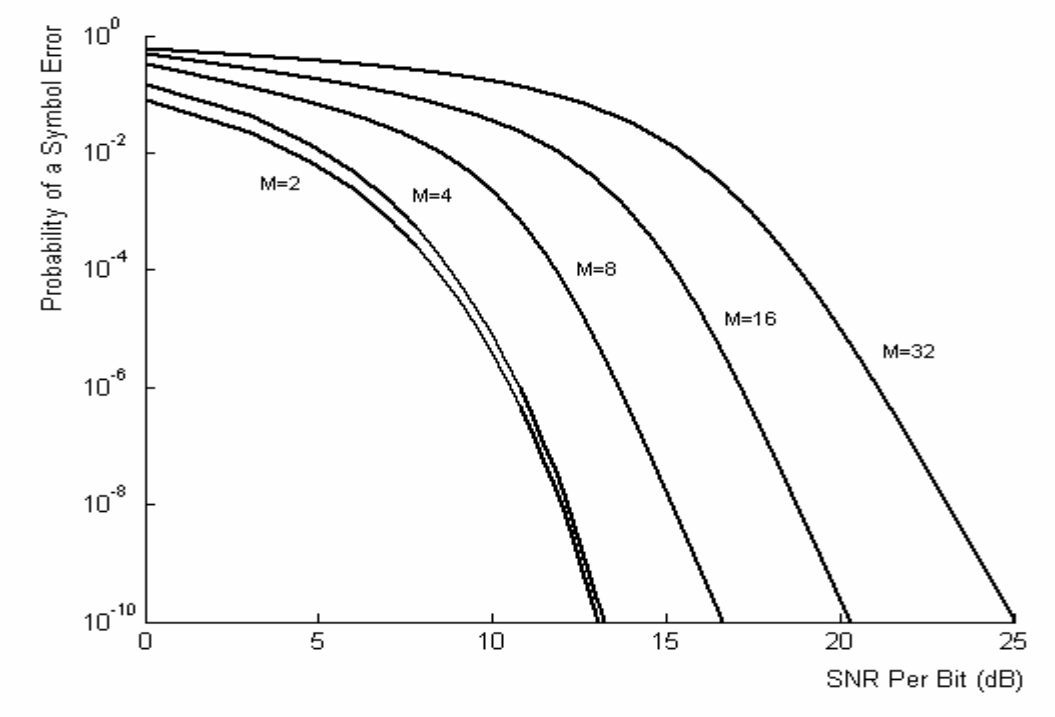


Figure 2-5: Probability of a Symbol Error for PSK Signals

The relationship between the symbol error probability and the bit error probability is not straightforward for M-ary PSK due to its dependence on the mapping of k-bit symbols into the corresponding signal phases. When a Gray code is used in the mapping, the equivalent bit error probability for M-ary PSK can be approximated as

$$P_b \approx \frac{1}{k} p_M$$

Spread spectrum technique is yet another implementation of the Shannon law by which the transmitted signal bandwidth B is much greater than the information bandwidth C. This excess bandwidth is used for "coding gain" to protect the signal from interference caused by multiple users in the same channel, as well as from the intentional jamming. Coding gain is usually defined as the difference between the required SNRs with and without the spread spectrum technique to achieve a certain BER requirement. Hiding a

signal by transmitting it at low power, spread spectrum techniques make it difficult for an unintended user to detect transmitted signal in the presence of noise. In addition, spread spectrum achieves message privacy in multiple users environment. For these reasons, spread spectrum techniques are widely used in digital communication.

In all such implementations, the theoretical and practically achieved BER vs. SNR curves serve to evaluate the overall capacity and coding gain that is equivalent to the increase in the system SNR. It is desirable to quickly obtain the BER performance of a manufactured device in all the cases.

2.2 BER Testing

All BER testers use the same basic principle: known test patterns (e.g. PRBSs, or PRWSs) are sent to a DUT, and the patterns are compared bit by bit with the output of the DUT after a certain period of time. The comparison process is synchronized at the start of the measurement. BER testing methods include software simulation and hardware emulation. In software simulations, each component of the communication system, including the communication channel, is built using software models; while with hardware emulation, all components are built in hardware.

2.2.1 AWGN Channel Model

Communication channels provide the connection between the transmitter and the receiver. There are different types of physical communication channels. It is convenient to construct a mathematical model to capture the most important characteristics of the transmission media. The model of the channel is used in the optimal design of the channel encoder and modulator at the transmitter and the demodulator and channel decoder at the receiver.

Additive white Gaussian noise channel model is the predominant model used in communication system analysis and design. The mathematical model of the additive white Gaussian noise (AWGN) channel is shown in Figure 2-6 [13]. This model applies to a broad class of physical communication channels.

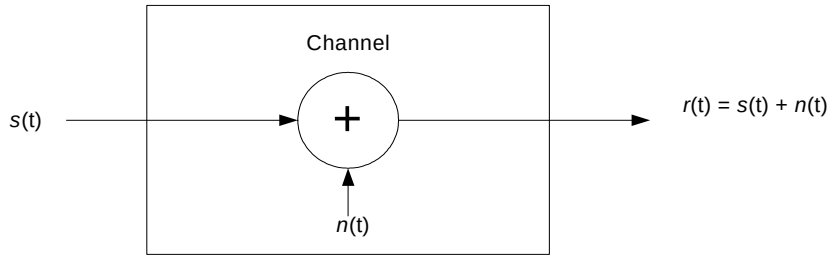


Figure 2-6: AWGN Communication Channel Model

In the AWGN channel model, the transmitted signal $s(t)$ is corrupted by noise $n(t)$. The noise is introduced by the channel, as well as by electronic components, including amplifiers at the receiver. This type of noise is most often characterized as a thermal noise, or statistically as a Gaussian noise process. The output of the communication channel is the sum of the deterministic signal and the random noise, which is expressed as

$$r(t) = s(t) + n(t)$$

where $s(t)$ is an analytical signal with amplitude A and $n(t)$ is a complex-valued, zero-mean, Gaussian noise. The real and imaginary parts of $n(t)$ are assumed to be mutually independent, each with variance σ^2. Therefore the input signal-to-nose ratio is $A^2 / (2\sigma^2)$.

According to the AWGN communication channel model, an AWGN generator is the key to build a channel emulator. Once the AWGN generator is available, it can be scaled to emulate the channel with different SNR conditions.

2.2.2 AWGN Theoretical Properties

Before characterizing the theoretical properties of AWGN, let us first have a look at the characteristics of random variables and the definitions of probability distribution function and probability density function.

Random variables are most often described by their statistics, whose most important properties are the mean, the mean-square, and the variance [13]. The definitions of these parameters are introduced in the following, where E[..] is the expectation operator.

Mean $m_x = \sum_n x_n P[x_n]$

Mean-Square $m_x{}^2 = E[x^2] = \sum_n x_n{}^2 P[x_n]$

Variance $\delta^2 = E[(X - m_x)^2 = E[X^2] - m_x{}^2$

Given a random variable X, for the event $\{X \le x\}$, where x is any real number in the interval $(-\infty, \infty)$, the probability of this event is written as $P(X \le x)$ and denoted simply by $F(x)$, i.e.,

$$F(x) = P(X \le x), \quad -\infty < x < \infty$$

The function $F(x)$ is called the probability distribution function of the random variable X. It is also called cumulative distribution function (CDF).

The derivative of the CDF $F(x)$, denoted as $p(x)$, is called the probability density function (PDF) of the random variable X. Thus, the following express is derived:

$$p(x) = \frac{dF(x)}{dx}, \quad -\infty < x < \infty$$

Or, equivalently

$$F(x) = \int_{-\infty}^{x} p(u)du, \quad -\infty < x < \infty$$

When the random variable is discrete or of a mixed type, the PDF contains impulses at the points of discontinuity of $F(x)$. In such cases, the discrete part of $p(x)$ may be described as

$$p(x) = \sum_{i=1}^{n} P(X = x_i)\ \delta(x - x_i)$$

where x_i, $i = 1, 2, \ldots, n$ are the possible discrete values of the random variable; $P(X = x_i)$, $i = 1, 2, \ldots, n$ are the possibilities; and $\delta(x)$ denotes an impulse at $x = 0$.

Of all the probability functions in digital communication systems, Gaussian density function is by far the most important. Gaussian distribution is also called normal

distribution. The probability density function (PDF) of a Gaussian random variable is written by

$$p(x) = \frac{1}{\delta\sqrt{2\pi}} e^{-(x-m_x)^2/2\delta^2}$$

where m_x is the mean and δ^2 is the variance of the Gaussian variable. The PDF plot of a Gaussian-distributed random variable is shown in Figure 2-7.

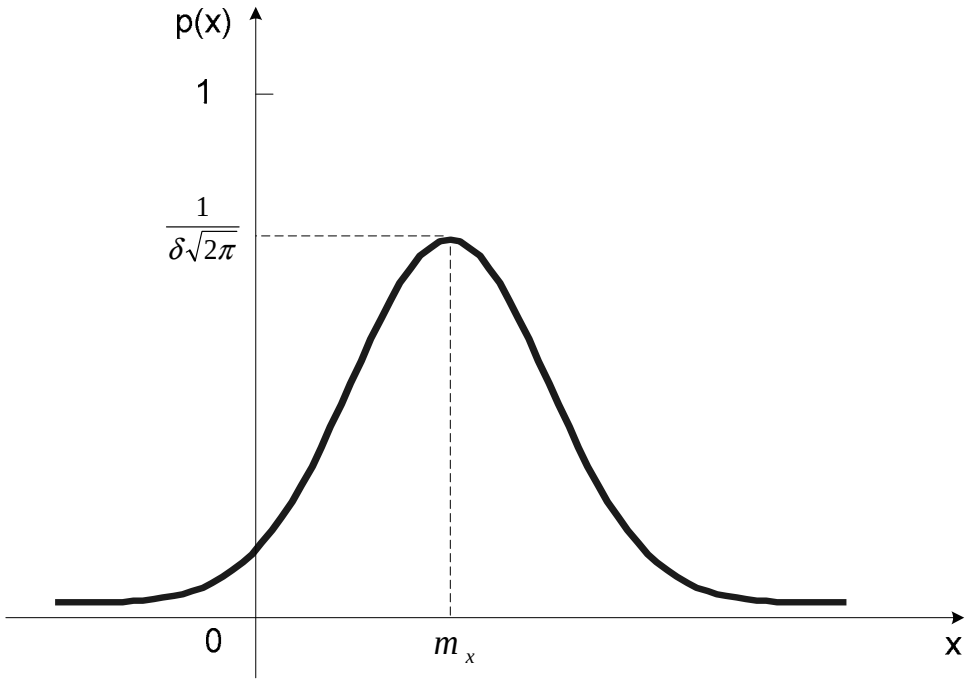


Figure 2-7: The PDF of a Gaussian Variable

The cumulative distribution function (CDF) of a Gaussian random variable is

$$F(x) = \int_{-\infty}^{x} p(u)du$$

$$= \frac{1}{\delta\sqrt{2\pi}} \int_{-\infty}^{x} e^{-(u-m_x)^2/2\delta^2} du$$

$$= \frac{1}{2}\frac{2}{\sqrt{\pi}} \int_{-\infty}^{(x-m_x)/\delta\sqrt{2}} e^{-t^2} dt$$

$$= \frac{1}{2} + \frac{1}{2} erf\left(\frac{x-m_x}{\delta\sqrt{2}}\right)$$

where erf(x) denoted the error function, defined as

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

The CDF $F(x)$ may also be expressed in terms of the complementary error function, which is

$$F(x) = 1\text{-}\frac{1}{2} erfc(\frac{x - m_x}{\delta\sqrt{2}})$$

where

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-t^2} dt$$

$$= 1\text{-}erf(x)$$

Note that erf(-x) =-$erf(x)$, $erfc(-x)$ =2-$erfc(x)$, erf(0)=$erfc$(∞)=0, and erf(∞)=$erfc$(0)=1. For $x > m_x$, the complementary error function is proportional to the area under the tail of the Gaussian PDF. The CDF plot of a Gaussian-distributed random variable is shown in Figure 2-8.

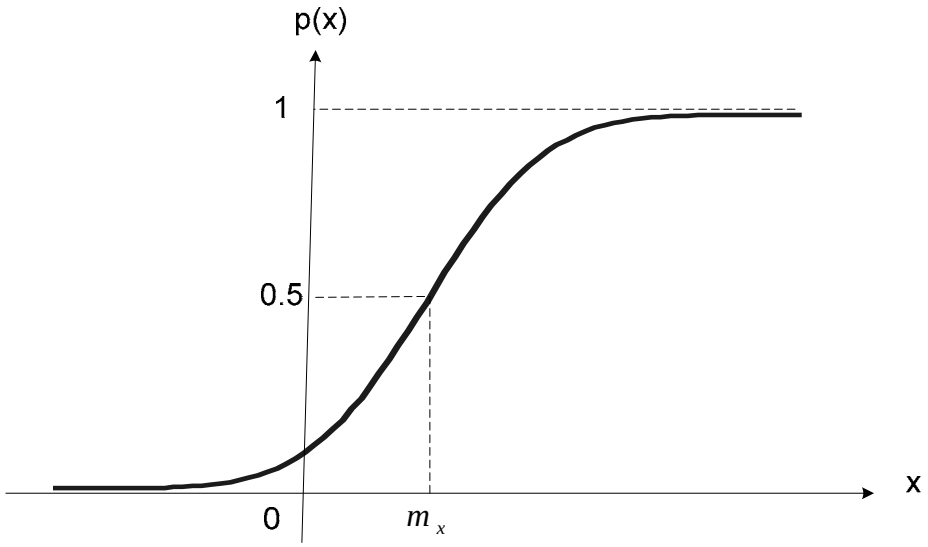


Figure 2-8: The CDF of a Gaussian Variable

Another important function used to characterize Gaussian destitution is Q function, which represents the area under the tail of the Gaussian density function. $Q(x)$ is the most important in computing the probability of error in communication systems. Normalized to zero mean and unit variance, $Q(x)$ is defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-t^2/2}\, dt, \quad x \geq 0$$

$$\approx \frac{1}{x\sqrt{2\pi}} [1 - \frac{1}{x^2}] e^{-x^2/2}$$

Hence we have

$$Q(x) = \frac{1}{2} erfc(\frac{x}{\sqrt{2}})$$

$$F(x) + Q(x) = 1$$

2.2.3 Software Simulation

In the development of a digital communication system, a first evaluation of the performance is usually done mathematically based on very basic principles. However, since a digital communication system suffers from a wide variety of effects that are often difficult to analyze accurately, gaining confidence by software simulation is an essential part of the early development stage. Simulation tools like MATLAB and Simulink [15] are therefore being used for this purpose. In software simulations, each component of a digital communication system, including the communication channel, is represented by a software model which exhibits the characteristics of the represented component. In this case, BER testing is performed based on these software models.

Figure 2-9 shows the software model of a BPSK communication system in MATLAB and Simulink [15]. The Modules in this system include a binary generator that produces information to be transmitted, a BPSK modulator, an AWGN channel, and a BPSK demodulator that recovers the transmitted binary information. All the modules are included in the Simulink library. When building the system, the user can take these models from the library. The BER performance is evaluated by comparing the transmitted information with the information received.

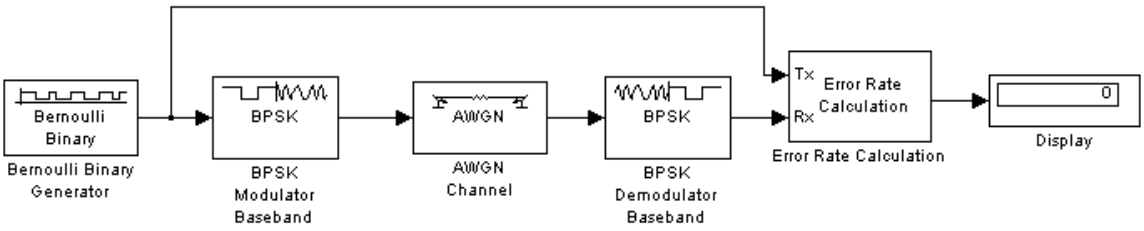


Figure 2-9: Software Model of a BPSK Communication System [16]

In a real digital communication system, the modulator and the demodulator can be real hardware implementations. The AWGN channel is the medium of the communication system, which might be the free space or the combination of several communication media. In the simulation setup in Figure 2.9, each model has many parameters that the user can set when using the model according to the simulated real communication system. The following is a brief introduction of the AWGN Channel block from [16].

The AWGN Channel block adds white Gaussian noise to a real or complex input signal. When the input signal is real, this block adds real Gaussian noise and produces a real output signal; when the input signal is complex, this block adds complex Gaussian noise and produces a complex output signal. This block inherits its sample time from the input signal. The AWGN Channel block uses the DSP Blockset's Random Source block [15] to generate the noise. The dialog box of the AWGN Channel block is shown in Figure 2-10.

In Figure 2-10, the initial seed parameter in this block initializes the noise generator. Initial seed can be either a scalar or a vector whose length matches the number of channels in the input signal. The variance of the noise generated by the AWGN Channel block can be specified using one of four modes [16].

As can be seen, in a software-based BER testing scheme, a communication channel is built using a software model. The BER performance of the communication system can easily be evaluated by running the simulations under different conditions of SNR.

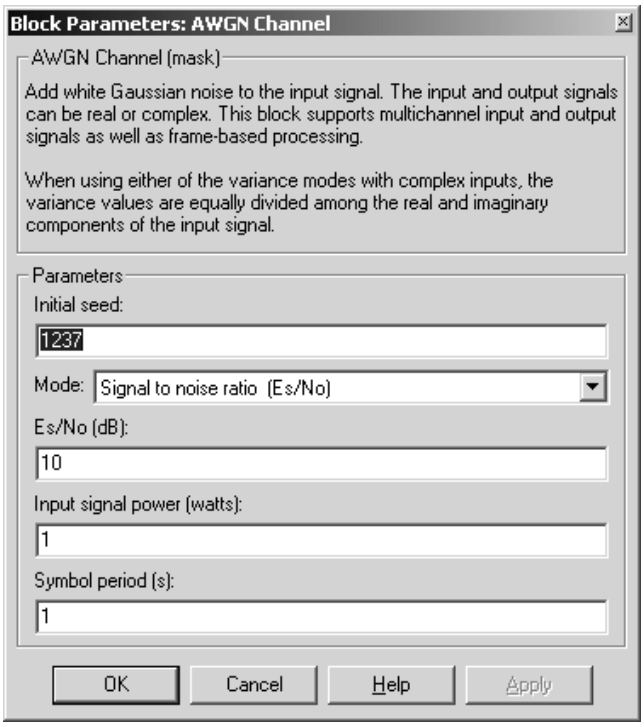


Figure 2-10: Dialog Box of AWGN Channel Simulation Block [16]

2.2.4 Hardware Emulation

Although software simulations are easy to set up to evaluate the BER performance of a digital communication system, they are very time consuming. Execution is done using workstation CPU processors or using acceleration methods. Execution speed depends on the level of abstraction of the simulation models. Due to vast amounts of data and run-time overhead, simulations generally are only suitable for the evaluation of a communication system with low BER performance (such as BER > 10^{-6}). For example, 10^{9} calculation iterations are needed to get an accurate (+-3.3%) estimation of a BER around 10^{-6} [17]; a simulation of BER= 10^{-8} with 10 errors takes days on a personal computer equipped with a 1 GHz Pentium 4 processor. In contrast, acceptable BERs in digital commercial communication systems go below 10 out of 10^{9} in many cases, such as data transmission. Moreover, many design variables, such as sampling frequency, digital format, carrier resolution, rounding, quantization and etc., have to be optimized while satisfying the best trade-off between performances and complexity, which would further lengthen the simulation process.

In order to speed up the BER evaluation process and final parameter optimization of a digital communication system design, performing direct hardware simulation (emulation) is proposed. As an alternative to simulation, emulation utilizes a different technology, such as FPGAs, to re-target all or parts of a design. Many software tools and dedicated hardware [18], [19] have been developed in the aim of automating this re-targeting process. In emulation, performance evaluation takes place in hardware, rather than in the virtual environment of a simulator.

Emulation enables performance evaluation done at system operating frequencies that exceed 20 MHz. A system operating at 20 MHz clock rate processes data 10^6 times faster than workstation-based simulation [6]. Emulation also makes it possible to run a design at a real time system. This feature is especially important for applications such as compression and decompression, where the final output (video or audio) needs to be observed in real time due to the subjective nature of the receiver (the human eye/ear). If such systems run in real time, the performance and quality of the system can be evaluated on the fly; otherwise, large vector sets need to be captured and replay mechanisms need to be created.

Overall, emulation can greatly reduce the design time for communication applications because of the real time test capacity. Additionally, it can enhance the quality of the final design by evaluating the subjective nature of a product under live tests, hence covering a much larger set of test conditions. For these reasons, hardware emulation is widely used in the development of communication applications for performance evaluation [20]-[23]. Hardware emulation can greatly speed up the whole design process.

For hardware-based BER testers, besides expensive standalone BER testing equipment, there are some FPGA-based BER testing solutions available. To evaluate the BER performance of a digital communication in hardware, a high-speed channel emulator and a BERT are essential. In [61], a BER testing solution is presented based on Xilinx RockIO FPGAs, but it does not include the channel emulator. Though a hardware-based

solution combining a BERT and an AWGN can be found in [62], it needs software involvement and the cost is still very high. There is an urgent need to develop a low-cost hardware-based BER testing scheme that combines an AWGN generator and a BERT.

2.2.5 BER Confidence Level

As can be seen in BER testing, BER is derived by calculating the ratio of the number of errors to the number of transmitted bits. But how many bits need to be transmitted in order to get a confident test result? BER confidence level is used to define how reliable the test result is.

For a given digital communication system or component, there usually is a minimum specification for the BER, $p(e)$. In practice, $p(e)$ is often estimated by calculating the ratio of detected bit errors (l) to total bits transmitted (n) in a fixed length test sequence, where the ratio is denoted by $p'(e)$. The accuracy of the estimation improves with the increase of the number of bits in the sequence, which is demonstrated in the following equation:

$$p'(e) = \frac{l}{n} \xrightarrow{n \to \infty} p(e)$$

In real BER testing, it is impossible to transmit infinite number of bits to get $p(e)$, as the test time would be infinite. The number of bits in the transmitted sequence depends on the desired BER confidence levels. Based on a set of measurements, BER confidence level is defined as the probability that the actual $p(e)$ is better than a specified BER level y (such as 10^{-12}). Confidence level (CL) is mathematically expressed as

$$CL = p[\, p(e) < y \mid l, n]$$

where $p[]$ indicates probability, y is a specified BER level, and $\mid l, n$ denotes a system where n bits are transmitted and l bits of errors are detected.

One interpretation of the confidence level is that, if the BER test is repeated many times and the value $p'(e) = l/n$ is recomputed for each test period, we expect $p'(e)$ to be better

than the BER level *y* for *CL* percent of the measurements. To measure BER with a constant confidence level, we need to use a variable length of test sequence [24], [60].

The BER confidence level can be calculated based on the binomial distribution function [25], [26] which models events that have only two possible outcomes, such as success/failure or error/no error. The binomial distribution function is generally written as

$$p_n(k) = \binom{n}{k} p^k q^{n-k}, \text{ where } \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

The above equation gives the probability that *k* events (i.e., bit errors) occurs in *n* trials (i.e., bits transmitted). In this equation, *p* represents the probability that an event occurs in a single trial (i.e. a bit error), and *q* represents the probability that the event does not occur in a single trial (i.e., no bit error); hence $p + q = 1$. Figure 2-11 represents the graph of binomial distribution with $n = 10^8$ and $p = 10^{-7}$.

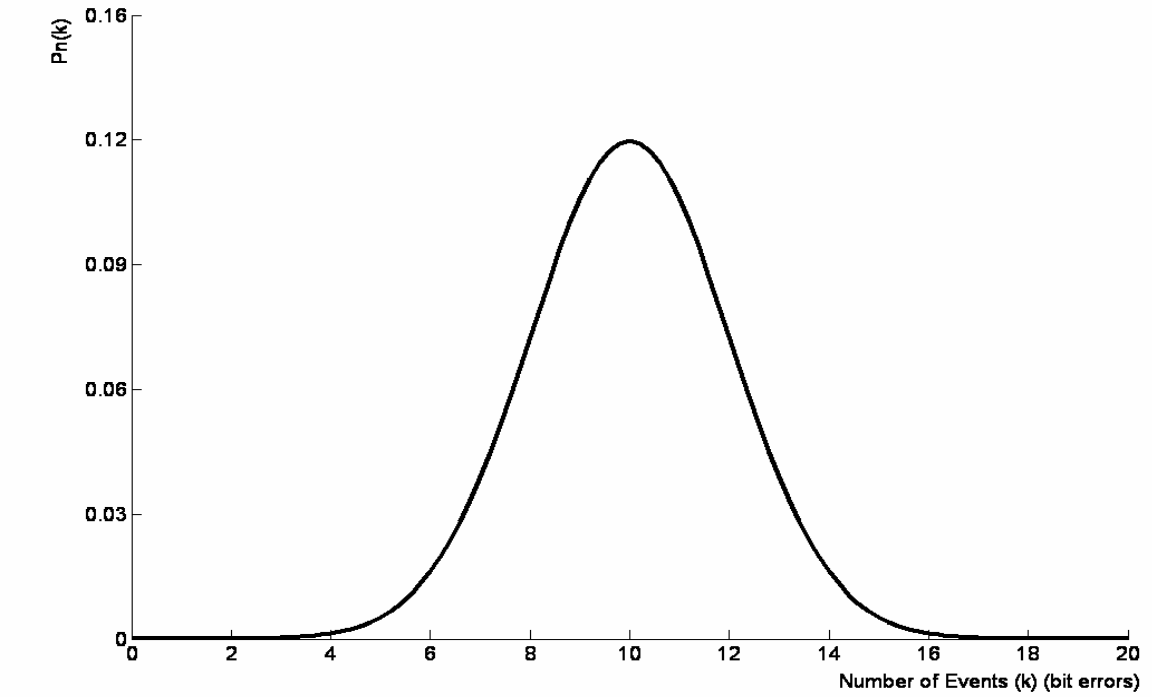


Figure 2-11: Graph of the Binomial Distribution ($n = 10^8, p = 10^{-7}$)

When Figure 2-11 is used for BER confidence level calculation, the n can be treated as total number of bits transmitted, p as the BER, and $p_n(k)$ as the probability that k bit errors will occur. We are interested in the probability that N or fewer events (bit errors) occur in n trials (transmitted bits). The probability is the cumulative binomial distribution function, which is written as

$$p(e \le N) = \sum_{k=0}^{N} p_n(k) = \sum_{k=0}^{N} \frac{n!}{k!(n-k)!)} p^k q^{n-k}$$

In terms of the cumulative binomial distribution function, the confidence level can be expressed as

$$CL = 1 - \sum_{k=0}^{N} [\frac{n!}{k!(n-k)!}] p^k (1-p)^{n-k}$$

In BER confidence level calculations, a hypothetical value of p and a desired confidence level (CL) are first chosen, then solve the above CL equation to determine how many bits (n) must be transmitted through the system with N or few errors to prove the hypothesis. It is difficult to directly solve n and N. Poisson theorem [25], which provides a conservative estimate of the binomial distribution function, can be used to simplify solving n and N. Poisson theorem is written as

$$p_n(k) = (\frac{n!}{k!(n-k)!}) p^k q^{n-k} \xrightarrow{n \to \infty} \frac{(np)^k}{k!} e^{-np}$$

Table 2-1 shows an example of the solutions for N and n for a communication system [27]. In this system, p is specified to 10^{-10}. It is impossible to achieve 100% confidence as it requires infinite test time. If the confidence level CL is set to 99%, for various values of N, corresponding values of n are solved.

Table 2-1: An Example of BER Estimation (CL=99% and p =10^{-10}) [27]

Bit Errors N	0	1	2	3	4
Required transmitted bits	$4.61*10^{10}$	$6.64*10^{10}$	$8.40*10^{10}$	$1.00*10^{11}$	$1.16*10^{11}$
Test time @ 622Mbps (s)	74.1	106	135	161	186

As can be seen from Table 2-1, in a 622Mbps system, if no bit errors are detected in 74.1s of testing, one bit error occurs in 106s, or two bit errors occur in 135s, we have a 99% confidence level that $p(e) < 10^{10}$.

Theoretical analysis shows that test time is proportional to –ln(1-*CL*). Figure 2-12 shows this relationship. As it is impossible to achieve 100% confidence level in BER testing, the BER measured by BERT equipment is only an estimate of the true BER. If we want to achieve higher confidence level, the test must take longer time. We are hence forced to play tradeoff between confidence level and test time.

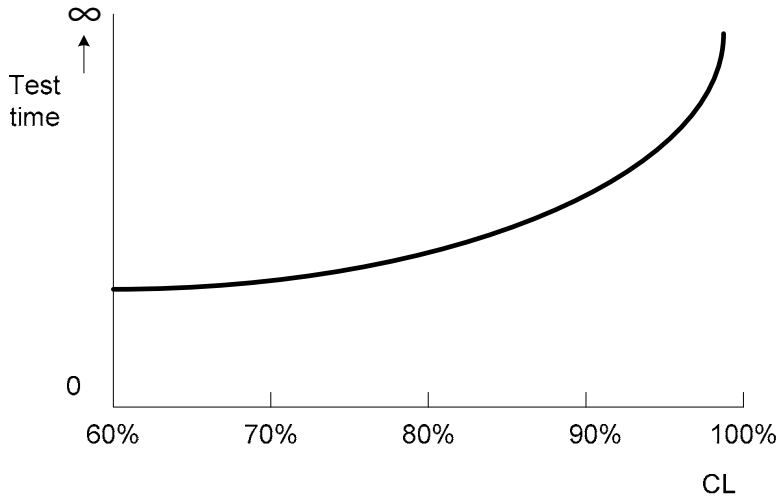


Figure 2-12: Test Time vs Confidence Level (*CL*)

2.3 Proposed BER Testing Scheme

As can be seen from the above of this Chapter, it is critical to quickly and precisely test the BER performance of a communication system. Traditional software simulation methods for BER testing are very time consuming to conduct. Though there exist hardware BER testers, they are very expensive and few of them include instrumentation that can emulate the communication channel which introduces the real-world impairment –noise, hence they are difficult to set up for BER testing under the presence of noise. To overcome these problems of existing methods of BER testing, a new BER testing scheme

is proposed and shown in Figure 2-13. This method can facilitate BER testing of various communication interfaces.

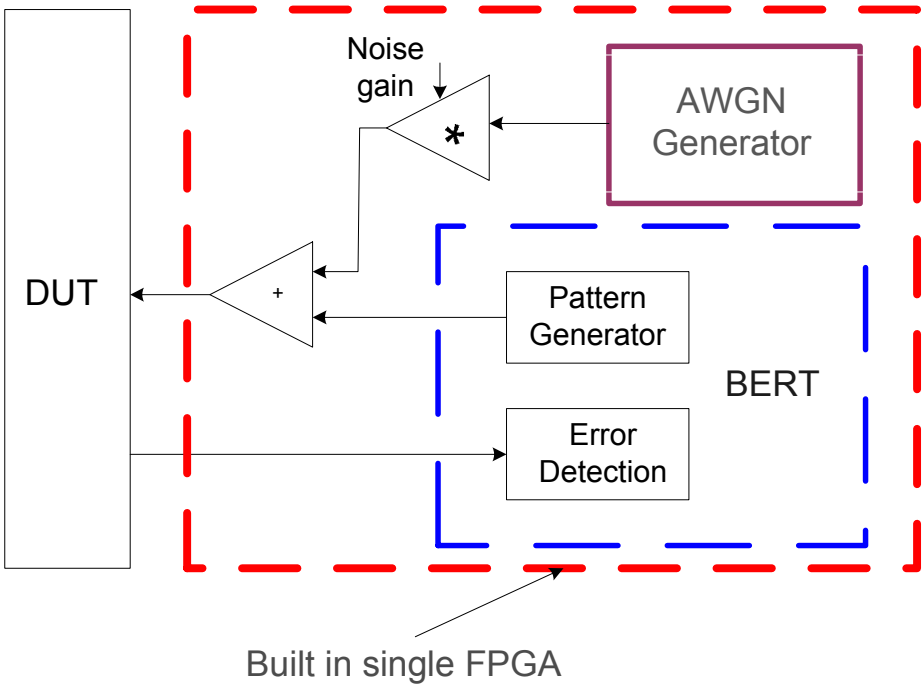


Figure 2-13: Proposed BER Testing Scheme

As shown in Figure 2-13, the solution combines a BERT and an AWGN generator in a single FPGA device. The scalable AWGN may be added or not according to applications. The proposed scheme can easily be set up to test the BER performance of a real DUT in different SNR conditions in real operations.

The DUT can be any communication interface or system that receives bit or word sequences and then restores the sequences after some signal processing or format changes, such as a transceiver (a transmitter and a receiver), the combination of a modulator and a demodulator, or the integration of an encoder and a decoder. Parameterized design enables the tester to interface a DUT either in serial, parallel or CDR format.

The detailed implementation and function of the BERT core and the AWGN core are discussed in the next two chapters.

Chapter 3 - BERT Core Design

3.1 Overview

As discussed in Chapter 2, the basic concept of BER measurement is simple: send a data stream to a DUT, compare the output of the DUT with its input, and differences are registered as errors and evaluated. However, the design of a BER tester is not trivial; the following issues must be addressed:

- Test sequences generation: Over an infinitely long period of time, we can assume that a data transmission is a random process. In BER measurement, a pseudorandom data sequence is used, as we can not create a truly random signal using deterministic methods.
- Test synchronization: When a test sequence is sent to a DUT, it takes some time for the DUT to process the data and then send the sequence out. The sequence to the input of the DUT should wait for a proper period of time when it is compared with the output sequence of the DUT. The waiting time is the delay of the DUT, and this process is called test synchronization. The synchronization process is conducted at the start of the BER measurement.
- Bit slip detection: The synchronization process is achieved at the start of a BER measurement. However, the synchronization may lost during the measurement process because of a bit slip or a bit repetition. In this case, synchronization should be re-built to get the true BER. A BER tester should be able to distinguish between a bit slip and an error burst.

Besides the above three issues, BER calculation and test result display are also needs to be addressed in a BER tester (BERT) design. In this Chapter, the detailed design of BERTs is presented. First, the design of a serial BERT is introduced. Then, the design of a parallel BERT is presented based on the design of the serial BERT. Finally, experiment results of the BERTs are given.

3.2 Serial BERT Design

A serial BERT sends bit sequence patterns to a DUT and then conducts bit-by-bit comparison of the received signal from the DUT. Based on the principle of the BERT, the structure of a serial BERT is proposed and shown in Figure 3-1. The serial BERT can test the BER performance of a serial digital communication link.

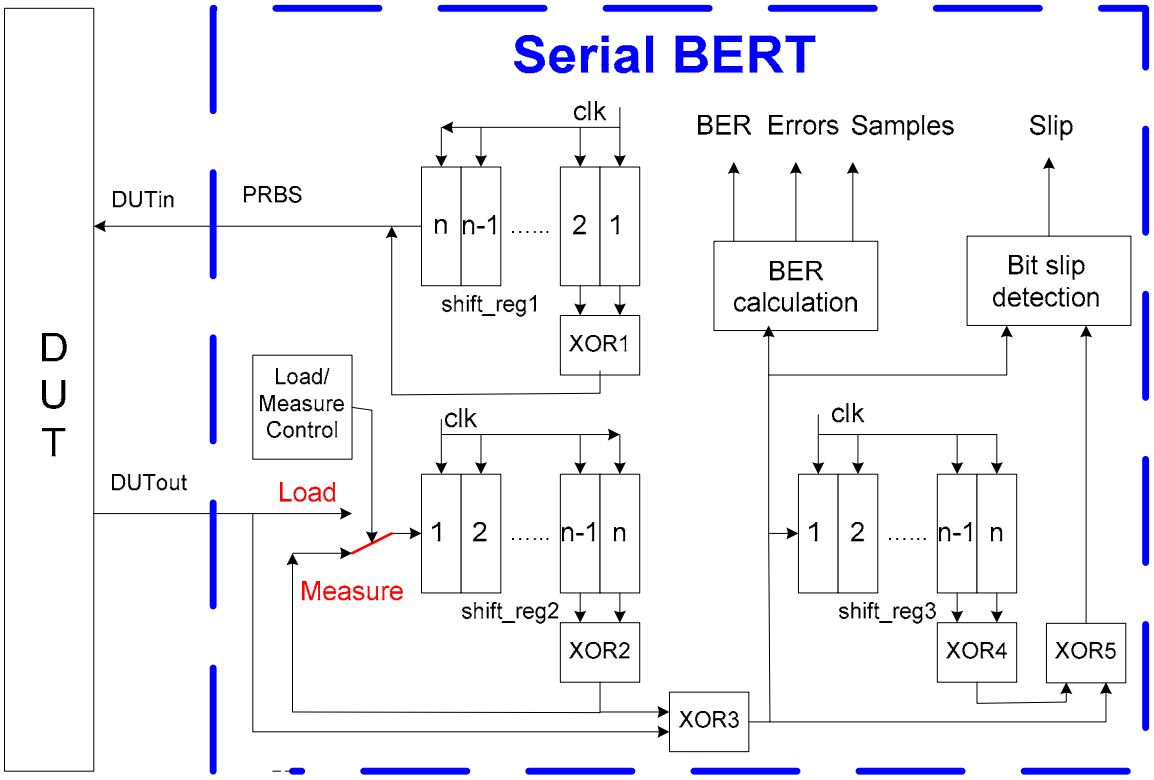


Figure 3-1: The Structure of the Serial BERT

In this scheme, the shift register *shift_reg1* and the gate *XOR1* form a linear feedback shift register (LFSR). The LFSR generates a pseudo random bit sequence (PRBS), and the sequence is sent to the DUT. Before a measurement begins, the *load/measure* switch is set to be in *load* state. When BER measurement begins, the switch is changed to *measure* state. The shift register *shift_reg2*, the switch and the gate *XOR2* are used for synchronization by replicating a delayed PRBS as the reference pattern. The gate XOR3 serves as a comparator, comparing the pattern from the DUT with the reference pattern.

The shift register *shift_reg3* and the two gates *XOR4* and *XOR5* serve for the purpose of bit slip detection. The detail of the BERT core is introduced in the following.

3.2.1 PRBS generation

In a real serial digital communication link, the bit states of the digital signal change frequently and unpredictably between one and zero. A BERT must provide facilities for simulating real operating conditions. For this reason, PRBSs are used to simulate the transmitted signals. The PRBSs are generated by a LFSR as shown in Figure 3-2. This structure is also shown in Figure 3-1, represented by *shift_reg1* and *XOR1*.

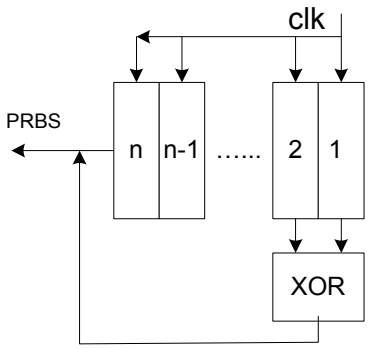


Figure 3-2: The Circuit for PRBS Generation

The sequence repeats periodically after a certain number of bits. In Figure 3-2, *n*, the stage number of the shift register, determines the length of the PRBS. The maximum period of the sequence is 2^n-1. The longest continuous sequence of ones within the sequence is *n*; the longest continuous sequence of zeros within the sequence is *n*-1. The bigger the *n* is, the closer the PRBS simulates the real transmitted data. More details about the principle and performance of the LFSR are discussed in Chapter 4-1.

In order to simulate real data transmission, it is necessary to choose a big number for *n*. However, a big number for *n* would result in long synchronization time, especially at low bit rate, as a BERT needs several periods of the PRBS to achieve synchronization. The synchronization principle is discussed in the next section. In the parameterized BERT core, users can set the value of *n* according to applications by balancing the

synchronization time and quality of randomness of the sequence. Some suitable values of *n* are 2, 3, 4, 6, 7 ….

3.2.2 Testing Synchronization

Basically, the synchronization between the transmitted and reference patterns is achieved by loading the shifter registers (*shift_reg2* and *shift_reg3* in Figure 3-1) with the transmitted PRBSs before the switch is turned to *measure* state from *load* state in Figure 3-1. The detailed principle is discussed in this section.

In order to easily demonstrate the serial BERT working process, we assume the length of all the shift registers in Figure 3-1 is 3, and assume the DUT is an adjustable shift register, which exhibits manageable outputs and delays. The delay and the output of the shift register are controlled by a 2-bit signal named *err_slip*. In this section, we only consider the normal state, in which *err_slip* is set to be "00", and the output signal of the shift register is the input signal delayed by three clock cycles. The other states will be discussed in the next section. In this case, the circuit related to synchronization is shown in Figure 3-3.

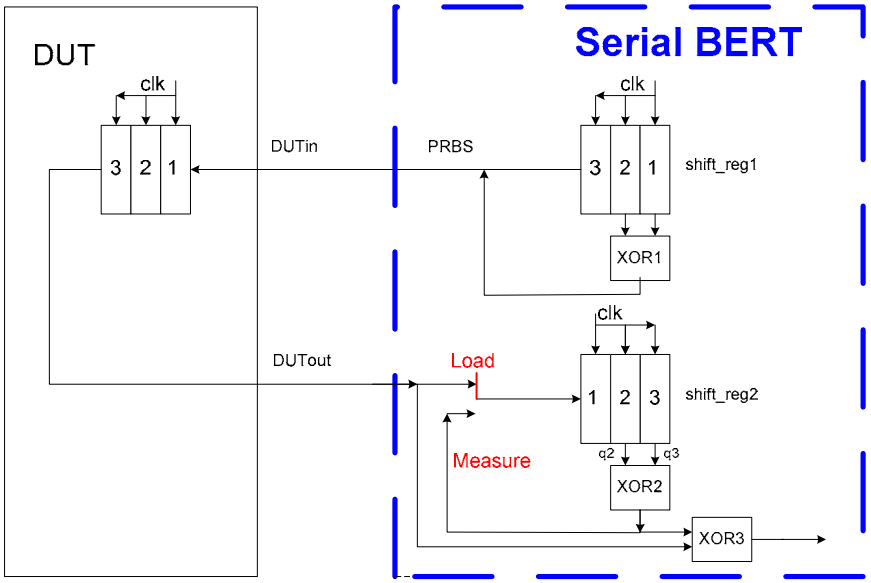


Figure 3-3: Synchronization Circuits (*n* =3, DUT delay = 3)

At the start of the BER measurement, the switch in Figure 3-3 is in *load* state. Assuming the initial state of the shift registers is not "000", the sequence sent to the DUT (*DUTin*) would repeat every 7 clock cycles after the reset signal is asserted. According to the circuit of Figure 3-3, each bit of the sequence *DUTin* is the XOR operation of the two bits that are immediately two bits before the bit. If the sequence *DUTin* is denoted by $x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ \ldots$, we have

$$x_4 = x_1 \oplus x_2$$

$$x_5 = x_2 \oplus x_3$$

$$x_6 = x_3 \oplus x_4$$

$$x_7 = x_4 \oplus x_5$$

$$x_1 = x_5 \oplus x_6$$

$$x_2 = x_6 \oplus x_7$$

$$x_3 = x_7 \oplus x_1$$

Table 3-1 shows the outputs of the circuit in Figure 3-3 for the first 16 clock cycles after the reset signal is asserted.

Table 3-1: The Outputs of the First 16 Clock Cycles in Figure 3-3

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DUTin	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2
DUTout	--	--	--	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2	x_3	x_4	x_5	x_6
q2	--	--	--	--	--	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2	x_3	x_4
q3	--	--	--	--	--	--	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2	x_3
XOR2	--	--	--	--	--	--	x_4	x_5	x_6	x_7	x_1	x_2	x_3	x_4	x_5	x_6
XOR3	--	--	--	--	--	--	0	0	0	0	0	0	0	0	0	0

As can be seen from Table 3-1, after a certain number of clock cycles, *XOR2* = *DUTout*, so *XOR3* = 0, where *XOR3* = *XOR2* $\oplus$ *DUTout*. The number equals to the sum of *n* and

the delay of the DUT. In the above case, both are 3; therefore, the number equals to 6. The above analysis is verified by the simulation waveform shown in Figure 3-4.

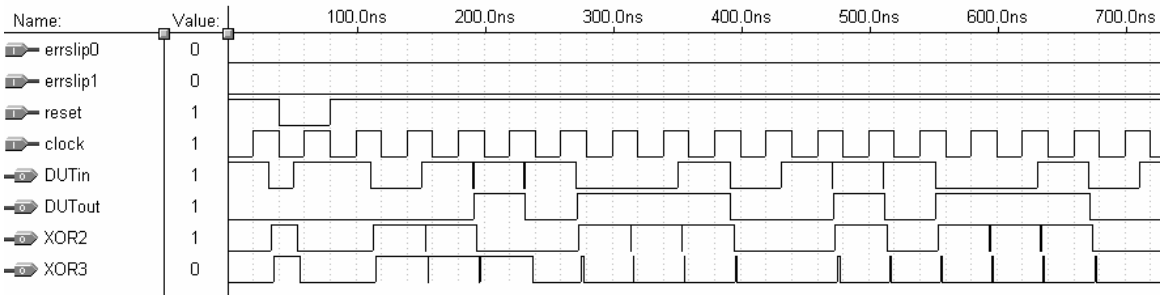


Figure 3-4: Simulation Waveforms of the Synchronization Circuit

As can be seen from the above figure, the output of XOR3 is zero after 6 clock cycles. The glitches are due to the delays of the XOR gates and the shift registers. All registers update their values at the rising edge of the clock signal.

In Figure 3-3, When the shift register *shift_reg2* is fully loaded with the transmitted PRBS, the output of *XOR2* equals to *DUTout*, so the switch can be changed to *measure* state without affecting the work state of the whole circuit. In this case, *shift_reg1* and *XOR1* form a LFSR, $LFSR_1$, and *shift_reg2* and *XOR2* constitute another LFSR, $LFSR_2$. The two LFSRs generate the same PRBS, but the sequence from $LFSR_2$ is d clock cycles later in timing than that from $LFSR_1$, where d is the delay of the DUT in terms of the number of clock cycles. Therefore, if the test patterns from $LFSR_1$ are correctly transmitted by the DUT, then the two inputs of XOR3 should be the same value in each clock cycle. In a real BER measurement, when the serial BERT is in measurement state, the output of XOR3 is the comparison result and is monitored every clock cycle: if a '1' is detected, a transmission error is counted; otherwise, the transmission is error-free.

3.2.3 Bit Slip Detection

In the above analysis, we assume that all the data is being correctly transmitted by the DUT in the synchronization process; otherwise, the process should be repeated. However,

a real communication system may encounter transmission errors. The errors can take the form of single errors, error bursts, or bit slips.

A single error can be caused by noise or mismatch between a transmitter and a receiver for specific data sequences. Single errors happen sporadically, and are the main form of errors for most data transmission systems.

An error burst can be caused by an event that leads to a large number of errors in a short period, such as poor contact, carrier fading in a radio link transmission or brief electromagnetic interfaces (e.g. switching).

Both the bit loss and the bit repeat are called bit slip. A bit slip results from the loss of certain sections of transmitted bit stream or the repeat transmission of some bits. The event causing bit slips may be an overflow of a digital buffer or clock problems at gateways. Bit slips lead to a phase shift between the transmitted and received sequences from the point view of the BERT.

After the synchronization process, both error bursts and bit slips can result in a large number of errors. For errors resulting from error bursts, they should be counted in BER calculation. For errors resulting from bit slip, they should not be totally counted in BER calculation; only the bits lost or repeated should be counted as errors. When bit slips happen, the number of bit errors measured will be infinite due to the phase shift between the received and reference patterns. In this case, the measurement must be interrupted and the BERT must be resynchronized.

When a large number of errors are encountered, the BERT must be able to determine whether the errors are caused by bit slips or error bursts. If bit slips happen, the BERT should be interrupted and resynchronized; otherwise, the measurement should continue.

A patented solution to the problem of distinguishing between error bursts and bit slips is offered by a few manufacturers of test equipment [43]. The solution is based on the fact

that the addition or superimposition of two PRBSs that are shifted in phase relative to each other produces another PRBS. Based on this principle, the bit slip detection circuit is devised and shown in Figure 3-5. We discuss how it works in the following.

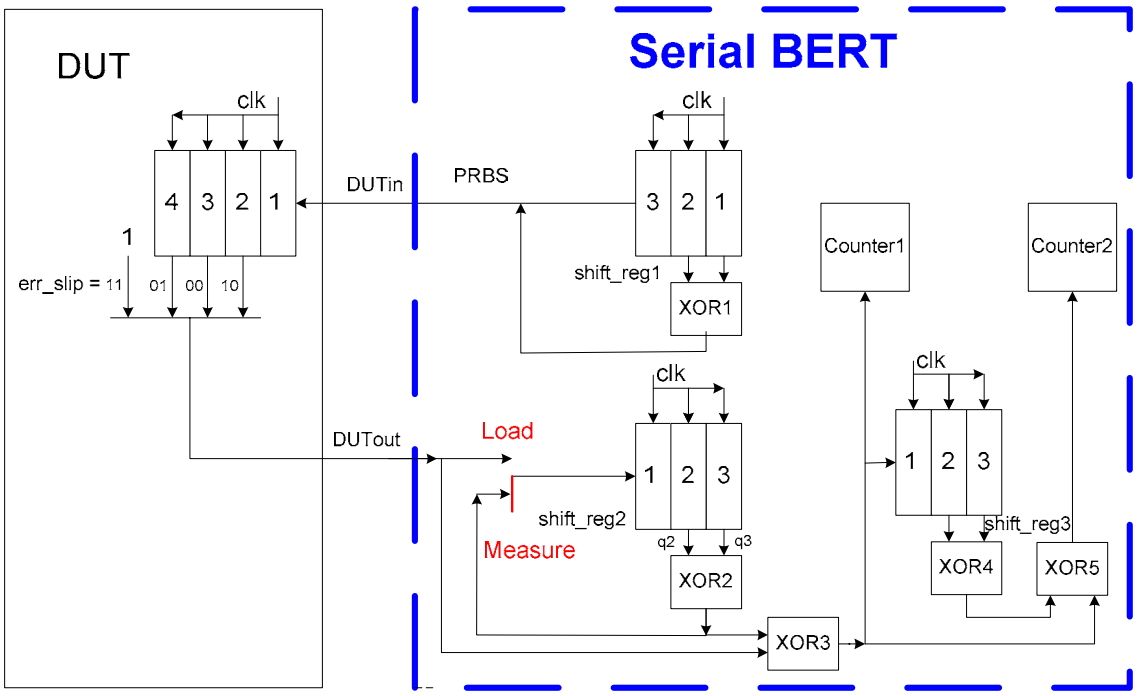


Figure 3-5: Circuit for Bit Slip Detection

In Figure 3-5, the DUT is revised on the base of the DUT in Chapter 3.2.2 for bit slip experiments. In Chapter 3.2.2, the signal *err_slip* is set "00"; the DUT is in normal state and exhibits a delay of 3 clock cycles, and the synchronization of the BERT is done in this state. Here another three states are added to the DUT to demonstrate the bit slip detection function: when *err_slip* = "10", the DUT exhibits a delay of 2 clock cycles, so it emulates a bit loss; when *err_slip* = "01", the DUT exhibits a delay of 4 clock cycles, so it emulates a bit repeat; and when *err_slip* = "11", the output of the DUT is always set to be 1s, so it emulates an error burst.

After the synchronization process, the BERT works in *measure* state. As discussed in Chapter 3.2.2, if all bits are correctly transmitted, the output from XOR2 and the output

from the DUT (*DUTout*) are the same PRBS in terms of both the value and the phase. Therefore, the outputs from XOR3 and XOR5 are all zero.

If a bit slip happens, the two patterns from XOR2 and the DUT are shifted in phase. To illustrate this case, we assume that in *measure* state *err_slip* is switched from "00" to "10" in clock cycle *t*, which emulates a bit loss. From clock cycle *t* on, the output sequence of XOR2 is set to be $x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7$ ….. The output of XOR2, the DUT and XOR3 are listed in Table 3-2. Please recall that $x_4 = x_1 \oplus x_2$, $x_5 = x_2 \oplus x_3$ and so on are from Chapter 3.2.2.

Table 3-2: The Output of the Comparator

Cycle	t-2	t-1	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	….
XOR2	x_6	x_7	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2	….
DUTout	x_6	x_7	x_2	x_3	x_4	x_5	x_6	x_7	x_1	x_2	x_3	….
XOR3	0	0	x_4	x_5	x_6	x_7	x_1	x_2	x_3	x_4	x_5	….

As can be seen from Table 3-2, when a bit slip happens, the output of XOR3 is the same PRBS value as the transmitted PRBS value, but may have different phase. The above is for the situation that only one bit is lost. It is easy to understand that when more bits are lost or bit repeat happens, the output of XOR3 is always the PRBS value, but with different phase. According to the synchronization principle as discussed in Chapter 3.2.2, when *DUTout* is a PRBS and the switch is in load state in Figure 3-5, the pattern from the output of XOR2 is the same as *DUTout* after *n* clock cycles, which results in zeros from the output of XOR3. Applying this principle, we know that, if the output of XOR3 is a PRBS, the two sequences from the outputs of XOR3 and XOR4 in Figure 3-5 are the same after *n* clock cycles, where *n* is the length of the shift registers. Therefore, the output of XOR5 is all zeros.

When error bursts happen, as the real bit errors are random and the synchronization is still maintained, the output of XOR3 is random. The output of XOR3 cannot be in phase with

the output of XOR4, so the output of XOR5 is not all zeros. During the measurement, the shift register *shift_reg3* is loaded continuously, and bit slips and error bursts are differentiated by adding two counters, *counter1* and *counter2*, as shown in Figure 3-5.

The counter *counter1* counts the number of bits that have been continuously correctly transmitted. In each clock cycle, the output of XOR3 is monitored. Once a 1 is detected, indicating an error happens or a bit slip happens, *counter1* is reset to zero; otherwise, it is increased by one until it reaches an upper bound threshold, *upper1*.

The counter *counter2* counts the number of consecutive zeros appearing in the output of XOR5. Consecutive zeros from XOR5 indicate the DUT is either in correct transmission state or in bit slip state. In each clock cycle, the output of XOR5 is monitored. Once a 1 is detected, indicating an error burst may happen, *counter2* is reset to zero; otherwise, it is increased by one until it reaches an upper bound threshold, *upper2*.

A bit slip is assumed and indicated by setting *slipflag* to be high when *counter2* reaches *upper2* and *counter1* does not reach *upper1*. In this case, the synchronization is lost and needs to be rebuilt, and the measurement should be repeated. Both the thresholds should be set at least bigger than the sum of the delay of the DUT and the length of the shift registers. The special synchronization monitor mechanism ensures that the measurement result is the actual number of errors, and is not influenced by the BERT itself. This feature also provides a way to search for the source of errors.

With the structure shown in Figure 3-5, when the thresholds *upper1* and *upper2* are all set to be 48, the above discussion regarding the bit slip detection principle is further verified by simulations. The waveforms are shown in Figure 3-6.

In Figure 3-6, the BERT works in *measure* state. Before the moment of time = 7.46us, *err_slip* is set to be 00, and the data is correctly transmitted. In this case, the output of XOR3 is all zeros, no errors occurring; both *counter1* and *counter2* reaches 48, so *slipflag* is low, no bit slip alarm signaled. After the moment of time = 7.46us, *err_slip* is set to be

"10", a bit slip occurring. In this case, the output of XOR3 is a PRBS, lots of errors appearing; the output of XOR5 is all zeros after 3 clock cycles; *counter1* is reset to zero frequently and *counter2* is increased by one continuously. Once *counter2* reaches the threshold 48, considering that *counter1* does not reach the threshold, the signal *slipflag* is asserted, indicating a bit slip occurs and the measurement should be interrupted and resynchronized. The glitches in the waveforms can be removed by adding a register before the output of each signal if necessary.

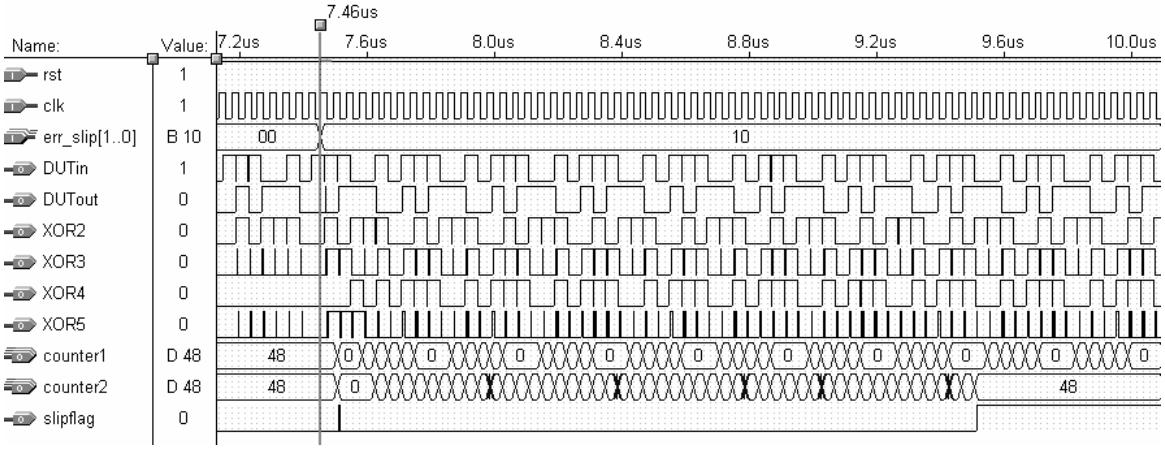


Figure 3-6: Simulation Waveforms of Bit Slip Detection

The method of distinguishing between error bursts and bit slips is also further verified by the simulation waveforms shown in Figure 3-7. The waveform is based on the design shown in Figure 3-5.

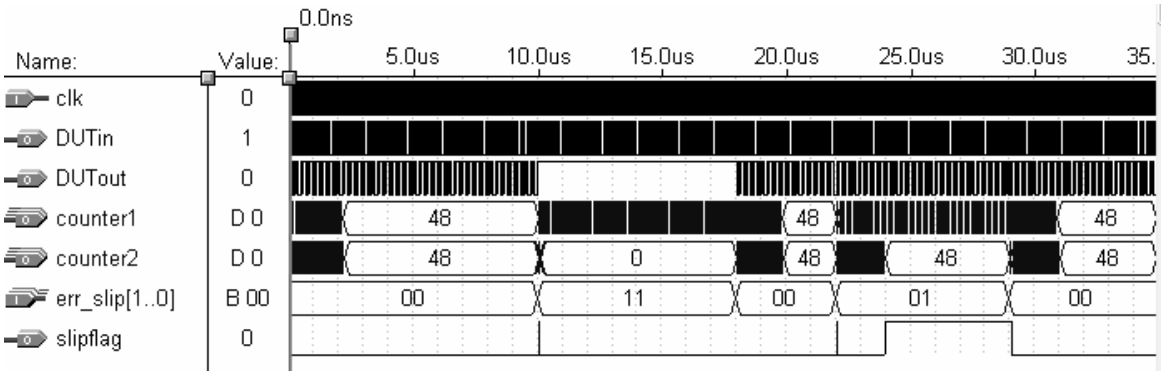


Figure 3-7: Waveforms of the Serial BERT (Error Burst and Bit Slip)

In Figure 3-7, before the moment of time = 10us, *err_slip* = “00”, all the PRBS data is correctly transmitted and the signal *slipflag* is inactive. During the period of 10us ~ 18us, *err_slip* = “11”, an error burst is created by setting *DUTout* to be one. Though a large number of errors occur during this period, the signal *slipflag* is still inactive, so the errors are considered as real transmission errors. During the period of 22us ~ 29us, *err_slip* = “01”, a bit slip is created by transmitting a bit twice. In this case, the signal *slipflag* is activated, indicating a bit slip occurs. During the periods of 18us ~ 22us and 29us ~ 35us, *err_slip* = “00”, the system goes back to correct transmission state, so the signal *slipflag* is inactive.

3.2.4 State Control and BER Calculation

In the above discussion, the BERT shown in Figure 3-1 is first set to be in *load* state to achieve synchronization before the measurement begins. After the synchronization is acquired, the switch can be changed to *measure* state.

The *load/measure* switch is controlled by a counter. When the reset signal is activated, the counter is reset to zero. Once the reset signal is inactive, the counter is increased by 1 each clock cycle until it reaches an upper bound threshold, *cntload*. According to Chapter 3.2.2, the threshold *cntload* should be greater than the sum of *n* (the length of the shift register) and the delay of the DUT. The switch is in *load* state only when the output of the counter does not reach to the threshold *cntload*. When the output of the counter reaches the threshold *cntload*, the switch is set to be in *measure* state, and will be kept in this state until the reset signal of the whole system is activated once again, such as for resynchronization of the current measurement or for a new measurement

Once the switch is in *measure* state, BER calculation is performed and the BER measurement begins. The BER calculation process is to continuously calculate the ratio of the number of error bits and the number of transmitted bits. In *measure* state, the number of error bits is calculated by a counter, whose content is increased by one if a 1 is detected from the output of the gate XOR3 at the rising edge of each clock cycle; the number of transmitted bits is calculated by another counter, whose content is increased by

one in each clock cycle. The minimum measurement time varies with applications and BER confidence requirements. Both counters are reset to zero in *load* state.

3.2.5 Output Display

The BER testing results from the counters are in binary form. The output display part enables the user to access the result more directly. This part is independent of the BERT core, and there are different means to realize it according to available hardware and software resources; therefore, the output display part is not included in the BERT core. Based on the available hardware resources, we have developed a VGA display core to directly display the BER measurement results on a monitor in decimal form.

VGA interfaces are included in many FPGA development boards, including the UP-1 board from Altera. With VGA format, the interface between the testing design and the display device is greatly simplified, only needing five signal lines.

3.3 Parallel BERT Structure

As discussed in Chapter 3.2, a serial BERT can be used to test the BER performance of a communication interface that transmits serial data. If a communication interface transmits parallel data, a parallel BERT is needed to test its BER performance.

3.3.1 Design Strategies

The design of the parallel BERT is based on the serial BERT presented in Chapter 3-2. Basically, a *k*-bit parallel BERT, where *k* is the width of the parallel data (bit0 ~ bit(*k*-1)) can be built using *k* independent serial BERTs that have the same load time. The parallel BERT sends pseudo-random word sequences (PRWSs) to the DUT. In order to qualify randomness of the generated sequences, the independence of each of the serial BERTs is very important. That means the length of the shift registers in each of serial BERTs should be different.

In the design, the width of the parallel BERT is parameterized, which ranges from 1 to 10. It can also be easily expanded to a width more than 10. For all the lengths of the shift

registers in the serial BERTs, being prime to each other can achieve a maximum period of the PRWS, 2^{m-1}, where *m* is the sum of all the lengths.

When *k* independent serial BERTs are directly put together to build a parallel BERT, each of the serial BERTs has circuits for the *load/measure* switch control and bit slip detection. The circuits for the *load/measure* switch control of each bit of the parallel data should change *load/measure* state at the same time, and the parallel BERT should be capable of distinguishing between error bursts and word slips instead of bit slips in a serial BERT. Therefore, only one of the *k* such control circuits is needed for switch control and word slip detection. For this reason, redundancies resulted from building a k-bit parallel BERT by directly combining *k* independent serial BERTs should be removed.

3.3.2 System Architecture

Based on the parallel BERT design strategies discussed in Chapter 3.3.1, the structure of the parallel BERT is developed and shown in Figure 3-8. In the parallel BERT design, the serial BERT circuitry for bit0 is used to control all the *load/measure* switches in synchronization circuits and to detect word slip.

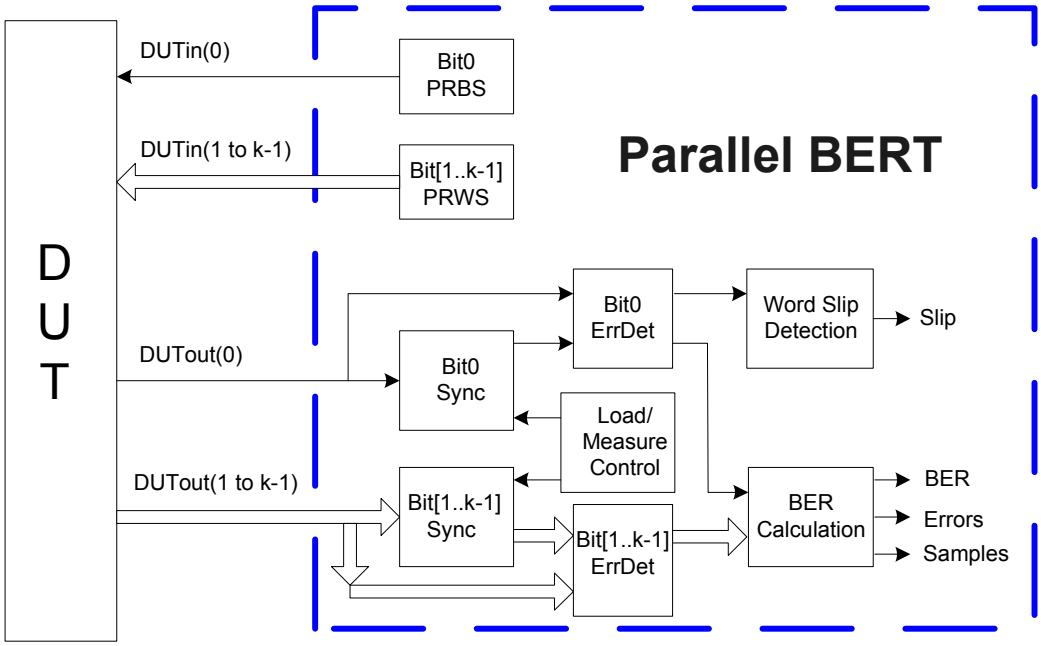


Figure 3-8: The Structure of the Parallel BERT

In Figure 3-8, the blocks *Bit0 PRBS*, *Bit0 Sync*, *Bit0 ErrDet*, *Load/Measure Control* and *Word Slip Detection* can be considered as a serial BERT which has the same circuit as shown in Figure 3-1. The blocks *Bit[1..k-1] PRWS*, *Bit[1..k-1] Sync* and *Bit[1..k-1] ErrDet* are the combination of *k*-1 independent modules. Figure 3-9 gives the circuit of one of the modules. Each module has the same structure, but the length of the shift registers in each module is different in order to maintain the randomness of generated word sequences.

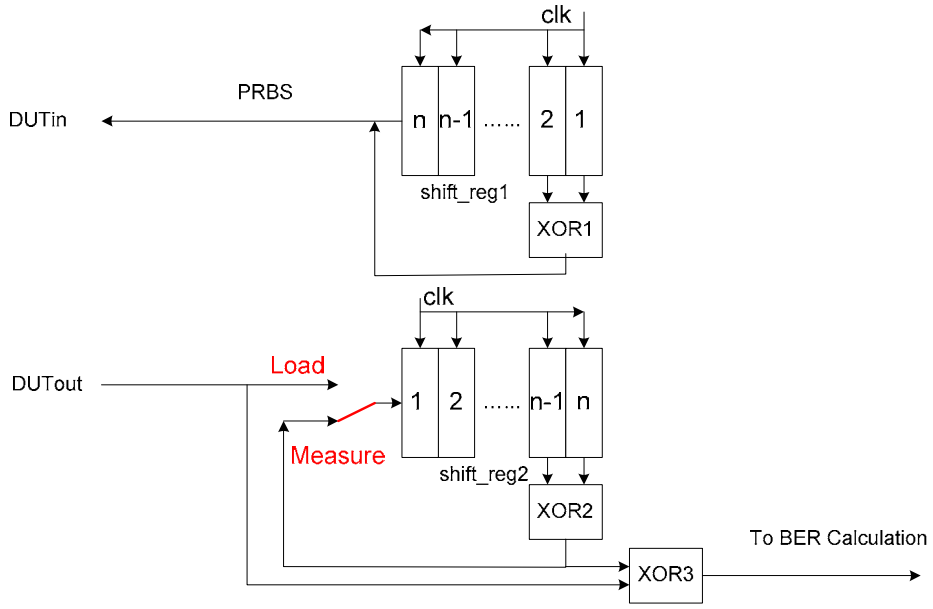


Figure 3-9: Sub-module Circuit of the Parallel BERT

In Figure 3-9, the structure is the same as that of a serial BERT discussed in Chapter 3.2. The shift register *shift_reg1* and the gate *XOR1* serve as PRBS generation; the switch, *shift_reg2* and XOR2 constitute a 1-bit synchronization block; the gate XOR3 is a 1-bit error detector.

In Figure 3-8, the *Load/Measure Control* block is used to control all the switches in *Bit0 Sync* and *Bit[1..k-1] Sync* blocks. The control principle is the same as that in the serial BERT. The load time is greater than the sum of the delay of the DUT and the longest length of the shift registers. In the parallel BERT, the *BER Calculation* block deals with a *k*-bit word in each clock cycle. Therefore, the counter counting error bits may be

increased by a number between 0 to *k* each clock cycle, depending on the number of error bits in the transmitted word. The counter counting transmitted bits is increased by *k* each clock cycle.

A parallel BERT interfaces a DUT with parallel data, which requires lots of connection wires and stringent timing specifications. The connection interface can be greatly simplified by inserting serial data transceivers between the parallel BERT and the DUT. A serial data transceiver consists of a transmitter and a receiver. A transmitter serializes the parallel data and encodes the clock signal into the serialized data, and a receiver recovers the clock signal and the transmitted parallel data (de-serialization) by dedicated clock data recovery (CDR) circuitry. If each side of the BERT and the DUT is appended with a serial transceiver, the parallel transmission between the BERT and the DUT can be simplified to serial transmission, which makes the interface more economical and easier to implement. More details about the CDR circuitry are discussed in Chapter 5 as a case study of an application of the BERT.

3.3.3 Function Verification

The functions of the parallel BERT are clearly demonstrated by the simulation waveforms as shown in Figure 3-10 and Figure 3-11. In the simulations, the parallel BERT is set to be 8 bits in width, and the DUT setting is the same as that in Chapter 3.2 except that a serial shift register is replaced by an 8-bit parallel shift register. In the waveforms, the signal *tmeasure* controls the state of the switches: when *tmeasure* = 0, the switches are in *load* state; when *tmeasure* = 1, the switches are in *measure* state. The signal *tdetout* represents the output of the error detector in the BERT. The signal *errnum* represents the number of error bits, and the signal *samples* represents the number of transmitted bits.

Figure 3-10 demonstrates how the parallel BERT responses when the switch is changed from *load* state to *measure* state. As discussed in Chapter 3.2.4, as soon as the switch counter in the BERT reaches the threshold, the switch is changed to *measure* state. As shown in Figure 3-10, the BERT is switched to *measure* state at 1.04us. In this state, the BER measurement begins, and the number of transmitted bits (*samples*) is increased by 8

every clock cycle. All the data is correctly transmitted before 1.3us (*errslip* = 00), so the number of error bits (*errnum*) remains zero. During the period of 1.3us -1.42 us, errors are injected (*errslip* = 11), so *errnum* is increased: reaching 8 in the first cycle, 13 in the second cycle, 19 in the third cycle, 22 in the fourth cycle of the injection, and remaining 22 as the error injection is removed (*errslip* = 00) after 1.42 us. Please note that there is a delay of two clock cycles between *errslip* and *errnum* due to the data processing time.

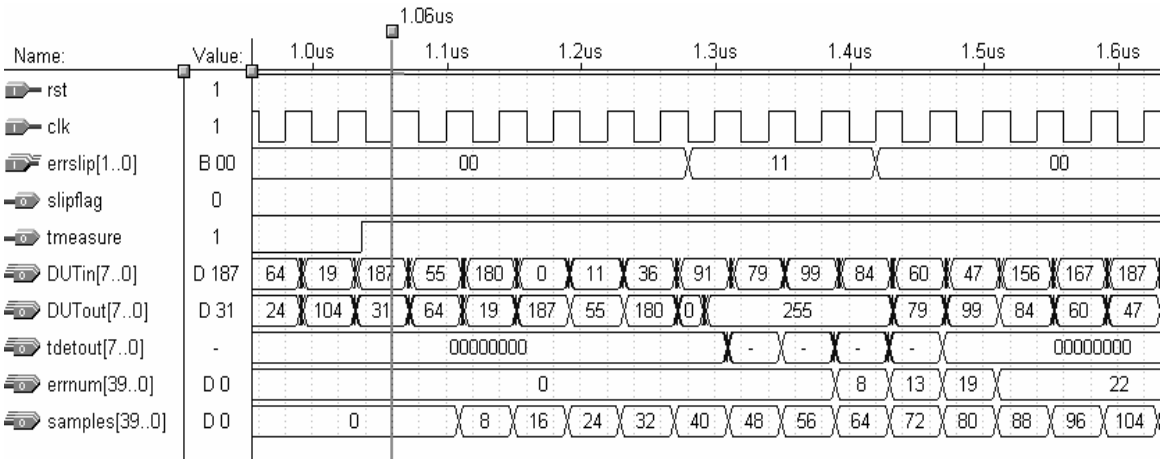


Figure 3-10: Waveforms of the Parallel BERT (Load → Measure)

As also can be seen from the signals *DUTin* and *DUTout* in Figure 3-10, the DUT exhibits a delay of three clock cycles in normal operation. When errors are injected, DUTout is set to be 11111111_B (255_D), and the output of the error detector (*tdetout*) is determined by the values expected to appear at *DUTout* in normal operation. For example, in the first clock cycle of error injection, *DUTout* is expected to be 00000000 in the normal state, so *errnum* is increased by 8; in the second clock cycle, *DUTout* is expected to be 00001011_B (11_D), so *errnum* is increased by 5 and reaches 13. The randomness of the transmitted data is also demonstrated by the contents of the signal *DUTin*.

Figure 3-11 demonstrates the ability of the parallel BERT to distinguish between a error burst and a word slip. During the period of 2.0us~6.0us, an error burst is injected (*errslip* = 11) and a lot of error bits (401) are generated. As the slip indicator *slipflag* is not

asserted, it demonstrates that *errnum* indicates real errors. During the period of 8.0us~12.0us, a word slip occurs (*errslip* = 10), and the BERT also detects a lot of error bits (818 – 401 = 417). Because the slip indicator *slipflag* is asserted shortly after the slip happens, it indicates that a word slip has happened and the synchronization should be reinitiated.

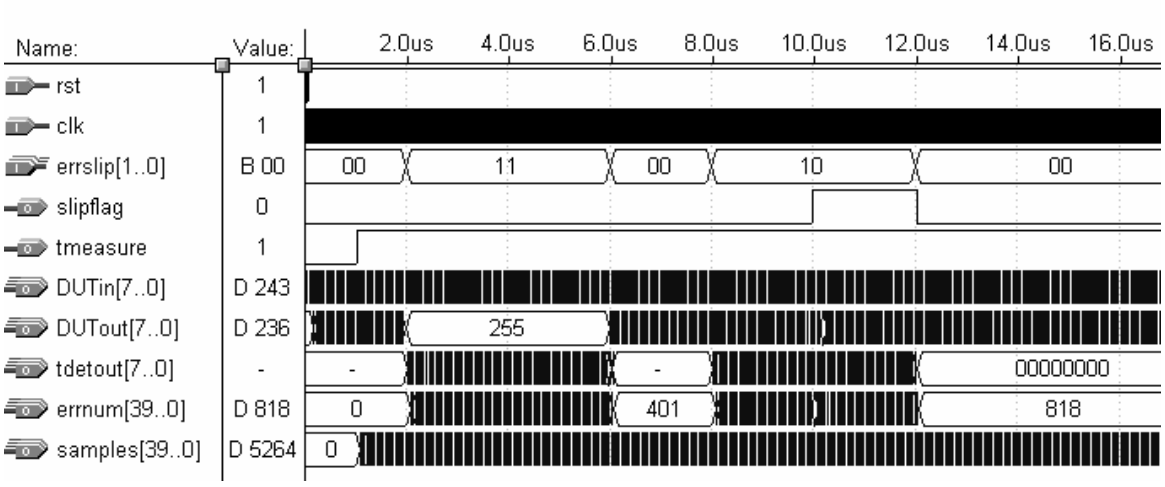


Figure 3-11: Waveforms of the Parallel BERT (Error Burst and Word Slip)

3.4 Synthesis Results

The BERT designs are built in VHDL, and can target almost any FPGA devices. The synthesis has been done using Quartus II tools by Altera. Table 3-3 shows the synthesis results of the parallel BERT design based on the Altera Mercury FPGA EMP120.

Table 3-3: Synthesis Results of the BERT

Function Block	Logic Elements	ESB Bits	fmax
BERT	384/4800 (8%)	0/49152 (0)	160.3 MHz
BERT + CDR*	837/4800 (17%)	320/49152 (<1%)	160.3 MHz

* CDR circuitry is discussed in Chapter 5.

As can be seen from Table 3-3, the BERT only occupies a small part the FPGA device. There are enough resources in the FPGA to implement other application-specified functions in a real BER testing, such as data storage, protocol implementations, special test controls, and user logic circuits.

Chapter 4 -AWGN Core Design

As discussed in Chapter 2.2.4, hardware-based BER testers are much faster than software solutions. For a hardware-based BER testing solution, a high-speed communication channel emulator is essential. A new method of implementing AWGN generator/generators in FPGAs is presented in this chapter. The whole scheme is implemented as an IP core, suitable for a single FPGA device. In this chapter, the detailed implementation of the AWGN core and its performance are presented.

4.1 AWGN Generation Method Overview

Existing methods are based on a variety of statistical techniques. After reviewing existing methods and their drawbacks in Chapter 4.1.1, we present our method in Chapter 4.1.2.

4.1.1 Existing Methods

4.1.1.1 CLT Method

The CLT method is based on Central Limit Theorem (CLT). According to CLT, if X is a random real variable of mean m_x and standard deviation δ_x, the random variable X_N defined as

$$X_N = \frac{1}{\delta_x \sqrt{N}} \sum_{i=0}^{N-1} (x_i - m_x)$$

tends toward the Gaussian distribution of zero mean and the unity standard deviation, when N tends toward infinity. In the above expression, x_i, are N independent instances of the variable X.

Traditionally, the CLT method is implemented using an accumulator. The AWGN generator in [28] is based on this method. This generator consists of four M-sequence generators, three adders and an accumulator. The M-sequence generators are linear

feedback shift registers (LFSRs) of lengths 28, 29, 30 and 31. By treating the last 10 bits of the shift register as a signed binary integer, a random number is generated. The AWGN generator produces one output every 12 system clock cycles by adding 48 10-bit random numbers. The output rate is 1 MHz.

If only the CLT method is used to generate Gaussian distribution, the convergence is very slow. Numerous independent random variables are needed to implement a high accuracy AWGN generator. In this case, either a very larger number of LFSRs and adders are needed or the output rate is very slow. So the CLT method is not suitable for high-speed applications.

4.1.1.2 Box-Muller Method

As a key tool in statistics, the Box-Muller algorithm can be applied to generate Gaussian distribution. This generator is shown in Algorithm 4-1.

1. Generate two independent random values x_1 and x_2, uniformly distributed over [0,1].
2. Obtain:
   $f(x_1) = \sqrt{-\ln(x_1)}$
   $g(x_2) = \sqrt{2}\cos(2\pi x_2)$
3. Generate Gaussian variable
   $n = f(x_1)\, g(x_2)$

Algorithm 4-1: Box-Muller Method

This method has the advantage of maintaining a one-to-one correspondence between the random numbers used and the Gaussian random variables produced, with every group of random values generating in line 1 producing one output in line 3 in Algorithm 4-1.

4.1.1.3 Mixed Method

A mixed method used to implement an AWGN generator in FPGAs is proposed in [17]. This method is based on the combination of the Box-Muller algorithm and Central Limit Theorem. The detailed hardware implementation and performance evaluation of the generator are presented in [17]. Figure 4-1 shows the block diagram of the mixed method.

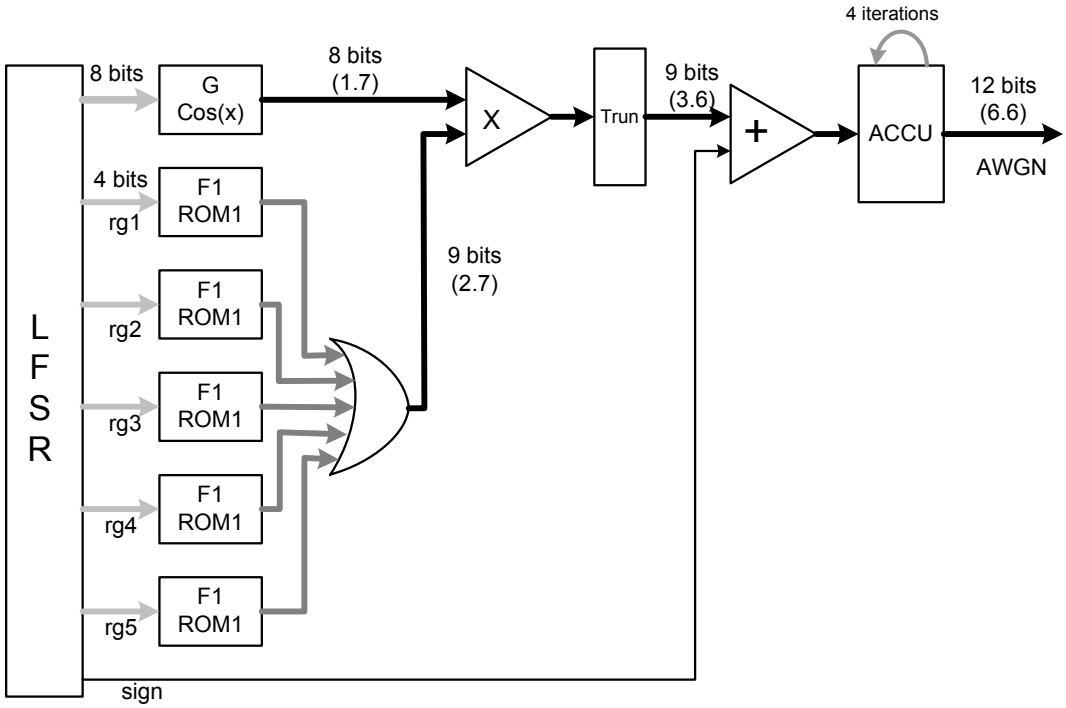


Figure 4-1: Block Diagram of Mixed Method [17]

In terms of speed and accuracy, the proposed implementation is very efficient and has been adopted by industry to generate AWGN [29], [30]. However, the Box-Muller method needs to implement both *ln* and *cos* functions. The methods of sampling and quantization for these functions need lots of considerations, such as number of recursions, relative position of the sample in a segment, etc. The efficient implementation is hence not straightforward. Moreover, the implementation of Central Limit Theorem in [17] greatly slows down the output speed.

4.1.1.4 Cellular Automata Based Method

The above existing methods all use LFSRs to produce pseudo-random numbers. LFSRs are very popular and effective for pseudo-random number generation, and have long been relied for generation of random numbers [31], [32]. However, when many sequences of random numbers are needed, the area consumed by LFSRs is large. One good alternative is using cellular automata to generate a large number of random numbers. In 1986, Wolfram [33] suggested that cellular automata could be used for efficient hardware implementation for random number generators. The generated random numbers can be transformed to Gaussian variables [34].

Cellular automata can be thought of as dynamic systems, discrete in both time and space [35]. The principle of cellular automata is that the next value of each register is calculated by a Boolean function from the current values of immediate neighbours and itself. The Boolean function is called computation rules and categorized by Wolfram [35]. One of the setups that can generate *m*-sequences is a careful mix of Rule 90 and Rule 150 as shown bellow:

$$Rule-90: a_i(t+1) = a_{i-1}(t) \oplus a_{i+1}(t)$$
$$Rule-150: a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t)$$

where $a_i(t)$ is the content of register *i* at time *t*. The positions of Rule-90 and Rule-150 in a register array can be determined according to [36], [37].

Due to its simplicity and regularity of design, cellular automata have been widely used for uniformly distributed random number generators [38], [39], [40], [41]. The transformation from uniform variables to Gaussian variables can be done based on CLT method. Another method for this transformation is illustrated in Figure 4-2 [34].

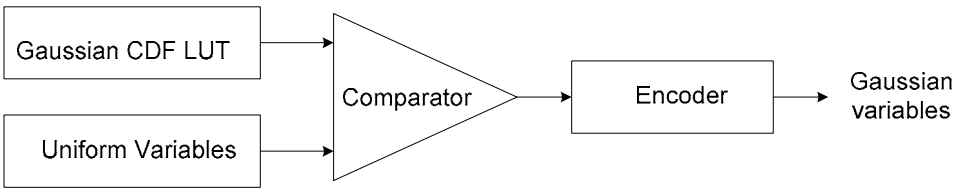


Figure 4-2: **Transformation from Random Variables to Gaussian Variables**

In Figure 4-2, an *n*-bit uniform variable is compared with the numbers in a Gaussian cumulative distribution function (CDF) conversion table and then encoded to an *l*-bit Gaussian random number. This process is equivalent to grouping all points in the area under a Gaussian PDF to several columns, randomly picking a point, and substituting the point with the one number that is the average value of the numbers in the column.

However, this transformation is usually difficult to implement for applications where high speed and high precision are required.

4.1.2 Our Method

In order to overcome the disadvantages of the existing methods, we propose a novel method to implement AWGN generators. Our method consists of Polar method as shown in Algorithm 4-2 and our CLT method.

1. **Do**
2. Generate two independent random variables, U_1 and U_2, uniformly distributed over [0,1].
3. Set: $V_1 = (2 * U_1) - 1$

   $V_2 = (2 * U_2) - 1$
4. Set: $S = V_1^2 + V_2^2$
5. If $S >= 1$, go back to line 2 and get new values for U_1 and U_2
6. **Loop** until $S < 1$
7. Set: $W = \sqrt{-2\ln(s)/s}$
8. Generate two independent Gaussian variables

   $X_1 = V_1 * W$

   $X_2 = V_2 * W$

Algorithm 4-2: Polar Method

As an improvement to the Box-Muller algorithm, Polar algorithm eliminates the trigonometric calculations. Polar algorithm provides a method to generate two independently distributed Gaussian variables with zero mean and the unity standard deviation [42]. For single channel emulation, we only need to generate one Gaussian variable (x_1 or x_2). The proof of the validity of this method is elaborated in [42].

Polar algorithm is faster than the Box-Muller algorithm because it uses few transcendental functions, even though it throws away, on average, 21% of numbers generated in the Do loop.

Our CLT method adopts pipelined architecture instead of an accumulator adopted by the traditional CLT method; therefore, our CLT method eliminates speed penalty while improving the accuracy of the AWGN generator.

4.2 Generating Random Variables

According to Algorithm 4-2, the first step to generate a Gaussian variable is to generate two independent random variables, U_1 and U_2, uniformly distributed over [0,1]. In the past, the random variable generation was mostly done by software. The software-based methods are well understood [44], [45], [46], but they frequently require complex arithmetic operations and thus are not feasible to be constructed in hardware. In this section, some techniques suitable for random number generation in hardware are first discussed, then the method used to generate U_1 and U_2 is introduced.

4.2.1 One Bit Random Number Generator

Ideally, the generated random variables should be uncorrelated and satisfy any statistical test for randomness. True randomness can be derived from certain physical phenomena, such as thermal noise in electronic circuit because of its well-qualified spectral and statistical properties. Figure 4-3 shows a representative implementation of a 1-bit true random variable generator [47]. In this circuit, the source *Vnoise*, which is the thermal noise of a precision resistor, is amplified and then passed to a high-speed comparator. The

reference voltage of the comparator, *Vref*, corresponds to the mean voltage of the amplified noise signal. The output of the comparator is sampled and latched to a register. The latched 1-bit signal exhibits true randomness.

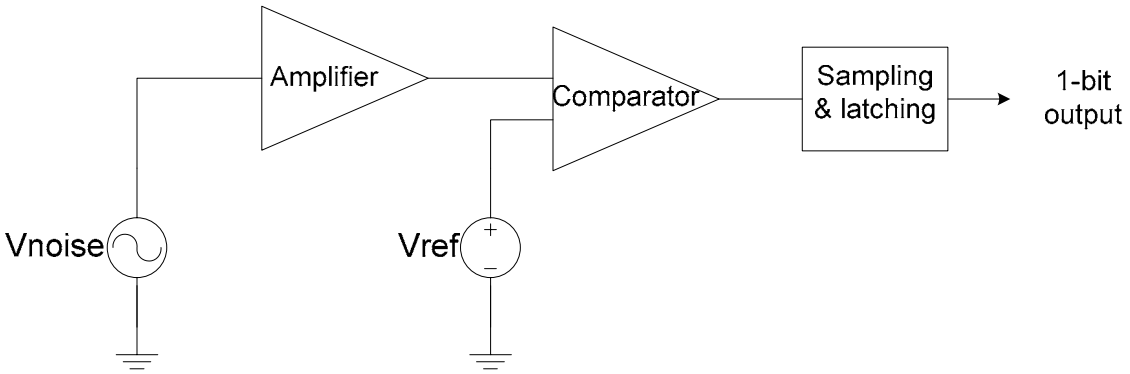


Figure 4-3: A True 1-bit Random Variable Generator

The true random variable generator consists of mainly analog components and cannot be implemented by pure digital circuitry. The mixed-signal implementation significantly increases the system complexity and is relatively slow, so this method is not suitable for high-speed digital circuit design.

One common solution is to use linear feedback shift registers (LFSRs) [48] to generate pseudo random variables. The sequence of a LFSR is based on specific mathematical algorithms. Though the generated pattern is repetitive and predictable, the sequence appears to be random if the cycle period of the LFSR is very large.

An LFSR uses feedback from the various stages of an *m*-bit shift register, connected to the first stage by means of XOR gates. The LFSR generating a single bit random number is based on the recurrence equation:

$$x_n = a_1 \cdot x_{n-1} \oplus a_2 \cdot x_{n-2} \oplus \cdots \oplus a_m \cdot x_{n-m}$$

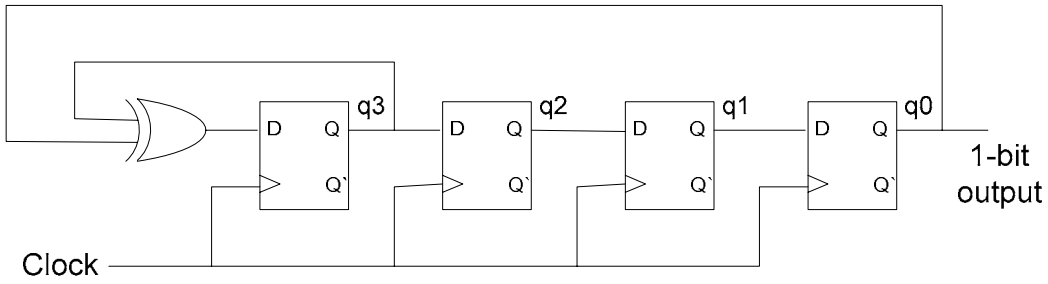
Here, x_i is the i^{th} number generated, a_i is a pre-determined constant that can be either 0 or 1, $\cdot$ is the AND operator, and $\oplus$ is the XOR (exclusive-OR) operator. This implies that a new number (x_n) can be obtained by utilizing *m* previous values $(x_{n-1}, x_{n-2}, \cdots, x_{n-m})$ through a sequence of AND-XOR operations.

In an LFSR, the maximum achievable period is determined by *m*, which is 2^m-1. In order to achieve the maximum period, a special set of a_is has to be used. In these sets, most a_is are 0; only two to four of them are 1. Thus, the actual recurrence equation is fairly simple, and the recurrence equations are different for different values of *m*. Many books, such as [48], [49], have tables that list the recurrence equations exhaustively. Table 4-1 lists the recurrence equations for *m* with values from 2 to 8.

Table 4-1: Sample Recurrence Equations

M	Recurrence equation
2	$x_{n-1}\ x_{n-2}$
3	$x_{n-1}\ x_{n-3}$
4	$x_{n-1}\ x_{n-4}$
5	$x_{n-2}\ x_{n-5}$
6	$x_{n-1}\ x_{n-6}$
7	$x_{n-1}\ x_{n-7}$
8	$x_{n-2}\ x_{n-3}\ x_{n-4}\ x_{n-8}$

As an example of the recurrence equation implementation in hardware, the circuit of an LFSR with *m* = 4 is shown in part (a) of Figure 4-4. A four-bit shift register, with the signals from the first and fourth stages fed back through an XOR gate, generates 15 different patterns during successive clock cycles. If the initial value of the shift register is set to $q_3\ q_2\ q_1\ q_0$=1000, then the output of each register and the generated 1-bit output can be determined. The results are shown in part (b) of Figure 4-4. As can be seen from Figure 4-4, in an LFSR implementation, an initial seed is needed to set the initial condition of the registers. The seed can be any state except for all 0 combinations, which causes the random sequence to be stuck at zero forever.



(a) Circuit

q_3	1	1	1	1	0	1	0	1	0	0	1	0	0	0	1	…
q_2	0	1	1	1	1	0	1	1	1	0	0	1	0	0	0	…
q_1	0	0	1	1	1	1	0	0	1	1	0	0	1	0	0	…
q_0	0	0	0	1	1	1	1	1	0	1	1	0	0	1	0	…
Output	0	0	0	1	1	1	1	1	0	1	1	0	0	1	0	…

(b) Generated Sequence

Figure 4-4: LFSR-based Pseudo Random Number Generator

As can be seen from Table 4-1 and Figure 4-4, an LFSR-based random number generator only needs an *m*-bit shift register and 1 to 3 XOR gates and thus the resulting circuit is very small and its operation is extremely fast. The generated sequence patterns have the characteristics of randomly created numbers. Furthermore, since the period grows exponentially with the size of the registers, large non-repetitive sequences can be easily generated. For example, with a 64-bit generator running at 1 GHz, the period is more than 500 years.

4.2.2 Multiple-Bit Random Number Generator

It is also possible to generate multiple-bit random numbers using a LFSR. For example, one can use the LFSR in Figure 4-4 to generator 4-bit random variables (i.e. $q_3\ q_2\ q_1\ q_0$). However, the generated random variables are highly correlated and fail many statistical tests since a new random number keeps most bits from the old number and contains only 1-bit new information. To over the correlation problem, it is necessary to replace all bits

in the random number rather than just one bit. One solution is to use parallel-LFSR method to generate multiple-bit random numbers. In this method, *m* independent LFSRs are used to generate *m*-bit random numbers.

Besides the parallel-LFSR method, there are other methods more efficiently utilizing FPGA resources to generate multiple-bit random numbers. For example, multiple-bit leap-forward LFSR method [50] is suitable for a small number of bits, and multiple-bit lagged Fibonnaci method [45], [50], [51], [46] is suitable for a large number of bits. However, their implementations are not as simple as the parallel-LFSR method.

In addition, as discussed in Chapter 4.1.1.4, cellular automata can also be used to generate random numbers, and is especially suitable for generating a large number of random variables.

In our AWGN generator design, the parallel-LFSR method is used to generate random numbers U_1 and U_2. The FPGA resources taken by implementing U_1 and U_2 is very small. In the design, each of the two variables in line 2 of Algorithm 4-2 is set to be four bits in width, so four single bit random number generators are used to form a four-bit random generator. There are totally eight independent LFSRs used to generate the two 4-bit independent random variables (U_1 and U_2) in Algorithm 2. The length of each of the LFSR is different, and all the LFSRs produce maximum periods. In this case, U_1 and U_2 are uniformly distributed between "0000" and "1111" (binary form). All these four bits represent the fractional part, so we get two independent random variables, U_1 and U_2, uniformly distributed over [0, 0.9375_D]. The maximum period of U_1 and U_2 is determined by the sum of all the lengths of the LFSRs, which can be adjusted to meet a required period.

4.3 Gaussian Variable Generation

In this section, the detailed implementation of AWGN generators is elaborated based on the generated random numbers U_1 and U_2. First the structure of a single AWGN generator

is presented, then the structure of two AWGN generators is derived. Finally, a novel accuracy improvement method is introduced.

4.3.1 Implementing a Single Generator

Algorithm 4-2 shows that the Polar method can generate two independent Gaussian variables with a single iteration. It can also be simplified to fit the structure of a single AWGN generator. Figure 4-5 shows the block diagram of a single AWGN generator. In this implementation, pipelined structure is adopted to optimize the output speed.

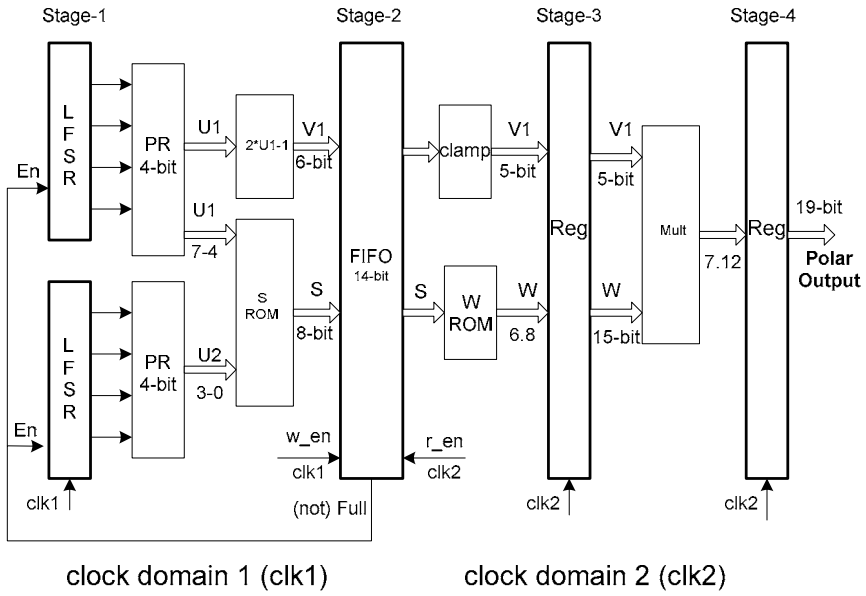


Figure 4-5: Block Diagram of a Single AWGN Generator

4.3.1.1 Generating V_1 and *S*

V_1 is generated using signed adders performing the computation

$$V_1 = U_1 + U_1\text{-}1$$

Before the addition, U_1 is converted to a 6-bit signed number.

Computing *S* involves lots of additions and multiplications, which are very time-consuming. Most modern FPGAs include embedded RAM blocks. These blocks enable us to implement complex arithmetic operations with ROM-based designs, which are

faster than the traditional arithmetic circuit implementations. Generating *S* takes advantage of this FPGA feature. ROM-based computation is used to implement the function

$$S = (2 * U_1 - 1)^2 + (2 * U_2 - 1)^2$$

The concatenation of U_1 and U_2 is set to be the address of the ROM and the values of *S* are set to be the data stored in the ROM. Both the address and data *S* are 8 bits in width. All the 8 bits for *S* represent its fractional part. If computed $S \geq 1$, the value of *S* stored in ROM is set to "00000000". As data "00000000" is only used to control the *w_en* signal of the FIFO (discussed in the next section), the effective range of data *S* is between "00000001" and "11111111" in binary form. In other words, the effective range of data *S* is over [0.00390625, 0.99609375] in decimal form.

4.3.1.2 FIFO Implementation

In Algorithm 4-2, a Do loop (line 1 to line 6) is used to generate qualified *S*, V_1 and V_2 for line 7 and line 8. On the average, line 1 to line 6 are executed 1.3 times of line 7 and line 8. To achieve a constant output rate, a synchronizing FIFO is used. The job of the FIFO is to synchronize the implementation of the loop and the implementation of line 7 and line 8 in Algorithm 4-2 without losing or corrupting data. The width of the FIFO is 14 bits, 6 bits for V_1 and 8 bits for *S*. The loop implementation logic sends data to the FIFO receiver and the FIFO transmitter sends out data to the implementation logic of line 7 and line 8. The structure of the synchronizing FIFO is show in Figure 4-6.

The FIFO uses two clocks, *clk* for the receiver and *clk2* for the transmitter. When *S* is not equal to "00000000", *w_en* is enabled, the FIFO receiving V_1 and *S* at the rising edge of *clk.* Otherwise, no data is written to the FIFO and the next value of *S* is checked. In this case, the receiving data rate is a variable. In order to let the FIFO send data out at a constant rate (*clk2*), *clk2* must be smaller than the average rate of receiving data. By setting the depth of the FIFO to be 16 and *clk2* to be half of *clk*, a constant output rate is achieved.

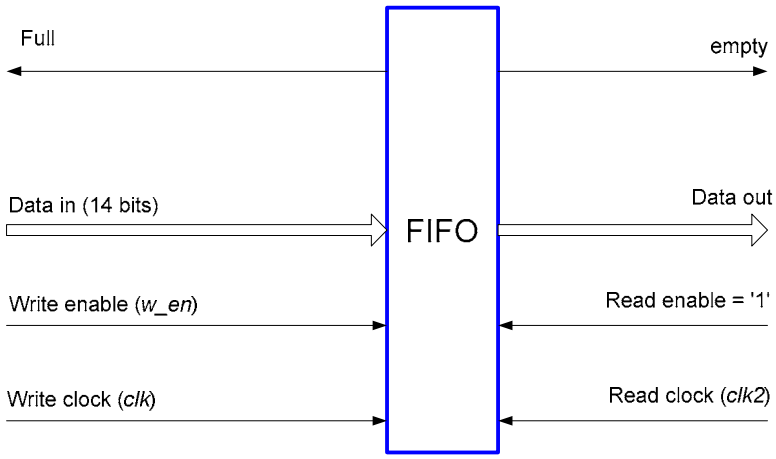


Figure 4-6: Structure of the Synchronizing FIFO

In the FIFO design, four extra parameters (*read_pointer*, *write_pointer*, *counter* and *full*) are used to deal with the issues of synchronization, overflow and underflow. The parameter *counter* indicates how many locations have been filled with data in the FIFO according to the equation

$$counter = write\_pointer - read\_pointer$$

In situation of *write_pointer* = *read_pointer*, we do not know whether we have an empty FIFO or full one. To prevent this problem, we consider the FIFO full when 15 out of the 16 locations are occupied with unread data. When *counter* =0, it indicates the FIFO underflow. When *counter* = 15, it indicates the FIFO overflow.

The FIFO is guaranteed not to overflow by the following mechanism: when *counter* >= 14, *full* signal is asserted and the LFSRs are disabled. The FIFO *w_en* is disabled one clock cycle later. In this case, the FIFO can still receive one group of data, so no data is missing. As the LFSRs are disabled, they stop generating data and no more data will be sent to the FIFO until *counter* < 14. Once *counter* < 14, LFSRs are enabled again and the FIFO *w_en* is enabled one clock cycle later if *S* does not equal to “00000000”. By this way, the counter will always be smaller than 16. The FIFO will never really overflow.

With the above mechanism, the FIFO begins to send out data once *counter* reaches 14. On average, the possible rate of writing data to the FIFO is around 1.5 times faster than the rate of reading data from the FIFO when the clock rate of *clk2* is set to be half of

the clock rate of *clk*. In this case, the FIFO, with a depth of 16, can still send data out even no data is written to it in 28 consecutive *clk* cycles. From the simulations, 5 was the maximum number of *clk* cycles in which no data was written to the FIFO (this number depends on the lengths and taps of LFSRs). In fact, a FIFO with depth of 8 is enough. We choose 16 to make our design more reliable. From the simulation results of 1 million clock cycles, the *counter* is always bigger than 10. It is concluded that the design is reliable enough to prevent underflow from happening.

4.3.1.3 Generating W

ROM-based design is also used to implement the function in line 7 in Algorithm 4-2

$$W = \sqrt{-2\ln(s)/s}$$

S denotes the address of the ROM. The width of *S* is 8 bits and all represent the fractional part. *W* denotes the data stored in the ROM. The plot of *W* as a function of *S* is shown in Figure 4-7.

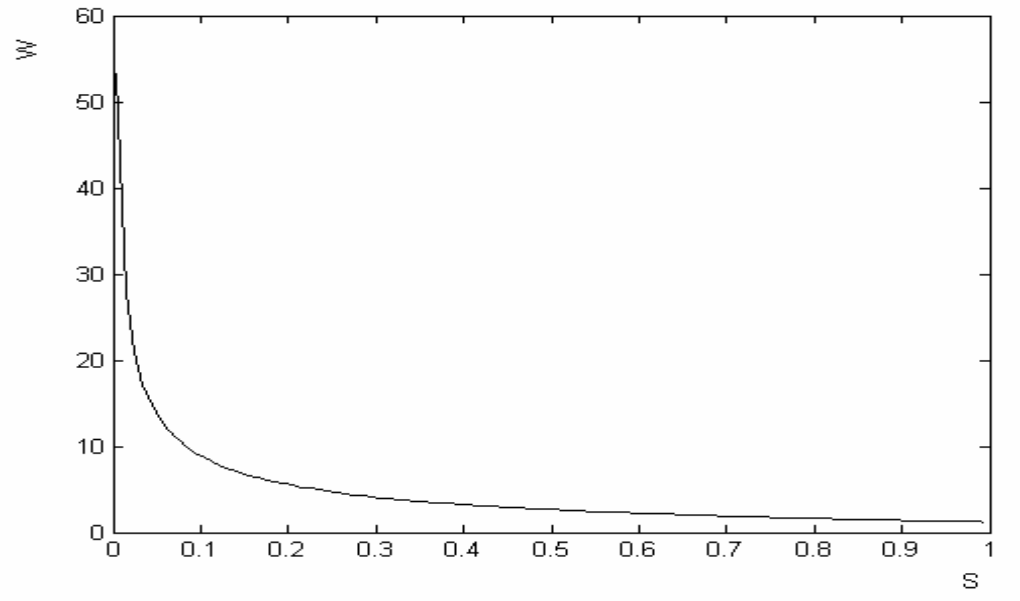


Figure 4-7: Plot of Function *W*(*S*)

As *S* is between 0.00390625 and 0.99609375, according to Figure 4-7, *W* is between 54.2835 and 0.0886. To represent *W* in binary form, 6 bits are needed to represent its

integer part. In our design, we use 14 bits to represent *W*, 6 bits for the integer part and 8 bits for the fractional part.

As the absolute value of V_1 from the FIFO is always smaller than 1, V_1 from the FIFO is clamped to 5 bits, 1 bit for the sign and 4 bits for the fractional part. The register *reg3*, which stores the values of *W* and clamped V_1, is clocked by *clk2*, the clock rate of reading data from the FIFO.

4.3.1.4 Generating Outputs

The last step of implementing the AWGN generator is to implement the function

$$X_1 = V_1 * W$$

This step is completed by a single signed multiplier. Before performing multiplication, one *'0'* is concatenated to the most significant bit of *W* to convert *W* to signed form. The output of the multiplication is 19 bits in width, 1 bit for sign, 6 bits for the integer part and 12 bits for the fractional part. This output is sent to the output register *reg4*. The output of the *reg4* is what we need, which behaves like a Gaussian random variable.

The output of the AWGN generator can however be truncated to different widths, depending on application needs.

4.3.2 Implementing Two Generators

Figure 4-5 shows the structure of a single AWGN generator. For modulated data like QPSK signals, two noise generators might be needed for *I* and *Q* channels. According to Algorithm 4-2, the proposed one generator structure can be easily modified to implement two AWGN generators by adding V_2 implementation and another multiplier. The block diagram of two AWGN generators is shown in Figure 4-8.

In this structure, the width of the registers for each stage should be increased accordingly. As can be seen, the hardware cost is very small to add another AWGN generator based on

the structure of a single generator. The proposed method of AWGN generation is especially suitable for multi-channel emulation.

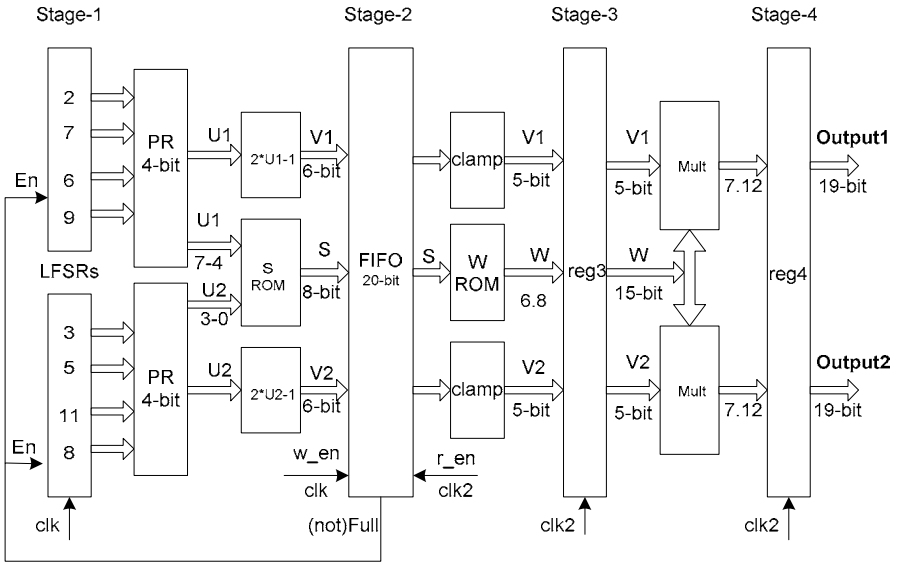


Figure 4-8: Block Diagram of Two AWGN Generators

4.3.3 Accuracy Improvement

In our implementation, Central Limit Theorem method can also be used to smoothen the variation of the distribution when high accuracy is need. As discussed in Chapter 4.1.1.1, CLT method traditionally uses an accumulator. However, the accumulator will slow down the speed of the output. For example, when $N = 4$, where N is the number of random variables to be accumulated, the output rate after the accumulator is only one-fourth of that before the accumulator. As our implementation can produce two AWGN generators with little hardware cost, we can achieve one AWGN generator with better performance by simply adding the outputs from the two AWGN generators shown in Figure 4-8. This implementation does not incur the speed penalty.

To overcome the speed penalty problem, we propose a new CLT method for accuracy improvement. The block diagram of this method is shown in Figure 4-9, which implements the case when $N = 4$. The proposed scheme does not exhibit the speed penalty while improving accuracy.

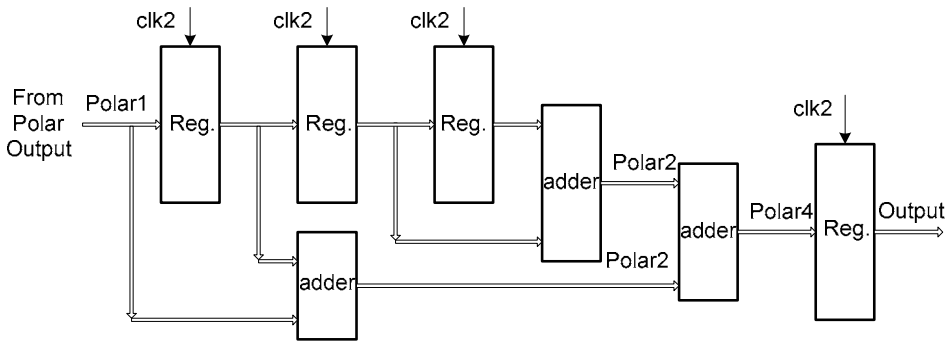


Figure 4-9: New CLT Method

4.4 Experimental Results

In this section, the statistical results based on the outputs of our AWGN generators are presented and compared with the theoretical properties of Gaussian distributions and other methods. The AWGN theoretical properties are discussed in Chapter 2.2.2. Experimental results demonstrate the suitability of our AWGN generators for channel emulation.

4.4.1 Experimental Statistical Properties

As the outputs of an AWGN generator are random variables with a mean of m_x and a standard deviation of δ, its performance evaluation should be based on statistics of the real outputs of the AWGN generator. According to the lengths of the LFSRs used to generate U_1 and U_2, the period of the generator may reach the range of 2^n, where *n* is the sum of the lengths of the LFSRs. When *n* is equal to 50, the period is greater than 10^{15}. In this case, the complete verification of the AWGN generator should be based on the statistics of a very large number of samples, at least greater than 10^{15}. Statistically evaluating the performance of such larger number of samples needs a lot of hardware resources and time. Our experiments show that statistical results of thousands samples are a good approximation for the performance evaluation of the real AWGN generator. In this part, we show the statistical properties of 10,000 and 500,000 samples from the output of

our AWGN generator. The process of getting the statistical properties consists of the following four steps:

1) Write the AWGN generator (VHDL top-level design) binary outputs to a text file.

2) The output data is imported to a *C* program that generates the probability density function (PDF) of the outputs from the generator by sorting the outputs and computing the probability density $P[x_n]$ of each output.

3) The mean m_x and standard deviation δ of the AWGN generator are calculated from its PDF according to the following definitions.

Mean $$m_x = \sum_n x_n P[x_n]$$

Mean-Square $$m_x^{\,2} = E[x^2] = \sum_n x_n^{\,2} P[x_n]$$

Variance $$\delta^2 = E[(x - m_x)^2] = E[x^2] - m_x^{\,2}$$

4) $Q(x)$ of our AWGN generator is obtained according to

$$Q(x) = \sum_{i=1}^{n} x_i P[x_i]$$

where x_i, i = 1, 2, …, n are the possible discrete values from our AWGN generator that meet the condition of $x_i \geq x$; $P(x_i)$, i = 1, 2, …, n are the possibilities of x_i.

In this evaluation process, $Q(x)$ is the area under the tail of Gaussian PDF. It represents the probability that the Gaussian variable is between x and +∞. The theoretical value of $Q(x)$ is computed according to

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-t^2/2}\, dt$$

$$= \frac{1}{2} erfc\left(\frac{x}{\sqrt{2}}\right)$$

where $erfc(x)$ is the complementary error function.

Table 4-2 shows the mean, variance and standard deviation of our generator. Displayed are the cases of 10,000 and 500,000 samples, respectively.

Table 4-2: Performance of our AWGN Generator

Samples	10,000	500,000
Mean	0.015835	0.008649
Variance	0.867620	0.866119
Standard Deviation	0.931461	0.930655

As we can see from Table 4-2, the relative error of mean and standard deviation of our AWGN generator is very small. For 500,000 samples, the relative error of mean is 0.008649 and the relative error of standard deviation is 0.069345. We can also see from the above table that the relative error of the mean decreases when the number of samples increases.

The relative error of Q(x) of our AWGN generator is shown in Table 4-3. Relative errors are computed according to

$$relative \quad error = \frac{OurQ(x) - TheoryQ(x)}{TheoryQ(x)}$$

Table 4-3 shows the results of Polar algorithm only implementation (Figure 4-5) with 10,000 samples and the implementation combining Polar algorithm and Central Limit Theorem (Figure 4-5 + Figure 4-9) with 10,000 and 500,000 samples.

As can be seen from Table 4-3, our method with the parameters shown in Figure 4-5 implements a high precision AWGN generator even with a limited number of samples. Our proposed CLT method shown in Figure 4-9 can further smoothen the variation of the distribution. We can also see from the above table that the relative error of Q(x) decreases when the number of samples increases.

Table 4-3: Q(x) Relative Error of our AWGN Generator

x	Theory Q(x)	Q(x) Relative Error of Our Generator		
		Figure 4-5	Figure 4-5 + Figure 4-9	
		10,000 samples	10,000 samples	500,000 samples
0	0.5000	2.76 %	1.02 %	0.24%
0.2	0.4207	-2.50 %	0.50 %	0.42%
0.4	0.3446	1.69 %	0.26 %	0.55%
0.6	0.2743	-0.10 %	1.06 %	0.80%
0.8	0.2119	1.88 %	1.74 %	1.09%
1.0	0.1587	4.70 %	3.21 %	1.20%
1.2	0.1151	-7.17 %	3.99 %	1.49%
1.4	0.0808	-4.29 %	4.95 %	1.85%
1.6	0.0548	-3.06 %	6.57 %	2.37%

4.4.2 Synthesis Results

Table 4-4 shows the synthesis results obtained with the parameters and structure shown in Figure 4-5. The synthesis has been done using Quartus II tools by Altera.

Table 4-4: Synthesis Results of the AWGN Generator

FPGA Device	Logic Elements	ESB Bits	Output Rate *
EP1M120F484C7	336/4800 (7%)	5856/49152 (11%)	73.48MHz

* *Output rate is counted in words. The output is 19 bits in width.*

As seen from Table 4-4, the AWGN generator only takes a small part of the FPGA. It is possible to emulate a whole communication system or a test scheme, where a communication channel emulator is needed, in a single FPGA device.

4.4.3 Comparison

Table 4-5 shows the performance comparison of our method shown in Figure 4-5 and Figure 4-9 with the mixed method in [17]. The statistical properties are based on 500,000

samples. The mean, variance and standard deviation properties of the mixed method are from our design based on [17]. The output rate of the mixed method is from [17], where the FPGA device is 10K100EQC240-1.

Table 4-5: Comparison of our Method with Mixed Method

Implementation	Our Method	Mixed Method
Mean	0.008649	0.001009
Variance	0.866119	2.065968
Standard Deviation	0.930655	1.437347
Output Rate	73.48MHz	24.5MHz

As can be seen from Table 4-5, both methods are suitable for implementing high accuracy AWGN generators, but our method exhibits higher speed; especially, when N increases for higher accuracy, the output rate of the mixed method will further decrease while ours keeps constant.

Chapter 5 – Case Studies

This chapter addresses the applications of the proposed BERT core and the AWGN core. Two cases are studied: one is testing a gigabit clock data recovery (CDR) circuitry included in Altera Mecury FPGAs; the other is testing the BER performance of digital baseband transmission under different noise conditions. We demonstrate through the case studies that the proposed BER testing solutions exhibit advantages in cost and speed over existing test methods.

5.1 CDR Circuitry Testing

5.1.1 Significance of the Serial Communication Interface

Serial communication interfaces support multi gigabit data transmission. In recent years, the rapid development of two technologies makes the high-speed communication available. The first one is differential I/O standards, such as low-voltage differential signaling (LVDS), low-voltage positive emitter coupled logic (LVPECL) and pseudo-current mode logic (PCML). Typically, single-ended I/O standards, such as in PCI and VME bus standards, are noise limited and load limited to about 200 Mbps. They reach noise limitations at frequencies of about 250 MHz before signal integrity deteriorates. Differential I/O standards break the frequency barrier of single-ended I/O standards with common mode rejection and allow data transmission at higher speed, though the clock skew issue arises for differential I/O standards when the frequency nears 1 gigabit per second.

Another technology enabling high-speed serial communication is CDR. Removing clock skew concerns by encoding the clock into every data stream, CDR circuitry guarantees that the clock and data are always perfectly in phase. Hence, it eliminates frequency barriers faced by source-synchronous systems. Figure 5-1 shows the CDR functionality:

a transmitter embedding the clock in the data stream and a receiver recovering the clock from the data.

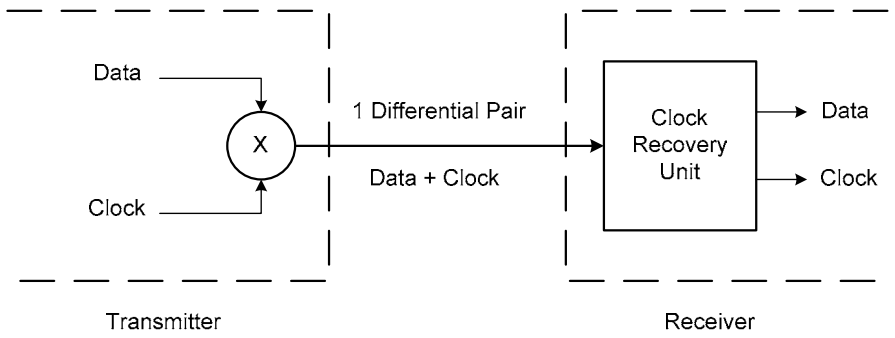


Figure 5-1: CDR Transmission Mechanism

At the present time, many serial transceivers, which employ the CDR technology and differential signaling, support applications that run up to 3.125 gigabits per second. The roadmaps of many companies point to 5 and 10 gigabits per second on each pair of wires. In addition, a wider pipe or datapath can be built by gluing multiple multi-gigabit transceivers. Figure 5-2 shows an example of transmitting 64 bits of data at 125 MHz through 8 transceivers, for an aggregated data rate of 8 Gbps.

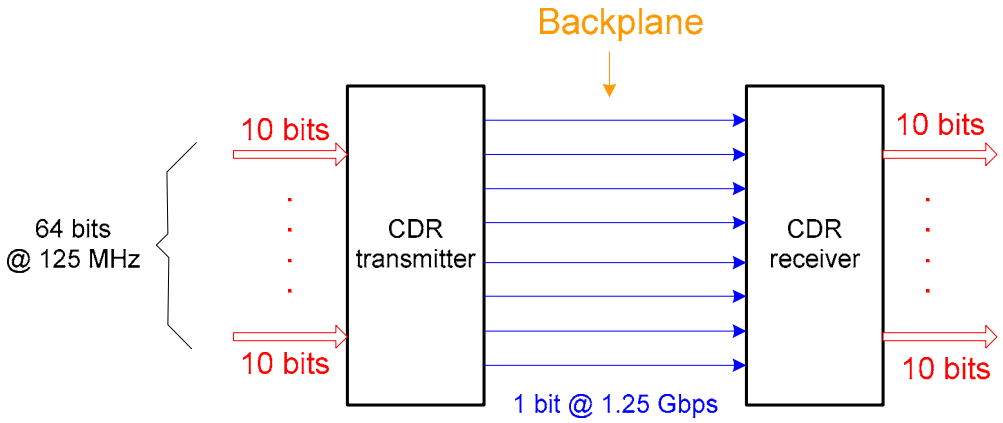


Figure 5-2: Applications of Multiple Transceivers

Besides the high-speed capability, the serial communication also provides simplified routing. In serial communication, data is transmitted one bit at a time down one wire. In

multi-gigabit transceivers, differential I/O standards are used and two wires are needed for each connection. But the wires are still much reduced from the parallel approach. Furthermore, in serial communication, the clock signal is embedded in the data and no clock skew exists. All these factors greatly simplify the routing of serial communication. Routing for parallel communication is always very channeling. Such as in an 8-bit parallel communication system, 8 or 16 wires are needed for data signals and another one or two wires are needed for the clock signal. Routing 9 or 18 wires across a board and keeping them all synchronized are hard, especially for long distance connections.

Because much less wires are used in a serial communication system compared with a traditional parallel system, this makes it possible to put more and more circuitry on one die or in one package. Serial communication greatly relieves the package pin count "bottleneck" problem for system-on-chip (SOC).

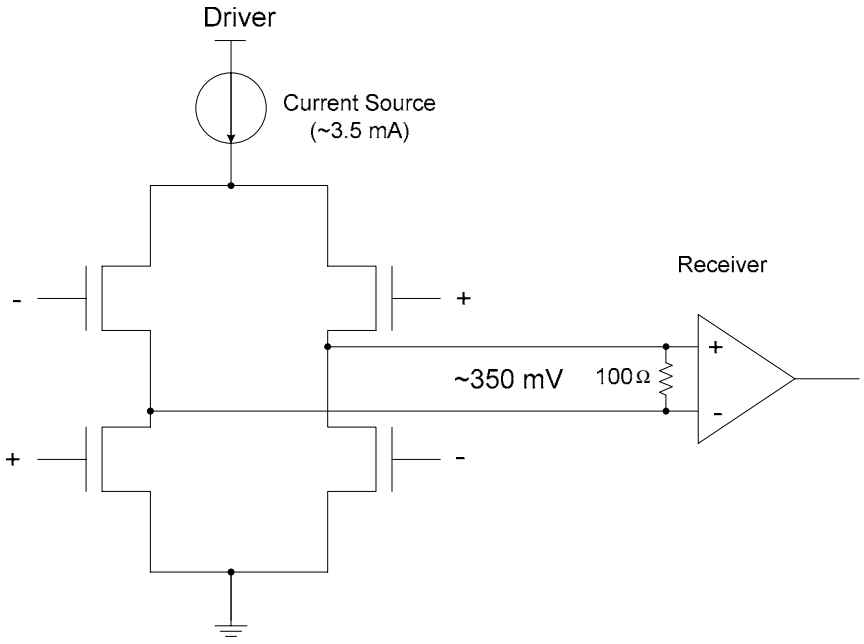


Figure 5-3: Current-Mode LVDS Driver [52]

Finally, a significant advantage of serial communication is the lower power requirement. For a 3.125 Gb link, it only consumes 300 mW. Low power consumption is mainly achieved by using low voltage differential signaling technologies, such as LVDS. As shown in Figure 5-3, LVDS technology uses a constant-current line driver rather than a

voltage-mode driver, so the supply current remains constant as the operating frequency increases, whereas the supply current for CMOS and GTL technology increases exponentially as frequency increases. The low power consumption of serial communication interfaces eliminates the need for either heat sinks or special packaging. Hence, serial communication reduces the system cost.

5.1.2 Structure of the Serial Communication Interface

A gigabit transceiver consists of two parts: a transmitter and a receiver. For most transceivers, the two parts are separated. They can function independently for half-duplex operation, or can be combined for full-duplex operation. The block diagram of a transceiver is shown in Figure 5-4.

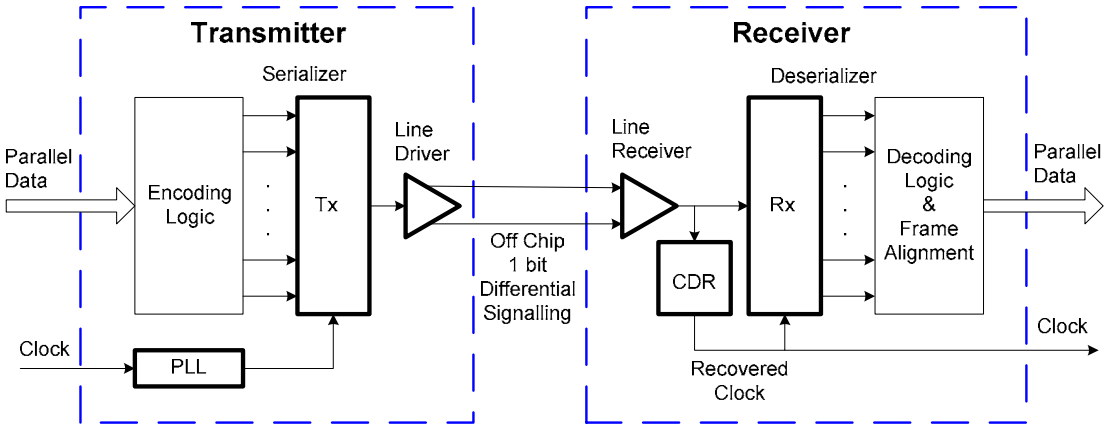


Figure 5-4: Block Diagram of a Transceiver

The transmitter receives parallel data and converts it into a serial format. The transmitter needs a clock input, which is synchronized with the parallel data. This clock is used to latch the parallel data and generate the internal high-speed serial clock for the serializer by a PLL circuit. The serialized data is sent to a differential line driver, which drives the serial data to the transmission media.

The receiver accepts high speed serial data and restores it to the original parallel format. The internal CDR circuit generates a recovered clock derived from the received serial

data, and re-times the data. Then the re-timed serial data is restored to parallel format by the deserializer.

In the above transmission mechanism, special encoding logic and decoding logic are needed to manipulate the transmitted data to make sure that the CDR circuit can function correctly. On the transmitter side, the clock is embedded in the serial data; on the receiver side, the CDR circuit extracts clock information by monitoring the transitions of the received data. Therefore, the data must have enough transitions on the transmitter side, no matter what data sequences are transmitted through the serial link. Otherwise, the clock information may be lost. For example, if we want to transmit an 8-bit word consisting of all zeros and the word is directly serialized to 8 consecutive zeros in the serial link, the receiver will have trouble to recover the clock as no transition exists.

One solution to guarantee enough transitions is to encode the original parallel data using an 8B10B encoder. An 8B10B encoder converts 8-bit words into 10-bit words, so it can always make sure there are bit transitions, regardless of what pattern you send. In the 8B10B encoding scheme, there are four different symbols for the zero character which gets interpreted as zero. This ensures that there are enough transitions for the clock recovery network to keep the system synchronized. On the receiver side, 8B10B decoding logic is used to convert the 10-bit format to the original 8-bit format. Before the decoder, a frame or word alignment block is needed to recognize the word boundary to correctly restore the transmitted parallel sequences. The restored sequences after the 8B10B decoder are presented on the output ports of the receiver.

When the transceiver is included as a macro cell integrated in an ASIC or FPGA, the parallel data input ports of the transmitter and the parallel data output ports of the receiver are connected to the internal circuit of the chip, while the serial port is interfaced to the outside media, such as an SMA cable, an optical link, a twisted pair wire, etc., depending on applications.

In the transmitter structure shown in Figure 5-4, The *Encoding Logic* block and *Decoding logic & Frame Alignment* block can be built with digital circuits; all other blocks can only be built with analog circuits. For transceivers embedded in FPGAs, the analog blocks are usually hard cores, but the users can set some parameters, such as PLL frequency boost factors and differential signaling formats. For digital blocks, the users have the freedom to use IP cores or develop their own designs.

5.1.3 Mercury Gigabit Transceiver

In this thesis, all the designs and testing schemes are implemented in an Altera Mercury FPGA. Its high performance and availability make the Mercury FPGA an ideal target device for the work.

Mercury devices seamlessly integrate a high-speed CDR-optimized programmable logic core, which provides speeds of up to 1.25 Gbps per channel and total CDR bandwidth of 45 Gbps to power next-generation high-speed connections that use standard protocols such as Gigabit Ethernet and SONET/SDH. The Mercury CDR circuitry is an ideal candidate for the case study of the serial communication testing.

In addition to the CDR function, there are many other excellent features in Mercury devices. These features also make Mercury devices suitable for implementing the AWGN core, the BERT core, and the whole testing scheme. For example, ESBs and distributed multiplier circuitry have already been used in the development of the AWGN core. More features will be used in the CDR transceiver testing setup. In Mercury family, there are two members, EP1M120 and EM1M350, which are shown in Table 5-1. The BER testing scheme can be implemented in any FPGA device, but we specifically target EP1M120 devices.

Table 5-1: Mercury Device Family

Device	Gates	Pin / Package	I/O Pins	CDR Channels	Logic Element	RAM Bits (ESB Bits)
EP1M120	120,000	484-Pin BGA	303	8	4,800	49,152
EP1M350	350,000	780-Pin BGA	486	18	14,400	114,688

The Altera Mercury gigabit transceiver is implemented in the high-speed differential interface (HSDI). The HSDI is dedicated to transmitting and receiving high-speed serial data streams between the Mercury device and other devices on a circuit board or across a backplane. Each HSDI transceiver consists of a transmitter and a receiver. Figure 5-5 shows the block diagram of one of the HSDI receiver and transmitter channels (channel 4) of a Mercury EP1M120 device.

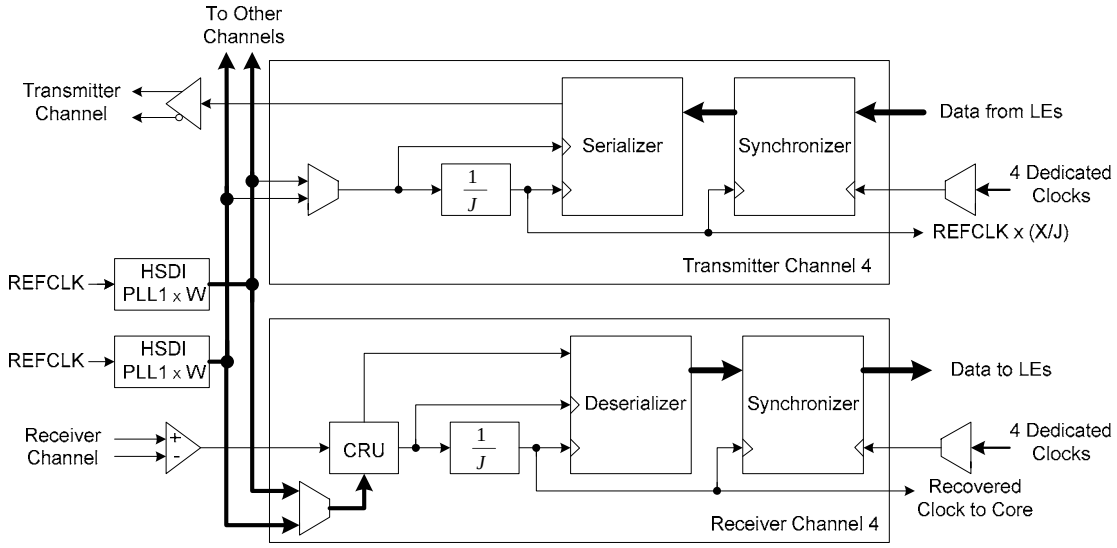


Figure 5-5: HSDI Circuitry Block Diagram

Each EP1M120 device contains 8 transceivers. Each channel has the same structure as the general transceiver as shown in Figure 5-4. But in Figure 5-5, more details are included. The transmitter channel has a dedicated synchronizer and a serializer; The receiver channel has dedicated circuitry consisting of a clock recovery unit (CRU), a deserializer, and a synchronizer. The HSDI PLL circuitry is dedicated to providing clock signals for the transceiver. The following gives a brief introduction about how the PLL and CRU in the Mercury HSDI circuitry work. The other parts in the CDR circuitry and extra logic needed to build a system, such as 8B10B coding and frame alignment, have already been discussed in Chapter 5.1.2.

In CDR mode, an external reference clock is fed to one of the two dedicated HSDI PLLs, HSDI PLL1 or HSDI PLL2. The PLL multiplies the reference clock by a factor *W*. *W* is determined by the ratio of the reference clock frequency and the rate of transmitted data stream. The two dedicated HSDI PLLs are separated from the general-purpose PLLs in the Mercury device. Figure 5-6 shows a diagram of a HSDI PLL.

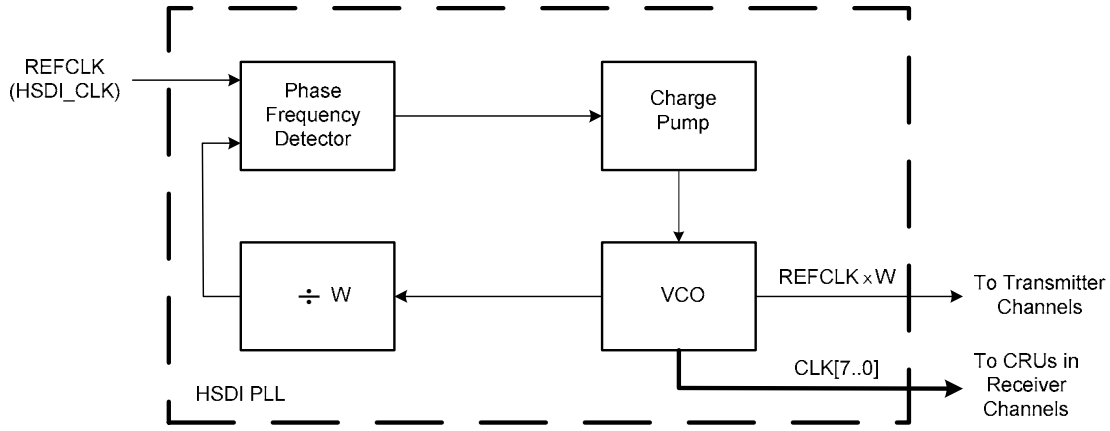


Figure 5-6: HSDI PLL Block Diagram

At each rising edge of the reference clock (HSDI_CLK), the phase/frequency detector of the HSDI PLL detects the phase difference between the reference clock and the clock generated by the voltage controlled oscillator (VCO) and divided by *W*. A voltage is generated in the charge pump by filtering the high-frequency changes in the phase difference, and this voltage drives the VCO. By taking outputs from the VCO, the PLL generates eight clocks with the same frequency as the serial input data. Each clock has a $1/8$ period phase shift.

There are no phase-relationship requirements between the reference clock and the serial input data. On each receiver channel, a CRU uses the multiplied reference clocks to generate a recovered clock in-phase with the received data. Figure 5-7 shows the CRU block diagram.

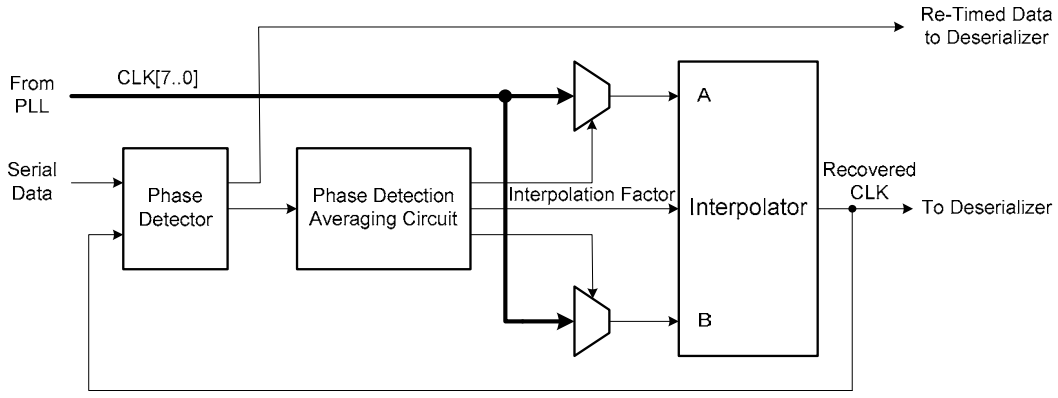


Figure 5-7: CRU Block Diagram

On each data transition, the phase detector decides if the current recovered clock is early or late. The decision of the phase detector is sampled and averaged by the phase detection averaging circuit. The averaging circuit drives two multiplexers to select the two clock phases that are closest to the ideally recovered clock. The interpolator uses the interpolation factor from the averaging circuit to generate a clock that is between the two clocks. Each of the eight equally-spaced phase clocks is divided into seven fractions; therefore, the resulting best-case clock granularity is $1/56$ of the clock period. Then, the recovered clock is used to deserialize and synchronize the data. The recovered clock can be driven to the global clock lines from channels 4 and/or 5.

5.1.4 Testing Setup

Based on the structure of the Mercury HSDI transceiver as discussed in Chapter 5.1.3 and the BERT core presented in Chapter 3, a setup to test the functionality of a Mercury HSDI transceiver is developed and shown in Figure 5-8.

The setup consists of four parts: a PLL, a HSDI transceiver, a BERT, and glue logic blocks. All the components are implemented in a Mercury FPGA, EP1M120. The BERT is the parallel BERT core presented in Chapter 3, and the data width of the BERT is 8 bits.

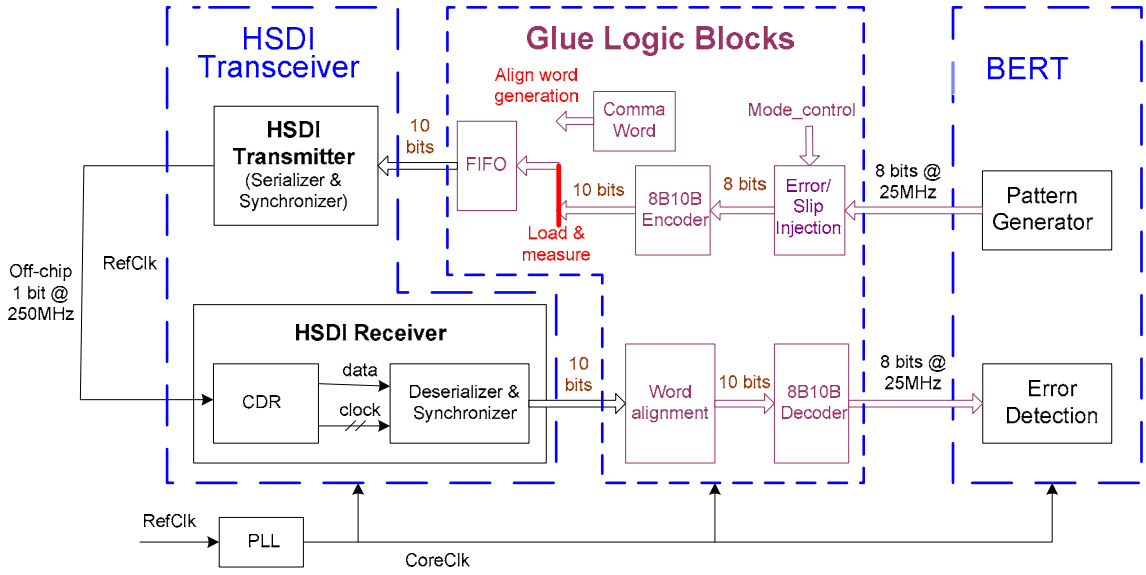


Figure 5-8: Testing Setup for the Mercury HSDI Transceiver

The PLL is one of the four general-purpose PLLs with programmable multiplication in the Mercury device, and it generates a stable core clock signal. In the testing setup, the PLL is implemented using the *MegaWizard Plug-In Manager* by instantiating the mega function *ALTCLKLOCK*. A mega function is a complex or high-level building block that can be used together with gate and flip-flop primitives in Quartus II design software [53]. Altera provides a library of mega functions, and *MegaWizard Plug-In Manage*r is a procedure used to instantiate the mega functions in Quartus II development tool. Based on the required core clock frequency, the PLL block generates the core clock signal (*CoreClk*) by multiplying and dividing the input clock signal *RefClk* by proper factors. For examples, if the available input clock signal is *RefClk* = 25 MHz and we need a core clock signal which is *CoreClk* = 50 MHz, we set the clock multiplication factor to be 2 and the clock division factor to be 1 when instantiating the mega function *ALTCLKLOCK*. The core clock signal, *CoreClk*, is used by all other blocks.

The transmitter and the receiver in the HSDI transceiver can be any one of the eight channels included in a Mecury EP1M120 device. In the test setup, we set both the transmitter and the receiver to be channel 5 when assigning input and output pins. The HSDI transmitter is implemented by instantiating the megafunction ALTCDR as a CDR

transmitter, and the HSDI receiver is implemented by instantiating ALTCDR as a CDR receiver [54]. The deserialization factor *J* is the width of the parallel data, which is set to be 10. The inclock boost factor *W* is set to be 20 when the input clock rate is 25 MHz and the core clock rate is 50 MHz, resulting in a 500 MHz clock signal for the serial data.

In this testing setup, the PLL, the HSDI transmitter and the HSDI receiver can only be built by instantiating the mega functions. These functions are hard macros in Mercury FPGA devices. They are not programmable logic cores, but we can set some parameters according to our applications.

The glue logic blocks are developed to interface the BERT and the transceiver. In the testing setup, the BERT sends 8-bit PRWSs to the glue logic blocks. The glue logic encodes the 8-bit sequences to 10-bit sequences. It also inserts synchronization words (comma words) at the start of the testing for word alignment. A FIFO is used to make sure that there is always data ready for transmission after a testing begins. The *Error/Slip Injection* block is used to inset errors or word slips for the purpose of testing the BERT. The *Word Alignment* block is used to make the received parallel data in phase with the parallel data in the input of the transmitter. The *8B10B Decoder* block recovers the 8-bit PRWSs sent by the BERT from the received 10-bit sequences, and then feeds the recovered 8-bit data back to the BERT for error detection.

The *Error/Slip Injection* block is built using an 8-bit parallel shift register controlled by a 2-bit control signal, *err_slip*, in a way similar to the control signal of the DUT described in Chapter 3. When *err_slip* = 00, the shift register delays the input by 3 clock cycles, emulating the normal operation; when *err_slip* = 01, the shift register delays the input by 2 clock cycles, so it emulates a bit loss; when *err_slip* = 10, the shift register delay the input by 4 clock cycles, so it emulates a word repeat; when *err_slip* = 11, the output of the shift register is set to be 1s, so it emulates an error burst.

The *8B10B Encoder* block encodes 8-bit data into 10-bit codes, and the *8B10B Decoder* performs the reverse. In this design, we use Altera 8B10B Encoder/Decoder MegaCore to

implement the encoding and decoding logic [55]. The encoding/decoding process is shown in Figure 5-9. In serial transmission, the least significant bit (LSB) is always transmitted and received first, while the most significant bit (MSB) is transmitted and received last.

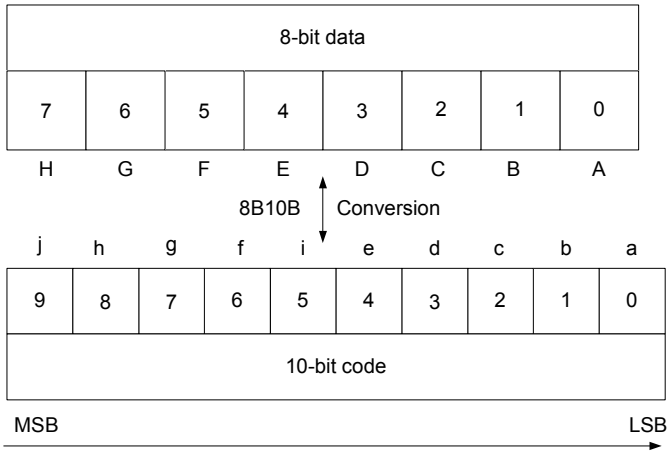


Figure 5-9: 8B10B Coding Process

As can be seen from Figure 5-9, the eight bits named *A, B, C, D, E, F, G*, and *H*, are split into two group: the five-bit group *A, B, C, D, E,* and the three-bit group *F, G, H*. *A* is the LSB and *H* is the MSB. The coded 10 bits named *a, b, c, d, e, i, f, g, h* and *j*, are also split into two groups: the six-bit group *a, b, c, d, e, i,* and the four-bit group *f, g, h, j*. The order of the 10 bits code is not alphabetical.

The 8B10B encoder/decoder core maintains a neutral average disparity. Disparity is the difference between the number of 1s and 0s in the encoded word. Neutral disparity indicates the number of 1s and 0s are equal, while positive disparity indicates more 1s than 0s and negative disparity indicates more 0s than 1s; therefore, average disparity determines the DC component of a serial line. Running disparity, which is done by the encoder, is a record of the cumulative disparity of every encoded word. To guarantee neutral average disparity, a positive running disparity must be followed by a neutral or negative disparity; a negative running disparity must be followed by a neutral or positive disparity. Details on running disparity rules can be found at [56].

In the coding scheme, in additional to the 256 data characters, the 8B10B codec defines twelve out-of-band indicators, which are also called special control (*K*) characters. Special *K* characters can be used for word alignment (packet delimiters), idle indicators, or other special purposes. In the testing setup, only the comma character (K28.5 in 10-bit special *K* code) is used for alignment purposes. The *Comma Word* block generates comma characters.

The FIFO can be built using the method as presented in Chapter 4. It can also be built using the FIFO mega function provided in Quartus II software.

The *Word Alignment* block realigns the received parallel data to generate parallel data in phase with the data in the input of the transmitter. The block diagram of the *Word Alignment Block* is shown in Figure 5-10.

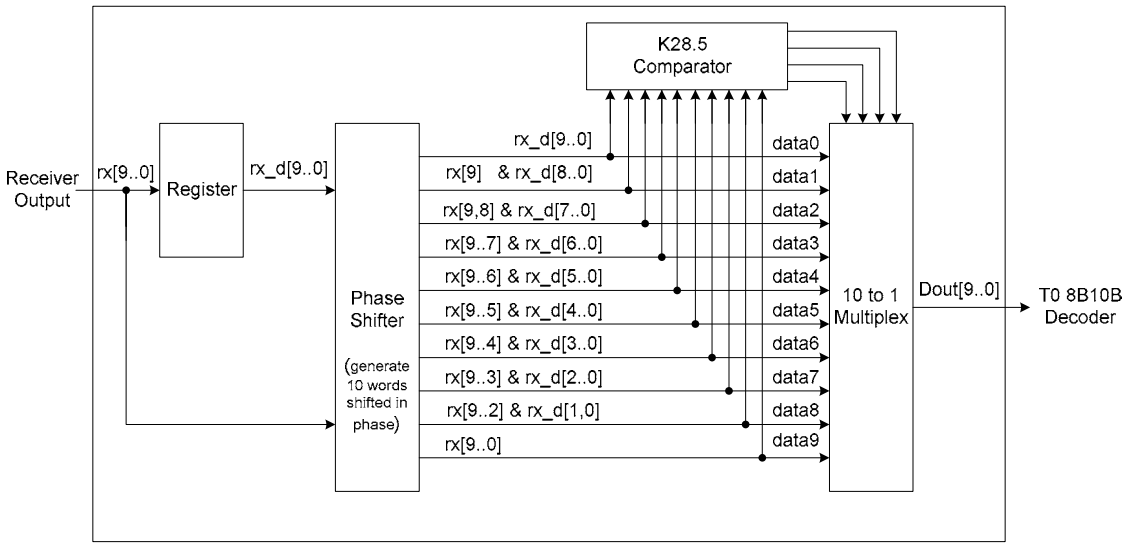


Figure 5-10: Block Diagram of Word Alignment

The deserialized 10-bit parallel data rx[9..0] from the output of the HSDI receiver is not certainly in phase with the input parallel data of the transmitter. The data *rx_d*[9..0] is generated by delaying *rx*[9..0] by one clock cycle. Based on *rx*[9..0] and *rx_d*[9..0], the phase shifter generates 10 parallel words. Each of the word is shifted by one serial clock cycle from the next one. For the 10 10-bit data words, there must be one and only one

word that is in phase with the transmitted data. This word is detected by comparing each of the ten words with the comma character K28.5 at the beginning of the transmission. The comparator generates control signals of the multiplex based on the comparison results. The multiplex locks onto the first received comma character and keeps the same alignment until a new transmission begins. The output of multiplex is sent to the 8B10B decoder to restore the 8-bit data sent by the BERT.

All the blocks in Figure 5-8 are implemented in VHDL, targeting the Mercury EP1M120F484C7AES device using Quartus II development tool. The synthesized results are downloaded onto an Altera Mercury HSDI CDR Demo board. The outputs of the transceiver are connected to the inputs of the receiver by two SMA cables, and LVDS format is used for the transmission. The delay of the transceiver and the glue logic is 42 clock cycles. Both simulation results and zero BER (when *err_slip* = 00) obtained from the real test running in hardware demonstrate the functional correctness of the HSDI transceiver and the feasibility of the testing setup.

5.2 Baseband Transmission Testing

In this section, we present the baseband transmission testing setup and its test results in terms of BER as a function of SNR. The test setup mainly consists of the BERT core presented in Chapter 3 and the AWGN core presented in Chapter 4. The test results are very close to the theoretical values. The proposed BER test scheme presented in Chapter 2.3 is verified through the experiment.

5.2.1 Baseband Signal Formats

In digital baseband transmission systems, there are various time domain signal formats. Figure 5-11 illustrates return zero (RZ), non return zero (NRZ) and non return zero inverted (NRZI) signaling for the binary information data sequence 10011011. The NRZ and NRZI formats are commonly used in digital baseband transmission.

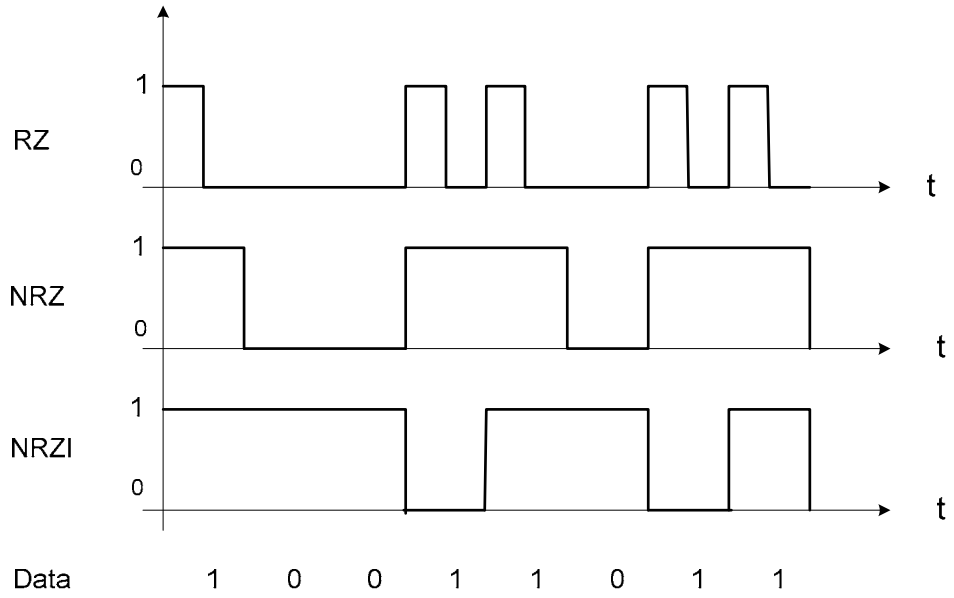


Figure 5-11: Baseband Signal Formats

In a RZ transmission system, the binary information digit 1 is encoded as a high signal represented by 1, but the high signal returns to zero state before reaching the end of the bit interval. For illustrative purpose, it is assumed that the return occurs at the midpoint of the interval in Figure 5-11. In a RZ transmission system, the binary information digit 0 is encoded as a low signal represented by 0.

In NRZ format, the binary information digit 1 is encoded as a high signal represented by 1, and the binary information digit 0 is encoded as a low signal represented by 0. NRZ is the simplest baseband signal format. The NRZ modulation is memoryless and is equivalent to a binary pulse amplitude modulation (PAM) or a binary PSK modulation in a carrier-modulated system [13]. The NRZ signaling format is more bandwidth efficient than RZ, as the pulses of NRZ signaling are wider than the RZ format.

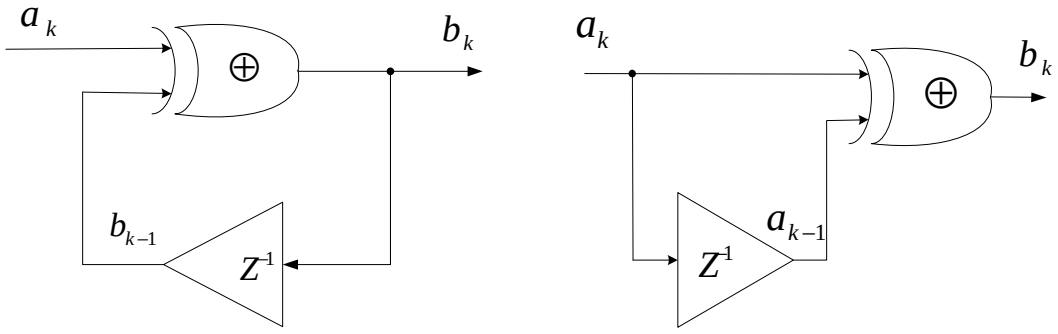
However, there are two particular problems associated with NRZ transmission. First, when the transmitted data is static, which means there is no change from one bit interval to the next, there is no transition in the transmitted waveform. This causes timing problems when establishing bit synchronization. The second problem occurs with data

inversion. If the levels of transmitted waveform are accidentally inverted during transmission, all the data is inverted, hence every bit is in error. Inversion can occur in several ways, such as a phase shift or losing track of the number of inversions.

To overcome these problems, NRZI format is introduced. NRZI signaling adopts differential techniques, in which the data is represented as changes in levels, rather than particular levels, of the signal. In NRZI format, the binary information digit 1 results in a signal transition, which can be either a low-to-high or a high-to-low; the binary information digital 0 results in no signal transition, which means the signal amplitude level remains unchanged. This type of signal encoding is called differential encoding. The coding operation is described mathematically by the relation

$$b_k = a_k \oplus b_{k-1}$$

where a_k is the binary information sequence into the NRZI encoder, b_k is the output of the encoder, and $\oplus$ denotes the exclusive-OR operation (addition modulo 2). Based on the mathematical model, it is easily to get the structure of the NRZI encoder, as shown in Figure 5-12 (a), where Z^{-1} denotes one-cycle delay.



(a) Encoder (b) Decoder

Figure 5-12: The Structure of a NRZI Encoder and Decoder

The NRZI decoder can be implemented as shown in Figure 5-12 (b). It compares the NRZI encoded signal to a delayed version of itself. If the two signals are the same in an interval, we know that 0 is being sent; if the two are different, a 1 is being sent. An exclusive-OR gate performs this decision process.

5.2.2 SNR Setting

Recall that from Chapter 2.1.2.1, when the correlation $\rho = 0$ in baseband transmission, we have $P_e = Q(\sqrt{\frac{E}{N_o}})$, where P_e is *BER*, and $\frac{E}{N_0}$ is *SNR*. In other words, the theoretical relationship between *BER* and *SNR* is characterized by

$$BER = Q(\sqrt{SNR})$$

As can be seen from the above equation, *BER* is only determined by *SNR* . In NRZ transmission systems, one (high level signal) is used to transmit data '1' and zero (low level signal) is used to transmit data '0'. We assume that data 1s and 0s have equal occurring probability, the average energy of two signals is

$$E = \frac{E_0 + E_1}{2} = 0.5$$

In an AWGN communication channel, the noise is Gaussian and characterized by a mean of zero and a variance of δ^2. The energy of the noise can be represented by

$$N_o = 2\delta^2$$

Though the above equations are derived from NRI signaling, they are also applicable to NRZI signaling. This is verified at the end of this case study. Combining the equations for E and N_o, we can get the equation for the *SNR* of baseband transmission. The *SNR* is determined by the variance of the noise and expressed as

$$SNR = \frac{1}{4\delta^2} \qquad (5.2\text{-}1)$$

In Chapter 4, an AWGN generator with zero mean ($m_x = 0$) and unity variance ($\delta^2 = 1$) has been developed. The variance of the generator can be changed to any value by adding a divider at the output of the generator. Suppose the original output is denoted by x, and it is divided by a, then the new variance of the generator becomes

$$\delta^2 = E[(x' - m_x)^2]$$

$$= E[(x/a)^2]$$

$$= 1/a^2 \qquad (5.2\text{-}2)$$

Combining equations 5.2-1 and 5.2-2, we have

$$SNR = \frac{a^2}{4}$$

where a is the scaling factor of the AWGN generator with zero mean and unity variance. By changing a, we can get different SNR conditions for the AWGN communication channel.

5.2.3 Testing Setup and Results

Based on the AWGN communication channel model discussed in Chapter 2.2.1, the testing setup for digital baseband is developed and shown in Figure 5-13.

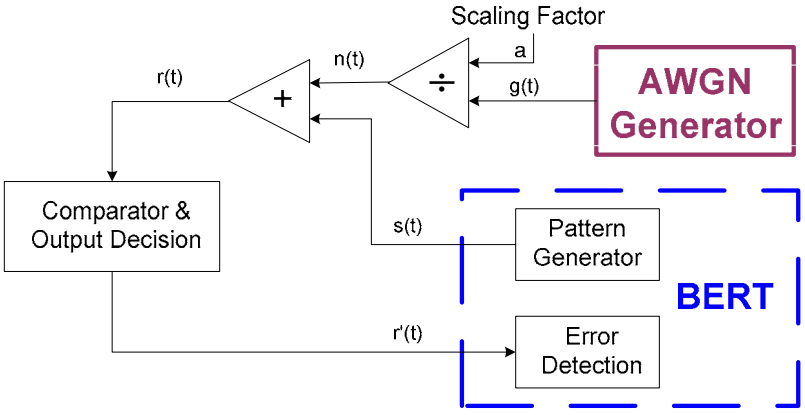


Figure 5-13: BER Testing Setup for NRZ Digital Baseband

In the testing setup shown in Figure 5-13, the AWGN generator block and the BERT block are the IP cores introduced in Chapter 3 and Chapter 4, respectively. This testing setup also constitutes a digital communication system. In this system, the transmitter consists of the pattern generator; the communication channel consists of the AWGN

generator, the divider and the adder; the receiver consists of the comparator and output decision block. The error detection block is used for performance evaluation.

The output from the pattern generator is PRBSs. The output sequence, denoted by $s(t)$, is the data signal to be transmitted. The data signal $s(t)$ is corrupted by the noise signal $n(t)$ in the AWGN communication channel. The noise signal $n(t)$ is derived from $g(t)$ by dividing $g(t)$ by the scaling factor a, where a is six bits in width, five for integer and one for fraction. By setting the value a, the SNR condition of the communication system can be set. The noise signal $g(t)$ is the output of the AWGN generator with zero mean and unity variance. As discussed in Chapter 4, the noise signal $g(t)$ is 19 bits in width, 1 bit for the sign, 6 bits for the integer and 12 bits for the fraction.

The noise corrupted data signal $r(t)$ is compared with a threshold to determine the output of the receiver. In NRZ transmission system, 0s and 1s are transmitted and they have equal occurring probability; therefore, the threshold is set to be 0.5. If $r(t)$ is bigger than 0.5, $r`(t)$ is set to be 1; otherwise, $r`(t)$ is 0. Finally, the received bit sequence $r`(t)$ is compared with a delayed transmitted sequence $s(t)$ bit by bit, and errors are counted. The measured BER is the ratio of the counted errors and the number of transmitted bits.

Table 5-2 lists the BER test results. The measurements are taken while running the AWGN core and the BERT core in an Altera Mercury FPGA board at the speed of 25 MHz.

Table 5-2: BER Measurements for Digital Baseband

Scaling Factor a	2	3	4	5	6	7	8
Variance δ^2	1/4	1/9	1/16	1/25	1/36	1/49	1/64
SNR	1	2.25	4	6.25	9	12.25	16
SNR (dB)	0	3.52	6.02	7.96	9.54	10.88	12.04
Error Count	2024	819	289	113	1195	176	162
Transmitted Bits	12448	12448	12448	20000	1000000	1000000	10000000
Measured BER	1.62e-1	6.58e-2	2.32e-2	5.65e-3	1.20e-3	1.76e-4	1.62e-5

In Table 5-2, the following equations are used. These equations have been discussed previously.

$$\delta^2 = \frac{1}{a^2}$$

$$SNR = \frac{a^2}{4}$$

$$BER_{measured} = \frac{ErrorCout}{TransmittedBits}$$

In the above digital baseband testing, the signal format is NRZ. For a NRZI baseband communication system, the testing setup shown in Figure 5-14 is used to test its BER performance.

In Figure 5-14, the structure of the NRZI encoder and the NRZI decoder is the same as these shown in Figure 5-12. The other blocks have already been introduced in the NRZ digital baseband testing setup. The NRZI testing results are the same as the test results shown in Table 5-2.

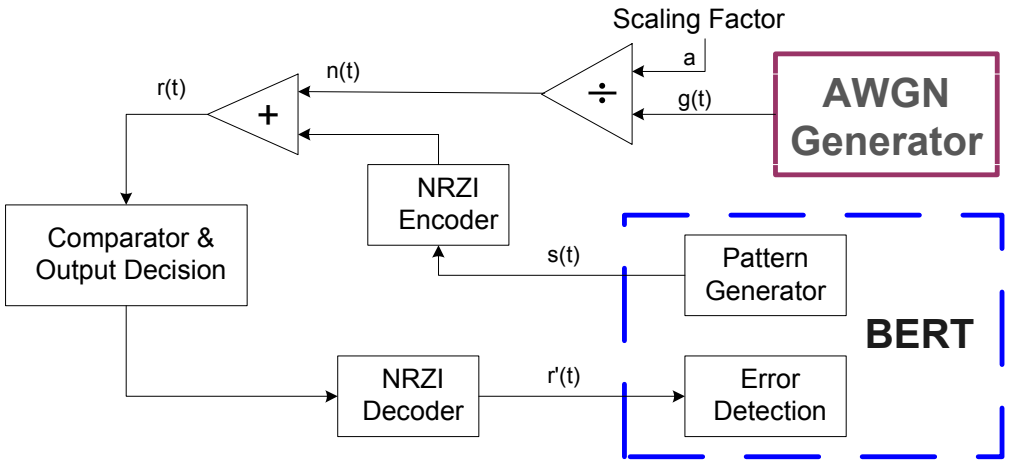


Figure 5-14: BER Testing Setup for NRZI Digital Baseband

Figure 5-15 shows the plot of the measured BER and theoretical BER of digital baseband as a function of input SNR. Recall from Chapter 2.1.2.1, the theoretical BER of digital baseband is given by

$$BER_{theoretical} = Q(\sqrt{SNR}) = \frac{1}{2} erfc(\sqrt{\frac{SNR}{2}})$$

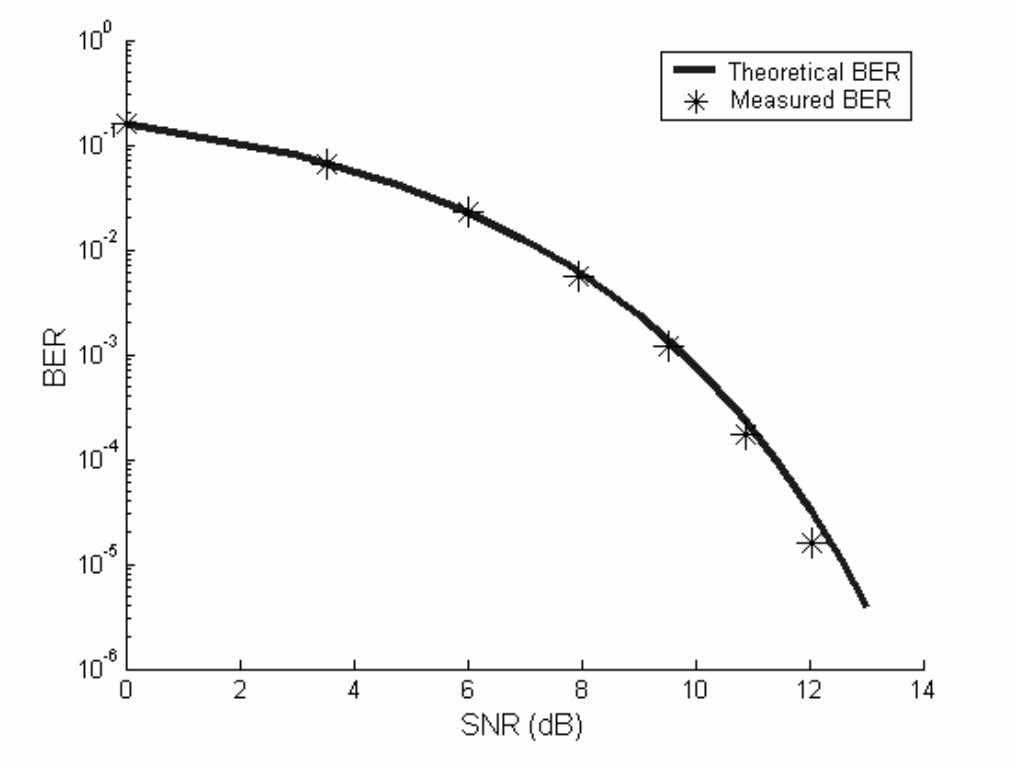


Figure 5-15: Plot of Measured BER and Theoretical BER for Digital Baseband

As can be seen from Figure 5-15, the measured BER closely matches the theoretical BER. This match can be further improved by optimizing the threshold voltage setting or increasing the number of samples. The plot demonstrates that the proposed AWGN core is suitable for communication channel emulation and that the BER testing results using the BERT core are reliable. In the above testing, it takes less than one second to generate the point at 1.62e-5 BER; in software simulations, this usually takes days. The proposed BER test scheme exhibits astronomical advantage in speed over traditional software simulation methods.

Although the experiment is based on testing a digital baseband system, the proposed BER testing scheme applies to any digital transmission system using AWGN communication channels. Only glue logic might need to be changed for different applications. For other communication channels, the proposed AWGN core can be modified to emulate the channels.

Chapter 6 - Conclusions

6.1 Conclusions

In this thesis, an FPGA-based BER testing scheme is presented. The scheme can measure the BER performance of a wide range of digital communication systems. Compared with traditional software simulations, the proposed BER testing scheme is a few orders of magnitude faster. Compared with traditional standalone BERT and ATE equipment, the proposed solution is much cheaper. It is also easy to set up for BER testing under different noise conditions as a novel implementation of an AWGN communication channel emulator is included in this scheme. In addition, FPGA-based solution makes it easy to interface different DUTs.

The proposed BER testing scheme mainly consists of two independent IP cores: a BERT core and an AWGN core. The two cores can also be used separately for different applications. In this thesis, two challenging testing cases were successfully conducted using the proposed IP cores. We demonstrate through case studies that the proposed BER testing scheme exhibits excellent performance in speed and cost.

6.2 Future Work

One can note that the DUT of the BER testing system must be the combination of an encoder and a decoder, or the combination of a modulator and a demodulator. If we want to test the performance of a decoder, a reference module (e.g. an encoder) needs to be added in the testing setup. It would be more convenient to test the BER performance of a receiver using the IP cores if a parameterized reference encoder and/or modulator can be included in the testing scheme.

In addition, as BER and jitter are closely related, jitter testing and jitter separation schemes can be devised based on BER testing schemes [57], [58], [59]. Jitter testing is a

very challenging issue and its importance has been widely recognized as the transmission speed is becoming higher and higher. The proposed BERT core and the AWGN core can be used as a good start point for jitter testing research. With the continuing enhancement of FPGA performance, it is possible to build a jitter testing scheme in a single FPGA, especially in a SoC or DSP oriented FPGA device.

References

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

[2] Lattice Semiconductor Corporation. *SERDES Handbook,* September 2002

[3] Xilinx, Inc. *Virtex-II Pro Data Sheet*. San Jose, California, 2003

[4] ITRS. *The International Technology Roadmap for Semiconductors,* 2001 Edition

[5] E. A Newcombe and S. Pasupathy. "Error Rate Monitoring for Digital Communications," *Proc. IEEE*, vol. 70, no. 8, pp.805-825, Aug.1982

[6] M. Courtoy, "Rapid System Prototyping for Real-time Design Validation," *Proc. Ninth International Workshop on Rapid System Protyping*, pp. 108-112, 1998.

[7] Agilent Technologies. *81250 Product Overview Datasheet.* Palo Alto, California, 2002

[8] Anritsu Corporation. *48 Gb/s BER Test System Data Sheet.* Atsugi-shi, Kanagawa, Japan, 2002

[9] Advantest Corporation. *D3371 3.6 GHz Transmission Aanalyzer*. Tokyo, Japan

[10] LeCroy Corporation. *Serial Data Analyzers SDA Series Data Sheet*. Chestnut Ridge, New York, 2003

[11] Y. Fan and Z. Zilic, "Efficient FPGA Implementation of High Speed AWGN Communication Channel Emulators," The First Annual Northeast Workshop on Circuits and Systems, Montreal, Canada, June 2003

[12] Y. Fan and Z. Zilic, "Testing for Bit Error Rate in FPGA Communication Interfaces," Poster Presentation at 11th *ACM International Symposium on FPGAs*, Monterey, California, Feb. 2003.

[13] J. G Proakis. *Digital Communications*. McGraw-Hill High Education, Electrical and Computer Engineering Series, 2001

[14] M. S. Toden. *Analog and Digital Communication Systems*. Discovery Press, Los Angeles, California, 2001

[15] The MathWorks, Inc. Natick, Massachusetts
http://www.mathworks.com

[16] The Mathworks, Inc. *Using the Communication Blockset,* Natick, Massachusetts. July 2002

[17] A. Gazel, E. Boutillon, J.L. Danger, G. Gulak and H. Lamaari, "Design and Performance Analysis of a High Speed AWGN Communication Channel Emulator," IEEE PACRIM Conference, Victoria, B.C., Canada, Aug. 2001

[18] Altera Corporation, San Jose, California

[19] Xilinx, Inc. San Jose, California

[20] C. Change, K. Kuusilinna, B. Richards and R.W. Brodersen, "Implementation of BEE: a Real-time Large-scale Hardware Emulation Engine," *Proceedings of the International Symposium on Field programmable Gate Arrays*, February 2003

[21] Wai-Kei Mak, D. F. Wong, "Board-level Multiterminal Net Routing for FPGA-based Logic Emulation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, April 1997

[22] T. Matsumura, N. Yamanaka, T. Yamaguchi, K. Ishikawa, "Real-time emulation Method for ATM Swithching Systems in Broadband ISDN," *Proceedings of seventh IEEE International Workshop*, Jun 1996

[23] M. Courtoy, "Rapid Prototyping for Communications Design Validation," Southcon Conference Record, Jun 1996

[24] S. Berber, "Techniques for Bit Error Rate Measurement Using Chebyshev Inequlity," *Electronics Letters*, 6th July 1989, Vol, 25 No.14

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

[26] K. S. Shanmugan and A.M. Breipohl. *Random Signals: Detection, Estimation, and Data Analysis.* New York, John Wiley and Sons, 1988

[27] Maxim Integrated Products, Inc. *HFTA-05.0: Statistical Confidence Levels for Estimating BER Probability,* Application Notes, Sunnyvale, California, 2002

[28] P. Atiniramit. *Design and implementation of an FPGA-based adaptive filter single-use receiver.* Master Thesis, Department of Electrical Engineering, Virginia Polytechnic Institute and State University, 1999

[29] MSS (Mobile Satellite Services) Corporation, *Bit Error Rate Generator and Additive White Gaussian Noise Generator Specification*, Gaithersburg, Maryland

[30] Xilinx, Inc. *LogiCore Additive White Gaussian Noise Core Data Sheet*, San Jose, California, October 2002

[31] VR. C. Tausworthe, "Random Numbers Generated by Linear Recurrence Modulo Two," *Mathematical Computing*, vol. 19, pp201-209, 1965

[32] S. W. Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, 1967

[33] S. Wolfram, "Random Sequence Generation by Cellular Automata," *Advances in Applied Mathematics*, vol. 7, pp. 123-169, June 1986

[34] J. Chen, J. Moon and K. Bazargan, "A Reconfigurable FPGA-based Readback Signal Generator for Hard-drive Read Channel Simulator," *Proceedings of 39th Design Automation Conference*, Pages: 349-354, June 2002

[35] S. Wolfram, "Statistical Mechanics of Cellular Automata," *Reviews of Modern Physics*, vol. 55, pp.601-644, July 1983

[36] S. Zhang, D. M. Miller, J. C. Muzio, "Determination of Minimal Cost One-dimensional Linear Hybrid Cellular Automata," *Electronics Letters*, vol. 27, no.18, pp.1625-1627, Aug. 1991

[37] K. Cattell, S. Zhang, "Minimal Cost One-dimensional Linear Hybrid Cellular Automata of Degree through 500," *Journal of Electronic Testing: Theory & Applications*, vol6, no.2, pp255-258, April 1995

[38] M. Sipper and M. Tomassini, "Generating Parallel Random Number Generators by Cellular Programming," *International Journal of Modern Physics C*, vol. 7, no. 2, pp.181-190, 1996

[39] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud, "Generating High-quality Random Numbers in Parallel by Cellular Automata," *Future Generation Computer Systems*, vol. 16, pp. 291-305, 1999

[40] M. Tomassini, M. Sipper, and M. Perrenoud, "On the Generation of High-quality Random Numbers by Two-dimensional Cellular Automata," *IEEE Transactions on Computers*, vol.49, pp.1146-1151, October 2000

[41] B. Shackleford, M. Tanaka, R. J. Carter and G. Snider, "FPGA Implementation of Neighbourhood-of-four Cellular Automata Random Number Generators," *Proceedings of the Tenth ACM International Symposium on Field-Programmable Gate Arrays*, pp.106-112, Feb. 2002

[42] D. E. Knuth. *The Art of Computer Programming.* Addison-Wesley, 1998

[43] R. Kiefer. *Test Solutions for Digital Networks*. Huthig GmbH, Heidelberg. 1998

[44] F. James, “A Review of Pseudo-random Number Generators,” *Computer Physics Communications* 60, 1990

[45] P. L`Ecuyer, “Random Numbers for Simulation,” *Communications of the ACM*, 33:10, 1990

[46] G.A. Marsaglia, “A Current View of Random Number Generators,” *Computational Science and Statistics: The Interface*, Balliard, Elsevier, Amsterdam, 1985

[47] Quantum World Corporation, *QNG Model J20KP True Random Number Generator Users Manual*, 1998

[48] P. H. Bardell, W.H. McAnney and J. Savir, *Build-in Test for VLSI: Pseudo-random Techniques*, John Wiley and Sons, 1987

[49] P.Alfke, “Efficeent Shift Registers, LFSR Counters, and Long Psudo-Random Sequence Generators,” *Xilinx Application Note*, 1995

[50] P. Chu and R. Jones, “Design Techniques of FPGA-Based Random Number Generator,” Military and Aerospace Applications of Programmable Devices and Technologies Conferences, 1999

[51] P. L`Ecuyer, “Efficient and Portable Combined Random Number Generators,” Comm. ACM 31:6, 1988

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

[53] Altera Corporation. *Introduction to Quartus II Manual*. San Jose, California, July 2003

[54] Altera Corporation. *Mercury Gigabit Transceiver MegaCore Function User Guide*. San Jose, California, February 2002

[55] Altera. *8B10B Encoder/Decoder MegaCore Function User Guide*. San Jose, California, December 2002.

[56] IEEE 802.3z Specification, Paragraph 36.2.4.4

[57] M. P. Li and J. B. Wilstrup, “On the Accuracy of Jitter Separation from Bit Error Rate Function”, *Proceedings of IEEE International Test Conference*, pp. 710-716, Oct. 2002.

[58] T. J. Yamaguchi, M. Soma, M. Ishida, H. Musha and L. Malarsie, “A New Method for Testing Jitter Tolerance of SerDes Devices Using Sinusoidal Jitter”, *Proceedings of IEEE International Test Conference*, pp. 717-725, Oct. 2002.

[59] Y. Cai, S. A. Werner, G. J. Zhang, M. J. Olsen and R. D. Brink, “Jitter Testing for Multi-Gigabit Backplane SerDes”, *Proceedings of IEEE International Test Conference*, pp. 700-710, Oct. 2002.

[60] S. Berber, “Bit Error Rate Measurement with Predetermined Confidence,” *Electronics Letters*, 20th June 1991, Vol. 27 No.13

[61] Xilinx, Inc., “SAPP661: RocketIO Transceiver Bit Error Rate Tester, V2.0”, San Jose, California, June 2003

[62] Nallatech Ltd., “Complete Hardware-based Solution for Bit Error Rate Testing,” Glasgow, Scotland, September 2002