A Reconfigurable FPGA-Based Readback Signal Generator For Hard-Drive Read Channel Simulator

Jinghuan Chen, Jaekyun Moon, and Kia Bazargan
Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN 55455, USA.
{jchen, moon, kia}@ece.umn.edu

ABSTRACT

A hard disk readback signal generator designed to provide noise-corrupted signals to a channel simulator has been implemented on a Xilinx Virtex^{TM}E FPGA device. The generator simulates pulses sensed by read heads in hard drives. All major distortion and noise processes, such as intersymbol interference, transition noise, electronics noise, head and media nonlinearity, intertrack interference, and write timing error, can be generated according to the statistics and parameters defined by the user. Reconfigurable implementation enables an update of the signal characteristics in runtime. The user also has the flexibility to choose from a set of bitstreams to simulate particular combinations of noise and distortion. Such customized restructuring helps reduce the area consumption and hence virtually increase the capacity of the FPGA device. The time to generate the readback signals has been reduced by four orders compared to its software counterpart.

1. INTRODUCTION

In the hostile hard drive environment, advanced signal processing techniques are necessary in order to recover the user data from readback signals that are picked up by read sensors (read heads). Exploration of novel techniques are normally carried out through very intensive simulation. Conventionally, the simulation task is performed in software. The major drawback of the software approach is that it usually takes a very long time when low bit error rate (BER) is investigated. For example, a simulation of BER=10^{-8} with 10 errors that runs on a personal computer equipped with a 1GHz Pentium 4 processor can take days. In contrast, acceptable BERs in commercial hard drives normally go below 10 out of 10^{13} for desktop products and 10 out of 10^{15} for server-class products. Since it is very important to understand the behavior of many signal processing and coding techniques such as turbo codes and low density parity check (LDPC) codes at very low BERs, high-speed hardware-based or mixed hardware-software simulators are of great practical significance.

In view of the fact that noise sources and characteristics in hard drives are quite unique and different from those in other communication systems, a dedicated readback signal generator must be implemented in front of other signal processing modules in such a simulator. This signal generator should be accurate and, at the same time, easy to reconfigure. It shall incorporate all major noise processes and distortions and should be capable of generating very low probability events in accordance with the user-defined statistics. It should also provide the user with a flexibility of choosing from various combinations of noise and distortion characteristics. In this paper, we propose to implement the readback signal generator on a field programmable gate array (FPGA) utilizing its reconfigurability. This FPGA-based signal generator can also be used as part of a testbench for the data-recovery circuit (usually called the "read channel") since it is totally tunable compared to typical test spindle setups. The proposed FPGA-based signal emulator can provide a complete set of test conditions.

1.1 Recording Process and Noise/Distortion Categorization

In hard drives, the storage medium consists of one or more metal or glass disks coated with magnetic thin films. Binary information is stored in concentric circular tracks on the surface of disks. Each binary bit takes a very small portion of a track called a *bit cell*. During the write process, a write head magnetizes each bit cell to one of two tangential directions. Whether a 0 or a 1 is recorded is determined by the direction of magnetization. A read head picks up the magnetization flux emitted from the boundaries of bit cells and generates readback pulses (Fig. 1). These pulses will be passed through a pre-amplifier, analog/digital filters, and data-recovery modules and converted back to binary data. The read channel includes the filters and the data-recovery modules.

The imperfect nature of the magnetic thin film results in a zigzag boundary between bit cells instead of an ideal straight edge. This will cause the pulse position to shift in time domain (*transition jitter*) and pulse-shape to deform (*width variation*). The noise introduced by this sort of phenomenon is called *transition noise (TN)* and it is data dependent in the sense that noise arises only in the presence of transitions that make up a data pattern. Almost all of today's read heads are *magneto-resistance (MR)* or

This work was supported by NSF under Grant CCR-9805195.

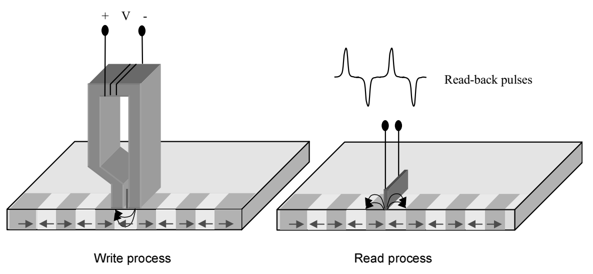

Figure 1: Write and read process in hard drives

giant magneto-resistance (GMR) heads which are not linear sensors. They introduce the *head nonlinearity (HNL)* that causes the positive and negative pulses to have different amplitudes. Two more types of media nonlinearity also occur during the write process: *nonlinear transition shift (NLTS)* and *partial erasure (PE)*. They generally create additional transition position shift and amplitude decline. Slow variant *write timing error (WERR)* stems from the irregularity of the disk spinning speed. A read head does not always position with 100% accuracy on the top of a track. It also senses the signal from the neighboring track and is influenced by *cross-talk* which is sometimes called *intertrack interference (ITI)*. Finally, *electronics noise (EN)* further corrupts the signal. Transition noise in hard-drive recording systems has been modeled in several ways. Moon[1] uses Tayler series to approximate transition jitter and width variation. Caroselli and Wolf [2] simplify the structure of recorded tracks by using a micro-track model. In our implementation, we mimic the physical reality by actually shifting the pulse position and changing the pulse-shape in runtime. The behavior of the transition noise and other noise processes as well as distortions follows the statistics laid down by the user. It should be noted that the statistics do not have to be Gaussian, as has been assumed in many previous modeling works.

1.2 Reconfigurable Signal Generation

Instead of running under static conditions, the signal generator may be required to change the noise properties during runtime. Furthermore, different kinds of noise and distortion characteristics mentioned above are not necessarily present in a system simultaneously. These features demand some degree of reconfiguration capability from the generator, which makes time-multiplexing FPGA the implementation method of choice.

In the next section, we will briefly describe how to model different types of noise processes and distortions observed in hard drives. Details of the modeling work is out of the scope of this paper. Further discussions will be given in another paper. We will deal with the design and implementation issues in Section 3. Then we will focus on pseudo-random number generators and structure and reconfigurability of the readback signal generator. Conclusions will be given in the last section.

2. MODELING OF READBACK SIGNALS

2.1 Distortion and Noise

A sequence of readback pulses are said to experience *inter-symbol interference (ISI)* when the length of each pulse extends over a number of symbols. The pulses overlap and are distorted. Expression of noiseless readback signal is given in Eq. (1). The response $h(t)$ is the isolated pulse and b_k is the differentiation of data bit $a_k \in \{-1, 1\}$ (Eq. 2). The sequence b_k takes value of $\{-2, 0, 2\}$ and a non-zero b_k indicates the existence of a transition. ISI happens when the length of $h(t)$, L, is longer than the symbol period T.

$$y(t) = \sum_k b_k h(t - kT) \tag{1}$$

$$b_k = a_k - a_{k-1} \tag{2}$$

The *position jitter* and *width variation* are categorized as noise sources because they introduce noise when the readback signal is sampled. Both types of noise can be modeled as random variables. We rewrite Eq. (1) to reflect the transition noise. Now the isolated pulse $H(t)$ is not deterministic but a random process. The position jitter parameter δ_j is a random variable falling in the range $-T < \delta_j < T$.

$$y(t) = \sum_k b_k H(t - kT + \delta_j) \tag{3}$$

Eq. (1) to (3) are well known and straightforward. Head nonlinearity can be characterized by a nonlinear function $F(x)$, which can be measured from a read head. NLTS and PE can be modeled as a data dependent nonlinear distortion of the pulses. Simplistic assumptions have been made on the data dependence [4] and [5]. In this paper, only the $1st$ order effect is considered. We express HNL, NLTS, and PE as follows:

$$y(t) = \sum_k F\{r_k b_k h_k(t - kT + \delta_j - \frac{\varepsilon T}{4} b_k b_{k-1})\} \tag{4}$$

where r_k represents the amplitude loss due to PE and ε is the amount of NLTS in units of T. Note that NLTS component reduces to zero effectively unless two consecutive transition occurs, i.e., $b_k b_{k-1} \neq 0$.

Slowly varying WERR can be simplified and modeled as a slow-paced *"random walk"*. Intertrack interference is implemented by summing up weighted output signals from two uncorrelated generators that simulate two interfering tracks. *Electronics noise* normally is modeled as an additive white bandlimited gaussian noise (AWGN).

2.2 Signal-to-Noise Ratio (SNR)

The behavior of statistically defined noise should be determined by its probability distribution. However, when the noise pdf is assumed to be Gaussian, the first and second order statistics can provide all necessary information about the noise process. In this case, the user of the signal generator only needs to specify the SNR, which effectively fixes the variance of the noise. In addition, the user shall specify the norminal isolated pulse $h(t)$, the percentage of the transition noise power, and the composition of the position jitter and the width variation. In the remainder of this paper, we will focus on Gaussian noise. We note, however, that our simulator can handle any user-specified pdf for the noise process.

3. IMPLEMENTATION

3.1 Gaussian Pseudo-Random Number Generator (PRNG)

Transition noise, electronics noise, and write timing error are all randomly generated in a non-uniform fashion. These in-system pseudo-random number generators must be uncorrelated and bear good randomness. In our design, uniform pseudo-random numbers are first produced and then transformed to non-uniform random numbers. One effective and popular uniform PRNG is the linear feedback shift register (LFSR) [6]. Its output is a maximum-length sequence (*m-sequence*) with a period of $2^n - 1$, where n is the number of storage units (registers). However, if we need many sequences of random noise, the area consumed by LFSR will be large, something that we cannot afford in an FPGA implementation. For example, with five 6-bit wide PRNGs that can generate random numbers with a period longer than 1000 ($n = 10$), the number of registers will be 300. Facing area constraints in this design, we adopted an alternative method, linear hybrid cellular automaton (LHCA) (for example, see [7] and [8]). The principle of a cellular automaton is that the next value of each register is calculated by a Boolean function from the current values of immediate neighbors and itself. A cellular automaton is said to have a *null boundary* condition when the left- and right-most registers in the array connect to zeros. A null boundary is preferred to a feedback boundary because it avoids long feedback paths. It has been shown that when the Boolean functions are carefully selected and after the LHCA evolves for a number of initialization clock cycles, such a register array will output an m-sequence that has the same period as that of an LFSR [7]. The Boolean functions are called *computation rules* and categorized by Wolfram [11]. One of the setups that can generate m-sequences is a careful mix (hybrid) of two Boolean functions, *Rule 90* and *Rule 150*, in the calculation of register contents [7]. Rule 90 and Rule 150 are defined as follows:

$$\text{Rule 90: } a_i(t+1) = a_{i-1}(t) \oplus a_{i+1}(t) \tag{5}$$

and

$$\text{Rule 150: } a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t) \tag{6}$$

where $a_i(t)$ is the content of register i at time t. How to determine the positions of Rule 90 and Rule 150 in a register array can be found in [9] and [10].

After a quick observation, we can find that not every output from registers in the array is a "good" bit because those from adjacent registers are correlated. In order to eliminate correlation, only one bit of every $\eta + 1$ bits will be used to form a random number where η is called the *site spacing parameter.* To compare with LFSR, if $\eta = 1$, the number of registers in a LHCA PRNG that can generate the same group of random numbers as mentioned above will be 60, a factor of 5 reduction in the number of registers.

Transformation from a uniform to a non-uniform pseudo-random number is illustrated in Fig. 2. An n-bit uniform random number is compared with numbers in a user-defined *cumulative distribution function (CDF)* conversion table and then encoded to an l-bit non-uniform random number. This exercise is equivalent to randomly picking a point in the area under the corresponding pdf, grouping all points in a column, and substituting them with the one

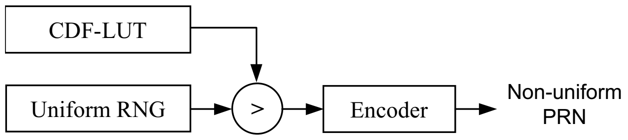

Figure 2: Generation of non-uniform pseudo-random number

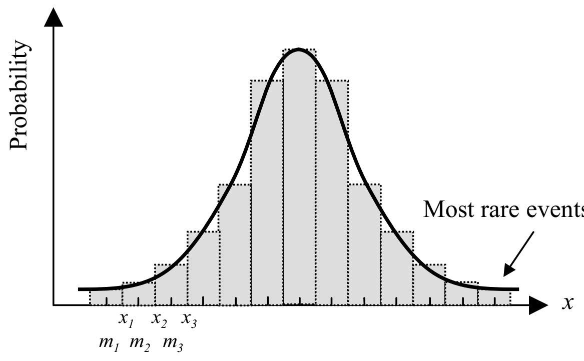

Figure 3: Splitting PDF into 2^m columns

number m_i (Fig. 3). Attention shall be paid to the choice of n and l since they determine the accuracy and precision of the PRNG. The width n of a uniform random number limits the smallest probability we can simulate, which is $\frac{1}{2^n-1}$. Encoding from an n-bit to an l-bit random number can be considered as quantizing a continuous random variable X to an l-bit discrete random variable X_q, if $n \gg l$. The width l controls the mean square loss associated with the quantization process.

Max [13] shows that the optimal quantizer is nonuniform when the pdf of the random variable is not uniform. However, the digital nature of the FPGA device decides the quantization levels $m_j (j = 1, 2, \ldots, q)$ to be uniform where q is the number of quantization levels, which is 2^l. We need to determine the boundaries $x_j (j = 1, 2, \ldots, q-1)$ for the suboptimal quantizer in term of minimizing the quantization loss. The quantization loss N_q is given in Eq. (7) and $f_x(x)$ is the pdf of X.

$$N_q = E[(X - X_q)^2] = \sum_{i=1}^{q} \int_{x_{i-1}}^{x_i} (x - m_i)^2 f_x(x) dx \tag{7}$$

In order to minimize the quantization loss, let

$$\frac{\partial N_q}{\partial x_i} = 0, \qquad i = 1, 2, \ldots, q-1. \tag{8}$$

One can obtain:

$$x_j = \frac{1}{2}(m_j + m_{j+1}). \tag{9}$$

Eq. (9) indicates that the suboptimal quantizer is a uniform quantizer. If noise is assumed to have Gaussian distribution, N_q can be computed analytically. Since quantization noise can be seen as an additive noise to the random

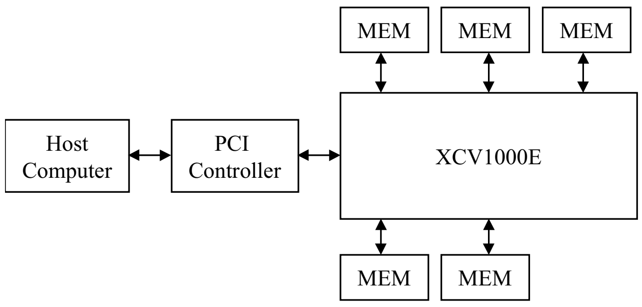

Figure 4: Simplified block diagram of an Annapolis Micro Systems FIREBIRD^{TM} PCI board

variable, we can define a signal to quantization noise ratio (SQNR) to evaluate the loss due to quantization as follows:

$$SQNR = \frac{\sigma_{RV}^2}{N_q} \tag{10}$$

where σ_{RV}^2 is the variance of the continuous random variable. For example, when $q = 64$ and σ_{RV}^2=9, $SQRT$ is 20.4 dB. Hence, as long as we use sufficiently many quantization levels, the quantization loss can be suppressed down to a negligible level.

3.2 Implementation Platform

The design is synthesized by Synplicity's Synplify^{TM} 6.2, simulated by ModelSim^{TM} 5.5, and implemented by Xilinx Foundation^{TM} 3.1 . The design has been implemented on a Xilinx Virtex^{TM}E XCV-1000E device which is mounted on an Annapolis Micro Systems FIREBIRD^{TM} PCI board. A simplified block diagram of a FIREBIRD^{TM} PCI board is shown in Fig. 4 [12]. The XCV1000E communicates with a host computer through PCI bus. The user's design, as well as some register space and a few interface blocks are implemented on the FPGA device (Fig. 5). By calling a group of application programming interface (API) functions, the user is able to command the board and establish communications between the host computer and the resister space as well as interface blocks. The user has access to 18 Mega bytes of RAM that is distributed in five modules on the board. Before the signal generator can produce readback signals, it has to be initialized with data pre-calculated by software. The initialization data is loaded into the on-board RAM before it is used to initialize the XCV1000E. The output signals are also stored into the RAM first and then transferred to the host computer. When the signal generator is integrated into the read channel simulator, however, the generated signal will be forwarded to the data-recovery module directly. Only recoverd binary data and BER will be sent back to the user. Clocks used by the XCV1000E and the RAM are synchronized and can run up to 150MHz.

The input interface of the generator in Fig. 5 bridges the host and the on-board RAM to the generator circuits and controls the timing of the initialization process. It reads the configuration values from the memory and from the host and initializes generator look-up tables. There are three types of tables need to be configured or reconfigured: pulse look-up tables, Gaussian conversion tables in PRNGs, and multiplier look-up tables.

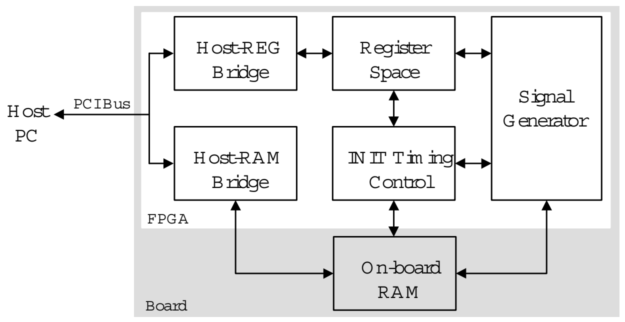

Figure 5: Interfaces between host, RAM, and the signal generator

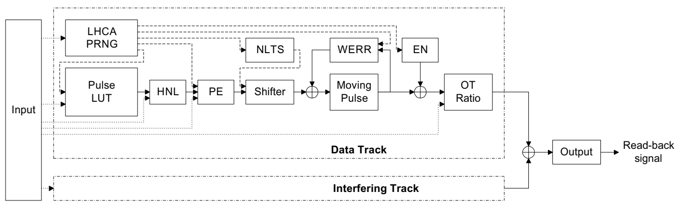

Figure 6: Block diagram of the readback signal generator

3.3 Readback Signal Generator

The architecture of the readback signal generator is modularized and pipelined. Its structure is illustrated in Fig. 6. The data-track and interfering-track modules are almost identical except the random numbers from LHCA-PRNG are independent and the off-track (OT) ratio r is different. The output of data-track is multiplied by the ratio r while the interfering track is multiplied by a different ratio r', ($r' \leq 1 - r$). ITI to the data track is embodied through adding two weighted signals. The peak of the summed signal will be degraded by a ratio of $r + r' \leq 1$. The value of r and r' are determined by the off-track position of the read head [14]. In the diagram, the solid arrows mark the signal paths, the dash ones indicate the random number paths, and the dotted lines are the configuration data paths.

The remaining seven modules in each track process the noise corruption and distortions on the pulses that are described in Section 2. An L-bit long isolated pulse is chosen from the *pulse look-up table* that is implemented in RAM. The look-up table contains N pulses that have different pulse shapes and widths. The amplitude and the shape of the selected pulse are altered by HNL and PE multipliers. In NLTS module, nonlinear transition shift is integrated into the overall moves to be executed in a 2-level Barrel shifter. After passing through the shifter, the pulse is superimposed on a sequence of pulses that moves forward by T bits in every clock cycle. The write error is added to the pulse sequence in the WERR module. Electronics noise is then incorporated onto each point before the output signal is multiplied by the OT ratio. Since one operand to every embedded multiplier is a known parameter, it is practical to implement the multipliers in look-up tables to increase the run speed.

3.4 Reconfiguration

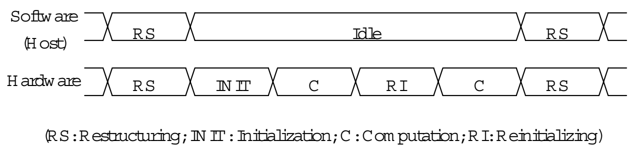
Software
(Host)
RS
Idle
RS
Hardware
RS
INIT
C
RI
C
RS
(RS:Restructuring; INIT:Initialization; C:Computation; RI:Reinitializing)

Figure 7: Time-duplexing

Reconfigurable computing in FPGAs provides a way to increase the device capacity and improve flexibility when solving a range of problems without duplicating hardware ([15], [16]). In the readback signal generator, two dimensions of reconfigurations are considered: *reinitializing* and *restructuring*. They are accomplished jointly through hardware and software.

3.4.1 Reinitializing

When a read head moves from one track to another, the pulse-shape, noise statistics, noise amplitude, and ITI are likely to change. This reality motivates the reconfiguration of the signal generator at runtime. The execution time is duplexed to two phases: *initialization phase* and *computation phase*. Prior to the execution of these two phases, configuration data is calculated by host software and loaded to the on-board RAM. In the initialization phase, the initialization control unit redefines look-up table contents in the signal generator with configuration data stored in the RAM. The execution time of reinitialization ranges from about one thousand to less than a hundred thousand clock cycles depending on the scale of the generator while the number of clock cycles in the computation phase is significantly larger. Meantime, the computation circuit idles for the initialization to finish and contents in all registers stay unchanged. When the generator completes reinitialization, it will generate readback signals characterized by the new configuration (see Fig. 7).

3.4.2 Restructuring

Although the noise statistics or pulse-shape may change in time, the kinds of noise processes and distortions remains the same in many cases when a hard drive system is put together. For instance, PE may happen for one type of magnetic media but be absent on another. In other words, not every module in the design has to be involved in the generation of a particular sequence of readback signals. Therefore, a customized structure is preferred to virtually increase the device capacity. This becomes more critical when the signal generator is integrated into the whole simulator. It is normally expected to economize on the generator to preserve more resource for data-recovery modules. Additionally, since the complicated generator takes a large share of the area budget, optimizing the generator structure is an efficient way to reduce the overall area consumption. To serve the interests of different simulation emphasis, we have chosen six typical combinations of noise and distortion (See Table 1). These combinations are by no means conclusive. In theory, there are $2^7 = 128$ possible bitstreams that can be prepared in advance. The task of choosing which bitstream to download is done by the software according to the user's demand. In other words, the functionality of the generator now is split between the hardware and software. The consumed resources will be reported in Section 3.6.

In summary, reconfiguration through time-duplexing and hardware-software co-design helps achieve resource reusing in the FPGA device. Hence, we can simulate readback signals with different noise processes and distortions by using significantly less resources.

Table 1: Implemented noise/distortion combinations

#	TN	HNL	PE	NLTS	WERR	EN	ITI	slices
0	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	12286
1	$\surd$							5010
2	$\surd$					$\surd$		7649
3						$\surd$		5716
4	$\surd$					$\surd$	$\surd$	12235
5	$\surd$	$\surd$	$\surd$	$\surd$				6410

3.5 Discussion

In the cases when the restructuring is required to be done during runtime, we can store the register contents to on-board RAM and load a different bitstream from host in a very short time. A control unit then configures the new generator with data readback from the RAM. However, a better approach will be laying out each module into physically bounded and floorplanned blocks and dynamically reconnecting different blocks into a new circuit. An extra credit by doing so is the simplification of the initialization control unit by putting various initial conditions into blocks as well. In other words, the reinitialization and restructuring can be merged into one reconfiguration scheme and it can be performed on the fly. A capable tool to achieve this approach is Xilinx's JBits ([17], [18]). Unfortunately, by the time of this publication JBits and the board in use are not compatible; hence, we are not able to implement this scheme. However, we believe it can help further improve the flexibility in utilization of this generator and reduce its resource consumption.

3.6 Implementation Report

A set of sample implementations (Table 1) have been constructed with the parameters given in Table 2.

The total area consumption of the implementaion with all modules (bitstream 0) is 99% of 12288 slices. This extreme number clearly indicates that it is vital to utilize reconfiguration in this design. The resource consumption of other bitstreams is listed in the last column in Table 1. When the size of design approaches the limit of the device, the number of used slices may not reflect the actual area consumption by the generator very well. From the table, it shows that by doing customized restructuring, we can significantly avoid wasting resources on unused modules. The readback signal generator can run with clock speeds up to 70MHz. The runtime of generating 10^{12} pulses on a FPGA-based generator is 4 hours compared to about 50000 hours in software. The runtime on a software-based generator is estimated on a PC eqipted with one 1GHz Pentium$^{\text{TM}}$ 4 processor and 256MB RAM. A sample output signal is shown in Fig. 8.

All the design parameters in Table 2 can be easily scaled up when a more precise generator is in demand and a larger

Table 2: Parameters of sample implementations

Parameter	Symbol	Value
Pulse look-up table size	N	16
ISI length	I	6
Symbol period	T	8
NLTS	εT	2
Signal-to-noise ratio	SNR	10 dB
Percentage of TN in total noise	α	90%
Percentage of jitter in TN	PJP	75%
LHCA length	l_{LHCA}	244
Site spacing parameter	η	1
Quantization levels	q	64
Uniform RN width	n	11
OT ratios	r and r'	0.9 and 0.1
Head nonlinearity	HNL	1 for pos. pulses 0.9 for neg. pulses

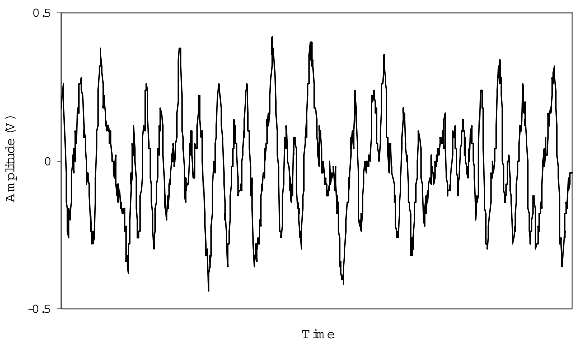

Figure 8: Generated readback signal

FPGA device is available. Every module can be easily removed from the design and a new module can be inserted without modification of others.

4. CONCLUSIONS

A readback signal generator for hard-drive systems has been implemented on a Xilinx Virtex$^{\text{TM}}$E FPGA device. It is capable of simulating all major noise processes and distortions that have been observed in hard-drive read and write processes. The design is modularized to meet the requirements of reconfiguration. The noise and distortion sources can be reconfigured in runtime and a flexibility of choosing from various noise and/or distortion sources is provided. The resource consumption can be reduced by a significant amount through reconfiguration.

5. REFERENCES

[1] J. Moon, “Discrete-time modeling of transition-noise-dominant channels and study of detection performance”, *IEEE Transactions on Magnetics*, vol. 27, no. 6, pp. 4573-4578, Nov. 1991.

[2] J. Caroselli and J. K. Wolf, “Applications of a new simulation model for media noise limited magnetic recording channels”, *IEEE Transactions on Magnetics*, vol.32, no.5, pt.1, pp.3917-19, Sept. 1996.

[3] H. N. Bertram, *Theory of magnetic recording*, Cambridge University Press, 1994.

[4] T. Yamauchi and J. M. Cioffi, “A nonlinear model for thin film disk recording systems”, *IEEE Transactions on Magnetics*, vol.29, no.6, pp.3993-95, Nov. 1993.

[5] B. Lin, E. Hild, and J. R. Cruz, “Magnetoresistive read/write channel models”, *IEEE Transactions on Magnetics*, vol.35, no.6, pp.4528-31, Nov. 1999.

[6] W. G. Solomon, *Shift register sequences*, Aegean Park Press, 1982.

[7] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller, H. C. Card, “Cellular automata-based pseudorandom number generators for built-in self-test”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits & Systems*, vol.8, no.8, pp.842-59, Aug. 1989.

[8] P. D. Hortensius, R. D. McLeod, H. C. Card, “Parallel random number generation for VLSI systems using cellular automata”, *IEEE Transactions on Computers*, vol.38, no.10, pp.1466-73, Oct. 1989.

[9] 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.1625-7, Aug. 1991.

[10] K. Cattell, S. Zhang, “Minimal cost one-dimensional linear hybrid cellular automata of degree through 500”, *Journal of Electronic Testing: Theory & Applications (Jetta)*, vol.6, no.2, pp.255-8, April 1995.

[11] S. Wolfram, “Statistical mechanics of cellular automata”, *Reviews of Modern Physics*, vol.55, no.3, pp.601-44, July 1983.

[12] Annapolis Micro Systems, Inc., *FIREBIRD^{TM} Hardware Reference Manual*, 2001.

[13] J. Max, “Quantizing for minimum distortion”, *IEEE Transactions on Information Theory*, IT-6(1), pp.7-12, March 1960.

[14] J. Chen and J. Moon, “Detection signal-to-noise ratio versus bit cell aspect ratio at high areal densities”, *IEEE Transactions on Magnetics*, vol.37, no.3, pp.1157-67, May 2001.

[15] S. Trimberger, D. Carberry, and A. Johnson, “A time-multiplexed FPGA”, *Proceedings of The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp.22-8, 1997.

[16] A. Dehon and J. Wawrzynek, “Reconfigurable computing: what, why, and implications for design automation”, *Proceedings of 1999 Design Automation Conference*, pp.610-15, 1999.

[17] S. A. Guccione, D. Levi, and P. Sundararajan, “JBits: A Java-based interface for reconfigurable computing”, *Proceedings of 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference*

[18] C. Patterson, “High performance DES encryption in Virtex FPGAs using JBits”, *Proceedings of 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, pp.113-21, 2000.