Modeling and Analysis of Communication Circuit Performance using Markov Chains and Efficient Graph Representations

Alper Demir Peter Feldmann

{alpdemir, peterf}@research.bell-labs.com

Bell Laboratories
Murray Hill, New Jersey, USA

Abstract

In high-speed data networks, the bit-error-rate specification on the system can be very stringent, i.e., 10^{-14}. At such error rates, it is not feasible to evaluate the performance of a design using straightforward, simulation based, approaches. Nevertheless performance prediction before actual hardware is built is essential for the design process.

This work introduces a stochastic model and an analysis-based, non-Monte-Carlo method for performance evaluation of digital data communication circuits. The analyzed circuit is modeled by a number of interacting finite state machines with inputs described as functions on a Markov chain state-space. The composition of these elements results in a typically very large Markov chain. System performance measures, such as probability of bit errors and rate of synchronization loss, can be evaluated by solving linear problems involving the large Markov chain's transition probability matrix. This paper first describes a dedicated multi-grid method used to solve these very large linear problems. The principal bottleneck in such an approach is the size of the Markov chain state-space, which grows exponentially with system complexity. The second part of this paper introduces a novel, graph based, data structure capable of efficiently storing and manipulating transition probability matrices for several million state Markov chains. The methods are illustrated on a real industrial clock-recovery circuit design.

1 Introduction

High-speed communication systems have extremely tight bit-error-rate (BER) specifications. For SONET/SDH applications it is not uncommon to have BER requirements in the order of 10^{-14}. Such specifications are practically impossible to verify through straightforward simulation because of the extremely long sequence that would need to be simulated in order to get meaningful error statistics. In the absence of a performance analysis tool, designers rely on the experience of previous designs, intuition, and good luck. This environment discourages innovative solutions and non-incremental applications.

On the other hand, the design process of communication systems would benefit significantly from the existence of a reliable design performance evaluation capability. Such a capability would permit the evaluation of a number of alternative algorithms, architectures, circuit techniques, and technologies in a short time and without the commitment of expensive resources.

A situation that illustrates the need for a reliable evaluation capability of the BER occurred in the design of a SONET-type application at a well-known micro-electronics company. The specification for a multiplexer chip required a BER of 10^{-14}. The prototype implementation, based on the modification of an existing design delivered performance that was more than an order of magnitude bellow the specification. The designers suspected that the main cause for the errors is the interference noise in the PLL-based clock recovery circuit, induced by the rest of the chip's circuitry. A number of circuit, technology, and packaging remedies were proposed, but the designers were frustrated by their inability to predict their effectiveness.

This paper introduces a method for performance evaluation of digital data communication circuit designs. Our analysis method computes the probability of errors directly from the design description, without relying on the simulation of long sequences. The system under evaluation is described as a number of finite-state machines (FSMs) with some of their inputs being random. The random processes describe stochastic models for the incoming data, noise, and jitter. The random inputs are modeled as functions on the state-space of Markov chains. It is shown that under these circumstances the entire system can be modeled by a larger Markov chain. The quantities of interest for our system, such as the probability of a detection error, or the mean time between failures due to detection errors are thus available from standard Markov chain analysis.

The first challenge is to develop numerical methods capable of handling the extremely large transition probability matrices associated with Markov chains that can easily reach millions of states for moderately complex systems. In this work we employ a specialized multigrid method which takes advantage of the underlying problem structure and is capable of solving million state problems in less than an hour on a beefed-up workstation.

The remaining challenge is to store and perform computations with the extremely large Transition Probability Matrices (TPMs) associated with Markov chains that can easily reach millions of states for moderately complex systems. For this purpose we introduce a novel graph-based data structure called **C**onditionally **O**rdered **C**onditional **P**robability **D**ecision **G**raphs (COCPDGs). COCPDGs are capable of storing and performing operations efficiently with TPMs resulting from multi-million size chain state spaces. In contrast to alternative data structures, proposed in the past, COCPDGs are efficient for any practical interconnection structure of the model FSMs. COCPDG storage requirements typically grow sub-linearly with the size of the Markov chain state-space. The cost of computing the product of a COCPDG-encoded TPM with an arbitrary unstructured vector is linear in the size of the vector. Multiplication of the COCPDG-encoded TPM with structured and graph encoded vectors can be performed in sub-linear time, but the use of structured vectors severely limits the choice of the numerical method for eigen computations and linear system solutions.

In this work we demonstrate the use of the COCPDG to encode a TPM resulting from the modeling of a real industrial clock and data recovery circuit. For one example, the Markov chain has more than two million states and the TPM has 1.35 billion non-zeros. With our data structure we encode the matrix with about 160 MBytes and perform a matrix-vector multiply in approximately 20 secs. The clock and data recovery circuit performance measures are computed through a simple power iteration in several hours.

2 Modeling and Performance Evaluation

Throughout the paper, we will be using the CDR circuit [1, 2] shown in Figure 1 to illustrate the stochastic model and the performance evaluation techniques. The framework we present here is by no means restricted to this particular circuit, and the general model we describe can be used for other discrete-time mixed-signal processing circuits.

The CDR circuit in Figure 1 consists of two coupled feedback loops. The first one (upper left) is a traditional "analog" charge-pump phase-locked loop (PLL) with a crystal reference and a voltage-controlled oscillator (VCO) that can generate multi-phase clocks (e.g., a ring-oscillator). The second loop (lower right) is *digital*, and has the purpose of selecting "the best" of the clock phases generated by the first loop in order to retime/align the data. This phase selection is continually updated by the loop. The currently selected phase and the incoming data are "compared" in the phase detector (PD) which

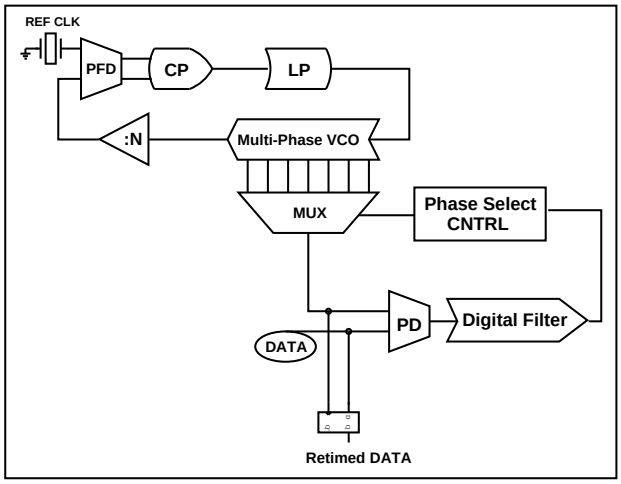


Figure 1: Clock and data recovery circuit

produces a *digital* phase error signal. The digital output of the PD is further filtered to produce the actual digital signal that controls the phase selection multiplexer.

For a timing recovery circuit, the BER specification for the retimed data has to be met for a given data characteristics in the presence of jitter. Moreover the phase detector can produce a phase error signal only when a transition occurs in the data signal. The input data stream is usually specified in terms of the longest possible bit sequence with no transitions and a maximal drift in frequency. The input data jitter is specified by eye opening, usually defined as uncorrelated timing jitter from a bit to the next. Sometimes correlated or cumulative jitter, i.e., a random walk, may also be specified. There are also specifications on the recovered clock jitter.

In this paper, we are going to concentrate on the digital phase selection loop. The major jitter source in most CDR applications is the incoming data, but the internally generated clock jitter due to device noise or interference from other circuits can also become significant. Once the internal clock jitter has been characterized using techniques covered elsewhere, it can easily be captured in our models and analysis.

It is virtually *impossible* to simulate the BER for the whole clock recovery system at once using *detailed transistor-level* models for the whole circuit. The size of the problem (in terms of the variables and the number of differential equations that describe it) is simply too large to handle, now or in the foreseeable future. Moreover, we are confronted with a mixed digital-analog circuit with large time constant feedback loops, i.e., stiff. The only use of transient simulation at the circuit level is for connectivity verification, and simple functional verification of the laid-out circuit. Obviously, such a verification is far from ensuring that the design meets the BER specification. To tackle the system level problem, we have to develop intelligent models that simplify the problem, but at the same time, capture the characteristics of the circuit that is essential for its operation.

The components in the digital phase selection loop, such as the phase detector and the digital filter, are highly nonlinear circuits with switching behavior if viewed from a differential equation (DE) perspective. A DE model noise analysis based on linearization (time-invariant or time-varying) is neither useful for, nor applicable to, this problem, because the noise or data jitter is too large for the linearization to remain valid. Moreover, the internal device noise sources (e.g., thermal noise) have little significance. The loop components are "almost" ideally functioning digital circuits and hence can be modeled as *discrete-time* digital systems. On the other hand, jitter and the phase error between the selected clock phase and data are continuous variables.

The simplest model that captures the essential behavior of the digital phase selection loop in Figure 1 can be expressed with the following difference equation

$$\Phi[k+1] = \Phi[k] - G\,\mathrm{sgn}(\Phi[k] + n_w[k]) + n_r[k] \qquad (1)$$

where Φ is the phase error between the incoming data and the recovered clock. The phase detector is simply modeled as a *memoryless* nonlinear function which produces the *signum* of its input at the output. n_w and n_r are random processes that model the jitter of the incoming data. n_w is a zero-mean *white*, i.e., uncorrelated in time, noise process that is usually Gaussian. n_w models the eye opening of the data and its characteristics can be readily deduced from the system specifications. n_r is usually a *nonzero* mean white noise process. From (1) one can see that if n_r has nonzero mean than the phase error will have a deterministic drift in the absence of the *signum* term which is responsible for phase corrections. One can also observe that the random part of n_r has a cumulative effect on the phase error: If the *signum* term and n_w was not present in (1) than the phase error would be a *random walk* with drift for a nonzero mean white n_r. Almost all jitter specifications on the incoming data can be represented together by n_w and n_r by assigning appropriate amplitude distributions (e.g., Gaussian with certain mean and variance). For instance, one can even "mimic" deterministic sinusoidally varying jitter by assigning the amplitude distribution of n_r appropriately.

The hardware implementation of the phase detector has to operate at the full data speed, hence it needs to be implemented by a relatively simple state machine. The same is true for any digital filtering that might be done at the output of the phase detector. Let us assume that $S[k]$ is a vector representing the state of the finite state machine (FSM) that implements the phase detector and the filter. We will now rewrite/revise (1) in the following more general form which will capture a real implementation

$$\begin{aligned} \Phi[k+1] &= \Phi[k] - f(\Phi[k] + n_w[k], S[k]) + n_r[k] \qquad (2) \\ S[k+1] &= g(\Phi[k] + n_w[k], S[k]) \qquad (3) \end{aligned}$$

Above, the functions f and g specify the phase-detector-filter FSM: g gives the next state of the state machine given its present state and present noisy phase error value. Similarly, f produces a value indicating the phase correction. In the implementation of Figure 1, f takes three possible values 0, G, $-G$ indicating no correction, phase delay or advance respectively. G is the smallest phase increment available from the internal clock.

The combined vector $X[k] = (\Phi[k], S[k])$ represents the state variables of the system described by the *nonlinear* difference equations in (2). Since there are noise sources as inputs to the system, $X[k]$ is best characterized as a stochastic process. We would like to analyze this stochastic process in order to evaluate the various system performance measures.

When the noise sources n_w and n_r are white, i.e., uncorrelated in time, $X[k]$ is a Markov process, that is, given its current state, its future is independent of its past. One way to analyze the system in (2) is using the machinery of discrete-time Markov chains, which requires that we discretize the phase error and also the noise sources to obtain a discrete state-space. The granularity of the discretization of the phase error and the noise sources is dictated by the number of clock phases and the magnitude of the noise source n_r. The discretization grid needs to be fine enough to accurately capture the small jumps in phase error due to n_r.

A Markov chain MC is completely characterized by its transition probability matrix (TPM) $\mathbf{P} = [p_{ij}]$

$$p_{ij} = \mathtt{Pr}\left(X[k+1] = x_j \,\middle|\, X[k] = x_i\right) \qquad (4)$$

where the state set $\{x_1, \cdots, x_L\}$ is the reachable state space of the MC, which is a subset of the Cartesian product of the discretized phase values $\{\phi_1, \cdots, \phi_M\}$ and the state set $\{s_1, \cdots, s_N\}$ of the phase detector/filter FSM. The entries of $\mathbf{P}$ are completely specified by the difference equations in (2) and the probabilistic characterization of the discretized noise sources. $\mathbf{P}$ is a very large but highly structured matrix. The structure is induced by the phase detector/filter FSM and the difference equations. In spite of its size, the matrix can be efficiently constructed as a composition of smaller components representing building blocks of the system using a graph based representation described in the sequel. This representation makes it possible to manipulate and store $\mathbf{P}$ even when the total state space is very large. Figure 2 shows

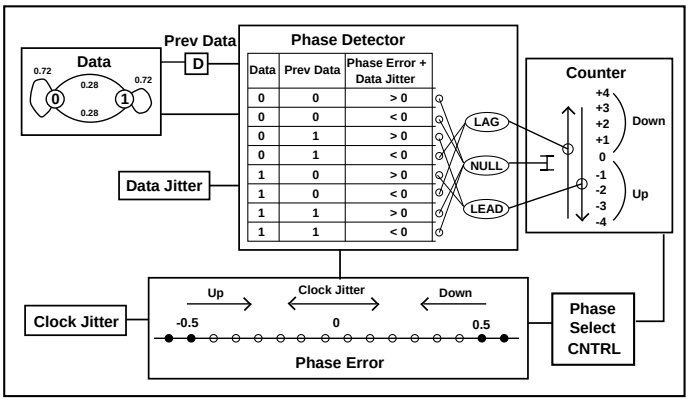


Figure 2: Model of clock and data recovery circuit

a more detailed compositional model of the clock recovery system of Figure 1 described graphically in the above formalism. This representation can be generalized to networks of FSMs with stochastic inputs to describe various high-speed communication circuits.

Now that we described our system in the MC formalism, we can compute various quantities that characterize the state of the system as a stochastic process. For the clock recovery system, whenever the phase error plus the data jitter, i.e., $\Phi[k]+n_w[k]$ in (2), becomes larger/smaller than half a clock cycle, the system might potentially produce bit errors. It would be highly desirable to compute the probability of this event happening. This probability can be directly obtained form the steady-state probability distribution of reachable states, which is the most basic analysis for MCs. This involves computing the eigenvector corresponding to the eigenvalue 1 of the stochastic matrix $\mathbf{P}$ [3]. Another measure of performance for CDR circuits is the average time between cycle slips. This translates into the computation of mean transition times between certain sets of MC states, which is another standard computation in MC analysis. It involves solving a linear system with the (modified) TPM.

3 Numerical Methods

The TPM, $\mathbf{P}$, of a MC is commonly called a stochastic matrix [3]: From its definition, we immediately deduce that it has non-negative entries (they are all probabilities), and its row sums are equal to 1 (a row expresses all the possibilities in a given state), i.e.,

$$\mathbf{P}\,[1,1,\ldots,1]^T = [1,1,\ldots,1]^T$$

It follows that $\mathbf{P}$ has $\xi = [1,1,\ldots,1]^T$ as a right-eigenvector corresponding to the eigenvalue 1. Let us denote the stationary state probabilities with $\eta_{[i]} = \mathtt{Pr}\,(z[n] = \sigma_i)$ as the entries of the $1 \times L$ row vector $\eta = [\eta_{[i]}]$. η satisfies [3]

$$\eta = \eta\,\mathbf{P} \tag{5}$$

Being a probability distribution, the vector η has nonnegative entries and the sum of its entries is equal to 1, and it is a left-eigenvector of $\mathbf{P}$ corresponding to the same eigenvalue 1. The computation of η is the most basic analysis for MCs. The information in η already makes it possible to compute some performance measures for the modeled system as discussed in Section 2. Moreover, computation of η is the prerequisite for computing other performance quantities such as the autocorrelation of a function defined on the states of the MC. Hence, we concentrate on methods for computing η, which can be posed either as an eigenvalue problem through (5), or as the solution of the following *homogeneous* linear system

$$(\mathbf{P}^T - \mathbf{I})\,\eta^T = 0 \tag{6}$$

with the normalization

$$\eta\,\xi = 1 \tag{7}$$

A variety of standard iterative techniques can be used to solve these problems. These techniques, however, do not exploit the properties of MCs.

For the eigenvalue problem, they range from simple power iteration to subspace projection methods, such as the Krylov subspace methods Arnoldi and Lanczos. For the solution of the linear system one can employ Gaussian elimination (specialized versions), (block) Jacobi, Gauss-Seidel or SOR iterations, and Krylov subspace methods, such as GMRES and the methods based on the Lanczos biorthogonalization procedure. The feasibility and convergence behavior (for the iterative ones) of these techniques are very much application dependent. If the TPM is available explicitly, stored in a sparse data structure, any of the above methods can be used. In some very large problems, however, matrix information may be available only implicitly through matrix-vector products, which limits the solution options. It is known that iterative methods may exhibit very slow convergence, or may not even converge, unless properly preconditioned.

A family of iterative techniques, specific to MC problems, are *aggregation/disaggregation*-type methods [4]. These techniques arise from and are related to the *lumpability* and *decomposability* concepts in MCs. Here we discuss only lumpability. Assume that we are given an N-state MC. We partition these N states into n disjoint sets with $n < N$, and form a new stochastic process by defining new states corresponding to the n sets. The value of the new stochastic process at time k is the new state that corresponds to the set that contains the state of the original chain at time k. This procedure could be used to reduce a MC with a very large number of states to a process with a smaller number of states, called the *lumped* process. It is often the case that we are only interested in these *coarser* states. For example, in the model of the clock recovery circuit, we are interested in the phase error which is only a component of the state vector. There are multiple states which correspond to the same phase error value. With the above procedure, we can define a process which is exactly equal to the phase error. However, the crucial question is, whether the newly defined process is *Markov* for *any* initial probability distribution for the states of the original MC. If so, we can treat the new process with MC methods and hence reduce the size of the problem. Unfortunately, the answer to this question in most cases is no, otherwise the model we originally developed was redundant and could have been simplified.

If we loosen the condition for the newly defined process to be Markov from *any* to *some* initial probability distribution, and if such an initial distribution exists, the MC is called *weakly lumpable*. In this case, the computation of the TPM for the reduced MC requires both the TPM of the original MC and the initial probability distribution [3]. This is basically the starting point for aggregation-disaggregation techniques for MCs that are used to accelerate the convergence of basic iterative methods such as Jacobi and Gauss-Seidel and possibly the Krylov subspace methods. For instance, let Jacobi be the iterative method. After performing a number of steps of Jacobi, the current iterate for the stationary vector and the TPM for the MC is used to compute a reduced stationary vector and a reduced TPM for the weakly lumped chain. Then, the reduced iterate vector is used as the initial guess to solve the weakly lumped chain exactly. Next, the solution of the lumped problem together with the initial guess is used to produce a correction to the *finer* level iterate. These steps are repeated till convergence [4]. This technique was generalized to more than two *lumping levels* by Horton and Leutenegger [5]. The multi-level method utilizes a set of recursively lumped versions of the original MC to achieve accelerated convergence. It can be interpreted as an algebraic multi-grid method.

The multi-level algorithm can achieve much better performance if the special structure in the MC or the underlying model composed of finite-state machines is exploited to develop a *coarsening* or *lumping* strategy. For the model of the clock recovery circuit in Figure 2, we employed a coarsening strategy which lumps the two states corresponding to consecutive discretized phase error values. In this way, the lumped problems resemble the original problem but with coarser phase error discretization. However, the coarsened problems do not capture all the behavior of the original model. For instance, for some of the problems, the phase error grid is too coarse for the effect of the small noise n_r in (2) to be represented accurately. Nevertheless, the coarse problems retain enough characteristics of the fine problem so as to help accelerate the convergence.

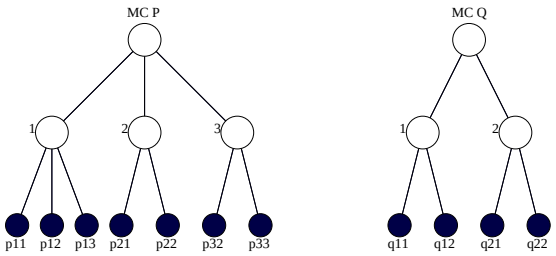


Figure 3: Graph representation of TPMs

We implemented such a multi-level algorithm in [6], where the lumping and expanding steps are interleaved with simple Gauss-Jacobi iterations and the coarsest problem is solved exactly with a direct method. In this implementation [6], we use flat sparse storage for both the fine and the coarse problems and this severely limits the size of the problem that can be handled. In order to overcome this limitation, we introduce a novel, graph based, data structure capable of modeling systems with state spaces larger by about two orders of magnitude. The data structure is described in detail in the following section.

This paper only reports results for the use of this data structure in conjunction with a simple power iteration algorithm. The extension of the more powerful algorithms (e.g., the multi-level algorithm) mentioned in this section for use with matrices represented by the graph based data structure will be reported in future publications.

4 Efficient Matrix Representation and Manipulation

Flat sparse storage of the TPM of the MC becomes prohibitive for problem sizes of over 1 million states and 100 nonzeros per row (requiring 1 GB of memory). Fortunately, TPMs for MCs resulting from FSM models are not arbitrary. Typical systems modeled with MCs are composed of a number of interacting FSMs. Therefore the corresponding TPM has a structure that can be exploited for efficient representation.

As an extreme example, the TPM for a composition of n independent 2-state MCs is in general a full matrix requiring 2^{2n} entries. By exploiting the structure of this matrix, it can actually be represented with only $4n$ numbers using Kronecker algebra. For most practical cases, the interaction between the component FSMs is more complex and some storage-efficient TPM representations have been proposed [7, 8, 9].

Unfortunately these schemes impose severe limitations on the types of interactions that the components may have. For example the generalized Kronecker representation [7] can only describe machines that interact unidirectionally (i.e., states of machine A affect the transitions of machines B and C, machine B affects C, but C can affect neither A nor B or a cycle will occur). While the graph based representations in [9] retain storage efficiency in the presence of cycles, the computational complexity of matrix-vector products is degraded.

We introduce a novel graph based representation of TPMs which represents a generalization of the Conditional Probability Decision Graphs (CPDGs) [9]. The new representation retains both storage and computational efficiency even in the presence of cyclic interactions among component FSMs.

We start by informally reviewing the Conditional Probability Decision Graphs (CPDGs) [9]. Examples of CPDGs are shown in Figure 3 which encode respectively the matrices

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & 0 \\ 0 & p_{32} & p_{33} \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix}$$

The white nodes correspond to rows in the TPM (the present state of the MC) and the dark ones correspond to columns (the next state of the MC). Every path from the root to the leaves corresponds to a nonzero entry in the TPM. Graphs for coupled machines can be constructed by composing the individual machine graphs.

For example if we assume that the two machines in in Figure 3 are independent, the graph for the composition TPM $R = P \otimes Q$ (Kronecker product) is the one shown in in Figure 4(a). While the number of nonzeroes in the composed matrix is the *product* of the number of nonzeroes of the individual matrix sizes (7×4 entries), the number of nodes in the composed graph is only the *summation* of the number of nodes in the individual graphs. More generally, when machine MC Q depends on the state of MC P, its TPM will be one of

$$Q^1 = \begin{bmatrix} q^1_{11} & q^1_{12} \\ q^1_{21} & q^1_{22} \end{bmatrix}, \quad Q^2 = \begin{bmatrix} q^2_{11} & q^2_{12} \\ q^2_{21} & q^2_{22} \end{bmatrix} \quad \text{or} \quad Q^3 = \begin{bmatrix} q^3_{11} & q^3_{12} \\ q^3_{21} & q^3_{22} \end{bmatrix}$$

conditional on the present state of MC P. In this case, the composed graph is shown in Figure 4(b). In the most general case, both machines depend on each other's states and the graph will have the form in in Figure 4(c). Note that every path from the root to the leaves corresponds to a nonzero entry in the TPM, which can be obtained as the product of the probabilities of the dark nodes along the path.

	flat	graph		
		independent	unidirectional dependence	cyclic dependence
Mem.	$(a^2)^n$	na^2	a^n	$n(a^n)$
CPU	$(a^2)^n$	a^n	a^n	$(a^2)^n$

Table 1: Storage and computational complexity comparison

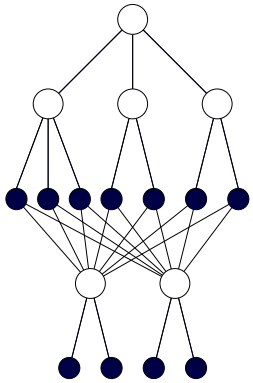
In all cases, graph representation leads to significant storage savings compared to the flat matrix representation as shown in Table 1 (where a is the number of states in an individual machine, and n is the number of machines being composed). The graph representation can be used to compute the result of the multiplication of the underlying TPM with an arbitrary unstructured vector. This is done by recursive descent in the graph where every node in the graph corresponds to a selection of rows and columns. The reconvergent paths in the graph indicate common partial results, which, when reused, lead to a decrease in the computational complexity of the multiplication. The complexities for the three cases are shown in Table 1. Note that in the independent and unidirectional dependence case, the multiplication is linear in the number of states of the composed machine. Unfortunately, there are no savings in computation for machines with cyclic dependence. From the discussion above it appears that CPDGs are only advantageous in comparison with flat storage when machines have at most unidirectional dependence. Unfortunately, most practical cases lead to models with cyclic dependent components.

We now introduce a generalization of CPDGs called Conditionally Ordered CPDGs, (COCPDG). We will show that COCPDGs can model machines with cyclically interacting components and still exhibit storage and computational complexities that are at most linear in the total number of states.

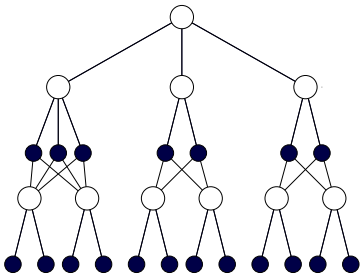
We first introduce the concepts of *conditional independence* and *conditional unidirectional dependence* needed for the generalization. Their formal definition is beyond the scope of this paper and we prefer to explain them through an example: In the model of the CDR circuit, in Figure 2, the occurrence of a data transition is modeled by an FSM the output of which is either *Transition* or *NoTransition*. This FSM operates independently of the other components, but both the phase-FSM and the counter-FSM depend on it and on each other.

When the output of the data-FSM is conditioned to be *NoTransition*, the FSMs that model the counter and the phase error become mutually independent, i.e., *conditionally independent* and can be composed like in Figure 4(a) for maximum savings in the corresponding subgraph. In the presence of data transition, the counter-FSM and phase-FSM have a cyclic dependence that we need to break. We do this by conditioning on the states of the counter-FSM. We distinguish the case when the counter is full. When the counter is not full, the phase-FSM becomes independent of the counter, and the cyclic dependence becomes *conditional unidirectional dependence*, and again the corresponding subgraph can be efficiently composed. The remaining situation (data transition and full counter), while still containing cyclically dependent components, can be described with reasonable complexity due to the significantly smaller subset of states involved. The structure of the resulting graph is illustrated in Figure 5.

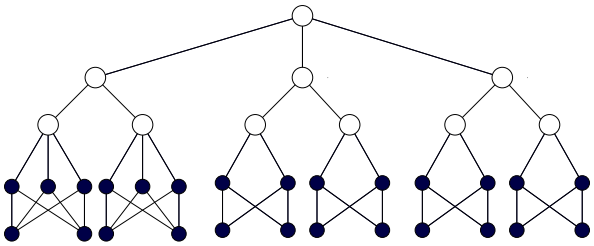
A graph node is labeled by the present-state or the next-state of a component FSM. In CPDGs, the order of the node labels from the



(a) Independent



(b) Unidirectionally dependent



(c) Cyclically dependent

Figure 4: Composed graph for TPM of two FSMs

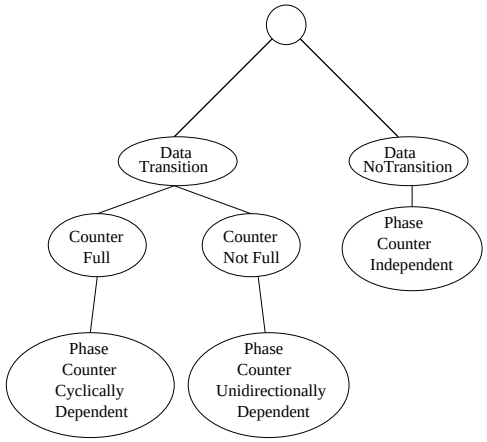


Figure 5: Independence and unidirectional dependence by conditioning

root to the leaves is the same for all the paths of the graph. In contrast, the COCPDG allows reordering of the labels in subgraphs so that maximal conditional independence and unidirectional dependence is achieved. Allowing different orders in subgraphs does not increase either the storage or the computational complexity. It only imposes a permutation on the multiplication vectors.

5 Examples

We built a compositional model of the clock recovery circuit in Figure 1. It consists of four interacting FSMs with stochastic inputs. The first FSM models the data statistics taken from SONET system specifications. The second one is the model of the phase detector and has present data, previous data and the noise source n_w (model of the eye opening) as its inputs. It produces a three-valued output: LAG, LEAD and NULL. Its output is the input to an up-down counter FSM that models the loop filter. The counter produces an UP-DOWN signal when it overflows, which is one of the two inputs to the FSM that has the phase error as its state. The other input to the phase error FSM is the noise source n_r.

All figures show the stationary probability density functions of the phase error Φ and the input to the phase detector, i.e, $\Phi + n_w$. The line above all of the density plots shows the counter length, the standard deviation of the stationary zero-mean white Gaussian noise n_w, the maximum value of the stationary white noise n_r (with a non-zero mean, non-Gaussian distribution with probability density function chosen to reflect SONET system specifications), and the BER computed by integrating the tails of the distribution computed using MC analysis.

The results in Figure 6 and Figure 7 were obtained using the multilevel algorithm with flat sparse storage. In these figures, the line below the density plots shows the size of the state space for the MC generated from the model, the number of multi-grid cycles required for convergence, the CPU time for generating the flat sparse storage for the TPM of the MC, and the CPU time spent for the stationary distribution computation.

In Figure 6, in the top plot, the noise levels are so small that the CDR system has negligible BER. When the standard deviation of the noise source n_w that models the eye data opening is increased 5 times, the BER increases to 1.23×10^{-11}, as seen in the bottom plot in Figure 6.

In Figure 7, we study the effect of the counter length on the BER performance, all noise levels being held constant. We set it to 4, 8 and 16. We observe that the best BER performance is obtained when counter length is set to 8, BER performance is 1.5 times worse with counter length 4, and 5 times worse with counter length 16. When the length is set to 4, the loop has high bandwidth. The system tends to follow the dominant noise source, n_w, and as a consequence detection errors occur. When the length is set to 16, the effect of the noise source n_r becomes predominant: the loop response becomes too slow to follow the drift caused by n_r and, again, bit errors occur. The length 8 is a good compromise, where both noise sources contribute to the BER.

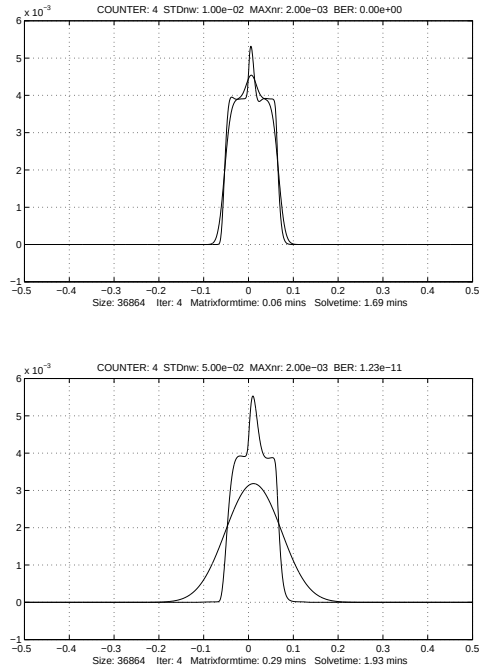


Figure 6: Phase error probability density and BER (multi-level algorithm with flat sparse storage)

counter length	state-space size	non-zeros in TPM (millions)	graph memory (MBytes)	multiplication time (CPU secs)
4	589824	405	61	5.23
8	1114112	719	93	10.6
16	2162688	1347	158	20.8

Table 2: Information on COCPDG representation of the TPM of the MC model of the clock recovery circuit

Hence, there is an optimal counter length for given levels of noise, the computation of which is enabled by the accurate and efficient analysis method described in the paper.

The number of non-zeros in the TPM of the MC generated from the model of the clock recovery circuit is dictated by the granularity of the phase error discretization. For the above results obtained with flat sparse storage, we were forced to use a relatively large discretization step resulting in a coarse approximation for the probability density functions of the noise sources n_r and n_w. It is desirable to use a finer phase error discretization so that the continuous density functions for the noise sources are closely approximated for more accurate results. This is enabled through the use of the graph data structure introduced in the paper. With finer phase error discretization, and with counter length set to 16, the size of the state space is above 2 million. If flat sparse storage is used to represent the TPM with *1.35 billion non-zeros*, 15 GBytes of memory would be needed. The graph data structure requires only 160 MBytes, two orders of magnitude less than what is required by flat sparse storage. In this case, the multiplication of the TPM with a flat unstructured vector takes about 20 secs. All noise levels being held constant, Table 2 shows the size of the state space for the MC generated from the model, the number of non-zeros in the TPM for a flat sparse storage, the memory of the graph representation, and the CPU time for the TPM-vector multiplication using the graph data structure, for counter lengths of 4, 8 and 16.

6 Conclusions

This paper introduced a new, non-Monte-Carlo analysis method, for the stochastic analysis of digital data communication circuits. The analysis is based on the modeling of the underlying system as a combination of finite state machines and Markov chains. The relevant system performance measures are derived from computations that involve the transition probability matrix of a large resulting Markov chain.

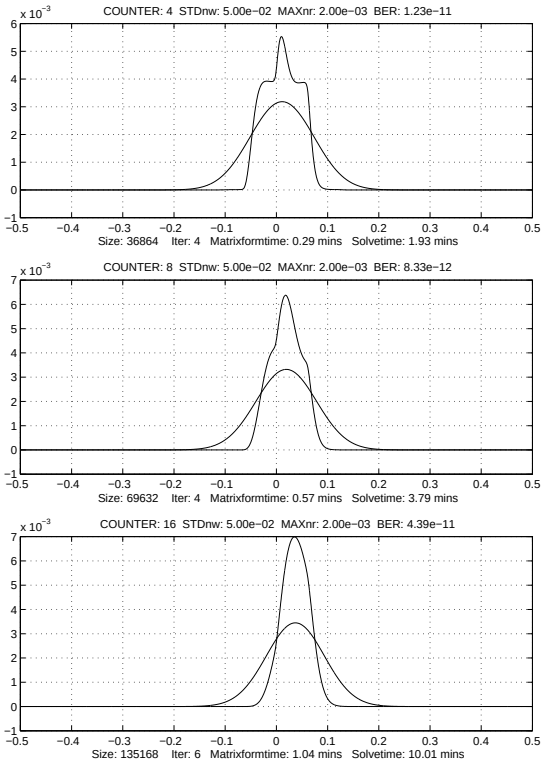


Figure 7: Effect of counter length on BER performance (multi-level algorithm with flat sparse storage)

Through the use of a specialized multi-grid method, very large systems can be solved in reasonable time on a powerful workstation. Furthermore, a novel graph based data structure for the representation of the Markov chain transition probability matrix was introduced, which requires *two orders of magnitude* less memory compared with flat sparse storage. The usefulness of the techniques introduced in the paper were illustrated through a real industrial design.

References

[1] J. Sonntag and R. Leonowich. A monolithic CMOS 10 MHz DPLL for burst-mode data retiming. In *IEEE International Solid-State Circuits Conference*, 1990.

[2] P. Larsson. A 2-1600 MHz 1.2-2.5V CMOS clock-recovery PLL with feedback phase-selection and averaging phase-interpolation for jitter reduction. In *IEEE International Solid-State Circuits Conference*, 1999.

[3] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, 1976.

[4] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[5] G. Horton and S. Leutenegger. A multi-level solution algorithm for steady-state Markov chains. *ACM Performance Evaluation Review*, 22:191–200, 1996.

[6] A. Demir and P. Feldmann. Stochastic modeling and performance evaluation for digital clock and data recovery circuits. In *DATE 2000*, 2000.

[7] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *Performance Evaluation Review*, August 1985.

[8] P. Buchholz. An adaptive aggregation/disaggregation algorithm for hierarchical Markovian models. *European Journal of Operational Research*, 116(3):85–104, 1999.

[9] M. Bozga, O. Maler. On the Representation of Probabilities over Structured Domains. In *Proc. CAV'99.* , Springer, 1999.