

Stochastic Modeling and Performance Analysis of Multimedia SoCs

Balaji Raman, Ayoub Nouri, Deepak Gangadharan, Marius Bozga, Ananda Basu, Mayur Maheshwari, Jérôme Milan, Axel Legay, Saddek Bensalem, Samarjit Chakraborty

▶ **To cite this version:**

Balaji Raman, Ayoub Nouri, Deepak Gangadharan, Marius Bozga, Ananda Basu, et al.. Stochastic Modeling and Performance Analysis of Multimedia SoCs. SAMOS XIII - Embedded Computer Systems: Architectures, Modeling, and Simulation, Jul 2013, Agios konstantinos, Samos Island, Greece. pp.145-154, 10.1109/SAMOS.2013.6621117 . hal-00878094

HAL Id: hal-00878094
https://hal.science/hal-00878094

Submitted on 29 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Stochastic Modeling and Performance Analysis of Multimedia SoCs

Balaji Raman[1], Ayoub Nouri[1], Deepak Gangadharan[2], Marius Bozga[1], Ananda Basu[1],
Mayur Maheshwari[1], Jerome Milan[3], Axel Legay[4], Saddek Bensalem[1], and Samarjit Chakraborty[5]
[1] VERIMAG, France, [2] Technical Univeristy of Denmark, [3] Ecole Polytechnique, France,
[4] INRIA Rennes, France, [5] Technical University of Munich, Germany.
E-mail: balaji.raman@imag.fr

***Abstract*—Quality of video and audio output is a design-time constraint for portable multimedia devices. Unfortunately, there is a huge cost (e.g. buffer size) incurred to deterministically guarantee good playout quality; the worst-case workload and the timing behavior can be significantly larger than the average-case due to high variability in a multimedia system. In future mobile devices, the playout buffer size is expected to increase, so, buffer dimensioning will remain as an important problem in system design. We propose a probabilistic analytical framework that enables low-cost system design and provides bounds for playing acceptable multimedia quality. We compare our approach with a framework comprising both simulation and statistical model checking, built to simulate large embedded systems in detail. Our results show significant reduction in output buffer size compared to deterministic frameworks.**

I. Introduction

As we are evolving towards next generation mobile devices, a number of emerging platforms are expected to support high-definition video and audio[1]. To support such high-end features, tomorrow's devices will require fast processors, large memories, and other resources. In fact, a sizable amount of on-chip memory is needed even for today's hand-held devices. For example, in a System-on-Chip (SoC) for processing multimedia, decoded frames are temporarily stored on-chip before display, consuming huge amount of on-chip memory. Figure 1 shows the video decoding portion of a SoC with an output buffer.

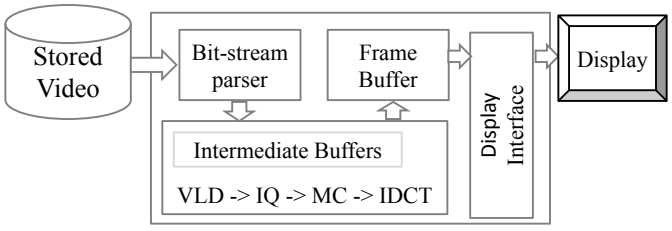


Fig. 1. Output buffer in the video accelerator of a SoC.

However, the required size of the on-chip buffer can reduce if some loss in output quality is tolerated. A standard approach in industry is to run exhaustive simulations (of a SoC model) to estimate buffer size for an acceptable output quality. The buffer size is chosen such that cost constraints of the SoC are met. This practice is time inefficient, since the process involves iteratively varying the buffer size.

To aid this architecture exploration, recent research proposes to complement simulation techniques with mathematical models, especially during the early design phase. The two main advantages of using analytical models are as follows: (1) for fast analysis in contrast to detailed simulation and (2) to gain key insights for tuning architecture and application parameters. Many of the analytical models proposed so far are faster than their simulation counterparts, but they often are incompatible to the systems modeled and are too inflexible to model key characteristic features of the application:

- Worst-case execution time modeling [16] cannot capture behavior of soft real-time systems, leading to pessimistic designs with exorbitant buffer size.
- Average-case execution analysis framework [30] cannot provide QoS guarantees and are thus hardly trustworthy.

To address the above problems we sought a mathematical framework for analyzing multimedia systems that account for the stochastic nature of the streaming application. More precisely, we need an analytical model characterizing input stream and execution of the multimedia stream as stochastic, instead of capturing event arrivals and executions with worst or average cases. The stochastic nature of the multimedia stream is due to high variability in the input and in processing these streams.

Recently, stochastic network calculus[2] based approaches have been proposed for performance analysis of multimedia systems [19], [21]. The proposal of this previous effort, however, did not use the probabilistic calculus in its entirety; the input stream objects of a multimedia stream (e.g., frames) and their execution behavior are assumed to be deterministic. Quite recently, there was another proposal to use probabilistic real-time calculus for hard real-time systems [23]. However, this effort did not rigorously explore using the calculus for any specific application domain. We believe that design of multimedia embedded platforms can benefit when stochastic

[1]Texas instrument's multimedia mobile platform includes support for 3D encoding and decoding [5]. Other companies have similar plans for bringing high-end media to mobile devices [17], [25], [18], [22].

[2]Stochastic network calculus was originally developed for performance analysis of computer networks [8].

network calculus is fully adopted for system-level design.

We experimented with the stochastic network calculus approach for analysis of a SoC architecture running video decoding applications. The architecture we modeled is an abstraction of the video decoding unit shown in Figure 1. The input video stored is fed to the input buffer in terms of stream objects such as macroblocks, frames, etc. A pipeline of functional units process the input stream. Processed items are temporarily stored in a frame buffer before sent to display. Now, we overview the performance analysis using our analytical model.

We can estimate output buffer size using our analytical framework. Previous work that estimate output buffer size using determinstc real-time calculus proceed as follows [16]. For all given video clips, Maxiaguine et al. construct upper bounds on the number of items that arrive to the input buffer and that execute in the processor. These two bounds together yield another upper bound on the number of items that arrive to the frame buffer. Thus, given a rate at which things are consumed from the output buffer, Maxiaguine et al. estimate the maximum buffer size required.

We too compute the deterministic upper bounds on arrival, execution, and output, however, only for a sub-set of given video clips; the remaining video clips can violate the deterministic bounds. For example, the number of items arriving to the output buffer over a time interval for a certain clip can be larger than the determinsistic output bound. Assume that the stream objects arrival and execution has stochastic. So, what is the probability that the number of stream objects that arrive to the input buffer over a time interval are larger than the deterministic arrival bound? First, we estimate the maximum probability of violating the deterministic bounds for the arrival and execution. Second, similar to the deterministic case, using the probabilistic bounds for the arrival and execution, we obtain the maximum probability with which the output determinstic bound can be violated. Given the constant consumption rate, we can estimate the probabilistic distribution of the buffer size.

We also estimate the initial playout delay: the delay after which the video starts to display. Previous work has shown that initial delay can be tuned to reduce the output buffer size. The trade-off is the loss in video quality. The context of the existing work, however, is in a determinstic setting, we extend it to our stochastic framework. Thus, we estimate the initial delay such that we satisfy the given QoS constraints and report the corresponding buffer size required.

We estimate the probability that the given QoS property is true using a simulation framework. A model of the multimedia SoC is constructed with the BIP framework. The design parameters are chosen such that there is no loss in video quality when any clip from a set of representative video clips is processed. Then the stochastic behavior of the system is simulated; the arrival times and corresponding execution of stream objects are randomly chosen from input distributions. The input distributions are constructed using a set of representative video clips, which are used to estimate the deviation of the upper and lower bounds in the analytical framework.

An inbuilt statistical model checking engine guides the simulation of the SoC model in that only sufficient and necessary simulations are performed. A precise estimate of the probability that a specific QoS property is true can be estimated using the statistical model checking framework. These estimates are compared with the results obtained from the stochastic real-time calculus.

To validate and compare probabilistic bounds obtained from the mathematical framework, we modeled the multimedia SoC using a rigorous simulation framework, namely, BIP [1]. We used statistical model checking to verify if the QoS property is satisfied. This design-flow, that is, detailed simulation followed by model checking, is an idea recently proposed for verifying large heterogeneous embedded systems [2]. Thus, we show the accuracy of our results obtained from our mathematical framework in comparison to the BIP approach.

Again, the main purpose of this study is to provide ways to design low-cost systems, especially towards solutions for reducing the on-chip memory size. In this context, the framework we are proposing in this paper is an apt choice for designing resource-constrained SoC for multimedia. The following sub-sections list the contributions of this work and sketches how the paper is organized.

A. Contributions and Organization

The main contributions of our paper are as follows:

- *Stochastic characterization:* The stochastic real-time calculus we use for performance analysis captures the probabilistic arrivals and executions of stream objects (and dependencies between them). The stochastic nature inherent in the multimedia applications is exploited to design low-cost systems. We provide probabilistic guarantees on specific QoS properties.
- *Statistical Simulation:* This work is among the very few to use statistical model checking techniques for multimedia system design. Towards time-efficient performance analysis of multimedia system design, we believe this statistical framework is an important candidate.

This paper is organized as follows. In Section II, we formally characterize the multimedia application mapped to the SoC with stochastic network calculus. In Section III, we provide a detailed description of the BIP framework and the statistical model checking approach, especially, on the application of the simulation framework to our case-study. Section IV presents analysis of both approaches for verifying specific QoS properties of the multimedia system. We discuss two important aspects of the analytical framework in Section V. Section VI and Section VII contains our conclusions.

II. Stochastic Real-Time Calculus Model

A. Background

This section presents the real-time calculus framework for the system model we use in our subsequent sections. Note that this section is not intended to be an exhaustive exposition of real-time calculus for embedded systems.

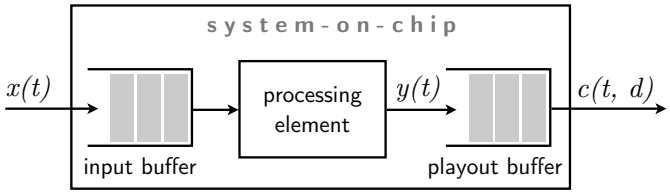


Fig. 2. Real-time calculus model

The system model used in this paper is sketched in Fig. 2. The architecture is composed of memory buffers and processing units. Fig. 2 shows a data stream x reaching the input buffer while the processed stream y is written to the playout buffer. The application mapped to the processing element is a multimedia task. However, there is no limitation on the number of applications running on the processing element. Consequently, there are no restrictions on the number of input streams fed to the processing element. Similarly, the number of architecture units in the system-on-chip can be any number of processing elements and memory units. Indeed, analysis using real-time calculus has been shown for a complex system containing shared memory, bus, and network-on-chip communication architectures. The data flow need not to be sequential in that there could be a feedback flow between the architectural components in the system-on-chip, and there could be splits and joins in the dataflow of the architecture.

The notations of the parameters involved in the description of our model are described in the following tables, divided as follows:

- Table I lists the application and architecture parameters that are given to the system designer.
- Table II records the functions that are obtained from simulations.
- The mathematical functions that are constructed from the known parameters from Table II are given in Table III.
- Similarly, mathematical functions constructed from simulation data from Table I are presented in Table IV.
- Finally, Table V and VI shows the functions and the models used in the real-time calculus framework.

The core part of the deterministic real-time calculus can be reduced to three inequalities. First, bounds on the arrival of the data stream are computed from the ϕ_l, ϕ_u functions, which gives the minimum and maximum number of bits constituting any k consecutive stream objects respectively.

$$\alpha_l(\Delta) = \phi_u^{-1}(r\Delta), \quad (1)$$

$$\alpha_u(\Delta) = \phi_l^{-1}(r\Delta), \quad (2)$$

where r is the bit-rate of the input video, and where $\alpha_l(\Delta), \alpha_u(\Delta)$ are the minimum and maximum number of items arriving over the time interval Δ respectively. The $\phi_{u,l}$ functions are computed from the $bits(k)$, which gives the number of bits per k consecutive stream objects.

The number of items arriving at the input buffer over the time interval $[0, t]$ ($x(t)$ is bounded by the previously introduced $\alpha_{u,l}$ functions:

$$\alpha_l(\Delta) \leq x(t+\Delta) - x(t) \leq \alpha_u(\Delta), \quad (3)$$

for $t, \Delta \geq 0$.

TABLE I
KNOWN APPLICATIONS AND ARCHITECTURE PARAMETERS.

	Notation and name		Description
Application	λ	Bitrate (in bits per second)	The bit rate of the input video is constant and is fed to the input buffer. This bit rate is assumed to be constant.
	c	Consumption rate (in macroblocks per second)	The output device reads items constantly from the playout buffer.
	d	Initial playout delay (in millisecond)	The delay after which the output device starts playing the video.
Architecture	b	Input buffer size (in macroblocks†)	The stream is fed to the input buffer. This is a logical buffer and physically it might be a part of memory.
	B	Playout buffer size (in macroblocks or bytes)	The processed stream is written to the playout buffer after which the consumer reads from the buffer.
	f	Processor frequency (in MHz)	The effective number of processor cycles available per unit time. The available processor cycles for the multimedia task is constant.

†*The input buffer can hold a given number of macroblocks and it is not given by a size in bytes because the application case study we use in this paper is video decoding. The compressed stream arriving at the input buffer consists of macroblocks of variable size whereas the playout buffer holds decompressed items of constant size.*

TABLE II
PARAMETERS FROM SIMULATION OF PROCESSOR AND FROM APPLICATION SOURCE CODE.

Notation	Name	Description
bits(k)†	Cumulative bits	Number of bits per k consecutive stream objects.
cycles(k)*	Cumulative cycles	Number of cycles consumed by k consecutive stream objects.

†*The number of bits for each and every macroblock is computed with the execution of the application once and the data flow between the software blocks.*

**The simulation performed to compute the cycles is not a system-level simulation. The processing element is independently simulated using a software simulation to compute the processor cycles consumed for each and every macroblock.*

TABLE III
MATHEMATICAL FUNCTIONS FROM SIMULATION DATA.

Notation	Name	Description
$\phi_l(k), \phi_u(k)$	ϕ functions	Minimum and maximum number of bits constituting any k consecutive stream objects.
$\gamma_l(k), \gamma_u(k)$	γ functions	Minimum and maximum number of cycles needed to process any k consecutive stream objects.

TABLE IV
MATHEMATICAL FUNCTIONS FROM INPUT PARAMETERS.

Notation	Name	Description
β-cycles(k)	Processor cycles over $[0,t]$	Product of time and frequency, β-cycles$(k) = tf$.
$c(t)$	Consumption function	Minimum and maximum number of cycles needed to process any k consecutive stream objects.

TABLE V
MATHEMATICAL FUNCTIONS AND INPUTS OF ARRIVAL, SERVICE AND OUTPUT.

Notation	Name	Description
$x(t)$	Input function	Number of items arriving in the input buffer over the time interval $[0,t]$.
$y(t)$	Output function	Number of items arriving in the output buffer over the time interval $[0,t]$.

TABLE VI
MATHEMATICAL MODEL.

Notation	Name	Description
$\alpha_l(\Delta), \alpha_u(\Delta)$	Arrival curves	Minimum and maximum number of items arriving over the time interval Δ. For example Δ is defined as $[0,t]$.
$\beta_l(\Delta), \beta_u(\Delta)$	Service curves	Minimum and maximum number of items guaranteed to be processed over the time interval Δ. For example Δ is defined as $[0,t]$.

The second core inequality is based on the service curve (β), which guarantees the number of processor cycles dedicated to that particular multimedia task over the time interval Δ. Given the processor frequency (f), we first compute the processor cycles available over time interval $[0,t]$, that is, β-cycles$(t) = tf$. Second, given the number of cycles consumed by k consecutive stream objects ($cycles(k)$), we compute the minimum number of cycles needed to process any k consecutive stream objects (γ_l). Third, we compute $beta$ as follows: $\beta_u(\Delta) = \gamma_l^{-1}(\beta\text{-cycles}(\Delta))$. Now, we present our second core inequality. Let $y(t)$ be the number of items arriving in the output buffer over the time interval $[0,t]$. Then, it can be shown that:

$$y(t) \leq (\alpha_u \otimes \beta)(t), \tag{4}$$

where $\otimes$ is the min-plus convolution operator defined as:

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t-s) + g(s)\}. \tag{5}$$

Finally, if the playout buffer never underflows, we have the last core inequality:

$$y(t) \leq c(t,d), \quad \forall t \geq 0. \tag{6}$$

, where $c(t,d)$ is the items consumed after the initial playout delay (d) by the display device over the time interval $[0,t]$.

B. Problem Formulation

In this section we propose a probabilistic framework for designing multimedia SoCs. We start with the construction of the synthetic traces.

Let the given set of input video clips be partitioned into two sets, S_A and S_B, based on the designer's requirement that all clips in S_B must be processed with no loss in video quality. The deterministic upper bound on the arrival and the output, introduced in the previous section, are constructed using clips in S_B. Now, we discuss how to synthetically generate clips in case we do not have a set S_B.

The information we have about the clips in set S_A are the number of bits and number of cycles corresponding to each macroblock. For certain macroblocks, we modify the number of bits and cycles, assuming we are given lower bounds[3] on these parameters. Any number of bits lower than the actual bound in the input traces are replaced with a value of the lower bound. Thus we obtain synthetic traces forming clips in set S_B. Now we explain how we estimate stochastic bounds using the actual clips (i.e. clips from set S_A) and synthetic traces (or if available clips from set S_B.

The upper and lower bounds on the arrival given from the formulae in the previous subsection are calculated for the synthetic traces. That is, from the modified inverse ϕ function, we compute $\alpha_l(\Delta)$ and $\alpha_u(\Delta)$. This leads to the definition of the stochastic arrival curve. Since by definition, $x(t)$ gives the number of items arriving over the time interval $[0,t]$, we obtain:

$$P\left(\sup_{0 \leq \Delta \leq t} (x(t+\Delta) - x(t) - \alpha_u(\Delta)) > a\right) \leq f(a), \tag{7}$$

for all $0 \leq \Delta \leq t$ and for all $a \geq 0$.

In the above description, the arrival curve bounds are checked with the actual number of items arriving at the input buffer over any time interval. The decreasing function $f(a)$ is an upper bound on the probability.

In practice, the designer is typically given a set of input clips to design the SoC with given display constraints. So, in our problem setting, the designer can construct an upper and lower bound using synthetic traces (which are obtained from actual traces) and use the actual traces to construct the bounding functions.

We define the stochastic service curve as follows. As formulated in the previous subsection, the output function from the processing element is guaranteed to be larger than the min-plus convolution of the arrival and service curves:

$$y(t) \leq (\alpha_u \otimes \beta_u)(t), \quad \forall t \geq 0. \tag{8}$$

For a stochastic service, the above inequality is restated as follows:

$$P\left(y(t) - (\alpha_u \otimes \beta_u)(t) > a\right) \leq g(a), \quad \forall t \geq 0, \tag{9}$$

where the stochastic bounding function g, ideally obtained from psychovisual models, is related to the acceptable loss

[3]In the discussion section, we present a technique to generate synthetic traces without this lower bound

of playback quality. Note that in defining a stochastic service curve we are also including the definition of the arrival curve. Thus dependencies are handled for both the arrival and the service curve.

The output from the processing element is bounded by the arrival functions α_l and α_u. The results from the stochastic network calculus provides bounds on the output curve [8]. If there is a stochastic arrival curve as defined in Eq. 7 and a stochastic service curve as defined in Eq. 9, the output curve is defined as follows:

$$P\left(y(t) - (\alpha_u \oslash \beta_u)(t) > a\right) \leq (f \otimes g)(a), \quad \forall t \geq 0. \quad (10)$$

Write $h(a) = (f \otimes g)(a)$, and the min-plus deconvolution operator is defined as follows:

$$(f \oslash g)(t) = \sup_{u \geq 0} \left\{ f(t+u) - g(u) \right\}, \quad (11)$$

Assume the output arrival curve is an arrival process to the playout buffer. The probability distribution at the playout buffer can be computed using the bounding function h.

$$P\left(B(t) > a\right) \leq h(a - (\alpha_* \oslash ct))(0), \ \forall a, t \geq 0. \quad (12)$$

In the above equation α_* is the output arrival function given by $(\alpha_u \oslash \beta_u)(t)$ which is denoted in Eq. 10.

In the results sections, we will show how to specify the QoS property by using Eq. 12.

III. Analysis with Statistical Model Checking

This section describes the model of the multimedia SoC built using the BIP framework as well as the performance analysis results obtained on it using statistical model checking.

A. Multimedia SoC models in BIP

BIP – *Behavior, Interaction, Priority* – [1] is a component based framework encompassing rigorous model based design. It allows building hierarchically structured systems (or composite components) from atomic components characterized by their behavior and their interface. Components are composed by layered application of interactions and priorities. Interactions express synchronization constraints between actions of the composed components while priorities are used to filter amongst possible interactions and to steer system evolution e.g. to express scheduling policies.

In BIP, *atomic components* are finite-state automata extended with variables and ports. Variables are used to store local data. Ports are action names, and may be associated with variables. They are used for interaction with other components. States denote control locations at which the components await for interaction. A transition is a step, labeled by a port, from a control location to another. It has associated a guard and an action, that are respectively a Boolean condition and a computation defined on local variables. In particular, probabilistic computation can be used to model random variables and hence stochastic behavior. In BIP, data and their transformations are written in C/C++. *Composite components* are defined by assembling atomic or composite using *connectors*. Connectors relate ports from different sub-components and represent sets of *interactions*, that are, non-empty sets of ports that have to be jointly executed. For every such interaction, the connector provides the guard and the data transfer, that are, respectively, an enabling condition and an exchange of data across the ports involved in the interaction. Finally, *priorities* provide a mean to coordinate the execution of interactions within a BIP system. They are used to specify scheduling or similar arbitration policies between simultaneously enabled interactions.

Figure 3 shows the BIP model of the SoC running the video decoding application. The functional units of the SoC are modeled as atomic components respectively, Generator, Processor, and Player. These functional components communicate explicitly through buffers, namely, Input Buffer and Playout Buffer, represented in BIP as atomic components as well. The lines represent connectors, namely write-push, pop-read are used to transfer macroblocks objects between a functional component and a buffer component. The tick connector synchronizes all the functional components, and is used to model explicitly the progress of the absolute (global) time. Now we describe the behavior of each of the functional components with more details.

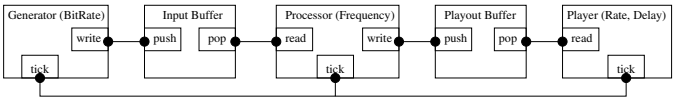


Fig. 3. BIP model of the SoC running a video decoding application.

Generator: This component models the generation of a stream of macroblocks. The stream is generated randomly and stored in the input buffer. The number of bits (the size) of every macroblock determines the arrival time of the macroblock to the input buffer. This number of bits for each macroblock (for a specific frame type) is randomly picked from a specific distribution, shown in Figure 4 and taken from [9], [10]. Moreover, the type of frames is chosen based on a Group of Pictures (GOP) pattern (IBBPBBPBBPBB)[4].

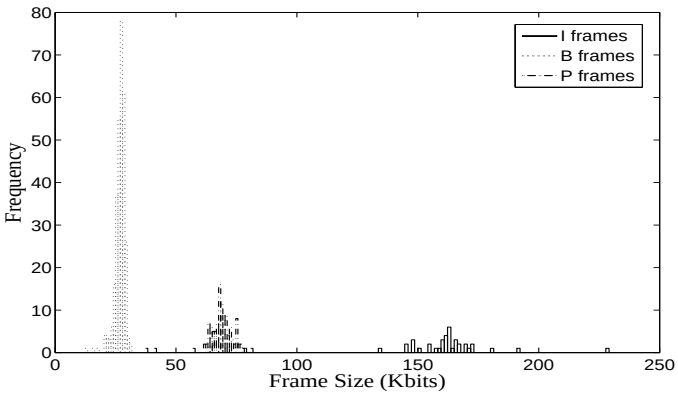


Fig. 4. Frequency distributions of I,B, and P frames in the video. I frames are smaller in size but larger in number than the B and P frames.

[4]The first GOP, IPBBPBBPBB,is different than the consecutive GOPs.

Processor: This component models the decoding of macroblocks, sequentially after reading them from the input buffer. The detailed behavior of the component is shown in Figure 5. The component has two states, IDLE and PROCESS and three ports read, write, and tick. In the IDLE state, the process is either waiting to read from the input buffer or waiting to write to the playout buffer. When there is a macroblock available, the process transits to the PROCESS state and remains there for the time required to process/decode the macroblock.

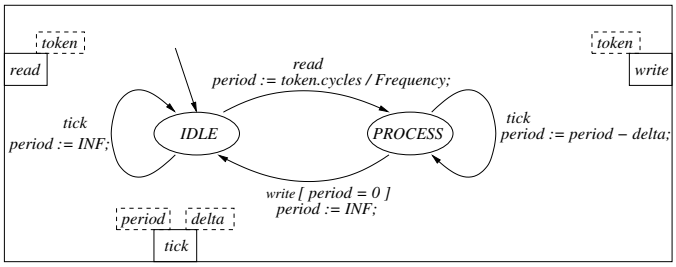


Fig. 5. *Processor* model as a BIP component. The processor unit executes a macro-block at a pre-specified speed. The execution cycles corresponding to each macro-block is randomly selected from a distribution or taken from a file in a sequential manner.

Player: The Player component models the consumption of the stream of decoded macroblocks. After an initial playout delay, the Player starts reading the macroblocks from the playout buffer at a constant rate. A buffer underflow occurs whenever the requested number of macroblocks is not available in the buffer. In this case, the request is postponed for the next iteration and the underflow is accumulated. For example, if the current buffer underflow is 2, then, at the next request, the Player seeks 3 macroblocks. If the buffer is still empty, the underflow became 3. Else, if the playout buffer has (at least) 3 items, then all three items are read at once the Player and the buffer underflow is reset to 0, etc.

B. Statistical Model Checking

Statistical Model Checking (SMC) has been proposed as an alternative to classical Model Checking techniques. It aims to avoid exhaustive state space exploration. The idea is to do verification on a sub-part of the state space (a sample) and then, using statistics, extrapolate the result to the whole system with some confidence. Concretely, given a stochastic system S and a property ϕ, *statistical model checking* refers to a series of simulation-based techniques that can be used to answer two questions : (1) **qualitative :** is the probability for S to satisfy ϕ greater or equal to a certain threshold θ ? and (2) **quantitative :** what is the probability for S to satisfy ϕ ?

The main approaches [29], [24] proposed to answer the qualitative question are based on *hypothesis testing*. Let p be the probability that $S \models \phi$. To determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A statistical-based solution (based on a sample) does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds is less or equal to α (respectively, β). Since it is impossible to ensure a low probability for both types of errors simultaneously, a solution is to use an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$.

Several hypothesis testing algorithms exist in the literature. Younes[29] proposed a logarithmic based algorithm that given p_0, p_1, α and β implements the *Sequential Ratio Testing Procedure* (SPRT) (see [27] for details). In [4], [12] Peyronnet et al. propose an estimation procedure (PESTIMATION) to compute the probability p for S to satisfy ϕ.

We applied statistical model checking to evaluate QoS properties on the BIP model of the multimedia SoC presented above. As explained earlier, this model is fully stochastic. We focus on a qualitative QoS property related to the playout buffer, that is, the buffer underflow within a second never exceeds two consecutive frames.

In order to evaluate this property on the traces of the model, we use the additional Observer component shown in Figure 6. This component runs in parallel with the systems and reacts to events (interactions) relevant to the satisfaction of the property. The component has three states: OK, PARTIAL, and FAIL. The FAIL state denotes the failure of the property, namely, the underflow of two consecutive frames within a second. If there is a loss of a single frame the observer moves from state OK to PARTIAL. Later, if there is an additional frame loss the Observer reaches the FAIL state. If no loss happens, the component moves back to the OK state.

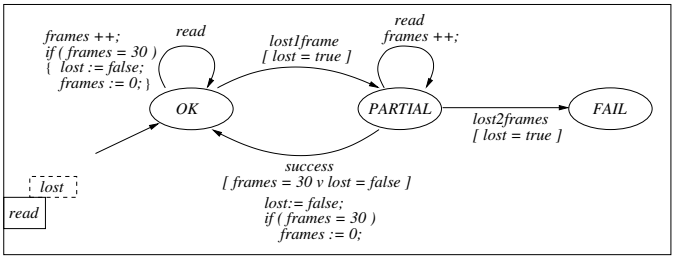


Fig. 6. *Observer* model as a BIP component. The observer models the QoS property to be verified. The variable *frame* counts the number of frames to check if two consecutive loss occurs within a second (i.e. within 30 frames). The read port of the Observer is synchronized with the read port of the Player. A variable *lost* is associated with the read port records a frame loss.

We used the SMC-BIP tool , The SMC-BIP tool takes as input (1) a stochastic system modeled in BIP, (2) a probabilistic bounded LTL property (refer [4]) or, alternatively, an observer component encoding the evaluation of the property, and (3) a series of confidence parameters needed for the statistical test (see [29]). Then, it proceeds according to the following steps:

- *Step* 1*:* an executable model of the SoC is created,
- *Step* 2*:* simulations traces of the system are iteratively generated,
- *Step* 3*:* the observer component checks the property on each input trace,
- *Step* 4*:* then gives a partial verdict for each of the trace checked,

- *Step* 5*:* steps 2, 3, and 4 are repeated until the SMC engine takes a decision to stop the simulation. The SMC engine implements the statistical algorithms introduced in the previous section.

Our tool is guaranteed to terminate its execution, when it has decided an answer for the input property to be verified on the input system based on the input statistical parameters. The guarantee for termination relies on the mathematical theory of the statistical model checking. [**?**]

IV. Results

This section sketches QoS probabilities estimated from the two analysis approaches presented in previous sections. We also tabulate the buffer size savings obtained using synthetic traces.

We implemented the analytical framework described in Section II in MATLAB. The experiments were conducted for a low-bit rate and low resolution clips ($352 * 240$) obtained from an open source [26]. The bit-rate of the input video is 1.5 Mbits per second and the frame output rate is 30fps. We used an MPEG2 implementation optimized for speed [14]. The MPEG2 source was annotated to get the number of bits corresponding to each compressed macroblock. The execution cycles for each macroblock is obtained from the software simulator SimpleScalar. Recapitulate that the number of bits and execution cycles per macroblock are inputs to the analytical framework. We chose the video files `cact.m2v`, `mobile.m2v`, and `tennis.m2v` for our experiments.

To construct the synthetic clips we set the lower bound for bits (e.g to 60) and the lower bound for execution cycles (e.g to 9000). Then from the actual trace containing the number of bits and execution cycles per macroblocks, synthetic traces are obtained; any value below the lower bound is modified to the lower bound. Figures 7, 9, and 11 show the probability that the buffer underflow is greater than two consecutive frames over any time interval (this refers to Equation 12 in Section II). Figures 7, 9, and 11 also show the probability estimates from the BIP framework. Following are the observations:

- Increase in playout delay decreases the amount of buffer underflow, so, probability that the buffer underflow is more than two consecutive frames decreases.
- The estimates from stochastic real-time calculus upper bound the statistical model checking results as the analytical framework captures the worst-case behavior.
- The delay values at which the statistical model checking starts to state that the property is true is not same for the analytical framework. The analysis using the stochastic real-time calculus should be used to determine a small set or range of delay values. Later, to precisely verify the property detailed simulation should be carried out.
- For each probability estimation, the number of traces statistical model checking simulated ranged from 44 to 1345. For each trace, the method took around 6 to 8 seconds to verify the property. The probability of error for the probabilistic estimates from the statistical model checking is bounded by 0.01.

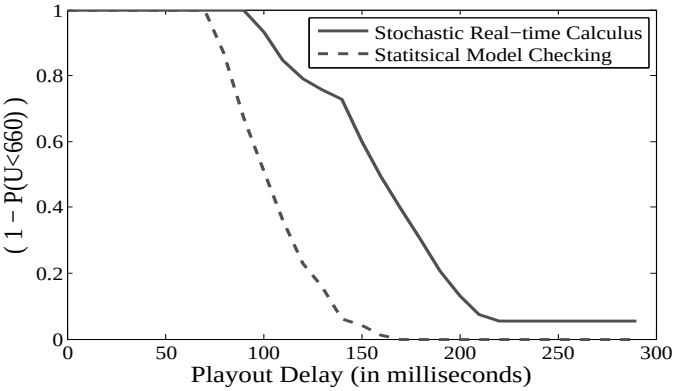


Fig. 7. Probabilistic bounds for `cact.m2v`

Figures 8, 10, and 12 plots the results of the buffer sizes for various playout delay values, and corresponding probabilistic bounds. These results correspond to simulation and statistical model checking. We observe that buffer size reduces substantially even for a small decrease of probabilistic value. For instance, there could be a buffer size reduction of 40% for a increase in the value of the probabilistic bound from 0 to 0.2 (Figures 10). In fact the buffer savings can be larger if we compare the buffer size required for no underflow and the memory required for the QoS property to be always true.

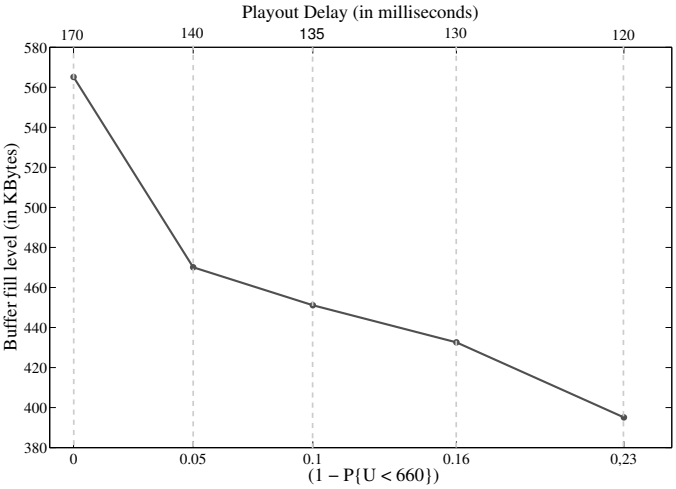


Fig. 8. Playout buffer fill level for `cact.m2v`.

V. Discussion

The results in the previous section confirms our hypothesis that, for tolerable loss in video playout, output buffer size could be significantly reduced compared to the buffer size required for playing lossless video. In this section, we focus on two requirements for our hypothesis to be used in practice: (1) extraction of synthetic clips from benchmark video clips, and (2) modeling loss of macroblocks as deadline misses (instead of dropping stream objects). We speculate the combined strengths of both the approaches when used together in a system design flow in the technical report.

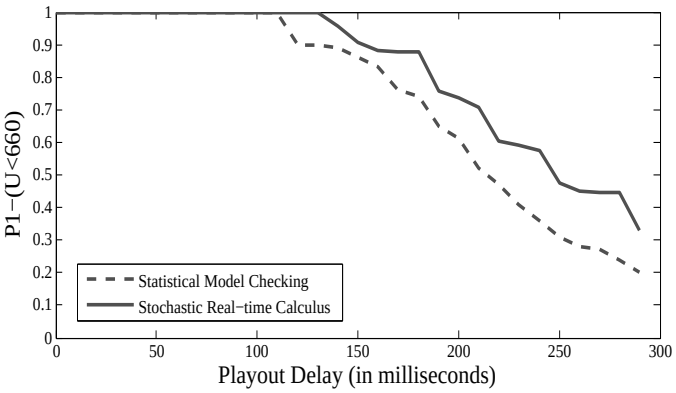


Fig. 9. Probabilistic bounds for `mobile.m2v`

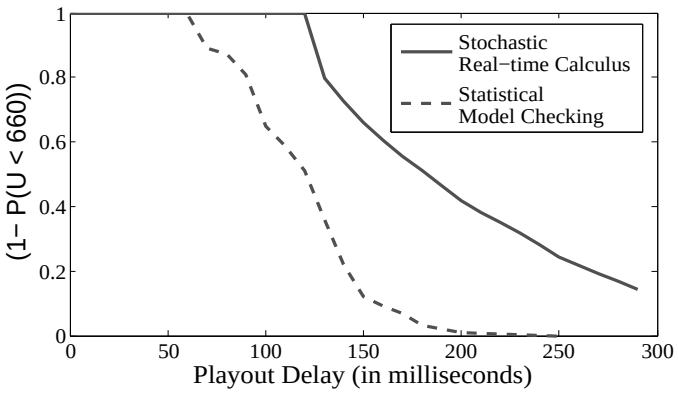


Fig. 11. Probabilistic bounds for `tennis.m2v`

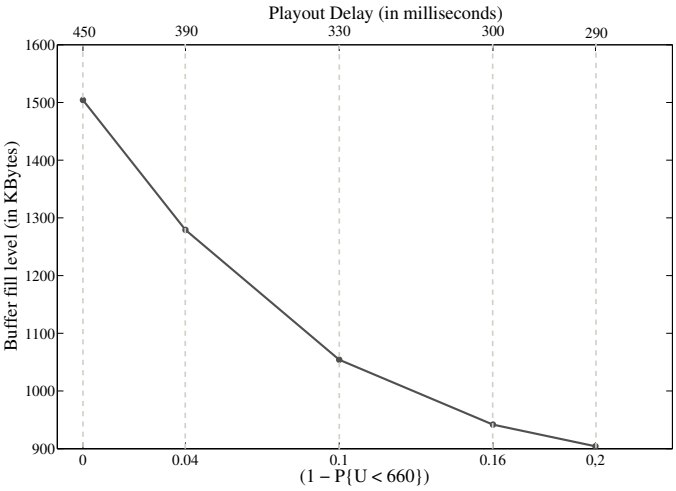


Fig. 10. Playout buffer fill level for `mobile.m2v`.

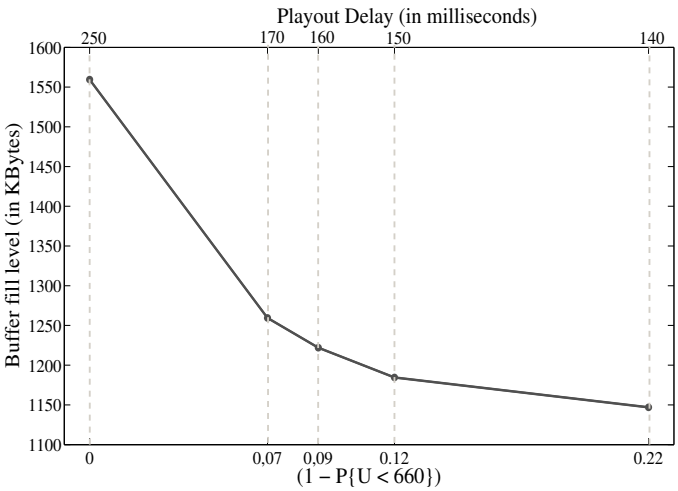


Fig. 12. Playout buffer fill level for `tennis.m2v`.

A. Synthetic Clip Generation

The primary motivation to generate synthetic clips is that a system designer would use the synthetic clips instead of actual video clips when deciding architecture parameters such as buffer size. The question is then how to generate synthetic clips from given actual video clips.

Currently, we choose at which point in frequency distribution of the input data (bits and cycles) we need to cut and reshape frequency distribution to generate the synthetic data. We choose this cut-off point based on two objectives: (1) to get significant reduction in playout buffer size, and (2) the loss in video is tolerable when the actual video clip is running in the SoC, which is designed using synthetic clips.

The approach we use to generate synthetic data can be time-inefficient as it involves iteration in making a choice to cut the distribution and analyze using our framework to check if our objectives are met. Instead of this trial and error technique, we are currently researching on a more sound technique to generate synthetic clips (for example, Yanhong et al., [15] use error percentage and eliminate some tail data from the distribution). To further this thought, notice in the analytical framework the need for synthetic clips arises when we compute the stochastic bounding functions. What if we use standard tail distribution functions that could characterize multimedia data accurately?

Assuming that these tail distributions could be found (refer Jelenkovic et al., [7]), then the task of generating synthetic clips reduces to a litmus test: does the synthetic clip generated conforms to the stochastic bounding function? In other words, without performing the iterative analysis — choosing a cut-off point first for synthetic clip generation, second checking if the video loss is tolerable, and then choosing a different cut-off point — we just check if a set of synthetic clips generated conforms to the stochastic bounding function.

B. Video Loss Model

In our model, when we say video loss, we precisely mean that the macroblocks missed their deadlines; we do not model video loss as drop in macroblocks or frames, as studied in [3], where the authors present an analytical framework to study the trade-off between buffer size and video quality required for a multimedia decoder in the context of frame drops. In correspondence to display of the video, as video loss is interpreted as deadline miss, the display device awaits until all

macroblocks are ready for display. If there were no deadline miss, for example, a frame would be displayed at the right time.

The QoS property we verified using our set-up required to check if there is a loss of two consecutive frames within 1 second. This should be read in the context of our model as the display of two consecutive frames being delayed within 1 second.

VI. Related Work

In this section, we present the state-of-the-art in characterizing the stochastic behavior of multimedia applications and compare it with our approach. We will focus mainly on analytical approaches (refer previous work for survey on simulation based approaches [2], [13]).

A domain-agnostic approach is to statistically analyze the execution variance of soft-real time applications (Kumar et al., [11]). Using profiling, components of the applications that lead to variable execution times are first identified. Then programmers can identify components that affect real-time behavior of the application in the context of other components. Thus, the programmers need not resort to ad-hoc methods for tuning applications for expected real-time behavior. Our approach differs from the domain agnostic techniques in the following way: our model tightly couples the application and architecture; in what follows, we also discuss how both the stochastic real-time calculus and statistical model checking are an excellent fit for streaming applications.

On the other hand, domain-specific techniques in a probabilistic setting perform analysis at task granularity (Yaldiz et al., [28] and Iqbal et al., [6]). In stochastic real-time calculus, our model captures input, execution, and output streams of the SoC. The granularity of the stream object can be at any level: bits, macroblock, frame, and group of pictures.

Iqbal and others [6] proposed scheduling techniques for soft-real time systems. The task execution times are stochastic and the solution for scheduling is based on an online Monte Carlo method on a joint space model of all tasks. The objective of this technique is similar to ours: reducing memory and computational requirements. The complexity of this technique, however, as the authors report, does not scale well for task graphs of huge size. Yaldiz and others [28] use stochastic modeling of the applications to obtain policies for energy savings and for providing probabilities for satisfying timing constraints. Their model considers set of concurrent tasks and takes into account data dependence, precedence relations and timing constraints.

Our methodology differs from the above two discussed approaches in the following way: (1) there is a tight characterization of inputs for streaming applications using arrival curves in the stochastic real-time calculus and randomly generated clips (based on distributions) in statistical model checking, and (2) we provide probabilistic guarantees instead of average-case analysis in the analytical framework.

Liu and others [15] introduced a new concept called approximate variability characterization curves (or Approximate VCCs), to characterize the average-case behavior of multimedia workloads in a parameterized fashion. The crucial difference in comparison to our work is that Approximate VCCs belongs to a family of average-case analysis; instead of probabilistic guarantees on buffer size they bound the error due to their analysis. Also, the framework remains in a deterministic setting after the alteration of workload curves. So, it is not elegantly able to capture stochastic nature of arrivals and stochastic nature of execution times.

VII. Conclusions and Future Work

The use of synthetic traces for SoC design for multimedia applications yielded significant reduction in buffer size. The trade-off of in quality over buffer size savings were analyzed using two independent approaches. The stochastic real-time calculus and the statistical model checking techniques estimated the probability that a certain QoS property is true. The analytical framework upper bounded the estimates from the statistical model checking.

Future extensions to this work are as follows: (a) modeling communication architectures such as bus, network-on-chip, and others; (b) modeling for variable bit-rate video and variable consumption rate; (c) integrating real-time calculus and BIP framework. This paper juxtaposes the analytical framework with a model checking technique, where both the techniques observe same QoS property of a multimedia system. This experience led us to envision a joint model, which can combine strengths of both the models (see discussion section in technical report [20]).

References

[1] A. Basu, B. Bensalem, M. Bozga, J. Combaz, M. Jaber, T.-H. Nguyen, and J. Sifakis. Rigorous Component-Based System Design Using the BIP Framework. *IEEE Software*, 28(3):41–48, May 2011.

[2] A. Basu, S. Bensalem, M. Bozga, B. Caillaud, B. Delahaye, and A. Legay. Statistical Abstraction and Model-Checking of Large Heterogeneous Systems. In *Proc. of the International Joint Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE)*, pages 32–46, June 2010.

[3] D. Gangadharan, L. Phan, S. Chakraborty, R. Zimmermann, and I. Lee. Video quality driven buffer sizing via frame drops. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2011 IEEE 17th International Conference on*, volume 1, pages 319–328. IEEE, 2011.

[4] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *Proc. of the Verification, Model Checking and Abstract Interpretation (VMCAI)*, pages 73–84, January 2004.

[5] T. Instruments. OMAP 5 mobile applicattions platform: Product Bulletin, 2011.

[6] N. Iqbal and J. Henkel. SETS: Stochastic execution time scheduling for multicore systems by joint state space and Monte Carlo. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 123–130, November 2010.

[7] P. R. Jelenkovic, A. A. Lazar, and N. Semret. The Effect of Multiple Time Scales and Subexponentiality in MPEG Video Streams on Queueing Behavior. *IEEE Journal on Selected Areas in Communications*, 15(6):1052–1071, August 1997.

[8] Y. Jiang and Y. Liu. *Stochastic Network Calculus*. Springer, 2008.

[9] M. Krunz, R. Sass, and H. Hughes. Statistical characteristics and multiplexing of MPEG streams. In *Proc. of the Conference on Computer Communications (INFOCOM)*, pages 455–462, April 1995.

[10] M. Krunz and S. K. Tripathi. On the characterization of VBR MPEG streams. In *Proc. of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 192–202, June 1997.

[11] T. Kumar, R. Cledat, J. Sreeram, and S. Pande. Statistically Analyzing Execution Variance for Soft Real-Time Applications. In J. N. Amaral, editor, *Languages and Compilers for Parallel Computing*, pages 124–140. Springer-Verlag, 2008.

[12] S. Laplante, R. Lassaigne, F. Magniez, S. Peyronnet, and M. de Rougemont. Probabilistic abstraction for model checking: An approach based on property testing. *ACM Transactions on Computational Logic*, 8(4):30–39, August 2007.

[13] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: an overview. In *Proc. of the International Conference on Runtime Verification (RV)*, pages 122–135, November 2010.

[14] libmpeg2. A free MPEG2 video stream decoder. `http://libmpeg2.sourceforge.net`, 2006.

[15] Y. Liu, S. Chakraborty, and W. T. Ooi. Approximate VCCs: a new characterization of multimedia workloads for system-level MpSoC design. In *Proc. of the ACM/IEEE Annual Design Automation Conference (DAC)*, pages 248–253, June 2005.

[16] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 131–136, January 2004.

[17] NVIDIA. NVIDIA Tegra 3. `http://www.nvidia.com/object/tegra-superchip.html`, 2012.

[18] Qualcomm. Snapdragon S4. `http://www.qualcomm.fr/products/snapdragon`, 2011.

[19] B. Raman. *Application-specific workload shaping in resource-constrained media players*. PhD thesis, School of Computing, National University of Singapore, July 2010.

[20] B. Raman, A. Nouri, D. Gangadharan, M. Bozga, A. Basu, M. Maheshwari, J. Milan, A. Legay, S. Bensalem, and S. Chakraborty. A general stochastic framework for low-cost design of multimedia SoCs. Technical Report TR-2012-7, Verimag Research Report, 2012.

[21] B. Raman, G. Quintin, W. T. Ooi, D. Gangadharan, J. Milan, and S. Chakraborty. On buffering with stochastic guarantees in resource-constrained media players. In *Proc. of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 169–178, September 2011.

[22] Samsung. Exynos 4. `www.samsung.com/exynos`, 2011.

[23] L. Santinelli and L. Cucu-Grosjean. Toward probabilistic real-time calculus. *ACM SIGBED Review*, 8(1):54–61, March 2011.

[24] K. Sen, M. Viswanathan, and G. Agha. Statistical Model Checking of Black-Box Probabilistic Systems. In *Proc. of International Conference on Computer Aided Verification (CAV)*, pages 202–215, July 2004.

[25] STEricsson. NOVATHOR U9500. `http://www.stericsson.com/products/u9500-novathor.jsp`, 2011.

[26] Tektronix. MPEG Elementary Streams. `ftp://ftp.tek.com/tv/test/streams/Element/index.html`, 1996.

[27] A. Wald. Sequential Tests of Statistical Hypotheses. *Annals of Mathematical Statistics*, 16(2)(2):117–186, June 1945.

[28] S. Yaldiz, A. Demir, and S. Tasiran. Stochastic Modeling and Optimization for Energy Management in Multicore Systems: A Video Decoding Case Study. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 27(7):1264–1277, July 2008.

[29] H. L. S. Younes. *Verification and Planning for Stochastic Precess with Asynchronous Events*. PhD thesis, School of Computer Science, Carnegie Mellon University, January 2005.

[30] N. H. Zamora, X. Hu, and R. Marculescu. System-level performance/power analysis for platform-based design of multimedia applications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(1):1–29, January 2007.