SYSTEM-LEVEL MODELING AND ANALYSIS OF MULTIMEDIA-SOC PLATFORMS

YANHONG LIU

(M.Eng., Institute of Computing Technology,

Chinese Academy of Sciences)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2007

Acknowledgments

Numerous people have supported me during the development of this dissertation, and my graduate experience more generally. Mentioning a few words here cannot adequately capture all my appreciation.

I would like to show my sincerest gratitude to my advisor Dr. Samarjit Chakraborty. I thank him for his devoted guidance and constant encouragement. I think I can never stop learning from his insight into the research area, intellect and inspiration. I also benefit a lot from the fact that Dr. Samarjit Chakraborty, as a generous and kind advisor, always helps students not only on academic growth, but also on their lives.

I also thank my other advisor Dr. Wei Tsang Ooi. I thank him for his generous help and guidance at the beginning of my life at the university. I am very impressed by his academic strictness. I would like to thank him for the continuous advising, suggestions and comments on the work related to this dissertation as well.

I have been lucky to have the opportunity of working with Dr. Radu Marculescu (from CMU) and Dr. Tulika Mitra and learnt a lot from them. I want to give my special thanks to Dr. Alexander Maxiaguine (from ETH). The cooperative work with him helps me to get a quick start of the simulation platforms used.

I would also like to thank the members of my dissertation committee, Dr. Wong Weng Fai and Dr. Ee-Chien Chang, for many useful interactions, and for contributing their broad perspective in refining the ideas in this dissertation.

I would like to thank the National University of Singapore for the research scholarship that makes this study possible and the administrative staff here for their support in the various aspects of academy and life.

Of many other friends and colleagues, I want to thank Dr. Yongxin Zhu for the help on some issues of simulations. Thanks also go to Lin Ma, Balaji Raman, Huaxin Xu, Qinghua

Shen, Zhiguo Ge, NGUYEN Dang Kathy, Yun Liang, Jimin Feng, Yu Pan etc. for the help and fun.

As always, I thank my family for their love and continuous encouragement. My parents always do whatever to avoid me to distract from the study. My brother helps to take care of my parents and to manage the family matters, which I should have shared with. My sister always devotes her help to me whenever need and often encourages me to do my best. I also thank the extended family members for their support.

Last, my most tender and sincere thanks go to my wife, Lili Zhang. Thanks for her self-giving help and support in innumerable ways.

List of Publications

1. Alexander Maxiaguine, Yanhong Liu, Samarjit Chakraborty and Wei Tsang Ooi. Identifying “Representative” Workloads in Designing MpSoC Platforms for Media Processing. In *2nd Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, Stockholm, Sweden, September 2004.

2. Yanhong Liu, Alexander Maxiaguine, Samarjit Chakraborty and Wei Tsang Ooi. Processor Frequency Selection for SoC Platforms for Multimedia Applications. In *IEEE Real-Time Systems Symposium (RTSS)*, Lisbon, Portugal, December 2004. (Rank 1 Conference)

3. Yanhong Liu, Samarjit Chakraborty and Wei Tsang Ooi. Approximate VCCs: A New Characterization of Multimedia Workloads for System-level MpSoC Design. In *Proceedings of the Design Automation Conference (DAC)*, Anaheim, California, June 2005. (Rank 1 Conference, Best Paper Award Nomination)

4. Yanhong Liu, Samarjit Chakraborty, Wei Tsang Ooi, Ashish Gupta, and Subramanian Mohan. Workload Characterization and Cost-Quality Tradeoffs in MPEG-4 Decoding on Resource-Constrained Devices. In *3nd Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, New York Metropolitan area, September 2005.

5. Yanhong Liu, Samarjit Chakraborty, and Radu Marculescu. Generalized Rate Analysis for Media-Processing Platforms. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Sydney, August 2006.

6. Samarjit Chakraborty, Yanhong Liu, Nikolay Stoimenov, Lothar Thiele, and Ernesto

Wandeler. Interface-Based Rate Analysis of Embedded Systems. In *IEEE Real-Time Systems Symposium (RTSS)*, Rio de Janeiro, December 2006. (Rank 1 Conference)

Contents

Summary

Currently there is a considerable interest in designing general-purpose configurable System-on-Chip (SoC) platforms specifically targeted towards implementing multimedia applications. Determining the optimal configuration for such platforms is especially difficult due to the various kinds of variabilities arising out of multimedia processing, such as the high variability in the execution requirements of multimedia streams and the burstiness in the on-chip traffic. System-level design and analysis methods are then desired for such platforms, which take into account such variabilities.

In this thesis we propose an analytical framework that can be used in the design space exploration and performance analysis of multimedia SoC platforms. Our work includes the following contributions.

Firstly, we adopt the concept of *variability characterization curves* to characterize the worst-case behaviours of multimedia workloads. An analytical scheme is also presented to obtain such characterization curves for a large library of potential inputs to the system.

Secondly, to illustrate the utility of our framework, we present analytical approaches for two typical system design cases. In the first case, we address the problem of identifying the frequency ranges that should be supported by different processors of a platform in order to run a target multimedia workload. In the other case, we determine tight bounds on the arrival rates of different multimedia streams at a platform such that predefined quality-of-service (QoS) constraints are met.

Finally, we propose the concept of *approximate variability characterization curves* to characterize the average-case behaviours of multimedia workloads. “Average-case” analysis using this concept can be used to derive tradeoffs between resource savings and QoS constraints. In this thesis we present error analysis algorithms to bound the extent to which such QoS constraints can be satisfied.

Our proposed framework can be used to precisely model multimedia workloads and estimate various performance parameters for multimedia SoC platforms in a seamless manner. Compared to purely simulation-oriented approaches, our framework provides provable performance guarantees and involves analysis times which are significantly shorter.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Today multimedia applications run on a wide range of consumer electronic devices, ranging from set-top boxes to PDAs and mobile phones. Because of flexibility, low design costs and time-to-market advantages, very often such devices are now designed using general-purpose configurable multiprocessor System-on-Chip (MpSoC) platforms. Examples of such platforms are the Eclipse architecture template [77, 79] and the Viper SoC architecture [31] from Philips that target advanced set-top box and DTV markets, OMAP from Texas Instruments [67] and PrimeXsys from ARM [71]. Many of these platforms are typically designed to process concurrent streams of audio and video data associated with broadband multimedia services and, at the same time, perform network packet processing to support high-speed Internet access.

One of the major problems that a designer has to address while using such platforms is the issue of platform configuration. Such platforms are typically designed for a *class* of applications. Given a particular application belonging to this class, the platform is tuned (or configured) to perform optimally when running this application. Configuring a platform may involve determining the size of on-chip buffers, bus width, cache configurations, etc. and also the parameters for different schedulers and bus arbitration policies.

Determining an optimal platform configuration is typically not easy and involves several design tradeoffs and constraints imposed by the platform itself. It should be fully considered about the flexibility, cost, performance and power consumption characteristics of the designed platform. For example, lowering the power consumption may imply degraded performance, and increasing flexibility is usually associated with increased cost and low performance. Additionally, a designer may face challenges due to rapidly changing protocols and time-to-market pressure. This problem becomes even more challenging in the context of *designing* SoC platforms for multimedia devices, because of the high computational demands, real-time constraints, and low power consumption requirements of such devices and various kinds of variabilities associated with multimedia processing. Also, the underlying *design space* is quite large and purely simulation-based techniques involve prohibitively high running time. Such considerations have led to an increasing demand for analysis techniques and system-level design tools for MpSoC platforms.

Research efforts have been paid to design multimedia SoC platforms using analytical techniques. Very little work, however, has fully taken into account the characterization of multimedia workloads during the design of SoC platforms. As we have mentioned, multimedia applications exhibit high computational requirements and various kinds of data-dependent variability. For example, arrival patterns of multimedia streams at the input of the system may have a bursty nature. The number of bits to encode a frame or macroblock is highly variable. The execution demand of a task may vary a lot from activation to activation due to data-dependent program flow. Such kinds of variabilities have a great impact on the selection of configuration parameters of SoC platforms and should be fully explored. Stochastic models (e.g. queuing models) fail to accurately model these variabilities and can only provide stochastic performance guarantees. A powerful analytical framework is desired for the design of multimedia SoC platforms that can fully capture the characteristics of multimedia workloads.

1.2 Thesis Contributions

This thesis presents an analytical framework for the system-level design of SoC platform architectures for multimedia applications. The proposed framework is based on the theory of *Network Calculus* [16], which was originally developed and is still largely used in the context of analyzing communication networks. Recently, it was extended to the domain of real-time systems. It was developed to analyze the SoC architectures in the context of network processors [21, 85] and further extended to the domain of general SoC platform architectures [20]. This research follows this line of development and extends the theory to analyze the SoC platforms for multimedia applications.

Firstly, we borrow the concept of *variability characterization curves* (VCCs) [63] to characterize the worst-case characteristics of multimedia workloads, which are based on the various concepts of “curves” introduced in the theory of network calculus. Using the concept of VCCs, we propose a methodology of identifying ”representative“ workloads from a large library of multimedia streams that can potentially run on the platform, the amount of which may be too huge to analyze all these streams. The VCCs measured for these set of selected streams are then used to represent the workloads imposed on the platform.

Secondly, based on the accurate model of the multimedia workloads (i.e. VCCs), we propose system-level analytical solutions for two typical cases of SoC platform design: on-chip processor frequency selection and rate analysis. In the first case, our analytical approaches can guide a system designer in identifying the frequency ranges that should be supported by the different processors of a platform architecture. In the latter case, we address the problem of determining tight bounds on the rates at which different multimedia streams can be fed into a platform architecture. We believe that under our proposed framework, effective analytical solutions can also be developed to determine other configuration parameters for SoC platforms.

Finally, we propose a novel concept of *approximate variability characterization curves* (or approximate VCCs) to characterize the “average-case” behavior of multimedia workloads. The concept is defined in a parameterized fashion, which denotes the amount of the worst-case scenarios that is discarded. Analysis algorithms are also developed to quantitatively account for the performance degradation and the associated resource savings corresponding to different values of the parameter.

The proposed analytical framework provides powerful and effective analytical approaches for the SoC platform design in the context of multimedia applications. It should be helpful in the design space exploration of such platforms and to greatly reduce the design cycle. It should help a system designer to achieve the various kinds of tradeoffs in the platform design, by considering multimedia workload characterization and the platform design in a uniform way. The proposed framework captures fully the characteristics of multimedia workloads imposed on the platforms, such as various kinds of variability arising from multimedia processing. It should be able to analyze various performance metrics for the targeted platforms and to determine various configuration parameters for a platform, given the applications to be supported by the platform. On the other hand, it should be able to determine the characteristics that the applications should satisfy given the platform whose parameters are known. The proposed scheme of average-case characterization of multimedia workloads may achieve great resource savings when applied in the design of SoC platforms, due to the high variability presented in multimedia processing.

1.3 Organization of the Thesis

The organization of the thesis is as follows. In the next chapter, we introduce the background and review the related literature. In Chapter 3, we conduct the overview of fundamental models, the concept of VCCs, basic methodologies and experimental setup that we have used. In Chapter 4, we present our methodology of identifying “representative” workloads, from which VCCs are measured. It is followed by the analytical approaches proposed for two typical system design problems: on-chip processor frequency selection and rate analysis, which are presented in Chapters 5 and 6 respectively. The concept of approximate VCCs is then introduced in Chapter 7 and algorithms are presented to quantify the performance degradation and resource savings for two system design cases. Finally, we summarize the thesis and talk about the future work.

Chapter 2

Background and Related Work

2.1 MpSoC Platforms

The ever increasing complexity of SoCs and the pressures of short time-to-market and low cost requirements for SoC designs, has led to new design paradigms such as platform-based design [47]. This paradigm encourages the extensive reuse of common architectural components that can be shared among a variety of applications as well as can support the future evolutions of applications, in order to reduce the overwhelming cost of chip design and manufacturing. Based on this idea, general-purpose configurable SoC platforms use complex on-chip networks to integrate multiple intellectual property (IP) blocks or cores from some libraries (such as the IBM Blue Logic Core Library [43]) (or a third-party vendor) on a single chip. Example of the IP blocks or cores that might be included in such a platform are configurable processors, parameterized caches, specialized memory hierarchies, flexible bus architectures, programmable logic and parameterized coprocessors etc. These IP blocks or cores are already predesigned and verified and hence the designer need not take care of the specific implementation of these individual components, while only concentrating on the overall system.

In a general-purpose configurable SoC platform, the interconnected components and/or architecture parameters can be customized towards the requirements of the target application (or applications) that might run on this platform. Examples of such generic platforms are PrimeXsys from ARM [71] and AcurX from Plamchip [3]. These platforms are targeted towards a wide range of applications starting from DVD players and set-top boxes, to network routers and network security processors.

Although application-specific hardware (e.g., ASICs and custom SoCs) are customized for a particular application domain and have the benefits of high performance capacity, low power consumption, and small size, they are usually associated with heavy engineering costs, slow time-to-market and inability to make provision for post-deployment upgrades (hence reduced time-in-market). On the other end, solutions purely based on general-purpose processors have the advantage of high degree of flexibility, enabling upgrades, and shorter design cycles, but often fall short of performance and power requirements. General-purpose configurable platforms, when used in a naive manner, still show a significant difference in the performance and power utilization characteristics, compared to more specialized solutions.

To bridge this gap, techniques are proposed to customize general-purpose configurable platforms for specific applications. Such application-specific platforms are customized for a particular application domain, but still support sufficient flexibility to allow them to be configured for specific products belonging to that domain. An example of such a platform is OMAP from Texas Instruments [67], which allow multimedia capabilities to be included in 2.5G and 3G wireless handsets and PDAs. The Eclipse architecture template [77] and the Viper SoC architecture [31], from Philips, are also examples of such application-specific platforms which target advanced set-top box and DTV markets.

2.2 Y-chart Scheme of Designing SoC Platforms

To get the optimal configuration of a complex SoC platform for target applications, the design space should be effectively explored, by taking fully into account both the application and architecture aspects of the platform under study. A common approach to follow in the design of SoC platforms is the Y-chart scheme [33, 48], as shown in Figure 2.1. This scheme requires to make a clear distinction between application and architecture to allow more effective exploration of alternative solutions, which is encouraged by the system design paradigm of *orthogonalization of concerns* [47]. Firstly, the designer characterizes the

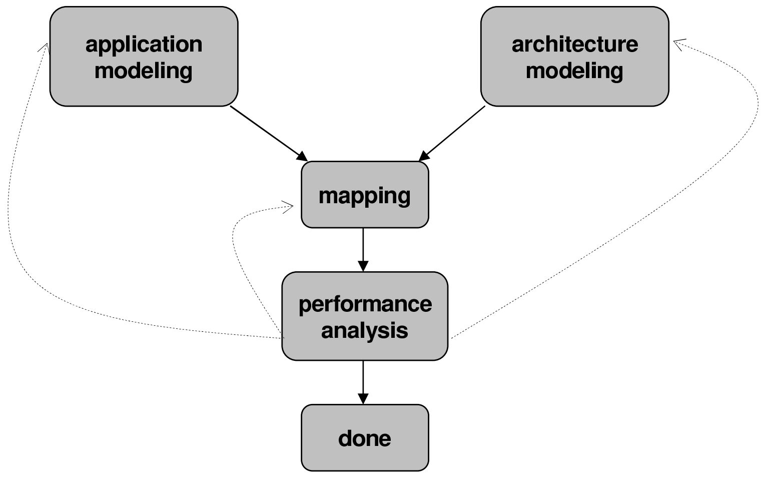


Figure 2.1: Y-chart scheme.

target application (applications), makes some initial calculations and proposes a candidate architecture. Then the application is partitioned and explicitly mapped onto the different architectural components. Next, performance analysis is conducted to quantitatively evaluate the application-architecture combination. According to the resulting performance numbers, the designer may decide to go ahead with the chosen architecture, or try to get better performance numbers by reconfiguring the architecture, restructuring the application or modifying the mapping of the application. This process is reiterated until satisfactory performance figures are achieved.

In Figure 2.1, both the application and the architecture are modeled separately. The application model is used to represent the application's functional behavior, which is often called *model of computation*. Model of computation is a mathematical model that specifies the semantics of computation and of concurrency for the application. The architecture model captures performance constraints of architecture resources, by defining architectural components that represent processors or coprocessors, memories, buffers, buses, and so on. An application model is independent from the specific architectural characteristics and hence a single application model can be used for evaluating different architecture models.

To explore the design space of complex SoC platforms, it is required that the performance analysis of the platform architecture is done at multiple abstraction levels for target applications. This makes it possible to control the speed, required modeling effort and attainable accuracy of the performance evaluations. Higher-level abstraction models are used to efficiently explore the large design space in the early design stages. More detailed models are applied at later stages to allow focused architectural exploration. Hence the models of the application and architecture should also be made at various levels of abstraction respectively to enable the stepwise refinement approach in the design space exploration. In this thesis, we are concerned with the modeling and performance analysis of multimedia SoC platforms at system-level.

2.2.1 Models of Computation

System-level models of computation typically describe the functional behaviors of an application as a hierarchical collection of tasks that are communicating with each other by means of events carried by channels. Based on the specification of the behaviors, the communication method, the implementation and validation mechanisms, and how the interconnected tasks are composed into a single one, the most important models of computation that have been proposed to date can be classified into being based on three basic models [56]: Discrete Event, Finite State Machines (FSMs) and Data Flow.

Discrete Event Model: In discrete event model, tasks communicate through multiple-writer and single-reader channels that carry globally ordered and time-tagged events. Task behavior is usually specified by a sequential language. As a task receives input events, it is executed and produces output events with the same or a larger time tag.

Finite State Machines: In finite state machines, task behavior is specified by a finite labeled transition system which is composed of states, transitions and actions. A state stores information that reflects the input changes from the system start to the present moment. The state executes the action (description of an activity) that is incurred when the required conditions (for example, entering/exiting the state, input conditions, certain transition) are satisfied. A transition indicates a state change, which is enabled only when a condition is fulfilled.

Data Flow Model: Data flow model is a special case of *Kahn Process Network* (KPN) computational model [45]. In a data flow process model, tasks communicate through one-way

FIFO channels. Each channel has unbounded capacity and carries a sequence (a stream) of data object. Each data object is written into the channel exactly once and read from the channel exactly once. Writes to channels are non-blocking, but reads are blocking (the read stalls when the input channel is empty). A task in data flow model is specified by a mapping from one or more input streams to one or more output streams.

2.2.2 Models of Architecture

The architecture is modeled as a set of interconnected modules and components along with their associated software to implement the functions imposed by applications. A module or component in the architecture model is defined with specified interfaces and explicit context dependency. The architecture is desired to be modeled in multiple abstraction levels. When the level of abstraction is closer to the final implementation, it is more effective in reducing cost and design cycles by reusing designs. Minimal variations in specification, however, may result in very different implementations. The models with higher-level abstraction can be more easily shared among different specifications and only a minimal amount of work is needed to achieve final implementation. Having multiple levels of abstraction, however, is important, since the lower levels may change due to the advances in technology, while the higher levels stand stable across product versions.

2.2.3 Performance Analysis

The application model is mapped onto the architecture model after both of these models are obtained, which is then followed by performance analysis of the application-architecture combination. The most common techniques for performance evaluation applied in industrial practice are simulation-based (e.g. VCC [88] and Seamless [80]). However, simulation possesses several disadvantages: it involves extensive running time, which fall behind the tight time-to-market demands today; it is also extremely difficult to find simulation patterns that lead to worst-case situations; it is hard to identify corner cases by simulation.

A great amount of research efforts have been put on presenting analytical techniques for performance analysis of SoC platforms as simulation-based methods fall short. Formal analysis guarantees full performance corner-case coverage and bounds for critical performance parameters, based on well-defined models.

Most of the formal analysis techniques are proposed for individual architectural components and a general framework for analyzing system-level designs is not offered, especially in the presence of heterogeneity. Few exceptions consider special cases of more complex architectures, for example, analysis of response times for static-priority process scheduling combined with a TDMA bus protocol [70]. Recently, an event stream interface model is introduced [76, 73, 74] and functions are provided for event model transformations. Based on identifying architectural components for which appropriate analysis methods already exist in the literature, a unified framework is presented to couple different local analysis techniques into a global compositional description of the complex system-level properties. These works have been extended [44], where standard event models are extracted from realistic systems that exhibit complex task dependencies such as multi-rate data dependencies, data rate intervals and multiple activating inputs. It is shown [58] that advanced performance analysis techniques can take into account *system contexts*, i.e. correlations between successive computation or communication requests as well as correlated load distribution, to yield tighter analysis bounds.

2.3 SoC Design for Multimedia Applications

Various methods and tools have been developed for SoC design, examples of which are Ptolemy [1], Milan [64], Metropolis [10], Mesh [13], Koski [46], etc. Due to the proliferation of consumer electronics products that support media processing, attentions have also been paid to design SoC platforms for multimedia applications. In the following, we introduce two directly related work. The first [68] is the project of *Architectures and Methods for Embedded Media Systems* (Artemis). The other is from Philips during the design of Eclipse architecture templates for media processing SoCs [78, 79, 86].

Application modeling: Artemis and Eclipse model multimedia applications using the KPN computational model. KPNs fit nicely with multimedia processing application domain, where application is structured by a directed graph with each node representing a task and each edge representing a data channel. Each data channel is a FIFO buffer, with one producer and one or more consumers. Tasks are executed concurrently and exchange information solely through the unidirectional data channels. The functional behavior of the KPN model, which is observed as the sequence of data items that communicate through channels, is independent of the order in which the tasks are executed. This deterministic property means that the same input always results in the same application output and the application behavior is independent of architecture models. Hence an application's performance metrics and resource constraints can be analyzed in isolation from the architecture.

Architecture modeling: Artemis aims to develop an architecture modeling and simulation environment for the efficient design space exploration of heterogeneous embedded-systems architectures at multiple abstraction levels.

In Artemis, the underlying architecture model does not model functional behavior, which has been caught by the application model. The architecture model is constructed from generic building blocks provided by a library, which contains performance models for various platform components such as processing cores, communication buses and different memory types. At a high abstraction level, various processing cores such as a programmable processor, reconfigurable component or dedicated hardware unit are abstracted as a processing-core model which functions as a *black-box*. To model the execution of an application event on a processing core, the architecture simulator assigns parameterizable latencies to the input events and thus simulates the timing behavior of the specific architectural implementation. The communication component within the architecture model (e.g. buses, memories), which the communicating Kahn channel is mapped onto, will account for the latencies associated with the data transfers.

Eclipse defines a heterogenous architecture template for designing high performance streaming-processing SoCs. This heterogenous architecture consists of fully programmable processor cores and various sophisticated hardwired function modules (coprocessors) optimized for high performance with minimum power consumption and silicon area.

Eclipse aims to present an architecture template that is flexible, scalable and cost-effective. The configuration flexibility of programmable cores is combined with high performance of hardwired modules. It achieves scalability by avoiding centralized control in the system. It allows hardwired modules to operate in parallel and independently, and can also run multiple applications concurrently. By introducing such high levels of parallelism and multi-tasking, cost-effectiveness is achieved.

Performance analysis. Artemis applies trace-driven cosimulation technique to achieve an interface that includes the mapping specification between application models and architecture models. Each executed task produces a trace of events that represents the application workload that this task imposes on the architecture. The trace events correctly reflect data-dependent functional behavior and refer to the computation and communication operations an application task performs. Hence the architecture models, driven by the traces, can simulate the performance consequences of the application events and then evaluate the architecture's performance.

Eclipse models the architecture as a flexible, cycle-accurate simulator. It obtains the performance measurements such as buffer filling, coprocessor utilization and data access latency at the application level (i.e. for each task and stream) through application simulation and tuning for particular architectural instance.

Artemis and Eclipse rely on simulation to measure the performance metrics. Simulation-based approaches, however, are known to suffer from the disadvantages of high running time, incomplete coverage and failure to identify corner cases, which are even severe in the context of designing multimedia systems.

Efforts have been put on presenting analytical solutions for performance analysis of multimedia SoC platforms. Mathematical algorithms have been presented [69] to explore the design space of system buses, the usage of which is believed to affect greatly performances and power consumption of the system. These algorithms are used to optimize the system bus usage by finding pareto-optimal solutions (supporting the target applications at the minimum cost in the sense of die area and energy consumption).

A formal technique for system-level power/performance analysis is presented [66], based on a proposed model called *Stochastic Automata Networks* (SANs). A process graph is used to model the application of interest and is translated to a network of automata, which is then used to generate the underlying Markov chain. The steady-state behavior of the SAN model is solved and performance measures are then derived. The technique, however, is purely probability-based and does not give any type of performance guarantees.

2.4 Characterization of Multimedia Workloads

A large amount of work has been conducted to model the video traffic in the context of network communications. A first model of *variable bit rate* video traffic models a video source as a first-order autoregressive process with marginal *probability distribution function* and an exponential autocorrelation function [57]. Later, a new methodology called *transform-expand-sample* is proposed to generate the number of bits in a frame following an arbitrary distribution and to model the frame correlation structure [55]. Lazar et al. [53] models the distribution and autocorrelation of a source bit stream accurately at the scene, the frame and the slice level.

The frame-size distribution for the three types of frames (i.e. I, P, and B) is also studied [81, 37, 40]. For example, a comprehensive characterization of MPEG video streams that captures the bit rate variations at multiple time scales is presented [50]. The sizes of different types of frames are modeled and intermixed as a complete model according to a given *group of pictures* pattern. The impact of scene changes on the long-term bit rate variations is also incorporated, in addition to modeling the marginal distribution and autocorrelation structure.

The above work concentrates on modeling the video traffic (i.e. the bit rate variations), but does not consider the variation in the execution time of multimedia streams.

Some previous work has been presented to predict the execution time of multimedia processing applications in order to employ real-time scheduling for efficiently implementing quality-of-service guarantees. Worst-case execution times (WCETs) of the MPEG-2 video decoding process are estimated [17] by integrating the WCET analysis into the decoder and taking into account of the actual input data. By considering frame type and size, a linear model of MPEG decoding is presented [11] to predict the actual decoding time for a frame.

Research has also been done on modeling the traffic and analyzing the execution time variability for multimedia applications in the context of computer systems design. The variability in the frame-level execution time on general-purpose architectures is analyzed for several multimedia applications [42]. It is concluded that execution time variability is mostly resulted from the application algorithm and the media input, and architectural features only contribute little to the variability in the execution time.

A recent work [87] addresses the modeling of on-chip traffic for the design of platforms for embedded multimedia appliances. It introduces that a fundamental property of self-similarity is exhibited by the bursty traffic between on-chip modules in typical MPEG-2 video applications. It quantifies the degree of self-similarity using the Hurst parameter and finds the optimal buffer-length distribution. In this work, a technique is also proposed to synthetically generating traces having statistical properties similar to real video clips and to speed up buffer simulations.

The above studies have mainly focused on modeling the video traffic and/or the execution time. They have not studied the design issues of the computer systems comprehensively and applied fully these modeling techniques to the design practice.

2.5 Network Calculus Theory

Network calculus is originally proposed as a theory of deterministic queuing systems for analyzing delay and backlog in a communication network, where the traffic and the service are characterized as envelope functions. This theory has been pioneered in the early 1990s for providing worst-case performance bounds for packet networks [28]. It is later developed to be placed in the *min-plus algebra* formulation [22, 15, 4], where the concept of *service* *curves* is used to express service guarantees to a flow. A comprehensive understanding of this theory can be referred to referred to the following textbooks [23, 16].

Recently, network calculus has been extended to analyze SoC architectures in the context of network processors [21, 85]. Analytical frameworks based on this theory are developed to explore the design space of network processor architectures in the early design stages. After a relatively small set of potential architectures are identified through analytical approaches, simulation techniques are used to get more accurate performance measures in the later design stages.

Network calculus theory is further extended [20] to the domain of general SoC platform architectures. It extends and generalizes the standard event models used in previous work [73, 76], as well as presents a framework for analyzing various system properties like timing analysis, on-chip memory demand and resource loads of heterogenous platform-based architectures.

The concept of *workload curves* is proposed [60] to characterize the variable execution demands of tasks, which provides tighter best-/worst-case bounds on the execution times of tasks than traditional WCET analysis mechanisms. This concept is generalized [63] to characterize (give best-/worst-case bounds on) the various kinds of variability arising from multimedia processing on an MpSoC platform, the result of which is a new abstraction called VCCs. This concept of VCCs is used to identify how the buffer requirements change with different scheduling mechanisms implemented on the processors, and to achieve the tradeoffs between savings on on-chip buffer sizes and scheduling overheads through analytical methods.

Our work in this thesis follows this line of development and concentrates on proposing a framework for system-level design and analysis of SoC platforms for multimedia applications. We will study the modeling techniques and effective analytical solutions for the design space exploration of such platforms. In the next chapter, we will introduce the fundamental concepts, models and techniques that are used in this thesis.

Chapter 3

Fundamental Models and Techniques

3.1 Models of Application and Architecture

Our models of multimedia application and architecture follows the traditional modeling techniques that have been extensively used in the literature [68, 78, 79, 86]. We model the multimedia application using the KPN computational model. Since we concentrate on the system-level study of the SoC platforms, we model the MpSoC platform architecture at higher abstract level. The KPN model representing a multimedia application is partitioned and mapped onto an abstract architecture model, as shown in Figure 3.1.

In this thesis, we consider the following system-level view of multimedia stream processing on an MpSoC platform. Here we discuss the processing of one stream, which can be easily extended to the case that multiple streams are processed. The platform architecture consists of multiple processing elements (PEs) onto which different parts of an application are mapped. An input multimedia stream enters a PE, gets processed by the task(s) implemented on this PE, and the processed stream enters another PE for further processing. At the input of each PE is a buffer (a FIFO channel of fixed capacity) used to store the incoming stream to be processed. Finally, the fully processed stream is written into a *play-out buffer* which is read by some *real-time client* (RTC) such as an audio or a video output device. For the sake of generality, we consider any multimedia stream to be made up of a sequence of *stream objects*. A stream object might be a bit belonging to a compressed bitstream representing a coded video clip, or a macroblock, or a video frame, or an audio sample—depending on where in the architecture the stream exists.

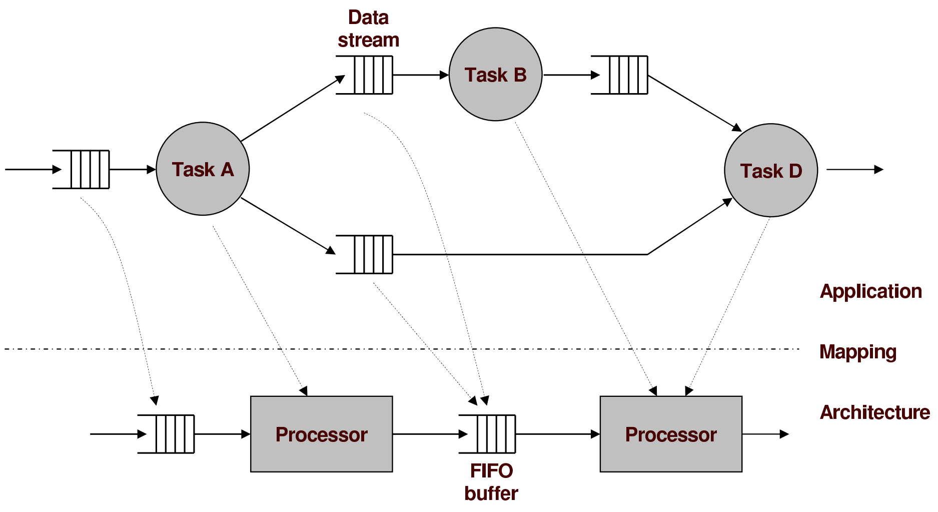


Figure 3.1: Illustration of the mapping of a multimedia application modeled as a KPN onto an MpSoC platform architecture modeled at abstract level.

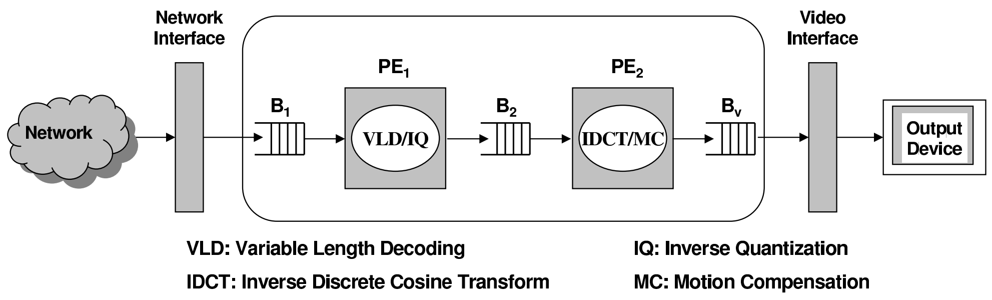


Figure 3.2: An MpSoC platform onto which an MPEG-2 decoder application is partitioned and mapped.

As an example, Figure 3.2 shows an architecture with two PEs (PE_1 and PE_2), implementing an MPEG-2 decoder application. The *variable length decoding* (VLD) and *inverse quantization* (IQ) tasks have been mapped onto PE_1, and the *inverse discrete cosine transform* (IDCT) and *motion compensation* (MC) tasks onto PE_2. A video stream, after being downloaded over a network, enters buffer B_1. PE_1 reads from B_1 and writes the resulting partially decoded macroblocks into buffer B_2. PE_2 reads from B_2 and writes the fully decoded macroblocks into the playout buffer B_v. The video output device reads from B_v at a pre-specified rate.

3.2 Multimedia Workload Characterization

To design MpSoC platform architectures for multimedia processing, the first task is to characterize the workloads imposed on the platforms by the target multimedia applications. Clearly, workload characterization should be based on *key properties* that are important in a particular design context. Usually these are properties that have a strong impact on the performance of the architecture being designed. For instance, in microarchitectural design such properties would be instruction mix, branch prediction accuracy and cache miss rates [32]. In this thesis, we hypothesize that *on the system level* the performance of multimedia MpSoC architectures is largely influenced by various kinds of *data-dependent variability* associated with the processing of multimedia data streams. This hypothesis rests on the observation that such variability is the major source of the burstiness of on-chip traffic in such multimedia MpSoC platforms [87]. The burstiness of the on-chip traffic necessitates the insertion of additional buffers between architectural entities processing the multimedia streams, and the deployment of sophisticated scheduling policies across the platform. Both of these inevitably translate into increased design costs and power consumption [42]. Therefore, it is certainly meaningful to characterize multimedia workloads with respect to their variability properties.

What are the sources of variability that are usually associated with the processing of multimedia streams on such MpSoC platforms? Firstly, arrival patterns of multimedia streams at the input of the system may have a bursty nature, i.e. stream objects may arrive on the system's input in highly irregular intervals. A typical example of this is a multimedia device receiving streams from a congested network. Secondly, each activation of a task may consume and produce a variable number of stream objects from the associated streams. For example, each activation of the VLD task in Figure 3.2 consumes a variable number of bits from the network interface, although, it always produces one macroblock at its output. Thirdly, the execution demand of a task may vary from activation to activation due to data-dependent program flow. Both the tasks in our running example of the MPEG-2 decoder—VLD and IDCT—possesses this property. Finally, stream objects belonging to the same stream may require different amounts of memory to store them in the communication channels. Again, in the example architecture shown in Figure 3.2, we note that the partially decoded macroblocks stored in buffer B_1, depending on their type, may or may not include motion vectors.

All these types of variability must be carefully considered and characterized during the workload design process. The concept of VCCs is a generic model that allows us to quantitatively capture the variability found in multimedia streams. In the following we describe this concept and give several examples of VCCs.

Variability characterization curves: VCCs are used to quantify best-/worst-case characteristics of *sequences*. These can be sequences of consecutive stream objects belonging to a stream, sequences of consecutive executions of a task implemented on a PE while processing a stream, or sequences of consecutive time intervals of some specified length. A VCC $\mathcal{V}$ is composed of a tuple $(\mathcal{V}^l(k), \mathcal{V}^u(k))$. Both these functions take an integer k as the input parameter, which represents the *length* of a sequence. Function $\mathcal{V}^l(k)$ then returns a *lower bound* on some property that holds for *all* subsequences of length k within some larger sequence. Similarly, $\mathcal{V}^u(k)$ returns the corresponding *upper bound* that holds for *all* subsequences of length k within the larger sequence. Let the function P be a *measure* of some property over a sequence $1, 2, \ldots$. If $P(n)$ denotes the measure of this property for the first n items of the sequence (i.e. $0, \ldots, n$), then we have $\mathcal{V}^l(k) \leq P(i+k) - P(i) \leq \mathcal{V}^u(k)$ for all $i \geq 0$ and $k \geq 1$. By default, $P(0)$ is assumed to be equal to 0. As examples, let us now consider the following different realizations of a VCC.

Workload curve $\gamma = (\gamma^l, \gamma^u)$: The VCC γ is used to characterize the variability in the execution requirements of a sequence of stream objects to be processed by a PE. In this case, given a sequence of stream objects, $P(n)$ denotes the total number of processor cycles required to process the first n stream objects. Hence, $\gamma^l(k)$ and $\gamma^u(k)$ denote the minimum and the maximum number of processor cycles that might be required by *any* k consecutive stream objects within the given sequence. Let us see an example as illustrated in Figure 3.3, $\gamma^l(4)$ ($\gamma^u(4)$) denotes the minimum (maximum) number of processor cycles required by

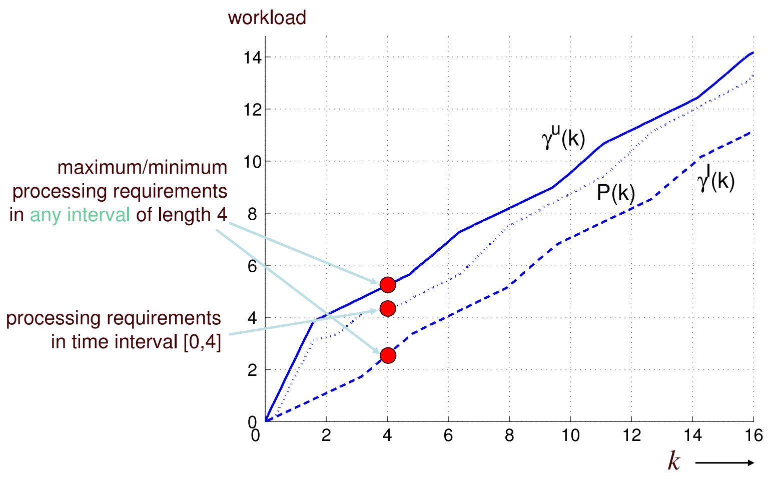


Figure 3.3: Illustration of workload curve γ.

any 4 consecutive stream objects within the given sequence, which records the minimum (maximum) value of $P(i+4) - P(i)$ for all $i \geq 0$. Hence, $P(4)$, which denotes the number of cycles required by the first 4 stream objects, is lower and upper bounded by $\gamma^l(4)$ and $\gamma^u(4)$ respectively.

Let $e_{\min}$ and $e_{\max}$ be the minimum and the maximum number of processor cycles required by any single stream object belonging to a sequence. For any reasonably large value of k, $\gamma^l(k)$ is clearly greater than $k \times e_{\min}$. Further, the difference between them increases with increasing values of k. Similarly, $\gamma^u(k)$ is clearly smaller than $k \times e_{\max}$. Hence, the VCC γ is more expressive compared to simple best- or worst-case characterizations commonly used in the real-time systems domain.

It is also meaningful to construct a *pseudo-inverse* of a VCC $\mathcal{V}$, which we denote as $\mathcal{V}^{-1}$. In the case of a workload curve, $\gamma^{l^{-1}}(e) = \min_{k \geq 0}\{k \mid \gamma^l(k) \geq e\}$ and $\gamma^{u^{-1}}(e) = \max_{k \geq 0}\{k \mid \gamma^u(k) \leq e\}$. Hence, $\gamma^{l^{-1}}(e)$ denotes the maximum number of stream objects that may be processed using e processor cycles. $\gamma^{u^{-1}}(e)$ denotes the minimum number of stream objects that are guaranteed to be processed using e processor cycles.

Arrival curve $\alpha = (\alpha^l, \alpha^u)$: This VCC is used to characterize the burstiness in the arrival pattern of stream objects. Given a trace of the arrival times of a sequence of stream objects at buffer b (e.g. the partially processed macroblocks being written into the buffer B_2 in Figure 3.2), $\alpha^l(\Delta)$ and $\alpha^u(\Delta)$ denote the minimum and the maximum number of stream objects that arrive within *any* time interval of length Δ. Given a PE that is processing

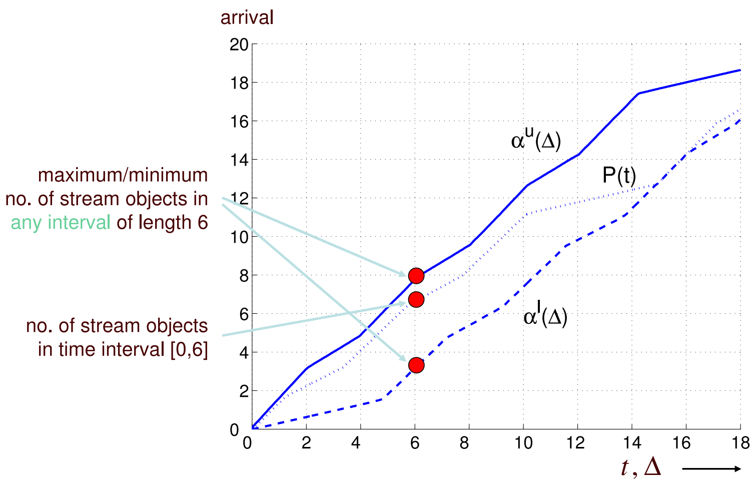


Figure 3.4: Illustration of arrival curve α.

a single stream, (α_x^l, α_x^u) are used to represent the incoming stream, (α_y^l, α_y^u) represent the processed stream and (α_c^l, α_c^u) represent the bounds on the rate at which the stream is consumed from the playout buffer. We will often refer to (α_c^l, α_c^u) as the *consumption bounds*. As illustrated in Figure 3.4, $\alpha^l(6)$ and $\alpha^u(6)$ respectively record the minimum and maximum number of stream objects that may arrive at buffer b over any time interval of length 6. Therefore, $\alpha^l(6)$ and $\alpha^u(6)$ show lower and upper bounds on the number of stream objects over any time interval of length 6 (e.g. $[0, 6]$).

Let us see one more example, let $\alpha_x^l(10) = \alpha_x^u(10) = 5$, which essentially means that within any time interval of length 10, at least and at most 5 stream objects can arrive at buffer b. Hence, the average arrival rate is one stream object in every two time units. Now suppose that we are also given that $\alpha_x^u(2) = 4$, which means that within a time interval of length 2 there might be a burst of at most 4 stream objects. Following this specification, if 4 stream objects arrive at b during the time interval $[0, 2]$, then over the time interval $(2, 10]$ at most 1 stream object can arrive. Hence, although the "long-term" arrival rate of the stream is 0.5 stream objects per unit time, there might be occasional bursts. The arrival curves α^l and α^u allow for the precise characterization of such bursts.

Service curve $\beta = (\beta^l, \beta^u)$: Due to the variability in the execution requirements of stream objects, the number of stream objects that can potentially be processed within any specified time interval varies (even when the processor runs at a constant frequency). We will use

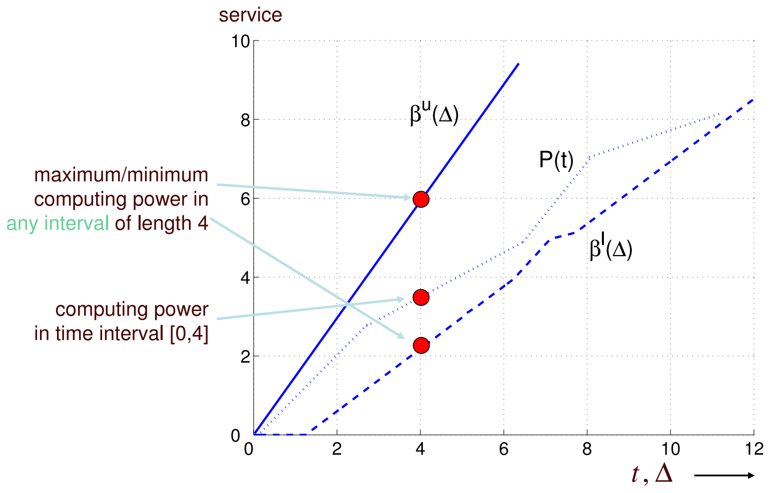


Figure 3.5: Illustration of service curve β.

$\beta^l(\Delta)$ and $\beta^u(\Delta)$ to denote the minimum and the maximum number of stream objects that can be processed (or served) by a processor within *any* time interval of length Δ. The curves β^l and β^u may also be derived from a trace of execution requirements of stream objects and the clock frequency with which the processor is being run. Figure 3.5 shows an example for service curves. The number of stream objects that can be served within any time interval of length 4 is lower and upper bounded by $\beta^l(4)$ and $\beta^u(4)$ respectively.

Note that this specification of *service* is stream dependent. It is also possible to specify the service offered by a processor in a stream-independent manner. Towards this, let $\sigma^l(\Delta)$ and $\sigma^u(\Delta)$ denote the minimum and the maximum number of processor cycles available within any time interval of length Δ. It is then easy to see that $\beta^l(\Delta) = \gamma^{u-1}(\sigma^l(\Delta))$ where γ^u is the workload curve associated with the stream (which was described above).

Consumption and production curves $\kappa = (\kappa^l, \kappa^u)$ and $\pi = (\pi^l, \pi^u)$: Let an input stream be processed by a task T. Each activation of T consumes a variable number of stream objects belonging to the input stream, and results in the production of a variable number of output stream objects, possibly of a different type. This variability in the consumption and production rates of T can be quantified using two VCCs κ and π, which we refer to as the consumption and the production curves respectively.

$\kappa^l(k)$ takes an integer k as an argument and returns the minimum number of activations of T that will be required to completely process any k consecutive stream objects.

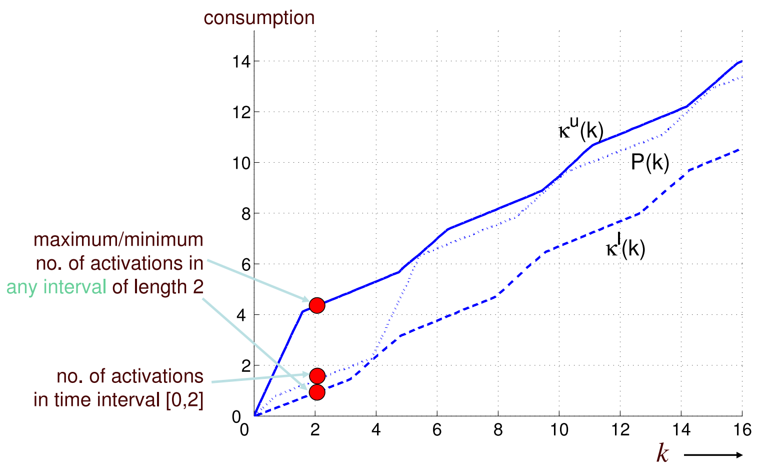


Figure 3.6: Illustration of consumption curve κ.

Similarly, $\kappa^u(k)$ returns the maximum number of activations of T that might be required to process any k consecutive stream objects. Let us see an example. As shown in Figure 3.2, the bit stream at buffer B_1 is processed by PE_1. Each activation of the VLD/IQ task processes one macroblock from buffer B_1. As illustrated in Figure 3.6, $\kappa^l(k)$ ($\kappa^u(k)$) returns the minimum (maximum) number of activations of the VLD/IQ task (i.e. number of macroblocks) that is required to process any k consecutive bits from buffer B_1.

On the other hand, we define $\pi^l(k)$ to be the minimum number of stream objects guaranteed to be produced due to any k consecutive activations of T. $\pi^u(k)$ is the maximum number of stream objects that can be produced due to any k consecutive activations of T. Therefore, k consecutive stream objects at the input of T will result in at least $\pi^l(\kappa^l(k))$ and at most $\pi^u(\kappa^u(k))$ stream objects at its output. As an example, the production curves $\pi^l(k)$ and $\pi^u(k)$ for PE_1 shown in Figure 3.2, are straight lines with slopes that correspond to the constant-rate production of one macroblock per task activation.

3.3 Performance Analysis

Given the MpSoC platform architecture that multimedia applications are mapped onto, the workloads imposed on the architecture are firstly characterized and represented by VCCs. We then evaluate the performance of this architecture and design/configure the architectural parameters, by taking into account the cost, application and architectural constraints etc.

Typical design constraints for a multimedia MpSoC platform architecture that we have modeled (e.g. the one shown in Figure 3.2) are (i) the playout buffers should not underflow, and (ii) none of the buffers should overflow. The constraint on the playout buffer underflow is to ascertain that stream objects can be read out by the audio/video output devices at the specified playback rate, and hence the output quality is guaranteed. The constraints on buffer overflow are motivated by the fact that typically on-chip PEs use static voltage and task scheduling policies. This is because using blocking write/read mechanisms efficiently to prevent buffer overflows/underflows either require a multithreaded processor architecture or substantial run-time operating system support for context switching.

We present an analytical framework for the performance analysis and design space exploration of multimedia MpSoC platform architectures. In contrast to simulation-based approaches, which usually follow a trial-and-error approach and is very time-consuming, our proposed framework can help a system designer to explore the design space in a very short time and to systematically tune a platform architecture. Our framework is based on the network calculus theory and extends this theory by developing new algorithms and models. In the following, we introduce some notation and a technical result that will be used in later chapters.

Notation. Throughout this thesis, all functions f are assumed to be wide-sense increasing, meaning that $f(x_1) \leq f(x_2)$ for $x_1 \leq x_2$ and $f(x) = 0$ for $x \leq 0$. For any two functions f and g, the *min-plus convolution* of f and g is denoted by

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

and the *min-plus deconvolution* of f and g is denoted by

$$(f \oslash g)(t) = \sup_{u \geq 0} \{f(t+u) - g(u)\}$$

We will use $f \wedge g$ to denote the infimum or minimum (if it exists) of f and g, and $f \vee g$ to denote the supremum or maximum (if it exists) of f and g.

Lemma 1 *For any functions f, g and h, $g \otimes h \geq f$ if and only if $h \geq f \oslash g$.*

This lemma follows from the definitions of the min-plus convolution and deconvolution operations and shows the relation between them.

3.4 Experimental Setup

We have conducted experiments to illustrate and validate our analytical framework. Since MPEG-2 streams have a complex nature and a rich set of characteristics [50], they represented an interesting target for our experiments. We studied the MpSoC platform architectures with an MPEG-2 decoder application mapped onto, one of which is that shown in Figure 3.2.

Our experimental setup consisted of the SimpleScalar instruction set simulator, a system simulator and an MPEG-2 decoder program. The MPEG-2 decoder program was used as an executable for the simulator and as a means to obtain traces of bit allocation to macroblocks.

The instruction set simulator was used to obtain traces of execution times for the VLD/IQ and IDCT/MC tasks of the MPEG-2 decoding algorithm. All the tasks processed the data stream at the macroblock granularity. The *sim-profile* configuration of the SimpleScalar simulator and the PISA instruction set were used to model on-chip processors of the architecture. Although this configuration does not model advanced microarchitectural features of the processor, it allows fast simulation and was therefore the most suitable choice. This choice is also justified by the fact that advanced features in the microarchitecture of a general-purpose processors do not have significant impact on the variability of multimedia workloads [42].

The system simulator consisted of a SystemC transaction-level model of the architecture. We used it to measure backlogs in the buffers resulting from the execution of the MPEG-2 decoder application on the platform.

Chapter 4

Characterizing Multimedia Workloads: Obtaining VCCs

To conduct performance analysis for given MpSoC platforms, we firstly have to characterize the workloads for the application that can be run on the platforms. In this chapter, we obtain the various kinds of characterization curves (VCCs) that represent the workloads imposed on the architecture. Due to the large volume of potential inputs to the system, it is impossible to cover every input stream and it is also time consuming. Hence good "representative" multimedia inputs should be selected and VCCs are then measured for these set of representative workloads. The measured VCCs are used to represent the workloads imposed by the large library of potential inputs, in the sense of best-/worst-case bounds.

Selecting a good "representative" input set is of course not a new concern—benchmark selection or workload design is a well recognized problem in the domain of microprocessor design. However, the main issues in that domain are microarchitecture-centric, where a designer is mostly concerned with program characteristics like instruction mix, data and instruction cache miss rates and branch prediction accuracy. On the other hand, the concerns in the case of system-level design of platform architectures are very different and these are not suitably reflected in a benchmark suite designed for microarchitecture evaluation.

In this chapter we attempt to address this issue of workload design in the specific context of system-level design of platform architectures for multimedia processing. Although simulation-oriented design and evaluation are widespread in the domain of system-level design, to the best of our knowledge the issue of methodically selecting representative inputs for architecture evaluation has not received any attention so far. Most of the work reported in the Embedded Systems literature, on novel system models or simulation schemes, shirk off this problem and leave the responsibility of choosing a representative input or stimuli to the architecture on the system designer (see, for example, [52]).

There are many reasons why this problem is interesting in the specific case of multimedia processing on MpSoC platforms. Firstly, many multimedia applications exhibit a large degree of data-dependent variability that complicates the problem of choosing a representative input set. Secondly, in contrast to general-purpose architectures, MpSoC platforms that are optimized for stream processing have heterogeneous parallel architectures. This fact further complicates the problem. Thirdly, multimedia processing is in general computationally intensive, which makes workload selection an important problem.

Arbitrarily selecting inputs to form the "representative" input set is certainly not a good idea. The goal of "representative" workload design should be to select inputs that represent *corner cases* for the target architecture, i.e. those inputs which impose worst- and best-case loads on different parts of the architecture. However, determining what constitutes a "corner case" is not a trivial undertaking due to the complex nature of most multimedia workloads. Attempts towards using some qualitative technique to judge the properties of multimedia streams based on their content (for example, by simply viewing video clips to be processed by the architecture and classifying them based on experience or intuition) might easily fail. Hence, a quantitative methodology is necessary, using which it should be possible to objectively assess and compare the properties of different multimedia streams. Based on such a comparison, a small *representative* subset of a large library of samples can then be chosen.

In this chapter we propose such a methodology to classify multimedia streams, which can be used to identify a small representative set meant for architecture evaluation. Towards this, we first hypothesize that all the characteristics of multimedia streams that influence the performance of an MpSoC platform architecture, are related to their "variability". Later in this chapter, we will show our preliminary experimental results that validate this hypothesis. Such variability manifests itself as data-dependent fluctuations of (i) execution time requirements and (ii) input-output rates associated with multimedia processing tasks. These fluctuations stem from the fact that execution time requirements of the tasks and the amount of data consumed and produced by the tasks depend on the properties of particular audio/video samples being processed. Now, given a library of multimedia streams, we classify two streams as *similar* if both of them exhibit the same kind of variability with respect to execution time requirements and input/output rates as mentioned above. Therefore, given a set of video streams which are *similar*, it would be sufficient to simulate an architecture with only one video stream from this set, as all the other streams would “stress” the architecture in the same way. The variability associated with a stream, with respect to an architecture, is quantitatively characterized with VCCs [63] which is summarized in Section 3.2.

We would like to point out here that the kinds of variabilities that should be considered in a multimedia stream for an effective classification would depend on the architecture and the application at hand, and a detailed discussion of this is beyond the scope of this chapter. Our work in this chapter also shows that the properties of multimedia streams, that should be considered for representative workload identification in the context of performance evaluation of SoC platforms, can be expressed in the form of VCCs.

Related Work: The construction of representative workloads for performance evaluation of computer systems has always been an area of active research since early 70s (see [83] and references therein). Since then the term *workload* has been widely understood as a mix of programs (or jobs, or applications) for which the performance of a computer system was evaluated. Domain-specific collections of such programs, called *benchmarks*, have been designed and widely used as a standard means to evaluate and compare computer architectures. Examples of these are MediaBench [54] and the Berkeley multimedia workload [82]. Design of such representative workloads was mainly concentrated on proper selection of the *programs* to be included in the workload. The selection of corresponding input data sets was limited to the definition of their size (e.g. sampling rate, resolution etc.) The dependency of program behavior on the values of the input data sets did not receive enough consideration in the process of forming such representative workloads.

Recently Eeckhout et al. [32] have shown that the *workload design space* may be very complex and therefore should be systematically explored during the construction of representative workloads. Their workload design space consists of *program-input pairs* that capture both, the variety of programs as well as various input data sets to those programs. They use techniques such as principle component analysis and cluster analysis to efficiently explore the space of possible workloads and select representative program-input pairs from it.

The problem of reducing simulation time has been addressed using *trace sampling techniques* (see [51] and references therein). The goal of such techniques is to identify representative fragments in the program execution and simulate only those fragments, thereby eliminating the need for simulating the entire program. Trace sampling techniques heavily rely on the characterization and classification of the workload imposed on the architecture by the different fragments in the program execution trace. However, it should be noted that all the above mentioned research efforts were primarily targeted towards characterization and composition of representative workloads in the domain of microprocessor design.

4.1 Measuring VCCs for Single Stream

In general, the construction of VCCs can be performed in many different ways. In some cases it is possible to derive the curves analytically from a formal specification of the system and its environment, whereas in other cases a simulation- and trace-based analysis approach may be necessary and indeed sufficient for the problem at hand.

Let us consider the platform architecture shown in Figure 3.2 and illustrate our measurement of VCCs. Here we will take the examples of the arrival curves at the buffer B_2 (α_x), the workload curves on PE_2 (γ) and the consumption bounds (α^l_C, α^u_C).

Suppose that we adopt the simulation- and trace-based analysis method to obtain γ and α_x. We collect execution traces from a simulation of an *abstract model* of the platform architecture and then analyze them to derive the required curves. To obtain the workload curves (γ^l, γ^u), we first collect a trace of execution demands for the pair of tasks IDCT and MC executing on PE_2. Suppose that the sequence of macroblocks being processed by PE_2 is m_1, m_2, ..., and the trace records a sequence of the number of processor cycles to process each macroblock. We then get the cumulative function $W(i)$ denoting the number of processor cycles required to process the first i macroblocks. For a given k, $\gamma^l(k)$ and $\gamma^u(k)$ record the minimum and the maximum values for the set of items $\{W(i+k) - W(i) | i \geq 0\}$ respectively. Hence $\gamma^l(k)$ and $\gamma^u(k)$ identify the minimum and the maximum processing demand imposed by any sequence of k consecutive macroblocks within the video sequence.

A trace of the arrival times of a sequence of macroblocks being written into the buffer B_2 (i.e. at the output of PE_1) can be obtained by measuring the execution demands of the VLD and IQ tasks for each macroblock in the video sequence and by taking into account (i) the constant arrival rate of the compressed bit stream at the input of PE_1, and (ii) the amount of bits allocated to encode each macroblock in the stream. We can then obtain the cumulative function $T(i)$ denoting the total time length during which the first i macroblocks arrive at the buffer B_2 shown in Figure 3.2. Using a similar method to the one for obtaining workload curves, we analyze the function $T(i)$ to obtain the pseudo-inverse of arrival curves (α_x^l, α_x^u). Finally we derive the arrival curves.

In similar way, we can derive the consumption bounds (α_C^l, α_C^u). However, in this case since we precisely know the characteristics of the real-time client we do not need to rely on the simulation. α_C^l and α_C^u can be constructed analytically by using the fact that the real-time client reads macroblocks from the playout buffer at a predefined constant rate.

4.2 Classification of Streams

We propose to classify streams based on the *shapes* of the VCCs associated with them. We hypothesize that if two streams are characterized by VCCs having similar shapes, then their behaviors, in the worst/best-case, will also be similar. Each stream might be associated with several types of VCCs, characterizing different aspects of variability within the stream.

Therefore, if two streams have similarly shaped VCCs of respective types, then they will impose similar workload on the architecture (in the worst- and best-case). For example, streams with similarly shaped workload curves will consume similar amounts of processing resources on a PE. Streams with similarly shaped workload curves and consumption curves will create similar maximum backlogs in the buffers of the architecture as a result of their processing.

4.2.1 Measuring Dissimilarity between Two Streams

To identify if two streams impose similar workloads on a platform architecture, the VCCs measured for them are used and the types of VCCs used are dependent on the problem studied. Firstly, we discuss how to compare two streams based only on a single variability type, i.e. the same type of VCCs. We define a measure of *dissimilarity* between two VCCs of the same type. Considering the general case of comparing two objects (VCCs here), an object is represented by a set of variables. The dissimilarity between two objects is found by computing some *metric* defined over these variables. In our case, a VCC, which is defined for a set of points $k = 1, 2, .., n$, can be seen as an object described by n variables. Intuitively, to see how dissimilar the shapes of two VCCs (of the same type) are, we need to compare their values for each of the points $k = 1, 2, .., n$. By noting that all n variables represent a VCC along essentially *separable* dimensions, we can quantitatively measure the dissimilarity between two VCCs using the City Block metric [35]. This metric is chosen, because in comparison to other known metrics (e.g. Euclidean Distance) it is more “sensitive” to differences in each of the dimensions, i.e. in our case, the metric is more “sensitive” to the differences in the shapes of two VCCs. Given below is a formal definition of the dissimilarity between two VCCs, based on the City Block metric.

Let $\theta_{ri}(k)$ ($k = 1, 2, .., n$) denote a VCC of type r associated with the ith stream. A measure of the pairwise dissimilarity between two streams i and j, with respect to a VCC of type r, is then defined as

$$d_{rij} = \sum_{k=1}^{n} \frac{1}{k} \, |\theta_{ri}(k) - \theta_{rj}(k)| \qquad (4.1)$$

where k denotes the length of the analysis interval that is used to normalize the differences $|\theta_{ri}(k) - \theta_{rj}(k)|$. The reason that we use this normalization is that the longer the analysis interval, the less *critical* is the difference in the values of the two VCCs corresponding to this interval. The absolute difference for a larger value of k is *distributed* over a larger number of stream objects than in the case of a smaller value of k, and therefore this difference becomes less critical.

In many cases it might be useful to characterize streams using more than one type of VCCs. How should the dissimilarity between streams be quantified in such cases? We believe that first, the measure of dissimilarity between VCCs having identical types should be computed using Eqn. (4.1). These measures can then be combined in various possible ways, one of them being simply computing the sum of all the dissimilarity measures for the individual VCC types. The pairwise dissimilarity between two streams i and j with respect to VCCs of types $r = 1, 2, .., p$ is then defined as

$$d_{ij} = \sum_{r=1}^{p} d_{rij} \tag{4.2}$$

4.2.2 Clustering of Similar Streams

Given a large library of streams, we need to *classify* them into different clusters. Streams within the same cluster impose similar workloads. To classify streams using the dissimilarity measure described above, we use a conventional hierarchical clustering algorithm based on the *complete linkage* algorithm [35] for computing distances between clusters. The rationale behind the choice of the complete linkage algorithm is the need to keep the clusters as dense as possible.

4.3 Empirical Validation

To see how the stream classification method described in the previous section performs on real data samples, we conducted a number of experiments with MPEG-2 video streams.

We considered the following scenario. A generic MpSoC platform, such as the one shown in Figure 3.2, has to be customized such that it supports real-time decoding of MPEG-2 video streams. Hence, we need to study the impact of different MPEG-2 streams on the platform and based on the results of our study, optimize the architecture accordingly. For this purpose we collected a large library of video clips that we believe our architecture should be able to support. However, due to time constraints we cannot afford to run simulations for all the clips in the library. Furthermore, simulation of an entire clip takes a prohibitively long time. Therefore, we are constrained to simulate only a limited number of *short fragments* extracted from *selected* video clips belonging to the library.

Data Selection: We assume that any video clip in the library contains only one *scene*. In a visual sense, a scene is "*a portion of the movie without sudden changes in view, but with some panning and zooming*" [50]. Distinguishing between different scenes is necessary, because even within a single MPEG-2 stream different scenes might have substantially different characteristics. For example, characteristics of MPEG-2 streams (such as bit rate) may *significantly* vary at a large time scale, i.e. across different scenes, while at a short time scale (i.e. within a scene) the variations are more moderate [50, 53]. If different scenes are not treated separately while deriving their VCCs, due to the nature of VCCs, important information about some scenes may be overshadowed by other scenes. Finally, we note that in practice it is always possible to split a long movie into a series of individual scenes (see [50] for the relevant references).

For our experiments, we used a library of MPEG-2 video clips that is shown in Table 4.1. The clips in this library contain two categories. Each clip in Category A is a 8 Mbps constant bit rate stream consisting of only one scene with a resolution of 704×576 pels and a frame rate of 25 fps, while clips in Category B are 4 Mbps constant bit rate streams consisting of only one scene with a resolution of 704×480 pels and a frame rate of 30 fps. We believe that the variety of scenes represented by this library is sufficient for a demonstration of our classification method.

To select representative streams for performance evaluation of our architecture, we classified the streams in the library based on (i) the variability in execution demand, and (ii) the variability in the production and consumption rates of the tasks running on the PEs of

category	index	video clip	index	video clip
A	1	100b_080.m2v	7	pulb_080.m2v
	2	bbc3_080.m2v	8	susi_080.m2v
	3	cact_080.m2v	9	tens_080.m2v
	4	flwr_080.m2v	10	time_080.m2v
	5	mobl_080.m2v	11	v700_080.m2v
	6	mulb_080.m2v		
B	12	100b_040.m2v	18	pulb_040.m2v
	13	bbc3_040.m2v	19	susi_040.m2v
	14	cact_040.m2v	20	tens_040.m2v
	15	flwr_040.m2v	21	time_040.m2v
	16	mobl_040.m2v	22	v700_040.m2v
	17	mulb_040.m2v		
Source: `ftp.tek.com/tv/test/streams/Element/MPEG-Video/`				

Table 4.1: MPEG-2 video clips used in our experiments.

the platform. The VLD (i.e. VLD/IQ) task has both these types of variabilities. For each activation, it consumes a variable number of bits from the input buffer and its execution demand also fluctuates. Hence, we characterized it using the workload curves $(\gamma^u_{vld}, \gamma^l_{vld})$ and the consumption curves $(\kappa^u_{vld}, \kappa^l_{vld})$. The IDCT (i.e. IDCT/MC) task was characterized using only the workload curves $(\gamma^u_{idct}, \gamma^l_{idct})$, because its execution demand is variable but consumption and production rates are constant.

Obtaining VCCs: Using the experimental setup described in Section 7.4, we simulated with the platform architecture as shown in Figure 3.2. The VCCs were obtained from the collected execution traces. To obtain an upper (lower) VCC we searched through the corresponding trace with time windows of different lengths and identified the maximum (minimum) execution requirements (or number of bits) occurring in the trace within each of these time windows. The maximum window size was determined by the maximum time interval over which the streams were compared. For each design scenario, this might be different. In our experiments we had set the maximum window size to 12 frames. This corresponds to the most frequently occurring length of group of pictures (GOP) in the MPEG-2 bitstreams.

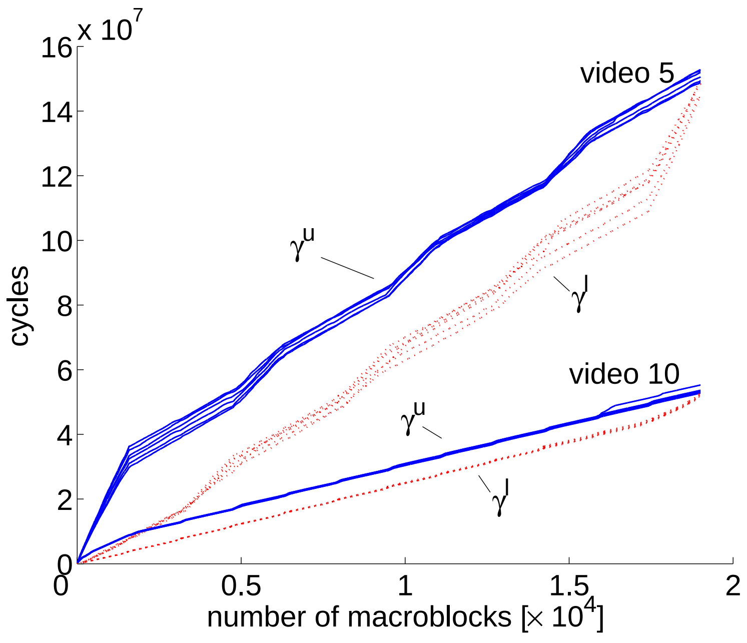


Figure 4.1: $(\gamma^u_{vld}, \gamma^l_{vld})$ for different fragments of video 5 and video 10.

Results and Discussion: Our first step was to compute the maximum dissimilarity between VCCs obtained from different fragments of the same scene. If this dissimilarity is sufficiently low then we can randomly pick a short fragment from a long video clip and use it as a representative of the whole video clip. If this dissimilarity is too high, then we may need to adopt other approaches to select short fragments. For example, fragments of the same scene can be classified first. Then *several* fragments can be chosen to represent that scene.

From each clip in our library, we extracted 10 unique fragments of the same length (30 frames) and measured their VCCs. Figure 4.1 shows results of the measurements for $(\gamma^l_{vld}, \gamma^u_{vld})$ for two video clips, i.e. clip numbers 5 and 10 from Table 4.1. Video 5 represents a natural full-motion scene, whereas video 10 is a video test pattern displaying a small running timer on a still background. By inspecting the plots in Figure 4.1 we can see that the dissimilarity between fragments of video 5 is larger than those between fragments of video 10. This can be explained by the higher degree of motion present in the scene of video 5. Nevertheless, we can see that the curves for different fragments of video 5 exhibit a similar behavior. For other videos in the library, we observed very similar trends.

Using Eqn. (4.1) for each VCC type we computed pairwise dissimilarities between

VCC	max.dissim	video	VCC	max.dissim	video
γ_{vld}^{u}	57151356	4	γ_{idct}^{l}	37220944	3
γ_{vld}^{l}	23548299	4	κ_{vld}^{u}	2146073	4
γ_{idct}^{u}	22903156	9	κ_{vld}^{l}	752238	4

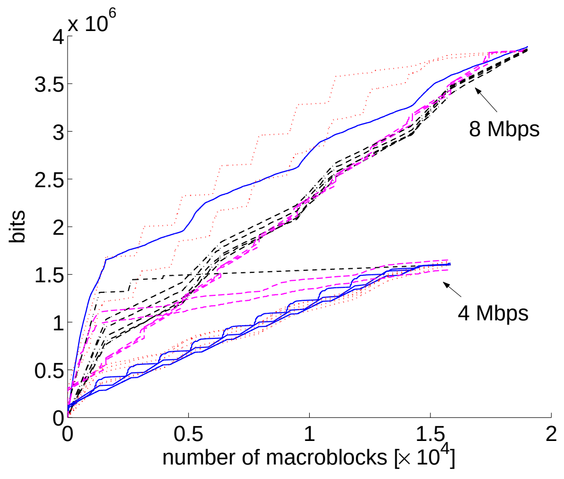
Table 4.2: Maximum dissimilarity between fragments of the same scene.

fragments of the same scene and selected their maximum value. Table 4.2 shows the obtained maximum values taken *over all* the video clips in Category A. From this table we can observe that video 4 probably contains a very complex and changing scene, because almost all the VCC types of its fragments exhibit a higher dissimilarity compared to those for the other clips.

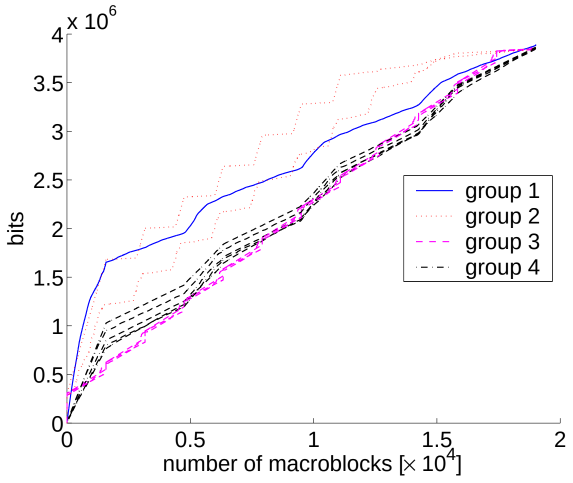
For the classification of the (full length) video clips we decided to randomly pick one fragment from each clip and then perform the classification based only on the selected fragments. Actually, the classification of the video clips can follow a hierarchical way. Firstly, in certain cases we can classify all the video clips in a large library into several coarse-grained groups, based on the different property values that each clip have (e.g. different resolutions, bit rates, contents etc.). We can then classify the clips in each group using the methods presented in this chapter. Further classification can be operated for any interested group that is already a classified result of previous steps.

In our example, we first classified all the video clips in the library into two groups: Category A and Category B, based on their different bit rate and resolution values. In Figure 4.2, we show that this coarse-grain division is meaningful in some cases, for example, when we perform the classification based on only *one* VCC type, κ_{vld}^{u}. Figure 4.2 (a) shows that based on the VCC shapes, all the video clips in the library are classified into two groups that just belong to Category A and Category B respectively. The further classification of Category A is shown in Figure 4.2 (b) and that of Category B is shown in Figure 4.2 (c).

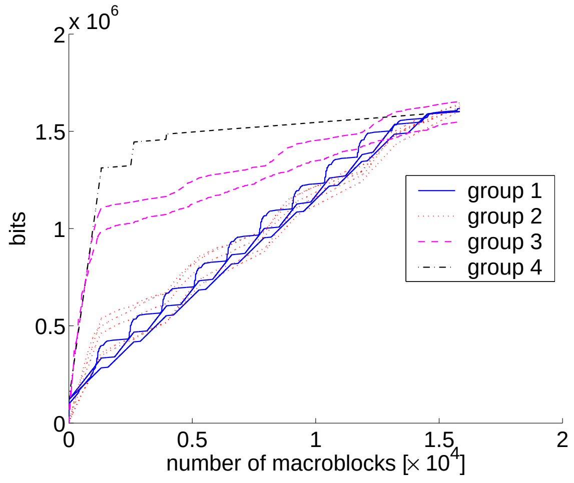
In the remainder of this chapter, we only show our classification for Category A. For the purpose of illustration, we first performed the classification based on only *one* VCC type. The results of the classification into four groups, based on the shape of γ_{vld}^{u}, are presented



(a) Category A and Category B



(b) Category A



(c) Category B

Figure 4.2: Classification based on κ^u_{vld} only for all the clips.

in Figure 4.3. As we can see in the figure, our method could correctly identify groups of curves having similar shapes. This indicates that the measure of dissimilarity defined by Eqn. (4.1) leads to a meaningful classification. The same observation can be obtained in Figure 4.4 as the classification is based on the shape of γ^u_{idct}.

Figure 4.5 shows a *dendrogram* of the hierarchical cluster tree obtained as a result of the classification based on *all* VCC types, i.e. by using Eqn. (4.2). In this dendrogram we can clearly distinguish between two major groups of clips: still and motion videos[1]. This kind of a coarse-grained division into two groups would have been possible to obtain just by viewing the videos on the screen. However, a more refined classification would be difficult to achieve using such a subjective technique. For example, before performing the

[1]Since video 10 is mostly still, it was assigned to the group of still videos by our method.

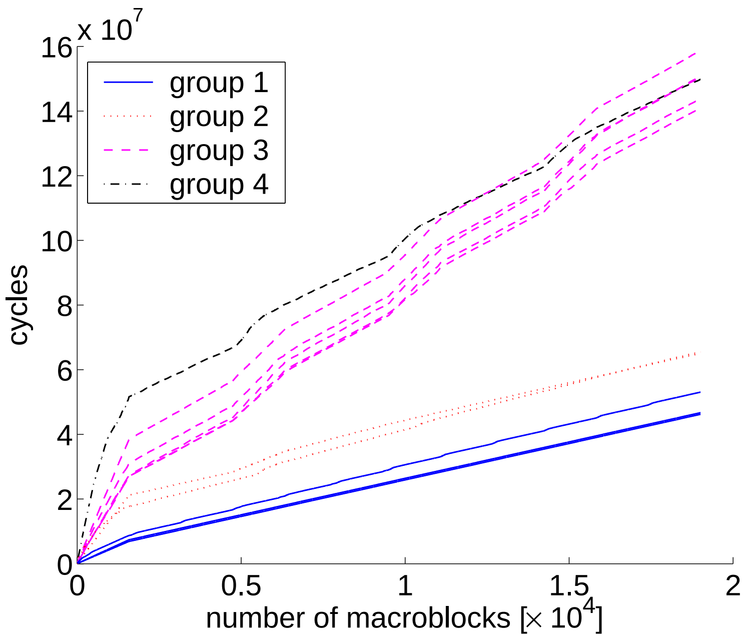


Figure 4.3: Classification based on γ^{u}_{vld} only for the clips in Category A.

experiments, by simply viewing the clips we could not predict that video 4 would have such different properties in comparison to the other motion videos. However, we can easily see this in the dendrogram: all other motion videos except video 4, form a tight cluster with the maximum linkage distance almost three times smaller than the maximum linkage distance when video 4 is included into the cluster.

Finally, to see how the results of the stream classification correlate with the actual impact of the streams on the architecture, we performed simulations of the system shown in Figure 3.2. We simulated the decoding of several *full-length* video clips from our library. As a measure of the *architectural impact* we decided to use maximum backlogs occurring as a result of the MPEG-2 processing in the buffers B_2 and B_V. The backlog in the buffer in front of PE_1 was not taken into account due to its relatively small size.

Table 4.3 summarizes the simulation results. Our measurements show that, for example, videos 1 and 7 produce very similar maximum backlogs in the both buffers. The maximum backlogs produced by videos 9 and 2 are *less similar* than the backlogs produced by videos 1 and 7. For videos 9 and 2, the differences in the backlogs in B_2 and B_V are 2110 and 245 macroblocks respectively. We can also see that video 9 is *more similar* to video 2 than to video 3. The maximum backlogs for video 3 and video 9 differ by 4935 and 405

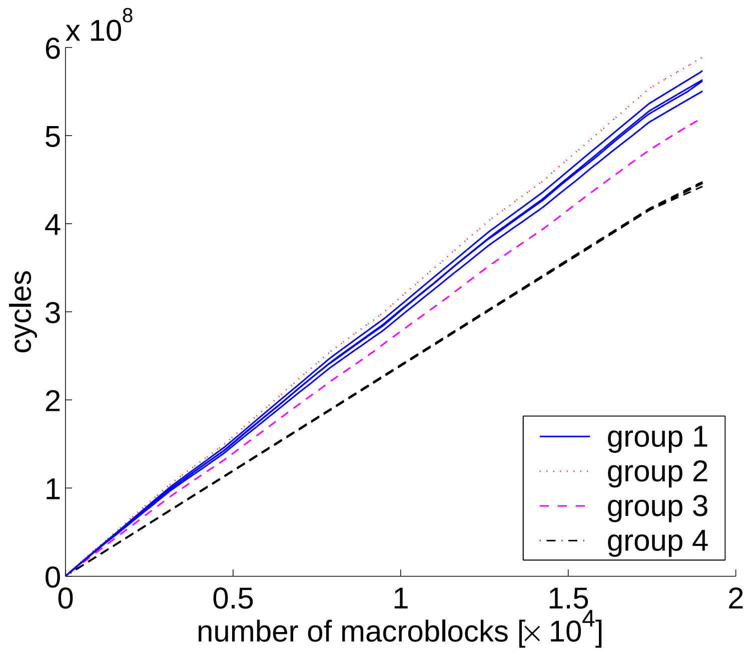


Figure 4.4: Classification based on γ^{u}_{idct} only for the clips in Category A.

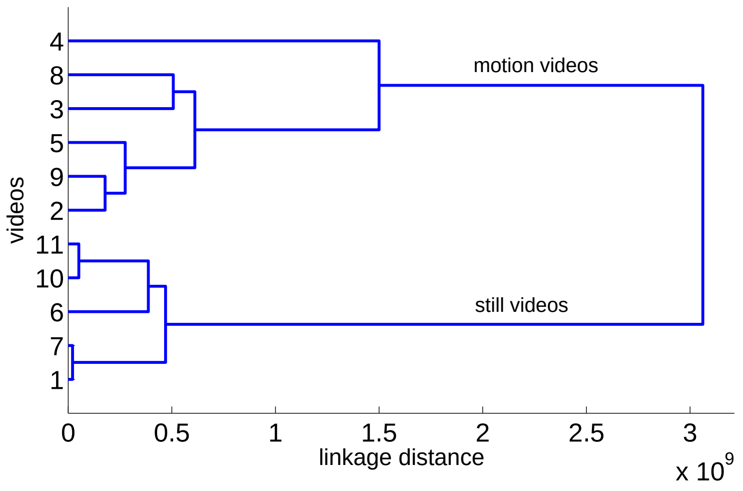


Figure 4.5: Cluster tree.

macroblocks in B_2 and B_V respectively. Hence, we can see that the simulation results exhibit the same tendency as that shown by the classification in Figure 4.5.

4.4 Summary

In this chapter we presented a promising approach for workload design for the specific context of system-level design of MpSoC platforms. "Representative" VCCs were identified that can be used in the performance analysis of MpSoC platforms, which is described in

video	B_2	B_V	video	B_2	B_V
1	8282	9433	6	5366	9190
2	5128	9027	7	8390	9593
3	7953	8867	9	3018	9272

Table 4.3: Measured maximum buffer backlogs.

detail later in this thesis. Our two main contributions in this chapter were: (i) identifying VCCs as a means for representing properties of multimedia workloads for system-level design of media processing platforms, and (ii) a classification method based on VCCs to cluster multimedia streams which exert similar influences on a platform architecture. We presented preliminary results that show the usefulness of this approach. However, there is considerable scope for further research in this direction. For example, to clearly identify the influence of multimedia workloads on buffer backlogs is not trivial, for which we need to study further what more types of VCCs should be considered. Hence, a more systematic study needs to be done to identify "variability types" beyond the ones considered in this chapter. We are not aware of any previous work in this direction and hope that our work in this chapter will encourage a systematic study of this problem, especially since simulation time is a widely recognized deterrent in the case of simulation-based performance evaluation of embedded systems.

Chapter 5

System Design Case I: Processor Frequency Selection

In this chapter, we apply our proposed analytical framework to study the issue of selecting frequency values for on-chip processors, which is one typical case in the design of energy-aware MpSoC platform architectures specifically targeted towards media processing in portable devices. Under the framework, we develop analytical approaches for solving this challenging problem.

General-purpose configurable SoC platforms generally provide embedded processor cores that offer a high degree of customization potential, such as instruction set tailoring and register file sizing. In recent years, dynamic voltage scaled (DVS) processors have appeared, and thus the operating voltage (and proportionally the operating frequency) can be customized. Choosing the number of *operating points* and the values for these operating points is becoming a part of this customization procedure. More levels imply more complicated design and more cost, but may result in more energy savings. Trade-offs between the cost and energy savings should be fully considered in the customization. Choosing the efficient operating points is especially critical in the context of multimedia applications because of the complex and bursty nature of on-chip traffic and the high variability in the execution times of multimedia processing tasks—both of these resulting in a highly variable demand on the computational resources available on the chip [87]. Hence, being able to control the processor frequency accurately to counter this variability is important.

The problems we are interested in addressing are of the following form. Suppose that we are given a multiprocessor SoC platform architecture “template” and a number of multimedia applications, all of which are required to be supported by this platform. Our job is to derive a (concrete) platform architecture from this template, by choosing appropriate processors, sizes of on-chip buffers and possibly other parameters such as bus widths and cache configurations. The processors to be chosen for this platform support software-controlled voltage and frequency scaling to allow different degrees of power consumption at run time. Therefore, we are also required to choose the frequency/voltage ranges that each processor should support. In this chapter we specifically focus on this last issue and identify how this range depends on the other parameters of the platform architecture, such as on-chip buffer sizes.

The results presented in this chapter also provide insights into questions such as: if a processor supports only a fixed number of operating points, where each such point is characterized by a voltage and a frequency value, then how many such operating points should a processor ideally support and how should these values be chosen? A processor which allows the voltage and frequency values to be changed continuously would typically be more expensive than one which allows these values to be changed in discrete steps and supports only a fixed number of these values or operating points. Today, processors of both these types are available—Intel’s XScale processor is of the former type and Transmeta’s Crusoe processor is of the latter type. Therefore, it is pertinent to ask questions like what kind of performance impacts choosing a processor of the latter type would have, over a more expensive processor which supports a continuous range of frequency values? Further, a platform designer would also be interested in identifying how the frequency range, that needs to be supported by a processor, varies with the available on-chip buffer size. Since on-chip buffers are available only at a premium because of their high area requirements [87], such information would help in choosing an appropriate tradeoff.

5.1 Our Results and Relation to Previous Work

The main contribution of this chapter is analytical approach which can guide a system designer in identifying the operating frequency range that different processors on a SoC platform architecture should support in order to run a given multimedia application or a class of applications. Identifying such a range accurately is not straightforward because of the reasons mentioned above, i.e. the complex nature of on-chip traffic arising out of multimedia processing and the variability in the execution times of tasks. Moreover, since different applications and input classes might have very different computational demands choosing an appropriate processor frequency range involves several tradeoffs between processor cost, flexibility and on-chip buffer requirements. Our approach can help a system designer in identifying these tradeoffs.

To save energy consumption, there has been a significant amount of work in developing voltage and frequency scheduling algorithms in the context of multimedia applications (see, for example, [2, 19, 26, 41, 65, 89], and also [6] and the references therein). Dynamic voltage and frequency scheduling (DVFS) methods are trying to run the processor in as low a frequency value as possible, while satisfying the quality-of-service requirements of the applications. The core of these methods is to predict the execution time of the future frames based on the history execution and thus to dynamically determine the lowest possible frequency value that can be run when an instance or instances of multimedia streams are actually running on the system. DVFS schemes are to select the optimal frequency values for given instances of multimedia streams on a given multimedia system where the voltage and frequency levels of the processors are already fixed. In other words, DVFS schemes do not involve in any issue of selecting voltage/frequency values that should be supported by a processor that is concerned during the design phase.

The problem of processor design and processor frequency selection from an energy-aware perspective has received considerably little attention so far. One representative work in this direction is [38], which addresses the selection of the processor core and instruction and data cache configuration in the design of variable voltage processors. Some other work has been done to select the multiple voltage levels at gate level [25][72][30]. However, at gate level it is not easy to capture the workloads imposed by the applications, which is especially important in the context of multimedia applications due to the high variability exhibited from multimedia workloads.

Little work has studied the problem of voltage selection at the application level, which is more related to what we do in this chapter. Quan and Hu [5] presented a technique to determine voltage settings for a variable voltage processor, where the processor is limited to utilizing a fixed-priority assignment to schedule jobs. The voltages of the processor are also assumed to be able to change continuously. Hua and Qu [39] studied the voltage selection problem in the case of only discrete voltage values being allowed. Analytical solution was derived for dual-voltage system, but for the multiple-voltage systems, numerical methods were used to approximate the solutions. Buss et al. [18] presented another work for the case of the discrete variable voltage processors. The task to be executed on a processor is firstly specified as a task graph whose vertices are annotated with execution requirements and deadlines. A linear programming based technique is then proposed to optimally select the number of operating voltage/frequency points and their specific distribution for optimal power savings.

Our work in this chapter follows the line of development in network calculus theory, but extends the underlying theory. None of the previous results provided means for computing the range of processor frequencies from an input specification. This extension is presented in Section 5.4. Our work presented here can be used to analyze a *class* of input streams and provide theoretical guarantees on the performance of an architecture for a class of inputs, for which a more elaborate theory is necessary—this is explained in detail in Sections 5.3.1 - 5.3.3.

The rest of the chapter is organized as follows. The next section formally states our problem. Given a specification of the application to be implemented on this architecture and the class of input streams to be processed, in Section 5.3 we compute bounds on the *service* that needs to be provided by each processor of this architecture. In Section 5.4 we show how such service bounds can be used to derive the operating frequency range of each processor. Finally, in Section 5.5 we present a case study involving an MPEG-2 decoder application to illustrate an application of the proposed approach, and also validate the results obtained using detailed simulations.

5.2 Problem Formulation

In this chapter, we consider the system-level view of multimedia stream processing on a SoC platform shown in Figure 3.2. A model of this platform architecture is shown in Figure 5.1.

In Figure 5.1, let $x_i(t)$ denote the number of stream objects that arrive at a PE PE_i during the time interval $[0, t]$. Let $y_i(t)$ (equal to $x_{i+1}(t)$) denote the number of processed stream objects at the output of PE_i (or the input of PE_{i+1}) during the time interval $[0, t]$. The real-time client RTC consumes stream objects from the playout buffer at a rate $C(t)$, which again denotes the number of stream objects consumed within the time interval $[0, t]$.

The *service* received by the stream entering the processing element PE_i is denoted by a service curve β_i, which is specified by a tuple (β_i^l, β_i^u). Within *any* time interval of length Δ, it is guaranteed that PE_i will process *at least* $\beta_i^l(\Delta)$ number of stream objects and it will be able to process *at most* $\beta_i^u(\Delta)$ number of stream objects. The functions β_i^l and β_i^u therefore represent lower and upper bounds [1] on the service provided by PE_i and is determined by the time required to process each stream object, the scheduling policy implemented on this PE (in case multiple streams are being processed by it), and also by the voltage/frequency scheduling policy implemented on it. Lastly, each PE PE_i is also associated with a workload curve $\gamma_i = (\gamma_i^l, \gamma_i^u)$, where $\gamma_i^l(k)$ and $\gamma_i^u(k)$ denote the minimum and the maximum number of *processor cycles* respectively, that may be required to process *any* k consecutive stream objects belonging to the input stream. γ_i is therefore used to capture the variability in the execution requirements of the different stream objects.

Now recall from above that for each PE belonging to the platform, we would like to

[1]In this chapter, unless specially noted as general bound, any reference to upper bound or lower bound means the tightest bound, i.e. an upper bound (lower bound) means also the maximum (minimum) value under certain constraints.

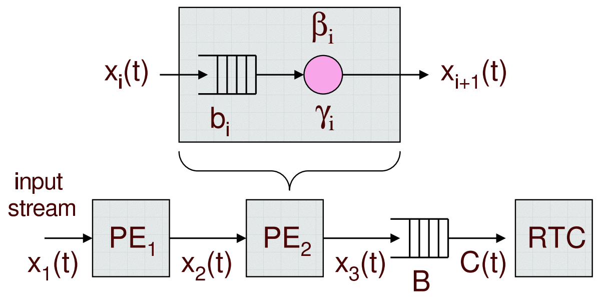


Figure 5.1: System-level view of multimedia processing on a multiprocessor SoC platform.

determine the operating frequency range that should be supported by it. If the processor supports only a fixed number of discrete frequency levels, then we would like to determine how should these frequencies be chosen and what kind of performance impacts will this decision have. Note that the platform should be designed to support a *class* (or several classes) of multimedia streams. For example, a portable multimedia device might have a wireless interface through which MPEG-2 coded video streams of two different classes come in—high-quality video clips with 8 Mbps input bit rate and low-quality clips with 4 Mbps input bit rate. The computational demands associated with these two input classes might vary widely, which translates to different operating frequency requirements for any PE on the platform. The input to a PE, when specified using the function $x_i(t)$, however, represents a concrete instance of a stream rather than a *class* of streams. Therefore, to specify the arrival pattern of a class or family of streams, we use one of the VCCs called *arrival curve* which is similar to the concept of *service curve* β_i described above. The arrival curve α_{x_i} representing the class of streams that might arrive at the input of PE_i is also specified by a tuple $(\alpha^l_{x_i}(\Delta), \alpha^u_{x_i}(\Delta))$, where the first and the second terms represent the minimum and the maximum number of stream objects that might arrive within any time interval of length Δ. In other words, $\alpha^l_{x_i}(\Delta) \leq x_i(t+\Delta) - x(t) \leq \alpha^u_{x_i}(\Delta),\ \forall t, \Delta \geq 0$. Therefore, any concrete arrival pattern $x_i(t)$ is lower and upper bounded by the functions $\alpha^l_{x_i}$ and $\alpha^u_{x_i}$ respectively. Similarly, we use $\alpha^l_{y_i}$ and $\alpha^u_{y_i}$ to denote lower and upper bounds on the arrival pattern of the processed stream at the output of PE_i.

Now, let us consider the last PE in the path of a stream, i.e. the PE whose output is written into the playout buffer (see Figure 5.1). Henceforth, for simplifying the notation, we drop the subscript i representing the PE identifier. Therefore, as described above, any input instance to this PE is specified by the function $x(t)$ and the class of all input instances is bounded by the arrival curve α_x. Any output arrival pattern from this PE is represented by the function $y(t)$ and the sizes of the internal and the playout buffers are b and B respectively. The consumption pattern of stream objects from the playout buffer is specified by the function $C(t)$ as described above. Now, given $\alpha_x(\Delta)$, $\gamma(k)$, $C(t)$ and the buffer sizes b and B, the problem is to compute the set of all possible processor frequencies at which this PE might be run, such that the following constraints are satisfied: (i) the playout buffer never overflows, (ii) it never underflows, and (iii) the internal buffer never overflows.

Our solution to the above problem consists of two parts. In Section 5.3 we compute lower and upper bounds on the service (β^l and β^u) that needs to be provided by the PE in order to satisfy the above mentioned buffer constraints. In Section 5.4, we then show how to compute the frequency range that needs to be supported by the PE in order to realize these service bounds. The extension of these results to any other PE in the path of a stream (i.e. one whose output is not written into the playout buffer, but instead into another PE) is fairly simple, and is also explained in Section 5.3.4.

Throughout this chapter we assume the following processor model: a PE can either support a continuous range of clock frequencies, or a fixed number of discrete frequencies, where this number can also be equal to one—i.e. the PE runs at a fixed frequency and does not support dynamic frequency scaling. Any clock frequency is associated with a minimum operating voltage that needs to be supplied to run the processor at this frequency. We assume that this is the voltage at which the processor is run for any frequency—i.e. voltage and frequency are tightly coupled and determining the frequency results in the voltage also being determined. Hence, we will only be concerned with determining the frequency range or the discrete frequency values for any given PE.

5.3 Computing Bounds on Service Requirements

Given $\alpha_x(\Delta)$, $C(t)$ and the buffer sizes b and B for the last PE in the path of a stream, in this section we compute the lower and the upper bounds $\beta^l(\Delta)$ and $\beta^u(\Delta)$ on the service that needs to be provided to this stream to satisfy the buffer overflow and underflow constraints described in Section 5.2. Within any interval of length Δ, if the service provided is less than our computed $\beta^l(\Delta)$, then either the internal buffer might overflow or the playout buffer might underflow. Similarly, if the service provided is greater than the computed $\beta^u(\Delta)$, then the playout buffer might overflow.

Following the notation introduced in Section 5.2, we use $x(t)$ to denote any arrival pattern of stream objects at the input of the PE and $y(t)$ to denote the arrival pattern at the output of the PE. Recall that the functions x, y and C always denote *cumulative values* over the time interval $[0, t]$, whereas the functions α_x and β take *time interval lengths* as the input parameter.

We assume that the first stream object arrives at the internal buffer b at time $t = 0$. The playback delay associated with the output device be equal to t_d, i.e. the first stream object is read out from the playout buffer B at time $t = t_d$. Then the constraint on the playout buffer underflow can be stated as (see Figure 5.1):

$$y(t) \geq C(t), \quad \forall t \geq 0 \tag{5.1}$$

Similarly, the constraint on the playout buffer overflow can be stated as:

$$y(t) \leq C(t) + B, \quad \forall t \geq 0 \tag{5.2}$$

Finally, the constraint that the internal buffer in the PE should not overflow, is given by:

$$y(t) \geq x(t) - b, \quad \forall t \geq 0 \tag{5.3}$$

The constraints (5.1) and (5.3) can be combined and stated as:

$$y(t) \geq C(t) \vee (x(t) - b), \quad \forall t \geq 0$$

Now, if $S^l(t)$ denotes the minimum number of stream objects that is guaranteed to be processed by this PE during time interval $[0, t]$, then it can be shown that $y(t) \geq (x \otimes S^l)(t)$, $\forall t \geq 0$ [16]. Hence, $(x \otimes S^l)(t)$ is the minimum value of $y(t)$ for any t, and therefore the above constraint on $y(t)$ can be reformulated as:

$$(x \otimes S^l)(t) \geq C(t) \vee (x(t) - b), \quad \forall t \geq 0 \tag{5.4}$$

From Lemma 1 we know that for any functions f, g and h, $g \otimes h \geq f$ if and only if $h \geq f \oslash g$. Using this result, Ineq. (5.4) can be reformulated as:

$$S^l(t) \geq (C(t) \vee (x(t) - b)) \oslash x(t), \quad \forall t \geq 0 \tag{5.5}$$

Since $\beta^l(\Delta)$ is the minimum number of stream objects that is guaranteed to be processed by this PE within any time interval of length Δ, we have

$$\beta^l(t) \geq S^l(t) \tag{5.6}$$

Ineq. (5.5) therefore gives a general lower bound on the service β^l that needs to be provided by the PE in order to satisfy the playout buffer underflow and the internal buffer overflow constraints.

If $S^u(t)$ denotes the maximum number of stream objects that could be processed by this PE during time interval $[0, t]$, then it can be shown that [16]

$$y(t) \leq (x \otimes S^u)(t), \quad \forall t \geq 0$$

Hence, following the same reasoning as above, $(x \otimes S^u)(t)$ is the maximum value of $y(t)$ for any t, and by using this, the constraint (5.2) can be reformulated as:

$$(x \otimes S^u)(t) \leq C(t) + B, \quad \forall t \geq 0 \tag{5.7}$$

To determine the maximum service that the PE should provide to satisfy the playout buffer overflow constraint, we are always concerned the case that enough data arrives at the input buffer, i.e. $x(t)$ is greater than $C(t) + B$. Note that in the case of $x(t) \leq C(t) + B$ for $t \geq 0$, S^u can be infinitely large, since no matter how much service is provided by the PE, the playout buffer can never overflow (this also follows directly from the definition of the min-plus convolution operator). However, it is not meaningful to compute such a bound that is infinite.

It can be shown [16] that for any functions f, g and h, $g \otimes h \leq f$ satisfying $g(t) > f(t)$ for all $t \geq 0$, if and only if $h \leq f \oslash g$. Ineq. (5.7) can then be reformulated as:

$$S^u(t) \leq (C(t) + B) \oslash x(t), \quad \forall t \geq 0 \tag{5.8}$$

Since $\beta^u(\Delta)$ is the maximum number of stream objects that can be processed by the PE within any time interval of length Δ, we have

$$\beta^u(t) \leq S^u(t) \tag{5.9}$$

Ineq. (5.8) therefore gives a general upper bound on the service β^u that can be provided by the PE, in order to satisfy the playout buffer overflow constraint.

5.3.1 Computing Service Bounds for a *Class* of Streams

The above bounds on β^l and β^u are based on a specific instance of the arrival pattern of a stream, i.e. $x(t)$. Hence, these bounds can only guarantee the buffer overflow and underflow constraints for this specific arrival pattern. However, we would like to derive the service bounds for a class of arrival patterns—i.e. all arrival patterns which are bounded by the arrival curve α_x.

5.3.1.1 Computing the Bound on β^l

For a concrete arrival pattern of stream objects given by $x(t)$, the bound on β^l is given by (from Ineqs. (5.5) and (5.6)):

$$\begin{aligned}\beta^l(t) &\geq (C(t) \vee (x(t) - b)) \oslash x(t) \\ &= \sup_{u\geq 0}\{(C(t+u) \vee (x(t+u) - b)) - x(u)\} \\ &= \sup_{u\geq 0}\{(C(t+u) - x(u)) \vee (x(t+u) - x(u) - b)\} \\ &= \max\{\sup_{u\geq 0}\{C(t+u) - x(u)\},\ \sup_{u\geq 0}\{x(t+u) - x(u)\} - b\} \\ &= \max\{(C \oslash x)(t),\ \alpha_x^u(t) - b\}, \quad \forall t \geq 0\end{aligned} \tag{5.10}$$

Since $x(t) \geq \alpha_x^l(t)$ for all $t \geq 0$, for any function f, $(f \oslash x)(t) \leq (f \oslash \alpha_x^l)(t)$ for all $t \geq 0$. Hence, the constraint specified by Ineq. (5.10) can be reformulated as:

$$\beta^l(t) \geq (C \oslash \alpha_x^l)(t) \vee (\alpha_x^u(t) - b), \quad \forall t \geq 0$$

Further, let us assume that the consumption pattern of stream objects from the playout buffer, as specified by the function $C(t - t_d)$ is lower and upper bounded by the arrival curve α_C, i.e.

$$\alpha_C^l(\Delta) \leq C(t - t_d + \Delta) - C(t - t_d) \leq \alpha_C^u(\Delta), \quad \forall t \geq t_d \ \&\ \Delta \geq 0$$

We assume that the bounds (α_c^l, α_c^u) hold over the time interval $[t_d, \infty)$ in order to obtain tighter bounds. We then have $C(t) \leq \alpha_C^u(t - t_d)$ and the above constraint on β^l can be stated as:

$$\beta^l(\Delta) \geq (\alpha_C^u(\Delta - td) \oslash \alpha_x^l(\Delta)) \vee (\alpha_x^u(\Delta) - b), \quad \forall \Delta \geq 0 \tag{5.11}$$

Ineq. (5.11) therefore provides a general lower bound on the minimum service that needs to be provided by the PE, in order to satisfy the playout buffer underflow and the internal buffer overflow constraints, where all arrival patterns at the PE are bounded by α_x and all consumption patterns from the playout buffer are bounded by α_C. Once again, recall that α_x and α_C represent a *class* or a *family* of arrival and consumption patterns of stream objects, and not any specific instance of an arrival or a consumption pattern.

5.3.1.2 Computing the Bound on β^u

From Ineqs. (5.7) and (5.9), the bound on β^u is obtained as:

$$\beta^u(t) \leq (C(t) + B) \oslash x(t), \quad \forall t \geq 0 \tag{5.12}$$

We know that any instance of an arrival pattern $x(t)$ at the input of the PE is upper bounded by α_x^u, and the lower bound on the consumption pattern of stream objects from the playout buffer is given by α_C^l, i.e. $C(t) \geq \alpha_C^l(t - t_d)$. Hence, the above constraint on β^u can be reformulated as:

$$\beta^u(\Delta) \leq \alpha_C^l(\Delta - t_d) \oslash \alpha_x^u(\Delta) + B, \quad \forall \Delta \geq 0 \tag{5.13}$$

The above general upper bound on β^u therefore guarantees that the playout buffer never overflows when the arrival pattern of stream objects at the PE is bounded by α_x and the consumption pattern of stream objects from the playout buffer is bounded by α_C.

5.3.2 Computing Service Bounds in Terms of Number of Processor Cycles

The lower and the upper bounds on the service that needs to be guaranteed by a PE, i.e. β^l and β^u, are specified in terms of the minimum and the maximum number of stream objects that need to be processed within any given time interval. However, due to the data-dependent variability in the execution times of multimedia tasks, the number of processor cycles required to completely process any stream object might be highly variable. As explained in Section 5.2, this variability can be captured by the function γ, which we refer to as the *workload curve*. $\gamma^l(k)$ denotes the minimum number of processor cycles required to process any k consecutive stream objects and $\gamma^u(k)$ denotes the maximum number of processor cycles that may be required to process any k consecutive stream objects. Therefore, it follows from the last subsection that $\gamma^u(\beta^l(\Delta))$ is the minimum number of processor cycles that must be provided to a stream within any time interval of length Δ to guarantee that the playout buffer never underflows and the internal buffer at the PE never overflows.

Similarly, $\gamma^l(\beta^u(\Delta))$ is the maximum number of processor cycles that may be provided to a stream within any time interval of length Δ to guarantee that the playout buffer never overflows.

Here we would like to point out that from our definition of the function β, it follows that

$$\beta^l(\Delta) \geq \beta^l(s) + \beta^l(\Delta - s)$$

for all $\Delta \geq 0$ and $0 \leq s \leq \Delta$. Similarly,

$$\beta^u(\Delta) \leq \beta^u(s) + \beta^u(\Delta - s)$$

for all $\Delta \geq 0$ and $0 \leq s \leq \Delta$. However, the bounds given by Ineqs. (5.11) and (5.13) need not satisfy these properties. Let us assume that Ineq. (5.11) is of the form $\beta^l(\Delta) \geq f(\Delta)$, $\forall \Delta \geq 0$ and Ineq. (5.13) is of the form $\beta^u(\Delta) \leq g(\Delta)$, $\forall \Delta \geq 0$, i.e. $f(\Delta)$ is the right hand side term of Ineq. (5.11) and $g(t)$ is the right hand side term of Ineq. (5.13). Now let us define two functions σ^l and σ^u as follows:

$$\sigma^l(\Delta) = \begin{cases} 0 & \text{if } \Delta = 0 \\ \gamma^u(f(\Delta)) & \text{if } \Delta = 1 \\ \max\{\gamma^u(f(\Delta)), (\sigma^l \overline{\otimes}' \sigma^l)(\Delta)\} & \text{if } \Delta > 1 \end{cases} \tag{5.14}$$

$$\sigma^u(\Delta) = \begin{cases} 0 & \text{if } \Delta = 0 \\ \gamma^l(g(\Delta)) & \text{if } \Delta = 1 \\ \min\{\gamma^l(g(\Delta)), (\sigma^u \otimes' \sigma^u)(\Delta)\} & \text{if } \Delta > 1 \end{cases} \tag{5.15}$$

where $\overline{\otimes}'$ and $\otimes'$ are redefined from the standard operations of *max-plus convolution* [16] and min-plus convolution in network calculus theory. Given any two functions f and g, they are defined as follows:

$$(f \overline{\otimes}' g)(t) = \sup_{0<s<t} \{f(t-s) + g(s)\}$$

and

$$(f \otimes' g)(t) = \inf_{0<s<t} \{f(t-s) + g(s)\}$$

The functions $\sigma^l(\Delta)$ and $\sigma^u(\Delta)$ are therefore defined over $\Delta = 0, 1, 2, \ldots$, and denote the minimum and the maximum number of processor cycles that should be provided to a stream within *any* time interval of length Δ for all the buffer overflow and underflow constraints to be satisfied. Moreover, it can be shown that these two functions satisfy the properties that any function which bounds the service provided by a PE should satisfy, i.e.

$$\sigma^l(\Delta) \geq \sigma^l(s) + \sigma^l(\Delta - s), \quad \forall\Delta \geq 0 \text{ and } 0 \leq s \leq \Delta$$

and

$$\sigma^u(\Delta) \leq \sigma^u(s) + \sigma^u(\Delta - s), \quad \forall\Delta \geq 0 \text{ and } 0 \leq s \leq \Delta$$

5.3.3 Bounding the Analysis Interval

So far, our computation of the service bounds σ^l and σ^u were based on the fact that the arrival curves α_x and α_C and the workload curve γ are known for all possible time interval lengths $\Delta \geq 0$. These curves would usually be derived by simulating the processing or execution of several representative audio/video samples on a template platform architecture. The traces collected from such a simulation—from the different parts of the platform architecture, such as the arrival pattern of stream objects in front of PE_2 in Figure 3.2—are then analyzed to derive the different arrival and workload curves. However, since these representative audio/video samples would always be of finite length, the curves or bounds derived from the resulting traces would also be of finite length. But the platform designed on the basis of these finite length traces might later be used to process larger audio/video samples. Hence, we would like to guarantee the buffer overflow and underflow constraints on input streams of any length (provided they satisfy the bounds dictated by the arrival and the workload curves), although the analysis and the design of the platform is based on only finite length representative inputs.

We would however like to point out here that in practice the above issue will not be of major concern to any system designer. He would use sufficiently long (but finite length) representative audio/video samples in the initial simulation phase to derive the bounds (i.e.

arrival and workload curves) that any input belonging to the class represented by these audio/video samples is expected to satisfy. Based on these bounds, the platform architecture in question would be designed. When such an architecture processes input streams which are longer in duration than the samples used for designing the architecture, it is assumed that the variability of the entire stream is bounded by the variability existing in the sample inputs. Such assumptions are not specific to our approach and are common whenever a system is designed based on *representative inputs* (for example, see [52]).

5.3.4 Extending the Analysis to Other PEs

The scheme presented so far is based on the assumption that the PE being analyzed is the last one in the path of a stream i.e. its output is directly written into the playout buffer. Now let us consider a PE, whose output is fed into another PE i.e. the next PE in the path of the stream. An example of such a PE is PE_1 in Figures 3.2 and 5.1. To derive the service bounds for this PE, let us denote the arrival curve corresponding to the arrival pattern of stream objects at the internal buffer of PE_2 as α_{x_2}. Similarly, let the arrival pattern of stream objects at the internal buffer of PE_1 be bounded by α_{x_1}, and let the size of this internal buffer be b_1. Then bounds on β^l and β^u (such as those given by Ineqs. (5.11) and (5.13)) for PE_1 can be calculated from α_{x_1}, α_{x_2} and b_1. The processed stream coming out of PE_1 must satisfy the bounds α_{x_2}. The only buffer constraint that needs to be satisfied in this case is that the internal buffer of PE_1 should not overflow. The resulting bounds on the service are therefore much simpler than the ones derived above, and hence we omit them here. This same scheme can be applied to other PEs in the path of the stream which are away from the playout buffer. If all of the PEs provide a service in accordance with the bounds computed for them, then it is guaranteed that none of the internal buffers in the architecture will overflow, and the playout buffer will neither overflow and nor underflow.

5.4 Computing Processor Frequency Range

Given the service bounds σ^l and σ^u for a PE, in this section we compute the discrete frequency levels or the frequency range that must be supported by the PE in order to realize these service bounds. For any given multimedia application and a class of input streams to be processed, accurately determining the appropriate processor frequency range is a non-trivial problem. The situation is much more complicated when the PE in question has to process multiple classes of input streams or multiple applications. In such cases, the different input classes might have different computational demands and hence require different processor frequencies. Here is it important to determine the range of processor frequencies that must be supported for each class. If these ranges overlap, then the processor might support some frequency belonging to this overlapping range. But if these ranges do not overlap, then multiple frequency levels need to be supported. Further, the processor frequency range to be supported by a PE is heavily dependent on the size of the on-chip buffers. A platform designer would therefore be interested in obtaining insights into this dependency. Our results presented below would help in obtaining such insights. These results can be summarized as follows:

1. For any application and a class of input streams, we can statically generate frequency schedules for a PE which satisfy all the buffer constraints. Such schedules specify the frequency with which the PE should be run at any time.

2. We derive a frequency range $(f_{\min}, f_{\max})$ such that *all* feasible frequency scheduling algorithms will only use frequencies within this range. Therefore, it would be sufficient if the PE supports frequencies belonging to this range only.

3. Finally, our schemes can also be used to identify how the bounds $(f_{\min}, f_{\max})$ change by changing the on-chip buffer sizes (both the playout and the internal buffers).

For simplicity, we assume that the processor frequency can be changed at each time unit. Then during any run of the processor over a time interval of length n (where n may be equal to $\Delta_{\max}$ defined in Section 5.3.3), let its frequency values be $f_1, \ldots, f_n$, i.e. f_i

is the frequency at which the processor is run during the time interval $t = (i-1, i]$. If the service being offered to a stream as a result of this schedule has to be bounded by the service curves σ^l and σ^u, then $f_1, \ldots, f_n$ is required to satisfy the following inequalities:

$$\begin{aligned}
\sigma^l(1) \leq & \quad f_i & \leq \sigma^u(1)\ \forall i = 1,2,...,n \\
\sigma^l(2) \leq & \quad f_i + f_{i+1} & \leq \sigma^u(2)\ \forall i = 1,2,...,n-1 \\
\sigma^l(3) \leq & \quad f_i + f_{i+1} + f_{i+2} & \leq \sigma^u(3)\ \forall i = 1,2,...,n-2 \\
\ldots & & \\
\sigma^l(n) \leq & \quad f_i + f_{i+1} + f_{i+2} + \ldots + f_{i+n-1} & \leq \sigma^u(n)\ \forall i = 1
\end{aligned}$$

The above constraints may be summarized as follows. For all $\Delta = 1,2,...,n$ and $i = 1,2,...,n-\Delta+1$,

$$\sigma^l(\Delta) \leq \sum_{j=0}^{\Delta-1} f_{i+j} \quad \leq \quad \sigma^u(\Delta) \tag{5.16}$$

From the constraints given by Ineq. (5.16), it may be seen that any frequency value f_i is dependent on all the previously assigned frequencies. To be more clear about how this dependency is, from Ineq. (5.16) we identify all those inequalities that only include f_i and the values in $f_1, ..., f_{i-1}$, which is shown in the following,

$$\begin{aligned}
\sigma^l(1) \leq & \quad f_i & \leq \sigma^u(1) \\
\sigma^l(2) \leq & \quad f_{i-1} + f_i & \leq \sigma^u(2) \\
\sigma^l(3) \leq & \quad f_{i-2} + f_{i-1} + f_i & \leq \sigma^u(3) \\
\ldots & & \\
\sigma^l(i) \leq & \quad f_1 + f_2 + \ldots + f_{i-1} + f_i & \leq \sigma^u(i)
\end{aligned} \tag{5.17}$$

The above inequality shows that f_i is dependent on the sum of all the previous k assigned frequencies proximate to it, for any $k = 0, \ldots, i-1$.

Randomly Generated Frequency Schedules: Suppose that all the previous frequencies $f_1, ..., f_{i-1}$ have been assigned, by solving Ineq. (5.17), the lower and upper bounds on any

f_i is given as follows:

$$\begin{aligned}
f_1^l &= \sigma^l(1) \\
f_1^u &= \sigma^u(1) \\
f_2^l &= \max\{\sigma^l(1), \sigma^l(2) - f_1\} \\
f_2^u &= \min\{\sigma^u(1), \sigma^u(2) - f_1\} \\
f_3^l &= \max\{\sigma^l(1), \sigma^l(2) - f_2, \sigma^l(3) - (f_1 + f_2)\} \\
f_3^u &= \min\{\sigma^u(1), \sigma^u(2) - f_2, \sigma^u(3) - (f_1 + f_2)\} \\
&\dots
\end{aligned}$$

Hence,

$$\begin{aligned}
f_i^l &= \begin{cases} \sigma^l(1) & \text{if } i = 1 \\ \max_{1 \le j \le i-1}\{\sigma^l(1), \sigma^l(i-j+1) - \sum_{p=j}^{i-1} f_p\} & \text{if } i > 1 \end{cases} \\
f_i^u &= \begin{cases} \sigma^u(1) & \text{if } i = 1 \\ \min_{1 \le j \le i-1}\{\sigma^u(1), \sigma^u(i-j+1) - \sum_{p=j}^{i-1} f_p\} & \text{if } i > 1 \end{cases}
\end{aligned} \tag{5.18}$$

f_i^l and f_i^u therefore give lower and upper bounds on the frequency that can be assigned during the time interval $(i-1, i]$, provided the frequencies at all the previous time intervals are known. To generate a static frequency schedule, we can choose any $f_i \in [f_i^l, f_i^u]$, and the chosen f_i will determine the range $[f_{i+1}^l, f_{i+1}^u]$.

Frequency Range: To compute the frequency range, that was mentioned above, we first compute the lower and upper bounds on f_i, by solving Ineq. (5.16). From equality (5.18), we know that the bounds on f_i is dependent on all values of $\sum_{p=j}^{i-1} f_p$. Like f_i, $\sum_{p=j}^{i-1} f_p$ is also dependent on all the previous frequencies $f_1, ..., f_{j-1}$, the constraints for which can be derived in a similar way to that for f_i (refer to Ineq. (5.17). We define two functions $(\sum_{p=j}^{i-1} f_p)^l$ and $(\sum_{p=j}^{i-1} f_p)^u$. The first provides a lower bound, and the second an upper bound on the sum $f_j + \ldots + f_{i-1}$.

$$
\begin{aligned}
(\textstyle\sum_{p=j}^{i-1} f_p)^{min} &= \begin{cases} \sigma^l(i-1) & \text{if } j = 1 \\ \max_{1\le q\le j-1}\{\sigma^l(i-j), \sigma^l(i-q) - (\sum_{p=q}^{j-1} f_p)^{max}\} & \text{if } j > 1 \end{cases} \\
(\textstyle\sum_{p=j}^{i-1} f_p)^{max} &= \begin{cases} \sigma^u(i-1) & \text{if } j = 1 \\ \min_{1\le q\le j-1}\{\sigma^u(i-j), \sigma^u(i-q) - (\sum_{p=q}^{j-1} f_p)^{min}\} & \text{if } j > 1 \end{cases}
\end{aligned}
$$

Using the above two functions, we can derive the bounds on f_i as

$$
\begin{aligned}
f_i^{\min} &= \begin{cases} \sigma^l(1) & \text{if } i = 1 \\ \max_{1\le j\le i-1}\{\sigma^l(1), \sigma^l(i-j+1) - (\sum_{p=j}^{i-1} f_p)^{max}\} & \text{if } i > 1 \end{cases} \\
f_i^{\max} &= \begin{cases} \sigma^u(1) & \text{if } i = 1 \\ \min_{1\le j\le i-1}\{\sigma^u(1), \sigma^u(i-j+1) - (\sum_{p=j}^{i-1} f_p)^{min}\} & \text{if } i > 1 \end{cases}
\end{aligned}
\tag{5.19}
$$

$f_i^{\min}$ is the smallest possible processor frequency that can be assigned during the time interval $(i-1, i]$, and $f_i^{\max}$ is the largest possible processor frequency that can be assigned during this time interval. Then the frequency range is defined as: $f_{\min} = \min_{i=1,\ldots,n}\{f_i^{\min}\}$, and $f_{\max} = \max_{i=1,\ldots,n}\{f_i^{\max}\}$. A dynamic programming method is also proposed to compute the frequency range, as shown in Figure 5.2.

5.5 Case Study

In this section we present a case study to illustrate an application of the approach presented in the last two sections. Towards this, we study a platform architecture consisting of two PEs, as shown in Figure 3.2, onto which an MPEG-2 decoder application is mapped. The goal is to compute the processor frequency range that needs to be supported by one of the PEs and also identify how this range changes by changing the on-chip buffer size.

As shown in Figure 3.2, the MPEG-2 decoder application is partitioned into a set of tasks executing in parallel on two PEs of the system architecture. PE_1 executes the VLD and the IQ tasks, while PE_2 executes the IDCT and the MC tasks. A compressed video bit

```
**Input:** service curves $\sigma^l(\Delta)$, $\sigma^u(\Delta)$ and time interval length $n$ ;
  **Definition:** arrays $F^l[n,n]$ and $F^u[n,n]$;
  **Initialization:** $F^l(i,k) \leftarrow \sigma^l(k)$, $F^u(i,k) \leftarrow \sigma^u(k)$ for all $1 \leq i,k \leq n$;
  **for** $i \leftarrow 2$ to $n$ **do**
      **for** $j \leftarrow i$ to 2 **do**
          $F^l(j,i-j+1) \leftarrow \max_{1\leq q\leq j-1}\{\sigma^l(i-j+1),\ \sigma^l(i+1-q) - F^u(q,j-q)\}$
            /*$F^l(j,i-j+1)$ stores $(\sum_{p=j}^{i} f_p)^{min}$*/
          $F^u(j,i-j+1) \leftarrow \min_{1\leq q\leq j-1}\{\sigma^u(i-j+1),\ \sigma^u(i+1-q) - F^l(q,j-q)\}$
            /*$F^u(j,i-j+1)$ stores $(\sum_{p=j}^{i} f_p)^{max}$*/
      **endfor**
  **endfor**
  **for** $i \leftarrow 1$ to $n$ **do**
      $f_i^{min} \leftarrow F^l(i,1)$, $f_i^{max} \leftarrow F^u(i,1)$;
  **endfor**
  $f_{\min} \leftarrow \min_{i=1,\ldots,n}\{f_i^{\min}\}$;
  $f_{\max} \leftarrow \max_{i=1,\ldots,n}\{f_i^{\max}\}$;
```

Figure 5.2: Algorithm of Computing Frequency Range.

stream arrives from the network interface into the input buffer of PE_1. After processing on PE_1 the partially decoded stream consisting of stream objects called *macroblocks* enters the buffer B_2 in front of PE_2. PE_2 reads the buffer one macroblock at a time and computes for each macroblock the IDCT and MC functions. After that the video stream emerges out of PE_2 as a fully decoded stream of macroblocks. This stream is written into the playout buffer B_v, which is read at a constant rate by the video output port V_{out}. The video output port represents the real-time client (RTC) in this setup. The rate with which it reads the playout buffer B_v is determined by the resolution and the frame rate of the decoded MPEG-2 video sequence. As we mentioned above, the service provided to the video stream on the PEs of such an architecture is dependent on the buffer constraints. In the above setup, at any point in time none of the buffers B_1, B_2 and B_v are allowed to overflow, and the playout buffer B_v should not underflow.

Determining the service that must be offered by the PEs to the stream, and thereby identifying their feasible clock frequency ranges under the given buffer constraints is not an easy task. The main complexity of the problem stems from the highly variable load imposed on the PEs of the architecture by the video stream. For example, let us consider the load imposed by the stream on PE_2. Firstly, the execution demand of IDCT and MC tasks performed by PE_2 varies for different types of macroblocks (e.g. because of the various kinds of motion compensation methods that have to be applied to the compressed macroblocks). Secondly, the arrival pattern of macroblocks into the buffer B_2 has a high degree of burstiness, which is caused by the variability in the execution demand of the tasks executing on PE_1. The overall result is a very complex and variable nature of the processing load imposed on PE_2. This further increases the burstiness of the macroblock stream emerging at its output and entering the playout buffer B_v.

Now, using the above example of PE_2 we will demonstrate how the proposed methodology can be applied to compute the required service bounds σ and the associated feasible clock frequency range of PE_2 for the given MPEG-2 decoder application.

5.5.1 Computing the Service Bounds and the Frequency Range for PE_2

Before we can compute the service bounds σ and the corresponding frequency range for PE_2, we need to obtain the arrival curves α_x and α_C and the workload curves γ characterizing the stream processed by PE_2, and the real-time client V_{out}.

Following the methods presented in Chapter 4, we obtain the arrival curves (α_x^l, α_x^u), (α_C^l, α_C^u) and the workload curves (γ^l, γ^u). Note that here we use a customized version of the SimpleScalar instruction set simulator for collecting the traces of execution demands of the MPEG-2 decoder tasks. Figure 5.3 shows the arrival curves (α_x^l, α_x^u), which we have obtained for a representative 4 Mbps video sequence $video_1$.

Now we apply the results presented in Section 5.3 to compute the cycle-based service bounds (σ^l, σ^u) corresponding to the service that must be offered by PE_2, to any video stream belonging to the class of streams bounded by the curves α_x, α_C and γ. The bounds (σ^l, σ^u) corresponding to the example video sequence $video_1$, for two different *system configurations* are shown in Figure 5.4. The two system configurations differ only in the sizes of the buffers B_v and B_2. By examining the plots on Figure 5.4 we can see that even

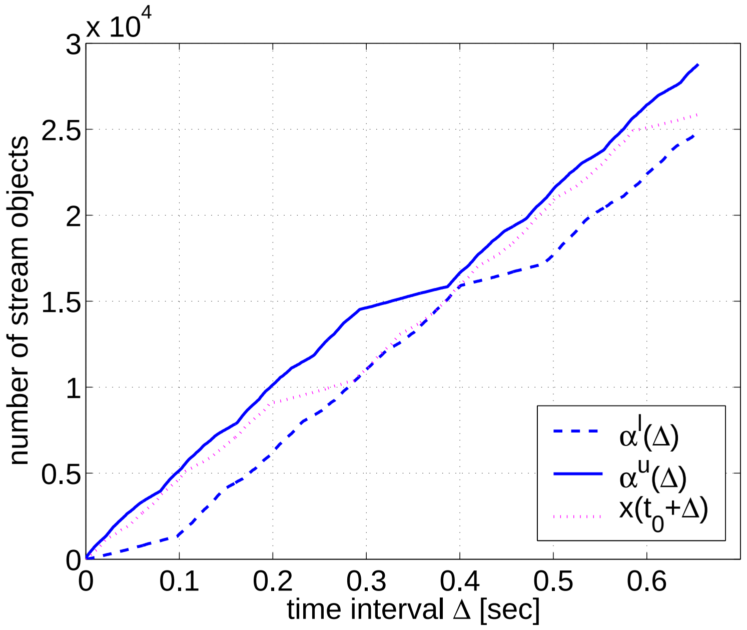


Figure 5.3: Arrival curves (α_x^l, α_x^u) of the macroblock stream on the output of PE_1 for the video sequence $video_1$. A fragment of the function $x(t)$ for $video_1$ is shown in this figure. Note that it is bounded by the corresponding arrival curves.

a relatively small change in the available buffer space may have a considerable impact on the service bounds. Furthermore, the distribution of the total on-chip buffer space among the different buffers may also have an impact on the service bounds..

We also compare with the service bounds computed by modeling the execution requirements of a sequence of stream objects using a simple best-/worst-case characterizations commonly used in the real-time systems domain. Let e_{min} and e_{max} denote the minimum and the maximum number of processor cycles required by any single stream object belonging to a sequence. The minimum and the maximum number of processor cycles that might be required by any k consecutive stream objects within the given sequence are modeled by $k \times e_{min}$ and $k \times e_{max}$. Figure 5.5 shows that the computed service bounds (denoted by (σ_s^l, σ_s^u)) using this simple modeling scheme are very pessimistic, compared to our computed service bounds (denoted by (σ^l, σ^u)) using workload curves. The frequency range resulted from this simple scheme is then computed to be (0, 1.107GHz), which is quite pessimistic compared to the computed frequency range (0, 1.672GHz) for our scheme. It is thus shown that our scheme using VCCs performs better than the simple modeling scheme.

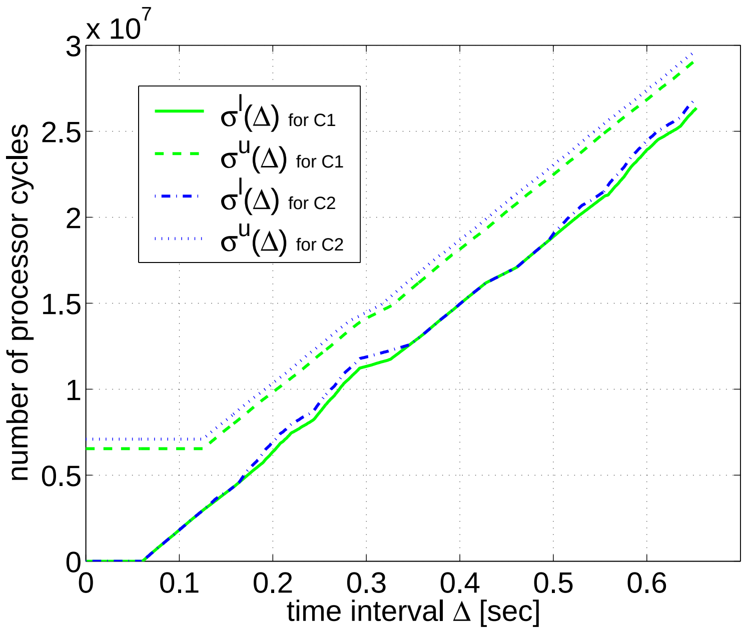


Figure 5.4: Service bounds (σ^l, σ^u) for $video_1$ for two different system configurations $C1$ and $C2$, where $C1 = \{B_2 = 4000, B_v = 7000\}$ and $C2 = \{B_2 = 4500, B_v = 6500\}$.

Video sequences belonging to different classes of streams may have very different on-chip buffer requirements. Therefore, the service bounds for these sequences, and hence their feasible clock frequency ranges might also be very different. This information about how different these ranges might be for different classes of video sequences can be efficiently obtained from the service bounds σ, as described in Section 5.4. In our example, using the proposed approach we have computed the frequency ranges for two sets of video streams. Each set has two classes of video sequences that are characterized by different input bit stream rates, i.e. for 4 Mbps and 8 Mbps MPEG-2 streams. One set of streams contains more motion and the other contains less motion. The computed frequency ranges for the above two sets of video streams are shown in Figure 5.6 and Figure 5.7. In the left of this subsection, we will discuss the results that can be concluded from Figure 5.6. The same observations can also be obtained from Figure 5.7.

Figure 5.6 shows the dependency of the frequency range on the playout buffer size for a 4 Mbps and a 8 Mbps MPEG-2 video streams. In this figure it can be seen that the playout buffer size has a considerable impact on the upper frequency bound f^u. By increasing

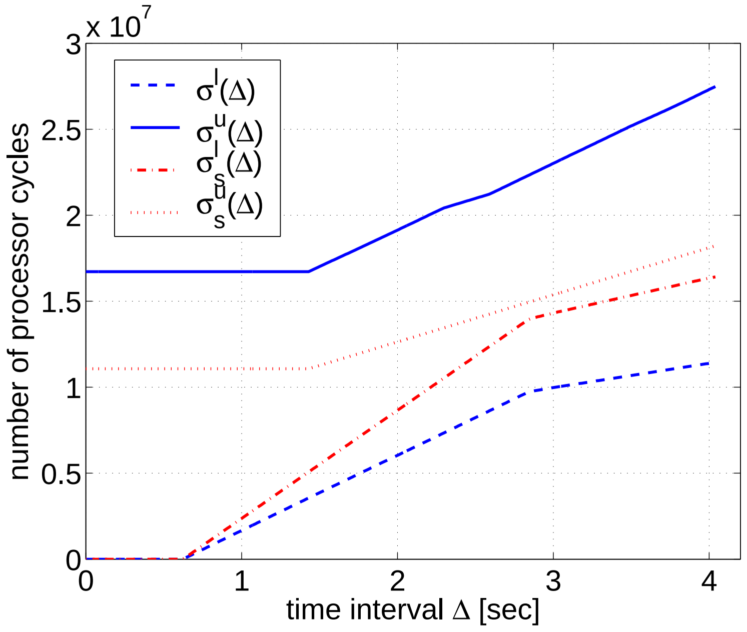


Figure 5.5: Service bounds (σ^l, σ^u) computed using VCCs and service bounds (σ^l_s, σ^u_s) computed using a simple modeling scheme for $video_1$ for system configuration $C = \{B_2 = 12000, B_v = 16000\}$.

the buffer size, the maximum frequency with which PE_2 can run, also increases. This corresponds to the intuitive understanding that the larger the playout buffer size, the more bursty the incoming stream can be.

Figure 5.6 shows an overlap in the frequency ranges of the two classes of video streams. This implies that for any *feasible* playout buffer size and a fixed size of B_2 (set to 3000 macroblocks), we can always find a clock frequency with which PE_2 can be run for video sequences belonging to both the input classes (i.e. 4 Mbps and 8 Mbps input rates). It may be noted here playout buffers only beyond a certain size are feasible—meaning that, for them feasible service bounds σ exists. It may also be noted that in general such a common clock frequency for any two input classes might not exist for a single system configuration. In such cases it will be necessary to support multiple frequency ranges/values, where the frequency level at which the processor is run depends on the class to which the input belongs. Alternatively, the configuration of the system can be changed (for example by increasing buffer sizes), till the frequency ranges of two input classes overlap. When this happens, once again it would be sufficient for the processor to support a single frequency

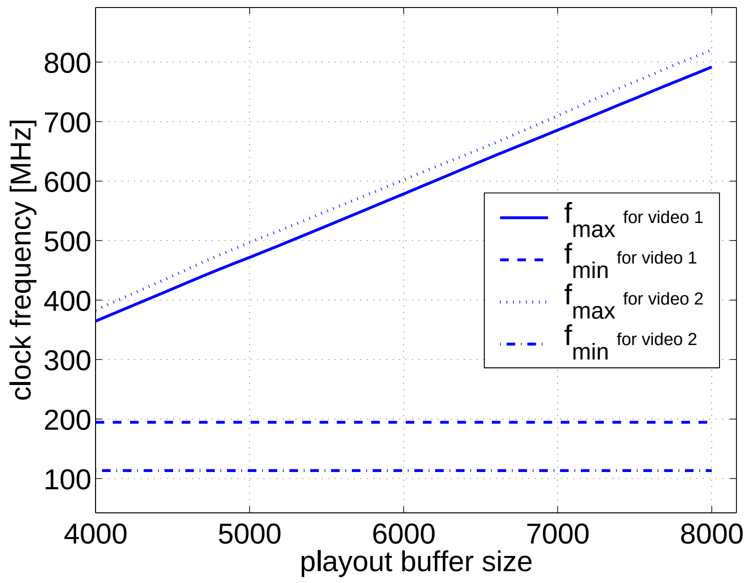


Figure 5.6: Dependency of frequency ranges on the playout buffer size for two different classes of the MPEG-2 video streams with more motion: 4 Mbps ($video_1$) and 8 Mbps ($video_2$). The size of buffer B_2 is fixed to 3000 macroblocks.

level belonging to this overlapping range. Our methodology can be used to efficiently identify such design tradeoffs in the case of configurable platform architectures.

5.5.2 Validation of the Analytical Bounds

To validate our approach for processor frequency selection we simulated the platform architecture using static schedules generated using the approach. Towards this, we used a detailed simulator of the system shown in Figure 3.2. The simulator consisted of a transaction level model of the system architecture written in SystemC, and the models of PEs were based on a customized version of the SimpleScalar instruction set simulator. Using this simulation setup, we measured the maximum backlogs and recorded any buffer underflows that occurred as a result of running the system with a static frequency schedule for PE_2 (which was generated using our approach). Table 5.1 shows a representative set of the simulation results.

We evaluated two frequency schedules for PE_2 that are bounded by the computed frequency range obtained using the proposed approach. These ranges correspond to different system configurations and classes of the video streams. In Table 5.1 these schedules are in-

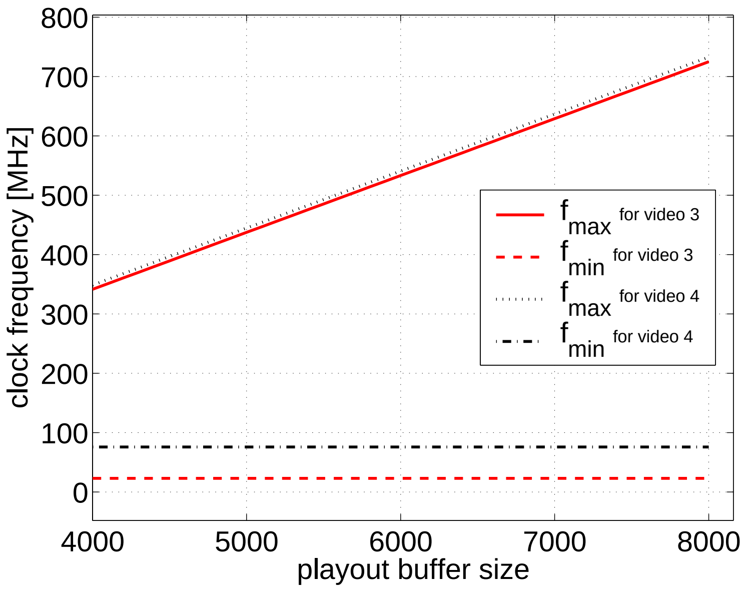


Figure 5.7: Dependency of frequency ranges on the playout buffer size for two different classes of the MPEG-2 video streams with less motion: 4 Mbps ($video_3$) and 8 Mbps ($video_4$). The size of buffer B_2 is fixed to 3000 macroblocks.

dicated as $rand1$ and $rand2$, which indicate randomly generated static schedules from the frequency bounds (f_{min}, f_{max}), as explained in Section 5.4. Two such randomly generated schedules are illustrated in Figure 5.10 for a class of video streams.

In all the simulations we performed, the maximum backlogs measured in the buffers never exceeded the buffer sizes. Furthermore, our simulation results also showed that playout buffer underflows never occurred for any of the simulated frequency schedules. It therefore validates the proposed framework and suggest its practicality. Finally, we would once again like to point out that obtaining equivalent results using purely simulation based approaches is extremely time consuming and such approaches usually fail to provide any formal performance guarantees.

5.5.3 Selection of the Analysis Interval

We also did experiments to see how the selection of the analysis interval (i.e. Δ_{max}) affects the frequency range computed. Given a sample class of video streams, Figure 5.11 shows the service bounds in terms of number of processor cycles for $0 \leq \Delta \leq 5.6$ sec. We then chose Δ_{max} to be $0.7, 1.4, ..., 4.9, 5.6$ seconds respectively and computed the frequency

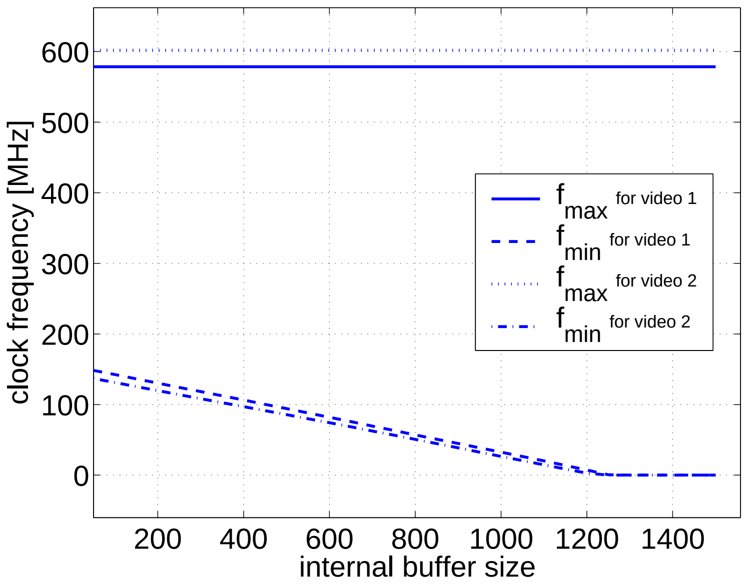


Figure 5.8: Dependency of frequency ranges on the internal buffer size for two different classes of the MPEG-2 video streams with more motion: 4 Mbps ($video_1$) and 8 Mbps ($video_2$). The size of buffer B_v is fixed to 6000 macroblocks.

ranges, as shown in Figure 5.12. It is observed that 0.7 sec is long enough for Δ_{max} such that the frequency range computed from it can bound (here, are equal to) those computed from any longer analysis interval than 0.7 sec.

5.6 Summary

In this chapter we presented our analytical approach that can be used for the design space exploration of parameters or configurations of SoC platform architectures for multimedia processing, that contain processor cores which support dynamic voltage/frequency scaling. Specifically, our approach studied how to choose the frequency range that should be supported by each processor under the architectural and application constraints. In contrast to simulation based approaches, which usually follow a trial-and-error approach and involve very high simulation times, the proposed approach can provide useful insights into the design space and can aid a system designer in systematically tuning a platform architecture for a class of applications.

Throughout this chapter we have assumed that a PE processes a single input stream.

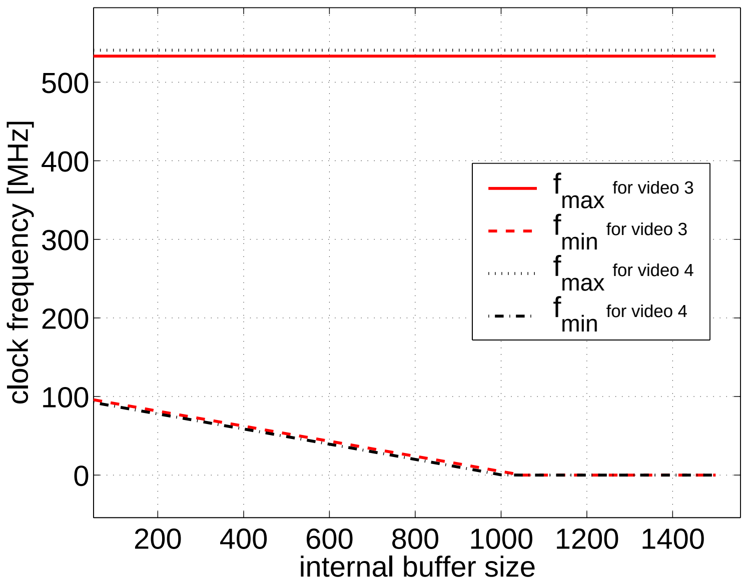


Figure 5.9: Dependency of frequency ranges on the internal buffer size for two different classes of the MPEG-2 video streams with less motion: 4 Mbps ($video_3$) and 8 Mbps ($video_4$). The size of buffer B_v is fixed to 6000 macroblocks.

However, in general a PE might process multiple input streams. For example, in the architecture shown in Figure 3.2, assume that on PE_2, in addition to the IDCT and the MC tasks, an MP3 decoder task is also implemented. In such a case, this PE processes an audio stream in addition to the video stream shown in the figure. Further, a task scheduler based on some scheduling policy chooses the stream to be processed at any time instant. Given a specification of the audio stream, our approach can be extended to identify the frequency range to be supported by the PE in this case. In addition to the theory presented in this chapter, this requires a modeling of the task scheduling discipline and how the *total* service offered by the PE is divided among the two streams. Some of ideas pertaining to such a scheme may be found in [20].

video sequence	buffer sizes		schedule	measured backlogs	
	B_2	B_v		B_2	B_v
$video_1$	4000	7000	rand 1	3234	4511
			rand 2	3259	4601
	4500	6500	rand 1	3761	4350
			rand 2	3334	4568
$video_2$	4500	7000	rand 1	3639	4721
			rand 2	3518	4910
	5000	6500	rand 1	3858	4912
			rand 2	3685	4941
$video_3$	4500	6500	rand 1	3118	4542
			rand 2	2966	4649
	5000	7000	rand 1	2781	5010
			rand 2	2668	5006
$video_4$	4000	6000	rand 1	3764	3615
			rand 2	3878	3589
	4500	5500	rand 1	3994	3556
			rand 2	4270	3482

Table 5.1: The maximum buffer fill levels obtained by simulating a static frequency schedule for PE_2 that was derived using the proposed framework. $video_1$ ($video_3$) and $video_2$ ($video_4$) are 4 Mbps and 8 Mbps MPEG-2 video streams respectively.

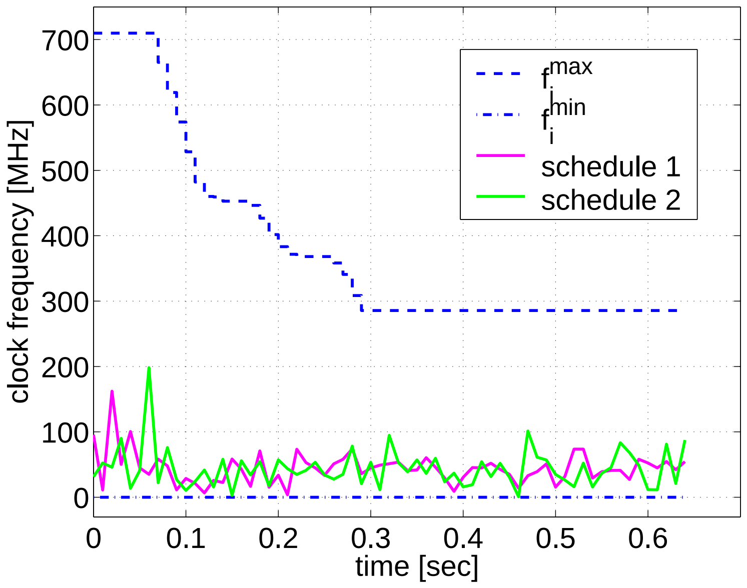


Figure 5.10: Two randomly generated schedules obtained from the service bounds σ.

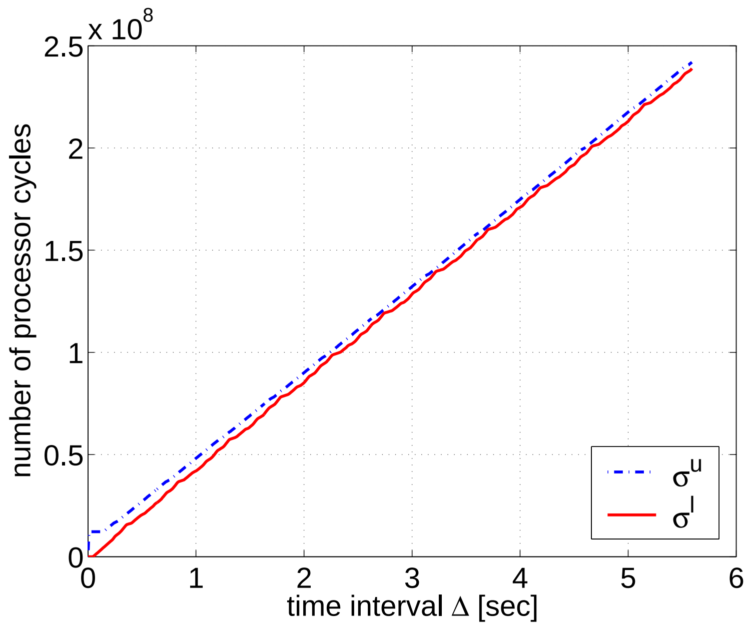


Figure 5.11: An illustration of the service bounds σ for a longer time interval.

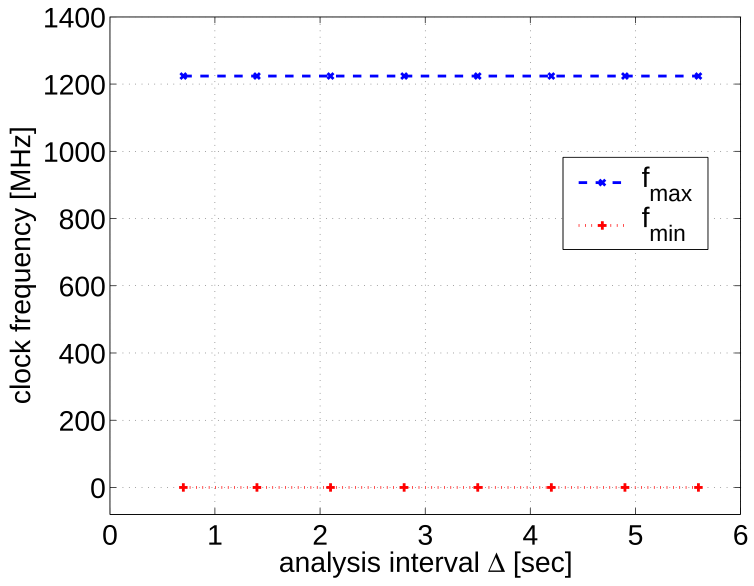


Figure 5.12: The frequency ranges computed for different values of the analysis interval.

Chapter 6

System Design Case II: Generalized Rate Analysis

In this chapter, we apply our analytical framework to specifically address a problem which we refer to as the *rate analysis problem*. Given a multiprocessor architecture and a multimedia application that has been partitioned and mapped onto it, the problem we aim at is to determine *tight bounds* on the rates at which different multimedia streams can be fed into this architecture. This is an important issue since when a stream arrives at a rate that is higher than a certain upper bound, this may lead to buffer overflow in the architecture. This problem is especially acute when dealing with architectures for portable devices (such as PDAs and portable audio/video players) which have a very limited on-chip buffer memory. On the other hand, when the stream arrives at a lower rate compared to a specified threshold, the quality of the output might suffer as well, i.e. the quality-of-service (QoS) constraints associated with the application may be violated. The goal of our rate analysis is precisely to compute these upper and lower bounds; this can help designing wireless interfaces and suitable buffering and traffic shaping mechanisms for multimedia streams.

The main difficulties associated with the rate analysis problem stem from (i) the high data-dependent variability in the execution time of multimedia tasks [79], (ii) the burstiness of on-chip traffic arising from multimedia processing on multiprocessor architectures [87] and (iii) the presence of on-chip buffers and different scheduling algorithms implemented on the different architectural components. As a result, the rate analysis problem should *not* be restricted to computing a constant "long-term" arrival rate of a multimedia stream. It

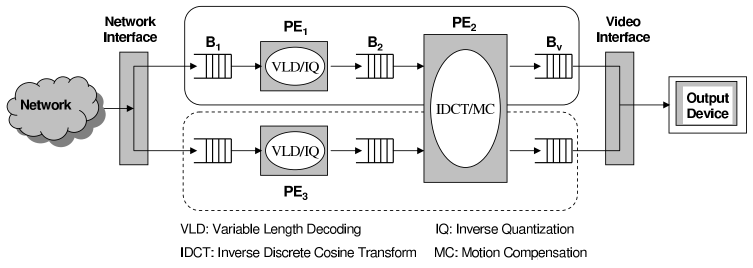


Figure 6.1: An MpSoC platform processing two concurrent MPEG-2 streams for a PiP application.

should be rather concerned with computing the allowable burstiness of a stream at different time scales. The wireless interface design and buffering mechanism mentioned above depend on these computed bounds on the burstiness.

Another use of rate analysis arises in the context of IP-based design of media processing architectures. Consider an architecture made up of different IP processor cores, where each such core runs some multimedia tasks. The stream to be processed gets processed at the first core and then, the partially processed stream, enters the next core for further processing. Two such processor cores might communicate via a bounded buffer (see Figure 6.1). Since designers usually treat these IP cores as black boxes, two cores may be connected if the rate which the partially processed stream comes out of the first core “matches” the rate at which a stream can be fed into the second processor core. Stated differently, the upper and lower bounds on the rate associated with the second core must “enclose” the output rates associated with the first core. We will show a concrete example in Section 6.1.

In the context of SoC platform configuration, the bounds returned by rate analysis depend on architectural parameters such as the amount of on-chip memory available, clock frequencies of different processors and bus arbitration policies. Further, these bounds also depend on application characteristics, including how the application is partitioned and mapped onto the architecture. If the input stream rates are dictated by the environment, a designer has to tune the platform configuration such that these rates can be supported by the architecture. The rate analysis framework that we present in this chapter can help a system designer precisely to solve this problem and also identify all the tradeoffs involved.

Although *scheduling* of multimedia streams has been extensively studied by both, the multimedia and the real-time systems communities, the rate analysis problem has not been addressed in sufficient detail, especially in the context of multimedia processing on multiprocessor architectures. Very restricted and simpler versions of this problem have however been studied before in the general domain of embedded systems design (e.g. computing maximum execution rates of concurrent processes interacting through synchronization). We believe that our work can generalize some of these previous results on rate analysis, studied in slightly different contexts. In addition, they can lead to new insights into less-studied problems, like sensitivity of schedulability analysis on input parameters, and eventually also inspire new techniques for solving them.

The rest of this chapter is organized as follows. In the next section we formally state our problem and introduce the necessary mathematical tools. In Section 6.2 we present our rate analysis approach. This is followed by our experimental results in Section 6.3. We discuss some related work in Section 6.4 and finally summarize in Section 6.5.

6.1 Problem Formulation

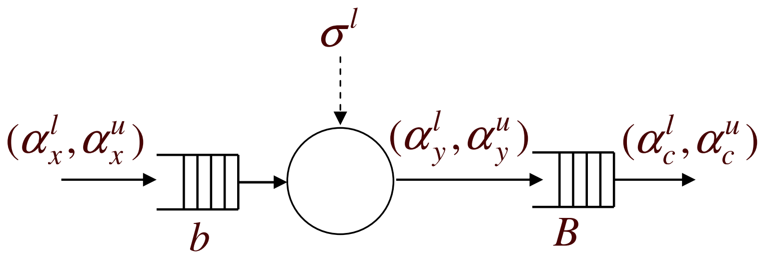
In this chapter, we consider the following system-level view of an MpSoC platform, as shown in Figure 6.1. This figure illustrates a picture-in-picture (PiP) application where two concurrent video streams are being processed by the platform architecture. An MPEG-2 decoder application is partitioned and mapped onto three processing elements PE_1, PE_2 and PE_3. The VLD and IQ tasks of the decoder application have been mapped onto PE_1 and also replicated on PE_3. Each of these two PEs process a different stream. PE_2, on the other hand, implements the IDCT and MC tasks and processes both the streams. A scheduler implemented on PE_2 schedules these streams, probably using different QoS parameters for each stream. The stream corresponding to the main window in the PiP application might be associated with a higher frame rate and resolution and will generate a higher workload on PE_2, compared to the stream associated with the secondary window. When a user switches off the PiP mode, the stream associated with the secondary window is switched off and all the processor cycles in PE_2 are used for the main stream. In this platform architecture, the processing for each stream is same as that shown in Figure 3.2.

As have been discussed in previous chapters, typical design constraints that need to be satisfied in a setup like this are (i) the playout buffers should not underflow (this would result in the output device missing a frame to be displayed), and (ii) none of the buffers should overflow. Note that because of factors like congestion in the network, the bitstream arriving at the buffer B_1 might be bursty in nature. However, the amount of burstiness would also depend on the kind of processing and buffering done at the network interface. In addition to this, the number of bits consumed by the VLD/IQ task to produce one partially decoded macroblock at the output of PE_1 is also highly variable. Lastly, the number of processor cycles required in this process (i.e. to generate one partially decoded macroblock) is also variable. For many multimedia tasks, the ratio between the worst-case and the average load on the processor can be as high as a factor of 10 [77]. As a result, the stream of partially decoded macroblocks that get written into B_2 will be highly bursty in nature.

Given a scheduling policy (and its associated parameters) for PE_2, the sizes of the buffers B_2 and B_v, and the rate at which B_v is read out by the output device, we want to compute tight bounds on the rate at which the stream objects can be allowed to arrive at B_2, such that the buffer overflow and underflow constraints are satisfied.

Recall from the above discussion that the rate at which stream objects are written out by PE_1 into B_2 is highly bursty in nature. If this rate "matches" the bounds mentioned above, then all buffer constraints will be satisfied for the system. However, if these rates do not match, then certain parts of the architecture need to be tuned accordingly. This tuning might include (i) changing buffer sizes, (ii) changing the scheduler in PE_2, (iii) changing the task mapping, or (iv) modifying the network interface. On the other hand, if the bounds allowed by PE_2 are much larger than those at which PE_1 outputs stream objects, then certain buffer sizes can be reduced to save cost.

Since we are concerned with the rates of bursty streams, it is not sufficient to specify such rates solely using the "long-term" arrival rates of stream objects. We would rather want to accurately specify the amount of burstiness in a stream. Towards this we shall use the concept of VCCs. Here we will show how they can be used to represent bounds on the



σ^l
(α_x^l, α_x^u)
(α_y^l, α_y^u)
(α_c^l, α_c^u)
b
B

Figure 6.2: Processing a single stream.

burstiness of a stream. We will also use VCCs to capture the data-dependent variability in the execution requirements of stream objects and the *service* offered by a processor to a stream.

The Rate Analysis Problem: We are now ready to formally state the rate analysis problem. Let us again consider Figure 6.2. Suppose that we are given α_c^l and α_c^u, a lower bound or guaranteed service offered by the PE (i.e. σ^l), the workload curves γ^l and γ^u and the buffer sizes b and B. The rate analysis problem consists of computing the functions α_x^l and α_x^u such that the buffer B does not underflow and neither of the buffers (i.e. b and B) overflow. It can then be guaranteed that any stream whose arrival process is bounded by α_x^l and α_x^u will satisfy the buffer underflow and overflow constraints.

Note that the VCCs always capture a *class* of streams. For example, the workload curves γ^l and γ^u capture *all* possible execution traces for which *any* k consecutive stream objects require a minimum of $\gamma^l(k)$ and a maximum of $\gamma^u(k)$ processor cycles. Hence, the problem specification given above holds for not just one concrete stream, but a class of streams.

When multiple streams are being processed by a PE, as shown in Figure 6.3, we are also given a specification of the scheduler running on the PE. The problem in this case is to compute the functions α_x^l and α_x^u for each of the individual streams. Again, the computed arrival curves are required to satisfy the buffer overflow and underflow constraints.

Lastly, in the case of architectures with multiple PEs connected in a pipelined fashion, the problem is to propagate the results of the rate analysis from one PE to the next, starting from the one closest to the output device (i.e. PE_2 in Figure 6.1). The final result of such

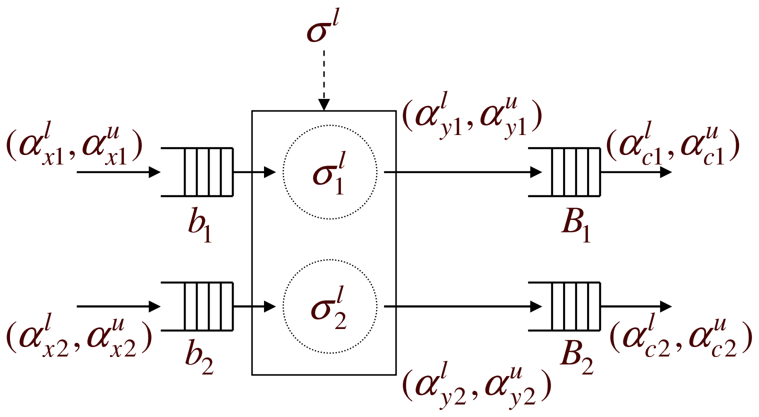

σ^l
$(\alpha^l_{x1}, \alpha^u_{x1})$
σ^l_1
$(\alpha^l_{y1}, \alpha^u_{y1})$
$(\alpha^l_{c1}, \alpha^u_{c1})$
b_1
B_1
σ^l_2
$(\alpha^l_{x2}, \alpha^u_{x2})$
b_2
B_2
$(\alpha^l_{c2}, \alpha^u_{c2})$
$(\alpha^l_{y2}, \alpha^u_{y2})$

Figure 6.3: Processing multiple streams.

an analysis will then be precise bounds on the input rate at which a stream can be fed into the platform architecture.

6.2 Rate Analysis

In this section, we first present our rate analysis framework for the case of a PE that is closest to the output device. We will then show how to extend it to the case of other PEs in the path of a stream.

6.2.1 The Single Stream Case

Let us again consider Figure 6.2. Using results from [63], it can be shown that the maximum backlog at the input buffer b is bounded by

$$\sup_{\Delta \geq 0}\{\alpha^u_x(\Delta) - \beta^l(\Delta)\} \tag{6.1}$$

where β^l can be obtained from γ^u and σ^l as discussed in Section 3.2. For the sake of notational simplicity, from hereon we will use b and B to denote both, the buffers and their respective sizes.

Using this result, the constraint that the buffer b never overflows can be stated as:

$$\alpha^u_x(\Delta) \leq \beta^l(\Delta) + b, \quad \forall \Delta \geq 0 \tag{6.2}$$

Similarly, the constraint that the playout buffer never overflows can be stated as:

$$\alpha_y^u(\Delta) \leq \alpha_c^l(\Delta) + B, \quad \forall \Delta \geq 0 \tag{6.3}$$

Let the playback delay associated with the output device be equal to t_d, i.e. the first stream object is read out from the plaout buffer B at time $t = t_d$. Thereafter, B is read out at a rate specified by the consumption bounds α_c^l and α_c^u. The time interval $t = [0, t_d)$, is often referred to as the *buffering time*. We assume that the bounds (α_c^l, α_c^u) hold over the time interval $[t_d, \infty)$ i.e. the buffering time is ignored. This is needed in order to obtain tighter bounds.

Ineq. (6.3) guarantees that the buffer B never overflows subject to the condition that it is empty at the time t_d and starts filling up from thereon. In reality, this is of course not true since stream objects are written into B during the buffering time. As a result, even if Ineq. (6.3) is satisfied, certain stream objects might be dropped. However, the maximum number of dropped stream objects can be bounded and we shall derive this bound towards the end of this subsection.

It can also be shown that $\alpha_y^u(\Delta) = (\alpha_x^u \oslash \beta^l)(\Delta)$. Using this result, Ineq. (6.3) is equivalent to

$$(\alpha_x^u \oslash \beta^l)(\Delta) \leq \alpha_c^l(\Delta) + B, \;\; \forall \Delta \geq 0$$

Using Lemma 1, this inequality can now be reformulated as:

$$\alpha_x^u(\Delta) \leq (\beta^l \otimes \alpha_c^l)(\Delta) + B, \quad \forall \Delta \geq 0 \tag{6.4}$$

By combining the Ineqs. (6.2) and (6.4), we obtain the following upper bound on α_x^u:

$$\alpha_x^u(\Delta) \leq (\beta^l(\Delta) + b) \wedge ((\beta^l \otimes \alpha_c^l)(\Delta) + B), \quad \forall \Delta \geq 0 \tag{6.5}$$

Next, we derive a lower bound on α_x^l. It may be noted that this will depend on the upper consumption bound α_c^u, the service curve β^l and the playback delay t_d. Let us fist consider the case where the playback delay $t_d = 0$. In this case, the output device has to wait for a maximum of ${\alpha_y^l}^{-1}(k) - {\alpha_c^u}^{-1}(k)$ time units until the k-th stream object is written into the playout buffer B. From this it is possible to bound the maximum time interval for which

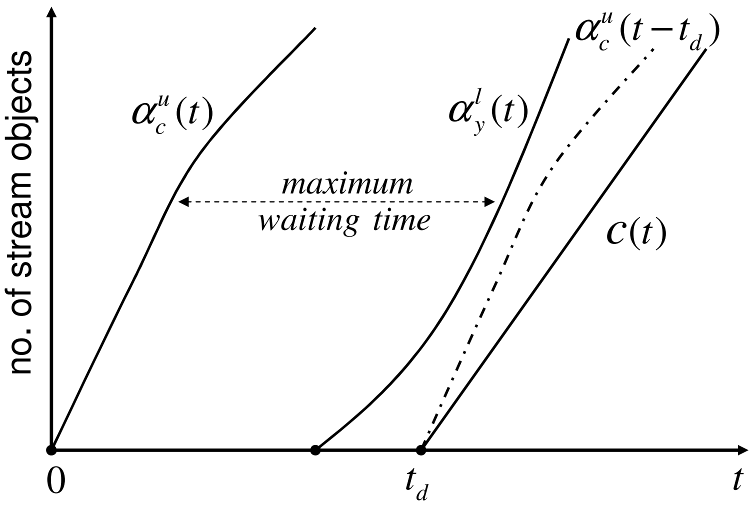


Figure 6.4: A graphical illustration of the playout buffer underflow constraint in terms of α_c^u, α_y^l and the playback delay t_d.

the output device might have to wait to read a stream object, as illustrated in Figure 6.4. This bound is given by:

$$\sup_{k\geq 0}\{\alpha_y^{l\ -1}(k) - \alpha_c^{u-1}(k)\}$$

Using this result, we have the following theorem for non-zero t_d.

Theorem 1 *Given the upper consumption bound α_c^u, the lower output bound α_y^l, and the playback delay t_d, the playout buffer B will never underflow if*

$$\sup_{k\geq 0}\{\alpha_y^{l\ -1}(k) - \alpha_c^{u-1}(k)\} \leq t_d$$

Proof: Let $C(t)$ denote the number of stream objects consumed by the output device and $y(t)$ denote the number of stream objects processed by the PE during the interval $[0, t]$. When the inequality $\sup_{k\geq 0}\{\alpha_y^{l\ -1}(k) - \alpha_c^{u-1}(k)\} \leq t_d$ holds, it follows that (see also Figure 6.4):

$$C(t) \leq \alpha_c^u(t - t_d) \leq \alpha_y^l(t) \leq y(t)$$

is also true. The guarantee that the playout buffer never underflows follows from $C(t) \leq y(t)$. $\square$

From this theorem it follows that in order for the playout buffer not to underflow, the following constraint needs to be satisfied:

$$\alpha_y^{l\ -1}(k) \leq \alpha_c^{u-1}(k) + t_d, \quad \forall k \geq 0 \tag{6.6}$$

For notational simplicity, we will use $\lambda_c^u(\Delta)$ to denote the pseudo-inverse of the function $\alpha_c^{u-1}(k) + t_d$. Ineq. (6.6) can then be written as: $\alpha_y^l(\Delta) \geq \lambda_c^u(\Delta), \forall\Delta \geq 0$. Further, it can be shown that $\alpha_y^l(\Delta) = (\alpha_x^l \otimes \beta^l)(\Delta)$. By combining this with the above inequality, we obtain that $(\alpha_x^l \otimes \beta^l)(\Delta) \geq \lambda_c^u(\Delta)$. From Lemma 1 we can then obtain the following lower bound on α_x^l:

$$\alpha_x^l(\Delta) \geq (\lambda_c^u \oslash \beta^l)(\Delta), \quad \forall\Delta \geq 0 \tag{6.7}$$

Inequalities (6.5) and (6.7) therefore give upper and lower bounds on the rate of the input stream.

Bounding the buffer overflow: Ineq. (6.3) guarantees that the playout buffer never overflows during the time interval $[t_d, \infty)$ subject to the condition that it is empty during $[0, t_d]$. In reality the assumption that B is empty during $[0, t_d]$ does not hold. However, it is possible to obtain an upper bound on the number of stream objects that can arrive within this time interval. This upper bound is the maximum number of stream objects that can overflow from the buffer as a result of the above assumption. We know that the maximum number of stream objects that can be processed within $[0, t_d]$ is $\gamma^{l^{-1}}(\sigma^l(t_d))$, and the maximum number of stream objects that can arrive at b within $[0, t_d]$ is $\alpha_x^u(t_d)$. Hence, the maximum number of stream objects that can arrive at B during $[0, t_d]$ is $\min\{\gamma^{l^{-1}}(\sigma^l(t_d)), \alpha_x^u(t_d)\}$, which is therefore equal to the maximum number of stream objects that can overflow from B.

6.2.2 The Case of Multiple Streams

In this subsection we are concerned with the case where multiple streams are being processed by a PE. We consider a PE processing two streams (see Figure 6.1) to illustrate our approach. The extension to more than two streams is straightforward. As in the single stream case, we again assume that the PE of interest is the one next to the output device. Let the playout and input buffers associated with each of the streams be of size B_n and b_n respectively, with $n = 1, 2$. Similarly, let the workload curves and consumption bounds associated with each of the streams be (γ_n^l, γ_n^u) and $(\alpha_{cn}^l, \alpha_{cn}^u)$ respectively. Finally, let the service curve offered by the PE be σ^l. The scheduler implemented on the PE divides the service σ^l among the two streams. Here we shall consider two scheduling disciplines—fixed-priority and time division multiplexing, but our approach can be used to analyze other schedulers as well. Figure 6.3 shows the problem setup that we discuss here.

6.2.2.1 Fixed-Priority Scheduling

Let the two streams being processed be s_1 and s_2, where s_1 is the higher priority stream. Since it is required that the buffer constraints associated with *both* the streams be satisfied, our derivation of the bounds on the input rates of s_1 and s_2 is based on the following reasoning. We first need to ensure that sufficient service is available for the low-priority stream for it to sustain its playout rate. The remaining service can then be offered to the high-priority stream and any unused service can again be used by the low-priority stream.

Since at most $\alpha_{c2}^u(\Delta)$ stream objects from s_2 can be consumed by the output device within any time interval of length Δ, to satisfy the playout buffer underflow constraint a minimum service of $\beta_2^l(\Delta) = \alpha_{c2}^u(\Delta)$ is required by s_2. The remaining service can then be potentially used by s_1. Note that all of this service might not be used by s_1 and whatever is leftover will then be used by s_2. Expressed in terms of the number of processor cycles, these service curves are:

$$\begin{aligned} \sigma_2^l(\Delta) &= \gamma_2^u(\beta_2^l(\Delta)) && \text{(service available to } s_2\text{)} \\ \sigma_1^l(\Delta) &= \sigma^l(\Delta) - \sigma_2^l(\Delta) && \text{(service available to } s_1\text{)} \end{aligned} \tag{6.8}$$

where γ_2^u is the upper workload curve that bounds the processor cycle demand of s_2. We assume that $\sigma^l(\Delta) - \sigma_2^l(\Delta)$ is wide-sense increasing (see Section 6.2), otherwise it is transformed into such a function.

When expressed in terms of the number of stream objects, the service curves given by Eq. (6.8) are equal to $\beta_1^l(\Delta) = \gamma_1^{u^{-1}}(\sigma_1^l(\Delta))$ and β_2^l. From these service curves it is possible to derive the bounds α_{x1}^l and α_{x2}^u on the input rates of the streams, using our results in Section 6.2.1. It should be noted that the minimum service available to s_2 may be larger than β_2^l (when s_1 does not use all the service available to it). As a result, it might seem that the playout buffer B_2 can possibly overflow even when the arrival rate of s_2 is bounded by α_{x2}^u (i.e. α_{x2}^u is not a correct upper bound since it underestimates the minimum service available to s_2). However, it can be shown that B_2 will not overflow even when the arrival rate of s_2 is equal to α_{x2}^u. To see this, let α_{y2}^u denote the upper bound on the output rate of the processed stream. Since $\alpha_{y2}^u = \alpha_{x2}^u \oslash \beta_2^l$, it follows that α_{y2}^u does not increase when β_2^l increases. Hence, Ineq. (6.3), which represents the playout buffer overflow constraint, continues to be satisfied with increasing β_2^l. It is therefore safe to compute α_{x2}^u using $\beta_2^l = \alpha_{c2}^u$. Exactly the same reasoning also holds for computing the upper bound α_{x1}^u for the higher priority stream s_1 (which is described below). Next, we derive the remaining two bounds α_{x1}^u and α_{x2}^l.

Bounding α_{x1}^u: From Ineq. (6.5), we have

$$\alpha_{x1}^u(\Delta) \leq \psi(\Delta), \quad \forall \Delta \geq 0 \tag{6.9}$$

where $\psi(\Delta) = (\beta_1^l(\Delta)+b_1) \wedge ((\beta_1^l \otimes \alpha_{c1}^l)(\Delta)+B_1)$. Now, based on our previous assumption that the stream s_1 does not use more than $\beta_1^l(\Delta)$ amount of service, we obtain that

$$\alpha_{x1}^u(\Delta) \leq \beta_1^l(\Delta), \quad \forall \Delta \geq 0$$

If the upper bound on the arrival rate of s_1 (i.e. $\alpha_{x1}^u(\Delta)$) is more than $\beta_1^l(\Delta)$ then the stream s_2 (being the lower priority stream) might not receive enough service and its playout buffer might underflow or its input buffer might overflow. Now, from the above two constraints, we obtain the following bound on $\alpha_{x1}^u(\Delta)$:

$$\alpha_{x1}^u(\Delta) \leq \psi(\Delta) \wedge \beta_1^l(\Delta), \quad \forall \Delta \geq 0 \tag{6.10}$$

However, in many cases this bound might be overly restrictive. For example, as illustrated in Figure 6.5, it might happen that for some Δ_s, $\psi(\Delta) \geq \beta_1^l(\Delta)$ when $\Delta \leq \Delta_s$ and for other values of Δ, $\psi(\Delta) < \beta_1^l(\Delta)$. Note from Figure 6.5 that it would have been possible to choose $\psi(\Delta)$ as an upper bound on $\alpha^u_{x1}(\Delta)$ had we not been concerned with the service available to the lower-priority stream s_2. But since s_2 requires a minimum of $\beta^l_2(\Delta)$ amount of service within any time interval of length Δ, $\alpha^u_{x1}(\Delta)$ now needs to be bounded by $\beta^l_1(\Delta)$ instead of $\psi(\Delta)$ for any $\Delta \leq \Delta_s$.

Note that the constraints imposed by the upper bound $\alpha^u_{x1}(\Delta)$ for small values of Δ have a greater influence on the allowable burstiness, than those imposed by larger values of Δ. As a result, the allowable burstiness in s_1 might be overly restrictive if $\beta^l_1(\Delta)$ (instead of $\psi(\Delta)$) is an upper bound on $\alpha^u_{x1}(\Delta)$ when $\Delta \leq \Delta_s$. For many applications, the high-priority stream might exhibit a higher degree of burstiness. It might also be the case that a stream is assigned a higher priority because it is more bursty. The above constraint might be especially restrictive in such cases, and therefore we would like to relax it. Towards this end, we choose a value t_s (where $t_s \leq \Delta_s$) and replace Ineq. (6.10) with the following.

$$\alpha^u_{x1}(\Delta) \leq \begin{cases} \psi(\Delta) \wedge \beta^l_1(t_s), & \forall\, 0 \leq \Delta \leq t_s \\ \psi(\Delta) \wedge \beta^l_1(\Delta), & \forall\, \Delta > t_s \end{cases} \tag{6.11}$$

The selection of t_s clearly involves a tradeoff between the allowable burstiness in s_1 and the service available to the lower-priority stream s_2 over intervals of length $\Delta \leq t_s$. A consequence of the reduced service available to s_2 is that its input buffer might *overflow*. We address this issue later in this section. It might seem that the playout buffer of s_2 may also *underflow*. However, note that the service available to s_2 over time intervals of length Δ larger than t_s continues to be lower bounded by $\beta^l_2(\Delta)$. Hence, from our results in Section 6.2.1, it is still possible to get a valid lower arrival bound α^l_{x2} for s_2 such that its playout buffer never underflows (given a sufficiently enough non-zero playback delay).

Bounding α^l_{x2}: Clearly, the higher-priority stream s_1 can consume at most $\alpha^u_{x1}(\Delta)$ amount of service within any time interval of length Δ. Now, recall from the above discussion that s_1 will not consume more than $\beta^l_1(\Delta)$ amount of service over any $\Delta > t_s$ (follows from Ineq. (6.11)). Hence, the service available to the lower-priority stream s_2 is lower bounded

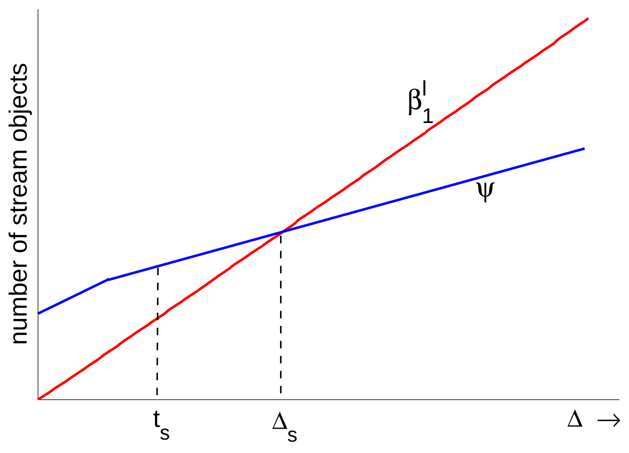


Figure 6.5: Illustration of deriving an upper bound on α^u_{x1}.

as follows.

$$\widehat{\beta}^l_2(\Delta) = \begin{cases} \zeta(\Delta), & \forall\, 0 \le \Delta \le t_s \\ \beta^l_2(\Delta), & \forall\, \Delta > t_s \end{cases}$$

where $\zeta(\Delta) = \max\{0, \gamma_2^{u^{-1}}(\sigma^l(\Delta) - \gamma_1^u(\alpha^u_{x1}(\Delta)))\}$. $\zeta(\Delta)$ represents the minimum service available for s_2 when the arrival rate of s_1 is upper-bounded by α^u_{x1} (see Ineq. (6.11)). Using $\widehat{\beta}^l_2(\Delta)$ as the service available to s_2, we can now compute α^l_{x2} following our results described in Section 6.2.1. When the service available to s_2 is greater than $\widehat{\beta}^l_2(\Delta)$, clearly the playout buffer underflow constraint would still be satisfied with α^l_{x2} as the lower bound on the arrival of s_2. Hence, it is safe to compute α^l_{x2} using $\widehat{\beta}^l_2(\Delta)$ (in terms of respecting the playout buffer underflow constraint).

A summary of the bounds we obtained so far on the arrival rates of the high- (s_1) and the low-priority (s_2) streams is given in Table 6.1. In this table, λ^u_{c1} and λ^u_{c2} denote the term λ^u_c in Ineq. (6.7), in the context of the streams s_1 and s_2 respectively.

Bounding the buffer overflow: As mentioned above, the input buffer b_2 associated with s_2 (see Figure 6.3) might overflow. However, we can bound the number of stream objects that may be dropped at b_2. First, we prove that b_2 might only overflow during the time interval $[0, t_s]$.

input arrival bounds	values
$\alpha^l_{x1}(\Delta)$	$(\lambda^u_{c1} \oslash \beta^l_1)(\Delta)$
$\alpha^u_{x1}(\Delta)$	$\begin{cases}\psi(\Delta) \wedge \beta^l_1(t_s), & \forall\, 0 \leq \Delta \leq t_s \\ \psi(\Delta) \wedge \beta^l_1(\Delta), & \forall\, \Delta > t_s\end{cases}$
$\alpha^l_{x2}(\Delta)$	$(\lambda^u_{c2} \oslash \widehat{\beta}^l_2)(\Delta)$
$\alpha^u_{x2}(\Delta)$	$(\beta^l_2(\Delta) + b_2) \wedge ((\beta^l_2 \otimes \alpha^l_{c2})(\Delta) + B_2)$

Table 6.1: Summary of the input arrival bounds.

Proof: Let $x_2(t)$ denote the number of stream objects that arrive at b_2 during the time interval $[0, t]$. When $t > t_s$, the maximum backlog at b_2 at time t is equal to $x_2(t) - \widehat{\beta}^l_2(t) \leq \alpha^u_{x2}(t) - \beta^l_2(t) \leq b_2$ (follows from Ineq. (6.2)). Since the remaining service for s_2 is at least equal to $\widehat{\beta}^l_2$, it implies that the buffer b_2 never overflows during the time interval $[t_s, \infty]$. Hence, b_2 might only overflow during the time interval $[0, t_s]$. □

As discussed in Section 6.2.1 (see Eqn. (6.1)), the maximum backlog at b_2 within $[0, t_s]$ is bounded by

$$\sup_{0\leq\Delta\leq t_s} \{\alpha^u_{x2}(\Delta) - \widehat{\beta}^l_2(\Delta)\}$$

Hence, the maximum number of stream objects that can be dropped at the input buffer b_2 (over the time interval $[0, \infty)$) is equal to $\sup_{0\leq\Delta\leq t_s}\{\alpha^u_{x2}(\Delta) - \widehat{\beta}^l_2(\Delta)\} - b_2$.

Similar to the single stream case, the playout buffers of s_2 and s_1 might also overflow. Suppose that the playback delay associated with s_2 is t_{d2} and that associated with s_1 is t_{d1}. It can be shown that the maximum number of processor cycles available to the low-priority stream s_2 (after processing s_1) within any time interval of length Δ is equal to $\sigma^u_2(\Delta) = \sup_{0\leq\tau\leq\Delta}\{\sigma^l(\tau) - \gamma^l_1(\alpha^l_{x1}(\tau))\}$. Hence, the maximum number of stream objects that may be dropped from the playout buffer associated with s_2 (also over the time interval $[0, \infty)$) is equal to $\min\{\gamma^{l\,-1}_2(\sigma^u_2(t_{d2})), \alpha^u_{x2}(t_{d2})\}$. For the high-priority stream s_1, the maximum service available to it during the time interval $[0, t_{d1}]$ is $\sigma^l(t_{d1})$. Hence, the maximum number of stream objects that can be dropped from its playout buffer is bounded by $\min\{\gamma^{l\,-1}_1(\sigma^l(t_{d1})), \alpha^u_{x1}(t_{d1})\}$.

The bounds on the buffer overflow that we derived above are summarized in Table 6.2.

buffer	maximum number of dropped stream objects
b_2	$\sup_{0\leq\Delta\leq t_s}\{\alpha^u_{x2}(\Delta) - \widehat{\beta^l_2}(\Delta)\} - b_2$
B_2	$\min\{{\gamma^l_2}^{-1}(\sigma^u_2(t_{d2})), \alpha^u_{x2}(t_{d2})\}$
B_1	$\min\{{\gamma^l_1}^{-1}(\sigma^l(t_{d1})), \alpha^u_{x1}(t_{d1})\}$

Table 6.2: Summary of the bounds on buffer overflow.

We would once again like to point out that all of these bounds correspond to the maximum number of stream objects that may be dropped over the time interval $[0, \infty)$, i.e. these are the *total* number of stream objects that can ever be dropped. This result is counter intuitive, because these bounds do not depend on the length of the audio/video clip.

6.2.2.2 Time Division Multiplexing

Analyzing a time division multiplexing scheduler is similar to the technique used for the single stream case. If $\sigma^l(\Delta)$ is the guaranteed service offered by the PE within any time interval of length Δ, then the service offered to the two streams are:

$$\begin{aligned} \sigma^l_1(\Delta) &= \tfrac{w_1}{w_1+w_2}\cdot\sigma^l(\Delta) \\ \sigma^l_2(\Delta) &= \tfrac{w_2}{w_1+w_2}\cdot\sigma^l(\Delta) \end{aligned} \tag{6.12}$$

where w_1 and w_2 are the weights associated with the two streams by the scheduler. When expressed in terms of number of stream objects, these bounds translate to $\beta^l_1(\Delta) = \gamma^{u-1}_1(\sigma^l_1(\Delta))$ and $\beta^l_2(\Delta) = \gamma^{u-1}_2(\sigma^l_2(\Delta))$. Bounds on the arrival rates of these two streams can then be computed by following the exactly same procedure as that described in Section 6.2.1.

6.2.3 Multiple Processing Elements

Our view of multimedia processing on a multiprocessor System-on-Chip platform, as outlined in Section 6.1, consists of multiple PEs processing any stream in a pipelined fashion. Between any two PEs a FIFO buffer stores the partially processed stream. The last PE in the path of a stream writes out the fully processed stream into the playout buffer, which is read out by an output device. The derivation of the bounds on the arrival rate of a stream, that we presented so far, was only concerned with this last PE, which feeds the playout buffer. Recall that the computed bounds pertain to the maximum and minimum rates at which a stream can arrive at the (input) buffer at the input to this PE. The constraints that the computed bounds were required to follow were (i) the playout buffer should not underflow, and (ii) none of the buffers should overflow. Among the inputs to our rate analysis problem were bounds on the consumption rate by the output device from the playout buffer, specified as upper and lower arrival curves (the consumption bounds (α_c^l, α_c^u)).

Let us now consider the PE (e.g. PE_1 in Figure 6.1) adjacent to this last PE in the path of the stream. To compute the bounds on the arrival rate of a stream at this PE, the input bounds computed for the downstream PE (i.e. PE_2 in Figure 6.1) serve as output bounds for this PE (i.e. the processed stream coming out of this PE must satisfy these bounds). The only buffer constraint that needs to be satisfied in this case is that the buffer at the input of this PE (B_1 in Figure 6.1) should not overflow. Deriving the input bounds on the arrival is therefore much more simpler than the case we considered above. This is because, only the following two constraints need to be satisfied: (i) Ineq. (6.2), and (ii) the bounds on the processed stream must be constrained by the input bounds computed for the adjacent downstream PE.

This process of computing the bounds on the arrival rate of stream is cascaded to all the upstream PEs, until the first PE in the path of a stream is encountered. The input bounds computed for this PE therefore serve as bounds on the arrival rate of a stream to be processed by the platform architecture.

6.3 Experimental Evaluation

We validated our analytical approach using a number of detailed simulations. Towards this end, we implemented a transaction-level model of the platform architecture shown in Figure 6.1 using SystemC [84]. The on-chip PEs were modeled using the SimpleScalar instruction set simulator [7], in which we used the *sim-profile* configuration and the PISA instruction set.

We modeled each video stream at the macroblock granularity. For any given video clip, we first simulated its execution (decoding) using SimpleScalar and obtained execution time traces of the VLD, IQ, IDCT and MC tasks. These traces record the execution requirement (in processor cycles) of each macroblock belonging to the video clip, for each of the above tasks. Based on these traces and the constant bitrate at which the video clip (which is a compressed bitstream) is fed into PE_1 or PE_3 (in Figure 6.1), it is possible to determine the arrival pattern of the stream at the input of PE_2 (i.e. at the buffer B_2) and also at the playout buffer (i.e. buffer B_v). For this we used the SystemC-based transaction-level model of the architecture, which was also used to model the scheduling policy on PE_2 when multiple streams (two in this case) are processed by the architecture. From the SystemC simulation, we measured the fill levels of the different buffers for any given video clip(s).

To validate our approach, we first compute bounds on the arrival rate of a stream at the input of PE_2 (e.g. at the buffer B_2). We then show using simulation that video clips which respect these bounds satisfy the buffer overflow and underflow constraints. At the same time, clips which do not respect the computed bounds, either result in buffer overflow or underflow, thereby showing that the computed bounds are not overly pessimistic. Note that these bounds are non-trivial, in the sense that they precisely capture the allowed burstiness in a stream. Obtaining them using purely simulation-based techniques is certainly not possible, due to the exhaustive simulation time involved. As discussed in the beginning of this chapter, these bounds can provide useful insights helpful for tuning the platform architecture (e.g. determining the optimal clock frequency of PE_1 and also designing the input network interface).

Recall that one of the inputs to our analytical framework, is the workload curve $\gamma(k)$ specifying lower and upper bounds on the number of processor cycles required by any k consecutive stream objects. Clearly, for the bounds on the arrival rate of a stream–that are computed by our approach–to be useful, they should hold good for a *class* of streams or video clips, not for just a single video clip. As an example, a *class* might be all video clips having the same bitrate and frame resolution. Therefore, the workload curve $\gamma(k)$ that we use as an input, should also specify the workload demand of the class of video clips we are interested in.

For our PiP application (see Section 6.1) that we used in our experiments, we chose two classes of video clips–those that have high motion content and second being made up of still images. The former class of clips are to be displayed in the main window of the PiP application; they are representative of usual video clips like movies. The latter class is representative of text messages or similar information about the main window being displayed in the secondary window of the PiP application. In what follows, for ease of exposition, we drop the term *class* when we talk about bounds on arrival rates; these bounds are always expected to hold for a class of streams and not just a single stream.

To obtain the workload curves corresponding to the above two classes, we simulated the execution of a set of MPEG-2 video clips using SimpleScalar, as mentioned above. We then analyzed the resulting execution time traces for the IDCT and MC tasks (which are mapped onto PE_2) and derived the bounds γ^l and γ^u (see Section 6.1). This procedure follows a recently developed technique described in [62]; we refer the interested reader to this paper for further details. The video clips for both these classes were encoded using a constant bitrate of 8 Mbps; they had a frame resolution of 704×576 pixels and a playback rate of 25 frames per second. Typically, the video clips displayed in the secondary window of a PiP application would have a lower resolution and bitrate than those displayed in the main window. However, for simplicity reasons, we decided to distinguish between the two classes only on the basis of their content (i.e. motion versus still videos).

For reporting our experimental results, we denote the computed arrival rates of a stream at the input of PE_2 using (α_x^l, α_x^u). To validate these bounds, we compare them with similar bounds obtained from simulation–we denote these bounds using (α_m^l, α_m^u) (where the subscript m denotes "measured"). Towards this end, we first record the trace of arrival times of partially decoded macroblocks at the input of PE_2 and then analyze these traces to obtain the bounds (α_m^l, α_m^u) (exactly as the workload curves γ^l and γ^u were derived). To measure the fill levels of buffers, if B_s is the specified buffer size and B_d is the computed upper bound on the number of stream objects that might be dropped (see e.g. Table 6.2) then $B_s + B_d$ is an upper bound on the maximum buffer fill level, which we want to validate using simulations. Similarly, the fill level of a playout buffer should always be greater than

class	scenario	input buffer size (mb.)	playout buffer size (mb.)	video clip	violation ?
motion	1	4000	5600	A	no
	2	4000	5600	B	yes
	3	3500	8000	C	no
still	4	5000	3000	D	yes
	5	4000	5600	E	no
	6	3000	3000	F	no

Table 6.3: Scenarios for the single stream case.

sched. policy	scenario	input buffer size (mb.)	playout buffer size (mb.)	video clip	violation ?
FPS	FPS1	4000	5600	A	no
	FPS2	3000	3000	F	no
TDM	TDM1	4000	5600	A	no
	TDM2	3000	3000	F	no

Table 6.4: Scenarios for the multiple streams case.

0 in order to satisfy the underflow constraint.

For our experiments, we used a selection of *scenarios* shown in Tables 6.3 and 6.4. Each scenario is specified by a class (of video clips), the input and playout buffer sizes and a video clip belonging to the class. The bounds on the arrival rates are computed from the class information and the buffer sizes. These are compared with the simulation results based on the buffer sizes and a concrete clip belonging to the class. For all the experiments, we run PE_2 at a constant frequency. Hence, the service offered by it can be represented as $\sigma^l(\Delta) = c \cdot \Delta$, where c is the frequency.

6.3.1 The Single Stream Case

In this case, the PiP mode is switched off. For clarity of presentation, instead of plotting the functions α_x^l, α_x^u, etc. directly, we plot the differences $\alpha_x^u - \alpha_x^l$, $\alpha_m^u - \alpha_x^l$ and $\alpha_m^l - \alpha_x^l$. Clearly, the arrival process of a video clip, which is captured in (α_m^l, α_m^u), violates the analytically computed bounds (α_x^l, α_x^u) whenever $\alpha_m^u - \alpha_x^l$ or $\alpha_m^l - \alpha_x^l$ crosses $\alpha_x^u - \alpha_x^l$ or goes below 0.

These plots are shown in Figure 6.6, 6.7 and 6.8 for three different scenarios. The same figures also show the fill levels of the input and the playout buffers i.e. B_2 and B_v in Figure 6.1. Note that for Scenario 1, the measured arrival patterns satisfy the analytically computed bounds (subfigure (a)). For this scenario, the measured fill levels of the input buffer (subfigure (b)) and the playout buffer (subfigure (c)) are less than the computed upper bounds. Also note that beyond the playback delay, the playout buffer does not underflow.

In Scenario 2, the measured upper arrival curve α_m^u violates the computed upper bound α_x^u. In this case, the measured buffer fill levels are greater than the sum of the specified buffer sizes and the upper bounds on the number of macroblocks that might be dropped. This is indicated as *buffer overflow* in Figure 6.7.

Finally, in Scenario 4, the measured lower arrival curve α_m^l violates the computed lower bound α_x^l. In this case, the simulation results show that the playout buffer underflows. Note that for all the scenarios, the input buffer sometimes underflows. However, this does not affect the performance of the system and we also do not specify it as a constraint.

Figure 6.9 shows the computed bounds on the buffer fill levels and the measured fill levels obtained using simulation, for all the six scenarios. From Table 6.3, note that apart from Scenarios 2 and 4, the measured arrival bounds always satisfy the computed bounds. Figure 6.9 confirms that it is only for these two scenarios that buffers either overflow or underflow, thereby validating our proposed approach.

We also compare with the bounds on the arrival rate computed using a simple best-/worst-case characterizations commonly used in the real-time systems domain. Let e_{min} and e_{max} denote the minimum and the maximum number of processor cycles required by any single stream object belonging to a sequence. The minimum and the maximum number of processor cycles that might be required by any k consecutive stream objects within the given sequence are modeled by $k \times e_{min}$ and $k \times e_{max}$. Figure 6.11 shows that the computed bounds (denoted by $(S_{\alpha_x^l}, S_{\alpha_x^u})$) using this simple modeling scheme are very pessimistic, compared to our computed bounds (denoted by (α_x^l, α_x^u)) using workload curves. Our scheme allows the bursts on the arrival rate to be as great as (α_x^l, α_x^u), while following the simple scheme the bursts can only be as great as $(S_{\alpha_x^l}, S_{\alpha_x^u})$. It is thus shown that our scheme using VCCs provides better bounds on the arrival rate.

6.3.2 The Case of Multiple Streams

As mentioned above, in this case the main window of the PiP application displays a regular video clip and the secondary window displays a still video (e.g. text information, program menu, etc.). We have experimented with two different scheduling policies on PE_2–fixed-priority and time division multiplexing. In Table 6.4, for the fixed-priority scheduler (i.e. FPS), the first class of streams is denoted as FPS1; this is the class of regular video clips. From this class, video clip A was used for the simulation. FPS2 denotes the class of still video clips, from which clip F was used for simulations. For each of the two streams being processed by PE_2, the corresponding buffer sizes are also specified in this table. The class FPS1 was assigned higher priority.

For the time division multiplexing scheduler (i.e. TDM), again TDM1 denotes the class of regular video clips and TDM2 denotes the class of still video clips. We associated weights 0.6 and 0.4 with the classes TDM1 and TDM2 respectively.

The results obtained for both these schedulers are summarized in Figure 6.10. From this figure, note that all the buffer overflow and underflow constraints are satisfied.

Our approach can also aid in selecting the TDM weights associated with the two streams being processed by PE_2. For the two classes of streams TDM1 and TDM2, Figure 6.12 shows the plot of $\alpha_x^u - \alpha_x^l$ for different values of TDM weights. In this figure, w_1/w_2 denotes the weights w_1 and w_2 associated with the streams TDM1 and TDM2 respectively. From Figure 6.12(a), note that the allowed burstiness in the stream TDM1 increases as the value of w_1 is increased ($w_1 + w_2 = 1$). Similarly, Figure 6.12(b) plots how the allowed burstiness in TDM2 increases as w_2 is increased. Note that as the service provided to a stream increases beyond a certain point, the allowed burstiness does not increase any more. For the stream TDM1, this happens when the ratio w_1/w_2 increases beyond $0.62/0.38$. Similarly, for TDM2 this happens when w_1/w_2 is less than $0.50/0.50$.

6.4 Related Work

The rate analysis problem has been studied before in the embedded systems domain, albeit in a different context. Broadly speaking, the setup considered before [59, 29] consists of a collection of concurrently executing embedded systems components/processes that interact through synchronization messages. The problem is to compute bounds on the execution rates of these processes, given certain resource constraints. Alternatively, given a number of rate constraints, the problem is to efficiently check if these constraints are consistent. Often, it is required to check these constraints in an interactive fashion and hence the emphasis in such cases has been on appropriate tool support.

In this chapter we were concerned with the rate analysis problem in the context of processing multiple concurrent multimedia streams. Rather than computing bounds on the execution rates of a process [59, 29], our aim has been to compute the allowable bursts in a multimedia stream over different time scales. Such bursts are specified as *arrival curves* which bound the minimum and maximum number of data items or events that can arrive at the system within any specified time interval length. We believe that our results can be combined with the previous work [59, 29] to model and analyze reactive systems consisting of a number of interacting processes that are triggered by bursty event streams. More specifically, the previous work [59] is only concerned with a periodic model, where the different interacting processes execute in a periodic fashion. As a first step, this restriction can be removed using our event model which allows the specification of arbitrary, but bounded bursts.

Within the real-time systems area, there has been a growing interest in the problem of computing the *parameter space* for which a system becomes schedulable. A recent paper [36] addressed the problem of computing the end-to-end feasibility regions of distributed aperiodic task systems under fixed-priority scheduling. The goal here was to compute the multidimensional space–with each dimension as the utilization of a resource–within which the system meets certain end-to-end deadlines. Similarly, it was addressed in [12] the problem of identifying task activation rates for fixed-priority scheduled systems that meet certain deadline constraints. The work that we presented here is in the same general direction as that of the above-mentioned two papers.

Our work has been inspired by a recent paper [61] which studied the rate analysis problem for multimedia streams. However, in contrast to our work, this paper computes the bounds on the arrival pattern of an input stream using two functions $x_{min}(t)$ and $x_{max}(t)$. Any arrival pattern $x(t)$, which is bounded by these two functions, i.e. $x_{min}(t) \leq x(t) \leq x_{max}(t)$, is guaranteed to satisfy all buffer overflow and underflow constraints (exactly as we specify here). The function $x(t)$ denotes the number of stream objects that can arrive at the system during the time interval $[0, t]$. The use of such a concrete arrival trace–rather than bounds on the burstiness, as we do here–considerably simplifies the formulation of the buffer underflow and overflow constraints. The downside of such a formulation is that the resulting bounds (x_{min} and x_{max}) are considerably more pessimistic than the bounds we have derived.

To see this, we have used Scenario 1 in Table 6.3 to analytically compute the bounds x_{min} and x_{max} based on the framework presented by Maxiaguine et al. [61]. These bounds are compared with the bounds α_x^l and α_x^u (that we obtained in this chapter) in Figure 6.13. Note that all these bounds analyze the arrival rate over the time interval $[t_d, \infty)$ in order to achieve a fair comparison. It follows from our approach that for any t, the value of $x(t)$ can be as small as $\alpha_x^l(t)$. On the other hand, $x(t)$ can only be as small as $x_{min}(t)$ if the bounds derived by Maxiaguine et al. [61] are to be used. The reason behind the bounds x_{min} and x_{max} being pessimistic is that these bounds do *not* capture the burstiness in a stream. Given a concrete arrival pattern $x(t)$, which is bounded by x_{min} and x_{max}, let α_x^l and α_x^u denote the arrival curves which bound $x(t)$. It is very likely that $\alpha_x^l(t)$ will be smaller than $x_{min}(t)$, especially for small values of t. Similarly, $\alpha_x^u(t)$ is likely to be greater than $x_{max}(t)$.

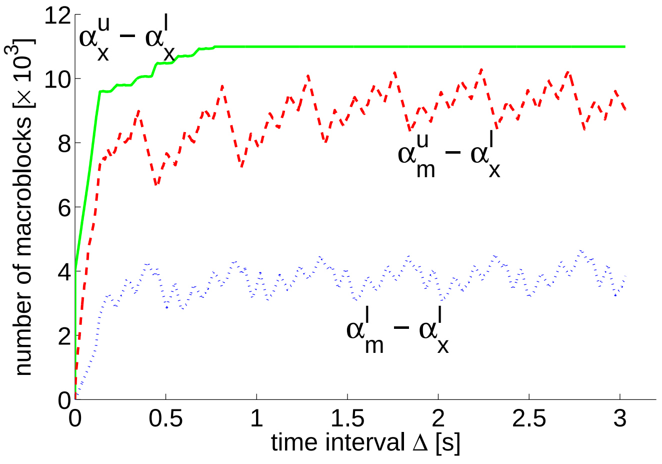
Apart from the fact that the formulation of the buffer overflow and underflow constraints are more difficult in the case we consider in this paper, we also exploit the variability in the execution requirements of a stream (captured using the workload curves). This variability is not exploited by Maxiaguine et al. [61]. However, the bounds shown in Figure 6.13 do not make use of the workload curves when deriving α_x^l and α_x^u. These bounds were computed with the service curve $\beta^l(\Delta)$ set to $C(\Delta)$ (the constant consumption rate of the stream from the playout buffer). The bounds x_{min} and x_{max} were also computed with the same $\beta^l(\Delta)$. This was done to achieve a fair comparison between the two schemes.

6.5 Summary

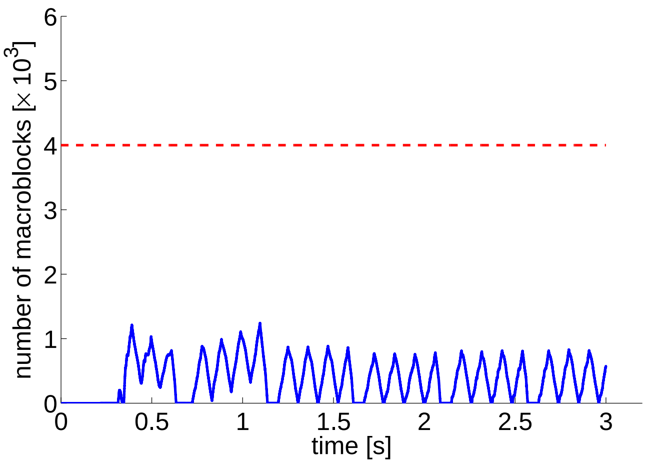
In this chapter we applied our proposed analytical framework and presented an approach for rate analysis of multimedia applications running on MpSoC platform architectures. In contrast to the recent paper by Maxiaguine et al. [61]–which inspired our work–the bounds on the arrival rate of a stream, that we derive, precisely capture the allowed burstiness. This is especially important in the context of (i) multimedia applications, since they exhibit a high degree of variability in their execution requirements [87], and (ii) such applications running on heterogeneous multiprocessor architectures, implementing different scheduling and arbitration policies [74, 76, 75].

Note that the approach we presented so far is purely functional in nature, i.e. it can not model the processing of streams where the arrival process or the service depends on the state of the system. For example, a PE might implement a protocol or a scheduler which adjusts the service provided based on the fill level of the buffer. An interesting research direction would be to extend the proposed approach to model and analyze such architectures.

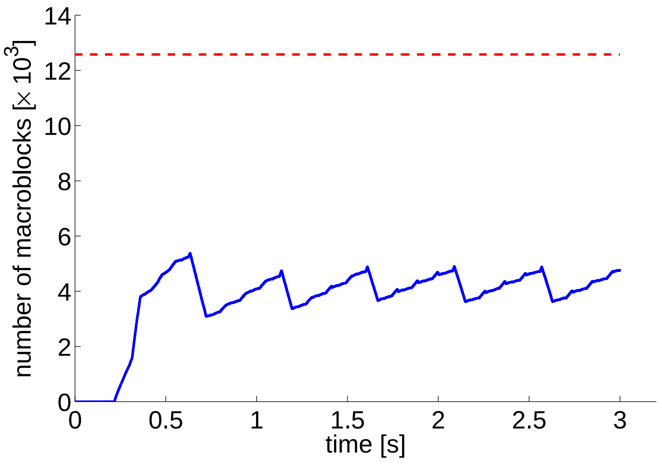
On the other hand, as mentioned in Section 6.4, it would also be worthwhile to explore possible combinations of the work by Mathur and Dasdan et al. [59, 29] with our approach. More specifically, our application model can be extended to allow for arbitrary task graphs along with deadline constraints, in addition to the buffer constraints that we addressed here.



(a)

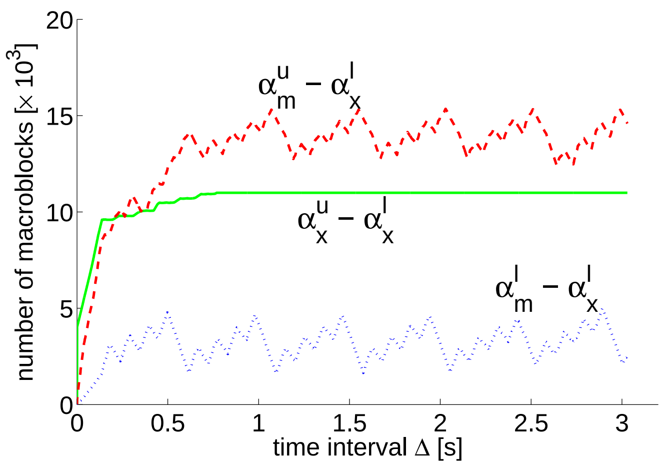


(b)

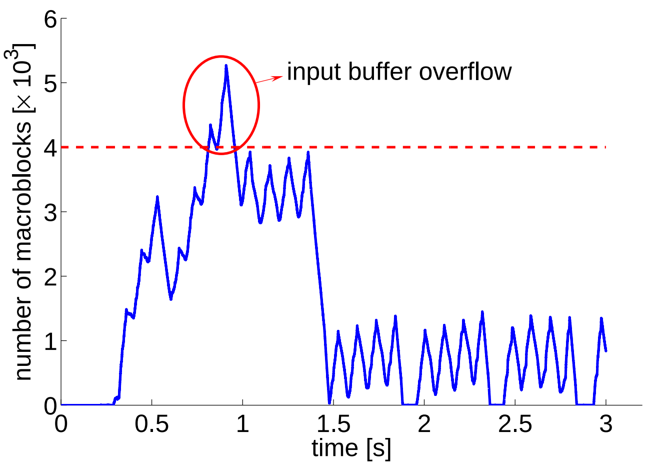


(c)

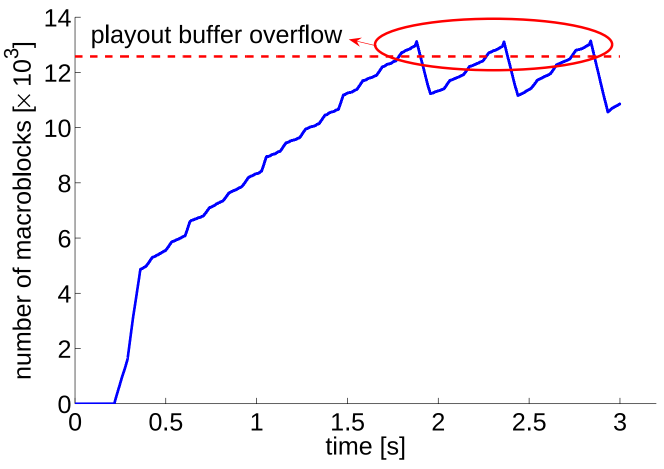
Figure 6.6: Scenario 1: (a) Computed and measured bounds on the arrival rate, (b) Measured input buffer fill level, (c) Measured playout buffer fill level.



(a)

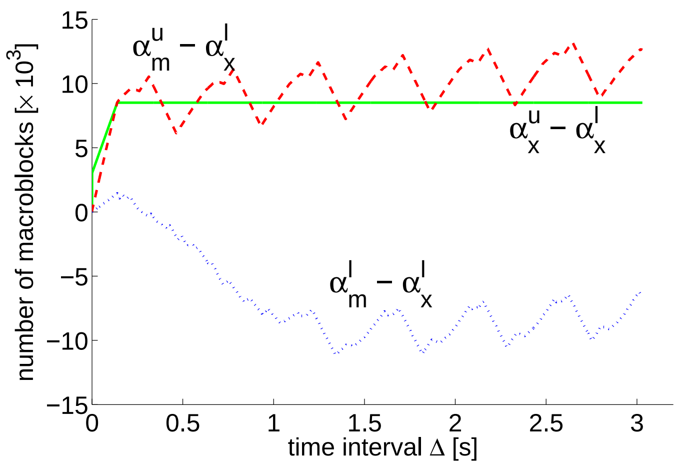


(b)

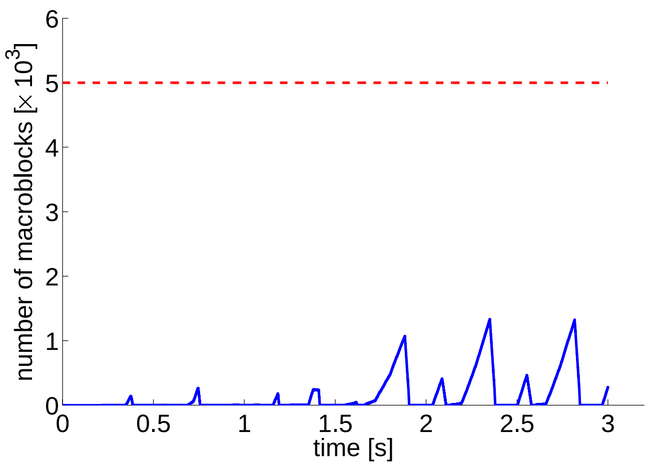


(c)

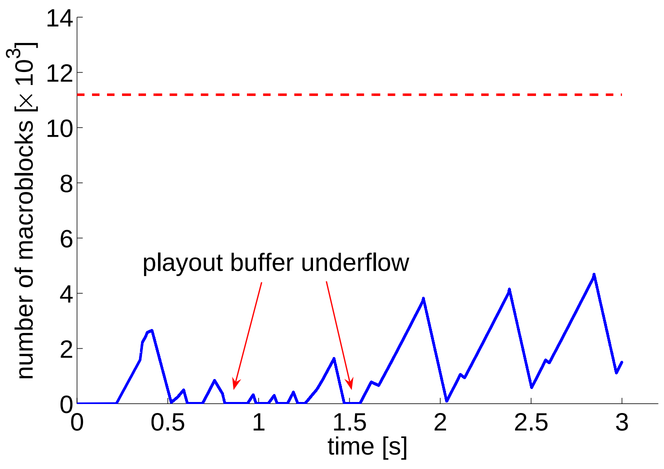
Figure 6.7: Scenario 2: (a) Computed and measured bounds on the arrival rate, (b) Measured input buffer fill level, (c) Measured playout buffer fill level.



(a)

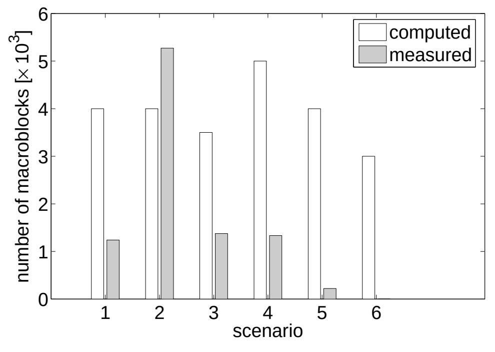


(b)

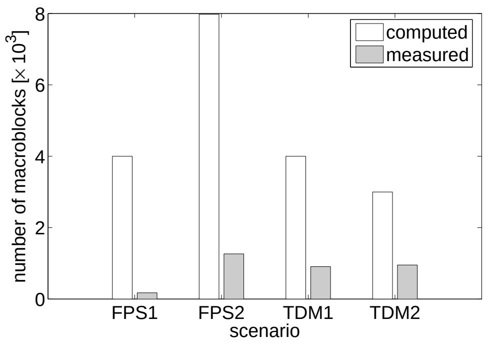


(c)

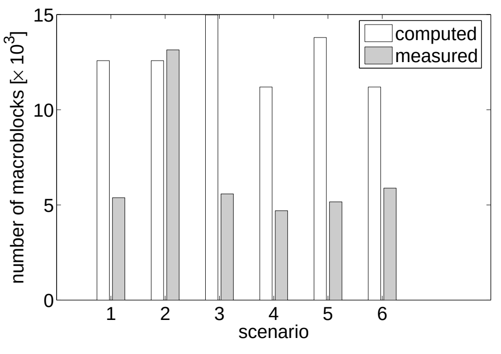
Figure 6.8: Scenario 4: (a) Computed and measured bounds on the arrival rate, (b) Measured input buffer fill level, (c) Measured playout buffer fill level.



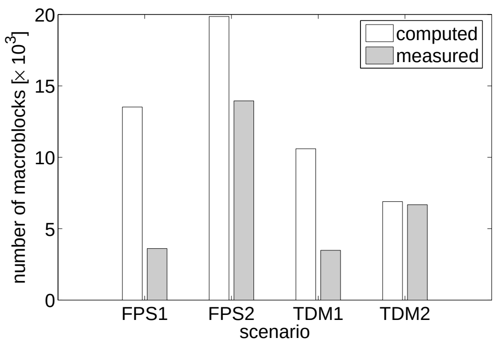
(a)



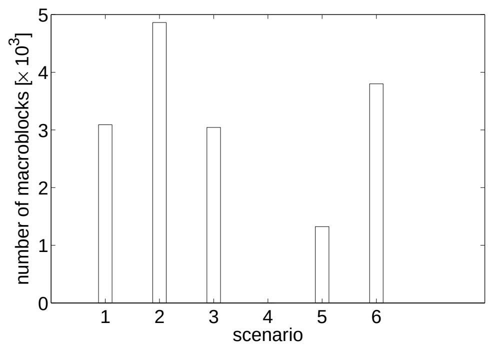
(a)



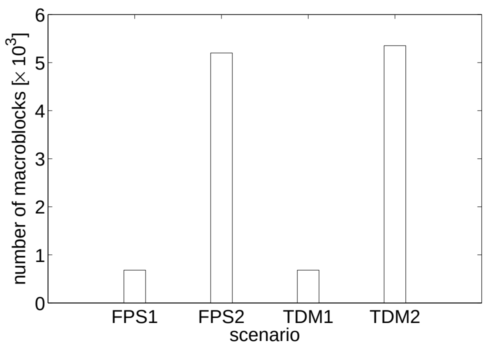
(b)



(b)



(c)



(c)

Figure 6.9: Buffer fill levels in the single stream case: (a) Computed versus measured maximum fill level of the input buffer, (b) Computed versus measured maximum fill level of the playout buffer, (c) Measured minimum playout buffer fill level.

Figure 6.10: Buffer fill levels in the multiple streams case: (a) Computed versus measured maximum fill level of the input buffer, (b) Computed versus measured maximum fill level of the playout buffer, (c) Measured minimum playout buffer fill level.

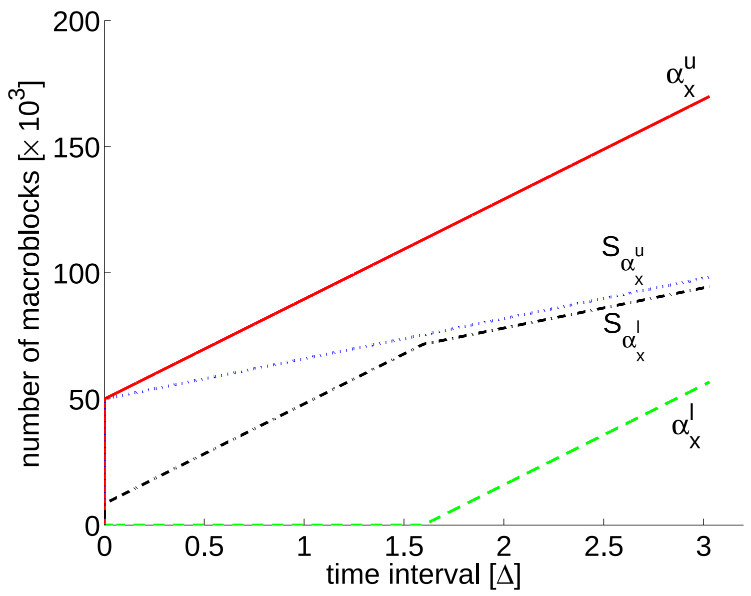
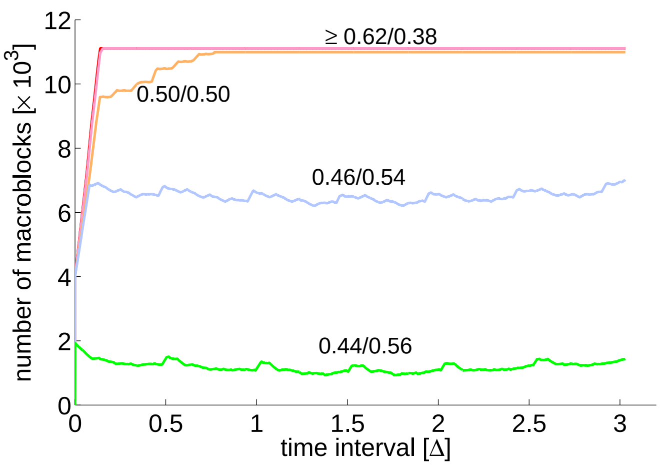
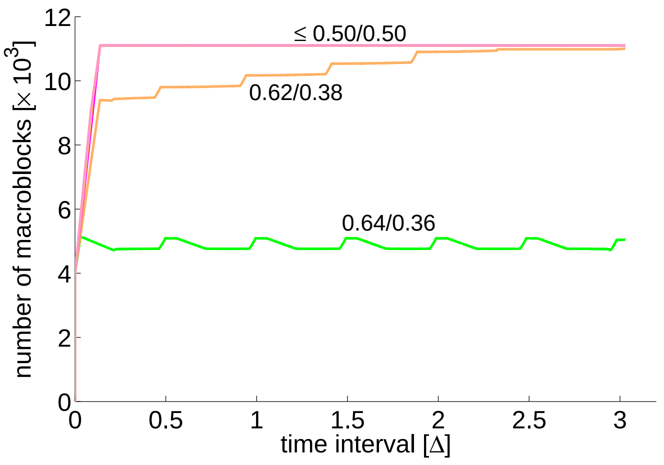


Figure 6.11: Bounds on the arrival rate computed using VCCs and a simple modeling scheme: (α_x^l, α_x^u) and $(S_{\alpha_x^l}, S_{\alpha_x^u})$.



(a) TDM1



(b) TDM2

Figure 6.12: $\alpha_x^u - \alpha_x^l$ for two scenarios, with different values w_1/w_2 for a TDM scheduler.

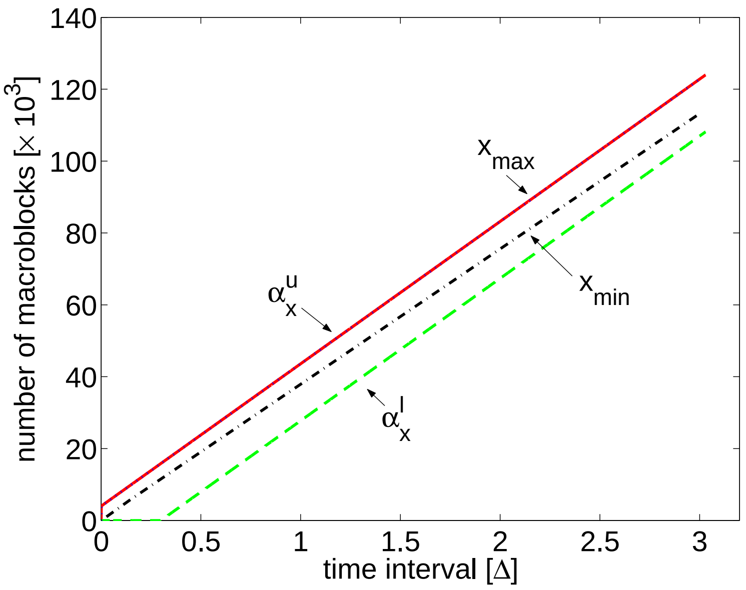


Figure 6.13: Bounds on the arrival rate of a stream (x_{min}, x_{max}) and (α_x^l, α_x^u) with playback delay value of 0.3 sec.

Chapter 7

Approximate VCCs: A New Characterization of Multimedia Workloads

To design a SoC platform architecture for a specific multimedia application, a common practice is to use a set of representative audio/video clips that would be processed by the application. The workload generated by such a representative set is then used to determine parameters such as on-chip buffer sizes, clock speeds of the different processors, bus widths and cache configurations. In the previous chapters, we have used the concept of VCCs to characterize the workload imposed by the multimedia application and developed analytical approaches for designing SoC platform architectures based on such characterization. We have known that VCCs capture the best/worst-case behaviours of multimedia processing. The designs based on VCCs (as have presented in Chapters 5 and 6) give worst-case solutions that guarantee the architectural and application constraints are satisfied in any case (even when the worst-case behaviours happen). In this chapter, we address the issue of characterizing the average-case behaviours of multimedia processing, which may improve or compensate the worst-case characterization schemes. At the same time, we conduct a preliminary study of applying this new scheme in the design process, where numbers on performance degradation can be bounded.

Multimedia workloads are known to exhibit a high variation in their resource demands. For example, the ratio of the worst-case and the average load on a processor running a

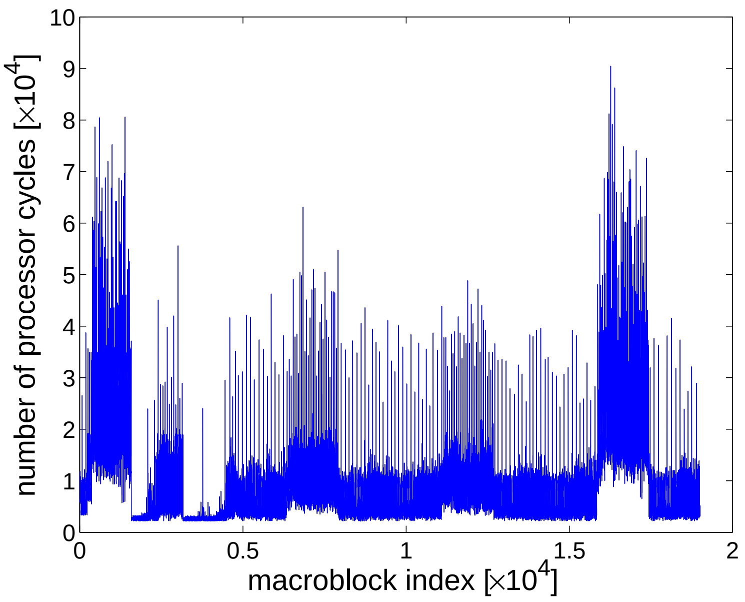


Figure 7.1: Processor cycle requirements of a sequence of macroblocks for an MPEG-2 decoder application.

multimedia task can easily be as high as a factor of 10 [77]. On the other hand, multimedia applications typically have soft real-time constraints. This allows certain tasks to miss their deadlines or a few data items to be occasionally dropped from a buffer, without significantly deteriorating the output quality. A consequence of the above two characteristics is that a worst-case analysis of multimedia workloads often lead to overly pessimistic results. At the same time, a straightforward average-case analysis does not suffice because of the high variability in the workload. Hence, appropriately characterizing multimedia workloads for system-level design is a tricky problem. Figure 7.1 shows the processor cycle requirements of a sequence of macroblocks for an MPEG-2 decoder application. The large variation in the processor cycle requirements for the different macroblocks is clearly noticeable.

To address the above problem, in this chapter we propose a new characterization of multimedia workloads, called *approximate variability characterization curves* (approximate VCCs) or ε-VCCs, that can be used to characterize the "average-case" behavior of a workload in a parameterized fashion. Towards this, we take into account the *frequency* with which the worst-case occurs and discard worst-case scenarios that do not occur frequently enough. Therefore, what we refer to as "average-case" (for the sake of simplicity), actually denotes the worst-case that occurs often enough. We quantify "often enough" using a

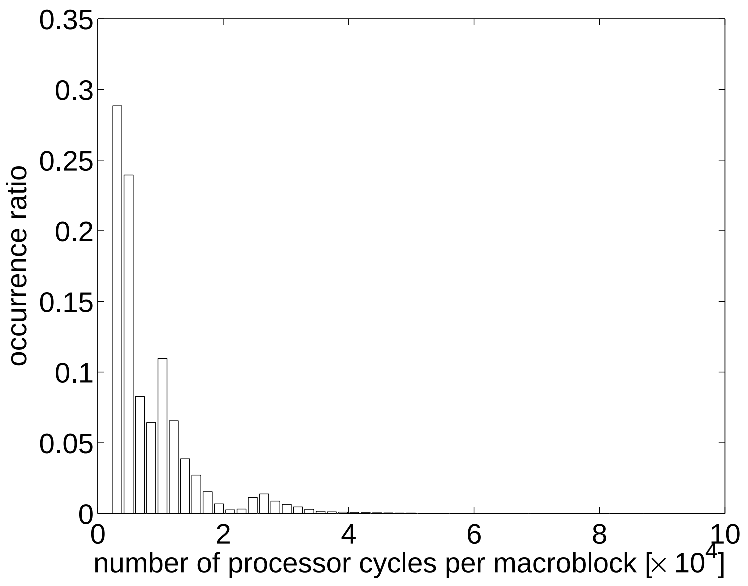


Figure 7.2: Histogram of the processor cycle demand per macroblock for an MPEG-2 video. The minimum and the maximum cycle demands are 2218 and 92247 respectively.

parameter that is specified by the designer. By ignoring worst-case scenarios that do not occur very frequently, significant amounts of resource savings are usually possible, with negligible loss in the output audio/video quality. Our proposed characterization can also be used to quickly identify the tradeoffs between the output quality and the potential resource savings. Using purely simulation-oriented techniques to determine such tradeoffs is not only expensive in terms of the simulation time involved, but is often also impractical.

As we have discussed, VCCs characterize *best*- and *worst-case* scenarios without considering the frequency with which such scenarios occur. Simulating the execution of an MPEG-2 decoder with a randomly chosen video clip shows that the worst-case processor cycle demand to decode a macroblock occurs in about 0.02% of the total number of macroblocks processed. For the execution trace in Figure 7.1, the histogram of the processor cycle demand per macroblock is shown in Figure 7.2. From this figure, it may be noted that the cycle demands of about 90% of all the macroblocks are less than half of the maximum/worst-case cycle demand of a macroblock. VCCs, as proposed in [63], would record this maximum value without taking into account the frequency of its occurrence.

An approximate VCC or ε-VCC ignores at most ε percentage of the data from the right-hand side of the histogram in Figure 7.2. The remaining data is then used to compute the worst-case scenario. As a result, all worst-case scenarios, whose cumulative frequency of occurrence is less than ε percent are ignored. Given a trace such as the one shown in Figure 7.1, we show how to bound the error corresponding to different values of ε, for typical system-level design problems. An example of this is to bound the maximum number of data items that may be dropped from a buffer, when the buffer sizing is done based on ε-VCCs. It may be noted that when buffer sizing is based on VCCs, it can be guaranteed that no data items will be dropped [63], albeit at the cost of much larger buffer sizes compared to when ε-VCCs are used. Since worst-case scenarios occur very infrequently (as discussed above), *significant savings* are achieved by using ε-VCCs, at the cost of *negligible loss* in output audio/video quality.

Related work: The concept of VCCs has its foundations in the theory of *network calculus* [16, 28]. Whereas the originally proposed network calculus may be seen as a deterministic queuing theory for analyzing communication networks, recently a number of extensions to this theory have been developed [14, 27]. These extensions are concerned with providing statistical service guarantees rather than deterministic guarantees, which often lead to resource over-provisioning. Along similar lines, Ayyorgun and Cruz [8, 9] have recently proposed a service model which allows a certain portion of network packets to be dropped based on a loss parameter. In contrast to the work presented in this chapter, they, however, concentrate on a multiplexing problem and study the necessary capacity of a multiplexer to provide deterministic service guarantees to each flow passing through it. As we already mentioned, all the above efforts focus only on the domain of communication networks, and the results obtained can not be applied to our problem setup (multimedia processing on MpSoC platforms) in any straightforward manner.

Within the embedded systems domain, the concept of *Stochastic Automatic Networks* (SANs) [66] has been proposed for average-case performance analysis of platform architectures. Whereas this is an automata-theoretic formalism, the workload characterization that we present here is purely "functional", where the "state" of the system is not modelled. The focus is primarily on modelling the variability in the arrival process and the execution demand of multimedia streams, rather than the *state* of the system processing these streams. We believe that there is a potential for integrating our work with the SAN formalism.

Organization of the chapter: To understand how ε-VCCs are defined, in Section 7.1 we formulate the concept of VCCs. This is followed by our definition of ε-VCCs in Section 7.2. In Section 7.3 we present an analytical method for bounding the error incurred while designing a system based on ε-VCCs. Experimental results which validate our method are presented in Section 7.4.

7.1 Formulation of VCCs

As we have described in Chapter 3, VCCs are used to quantify best-case and worst-case characteristics of *sequences*. Here we formulate the definition of VCCs. A VCC $\mathcal{V}$ is defined as a tuple ($\mathcal{V}^l(k)$, $\mathcal{V}^u(k)$), where k represents the length of the sequence. Let the function P be a measure of some property over a sequence. If $P(n)$ denotes the measure of this property for the first n items of the sequence, then $\mathcal{V}^l(k)$ and $\mathcal{V}^u(k)$ for all $k \geq 0$ are defined as follows.

$$\begin{aligned} \mathcal{V}^l(k) &= \inf_{i\geq 0}\{P(i+k) - P(i)\} \\ \mathcal{V}^u(k) &= \sup_{i\geq 0}\{P(i+k) - P(i)\} \end{aligned} \tag{7.1}$$

$\mathcal{V}^l(k)$ and $\mathcal{V}^u(k)$ therefore provide lower and upper bounds on the measure P, for *all* subsequences of length k, within a larger sequence. Let us now consider a few concrete examples of VCCs that will be used in this chapter and see how they are formulated.

Workload Curve $\gamma = (\gamma^l, \gamma^u)$: The VCC γ is used to characterize the variability in the number of processor cycles required to process a sequence of stream objects by a PE. In this case, given a sequence of stream objects, $P(n)$ denotes the total number of processor cycles required to process the first n stream objects (here specifically we denote it as $\mathcal{W}(n)$). Then $\gamma^l(k)$ and $\gamma^u(k)$ are defined by

$$\begin{aligned} \gamma^l(k) &= \inf_{i\geq 0}\{\mathcal{W}(i+k) - \mathcal{W}(i)\} \\ \gamma^u(k) &= \sup_{i\geq 0}\{\mathcal{W}(i+k) - \mathcal{W}(i)\} \end{aligned} \tag{7.2}$$

Hence $\gamma^l(k)$ and $\gamma^u(k)$ denote the minimum and the maximum number of processor cycles that might be required by *any* k consecutive stream objects within the given sequence.

Pseudo-inverse of Arrival Curve $\xi = (\xi^l, \xi^u)$: For notational simplicity, henceforth we will denote the pseudo-inverse of α (i.e. α^{-1}) as ξ. This VCC is used to characterize the burstiness in the arrival pattern of stream objects. Given a trace of the arrival times of a sequence of stream objects (e.g. the partially processed macroblocks being written into the buffer B_2 in Figure 3.2), $P(n)$ denotes the total time length during which the first n stream objects arrive (here specifically we denote it as $T(n)$). Then $\xi^l(k)$ and $\xi^u(k)$ are defined by

$$\begin{aligned} \xi^l(k) &= \inf_{i \geq 0}\{T(i+k) - T(i)\} \\ \xi^u(k) &= \sup_{i \geq 0}\{T(i+k) - T(i)\} \end{aligned} \tag{7.3}$$

Hence, $\xi^l(k)$ and $\xi^u(k)$ denote the minimum and the maximum time length for the arrival of *any* k consecutive stream objects.

7.2 Approximate VCCs

VCCs have been used to analyze and tune platform architectures for multimedia processing (see Chapters 5, 6 and reference [63]). However, in the above formulation, the best- and worst-case characterization using VCCs do not take into account the frequency with which the best- or the worst-case occurs. Approximate VCCs generalize the concept of VCCs and take into account the frequency with which the best-/worst-case occurs.

Recall our definition of VCCs, as given by Eqn. (7.1). Now, for any given k, let a set S be defined as follows: $S = \{P(i+k) - P(i) \mid i \geq 0\}$. Instead of computing the minimum and maximum value in the multiset S, to compute ε-VCCs, we first remove certain extreme observations from S and then compute the minimum and the maximum value from the remaining elements.

Let S^l_ε denote the set resulting from removing the smallest ε percent of items from the set S. Similarly, S^u_ε denotes the set resulting from removing the largest ε percent of items from S. An ε-VCC $\mathcal{V}_\varepsilon$ can now be defined as follows: $\mathcal{V}^l_\varepsilon(k) = \inf_{i \geq 0}\{S^l_\varepsilon\}$ and $\mathcal{V}^u_\varepsilon(k) = \sup_{i \geq 0}\{S^u_\varepsilon\}$.

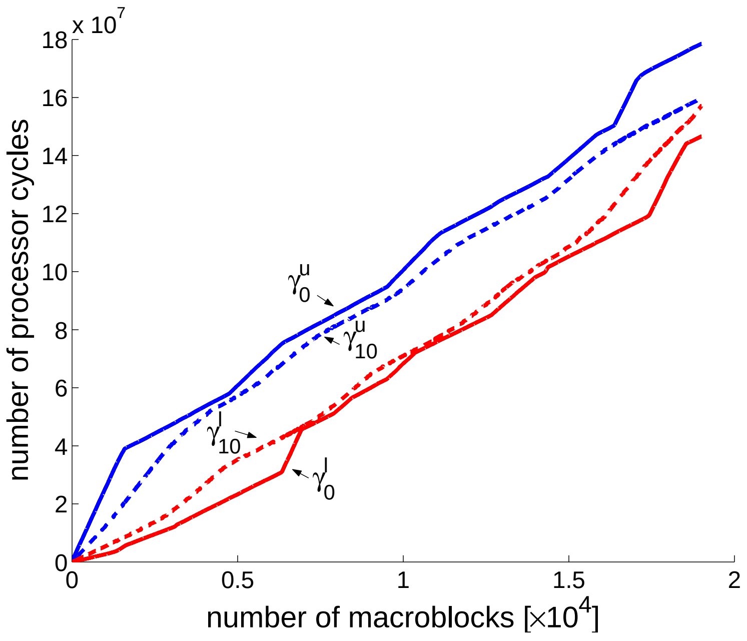


Figure 7.3: Approximate workload curves.

The above definition of ε-VCC implies that ε percent of items in S are less than $\mathcal{V}_\varepsilon^l$ and ε percent of items in S are larger than $\mathcal{V}_\varepsilon^u$. Since the set S can contain a potentially large number of elements, a computationally efficient algorithm is necessary to compute $\mathcal{V}_\varepsilon^l$ and $\mathcal{V}_\varepsilon^u$. We adopt a histogram-based algorithm [89] which is simple and efficient. Although the results obtained are not as accurate as percentile-based methods [24], they are sufficiently precise for the problem setups that we are interested in.

The histogram-based algorithm works as follows. Let D_{min} and D_{max} be the minimum and the maximum values of the elements in S. Suppose that the range $[D_{min}, D_{max}]$ is split into n equal-sized bins with the bin boundaries being $c_0, c_1, \cdots, c_n$. First, we construct a histogram for all the elements in S. We then compute r_i (for all $1 \le i \le n$), which is the ratio of the number of elements in the i-th bin $(c_{i-1}, c_i]$ to the total number of elements in S. Clearly, the sum $\sum_{j=1}^{i} r_j$ represents the fraction of items which are not larger than c_i. We then define a function F, where $F(c_i) = \sum_{j=1}^{i} r_j$ for $0 \le i \le n$ (note that F is defined only for these values). Finally, $\mathcal{V}_\varepsilon^l$ and $\mathcal{V}_\varepsilon^u$ are defined as follows.

$$\begin{aligned}\mathcal{V}_\varepsilon^l(k) &= \max\nolimits_{0\le i\le n}\{c_i \mid F(c_i) \le \tfrac{\varepsilon}{100}\}\\ \mathcal{V}_\varepsilon^u(k) &= \min\nolimits_{0\le i\le n}\{c_i \mid F(c_i) \ge 1 - \tfrac{\varepsilon}{100}\}\end{aligned} \tag{7.4}$$

It follows from the above definition that VCCs are a special case of ε-VCCs, with ε

set to zero. Figure 7.3 shows an approximate workload curve (for the VLD/IQ task in Figure 3.2) with $\varepsilon = 10$. The same figure also shows the corresponding workload curve (i.e. the case where ε is set to 0). It can clearly be seen that the approximate workload curves represent more conservative bounds on the execution requirements of sequences of stream objects, compared to the lower and upper bounds obtained from the (exact) workload curves.

7.3 Error Analysis

In a typical system design process, a designer would analyze a set of representative audio/video clips to obtain different ε-VCCs. These ε-VCCs would represent the workload that the system will be required to support. In the context of platform-based design, these ε-VCCs would determine different platform configuration parameters such as sizes of on-chip buffers, bus widths and clock frequencies of the different on-chip processors. Since ε-VCCs represent more conservative bounds and ignore infrequent best- and worst-cases, the resulting systems can also be more conservatively designed (and hence would be less expensive), albeit at the cost of small errors. For example, the minimum on-chip buffer sizes determined using ε-VCCs would be smaller compared to those determined using VCCs. The difference in size would depend on the value of ε chosen. However, the savings would come at the cost of occasionally some stream objects being dropped from the buffer. In this section we present an analytical method that can be used to bound the error incurred for any ε. We present this method in the context of two system design problems: optimal on-chip buffer sizing and processor frequency selection. By illustration, we still choose the target platform architecture to be that shown in Figure 3.2.

7.3.1 On-Chip Buffer Sizing

Consider a PE (such as PE_2 in Figure 3.2) processing a stream whose arrival process is bounded by the arrival curve α. Let β be the service curve offered by the PE. It can then be shown that the minimum size of the buffer (or the maximum backlog) at the input of this PE (i.e. B_2 in this case), denoted by $b_{\max}$, is equal to

$$b_{\max} = \sup_{\Delta \geq 0} \{\alpha^u(\Delta) - \beta^l(\Delta)\}$$

To see how β^l is obtained, let us assume that the PE runs at a clock frequency of f clock cycles/second. Given a trace of processor cycle requirements per stream object (such as the one shown in Figure 7.1) it is possible to compute the workload curve γ^u. It is then easy to see that $\gamma^{u-1}(f \cdot \Delta)$ is the minimum number of stream objects that are guaranteed to be processed within any time interval of length Δ. Hence, we set $\beta^l(\Delta)$ to be equal to $\gamma^{u-1}(f \cdot \Delta)$.

For the buffer sizing to be done using ε-VCCs, we proceed as follows. Instead of using the arrival curve α directly, we use its pseudo-inverse ξ. From a representative trace of arrival times of a sequence of stream objects, we compute $\xi^l_\varepsilon(k)$. From the trace of execution time requirements of the stream objects we compute β^l, as described above. The estimated maximum backlog is then given by:

$$b_\varepsilon = \sup_{k \geq 0} \{k - \beta^l(\xi^l_\varepsilon(k))\}$$

It may be shown from the proof below that

$$\sup_{k \geq 0} \{k - \beta^l(\xi^l_\varepsilon(k))\} = \sup_{\Delta \geq 0} \{\alpha^u_{\varepsilon'}(\Delta) - \beta^l(\Delta)\}$$

where the right hand side is similar in form to the computation of $b_{\max}$ shown above. Here, $\alpha^u_{\varepsilon'}(\Delta)$ is obtained by inverting $\xi^l_\varepsilon(k)$. It may be noted that by inverting $\xi^l_\varepsilon(k)$ we obtain an approximate arrival curve whose *approximation ratio* ε' is different from the approximation ratio ε of $\xi^l_\varepsilon(k)$.

Proof: Firstly we have $\alpha^u_{\varepsilon'}(\xi^l_\varepsilon(k)) = k$, since $\alpha^u_{\varepsilon'}(\Delta)$ is the pseudo-inverse of $\xi^l_\varepsilon(k)$. Let $\Delta = \xi^l_\varepsilon(k)$, then we have $\alpha^u_{\varepsilon'}(\Delta) = \alpha^u_{\varepsilon'}(\xi^l_\varepsilon(k)) = k$ and $\beta^l(\Delta) = \beta^l(\xi^l_\varepsilon(k))$, for all $\Delta \geq 0$ and $k \geq 0$. By combining the above two equations, it is easy to see that $\sup_{\Delta \geq 0}\{\alpha^u_{\varepsilon'}(\Delta) - \beta^l(\Delta)\} = \sup_{k \geq 0}\{k - \beta^l(\xi^l_\varepsilon(k))\}$. $\square$

Clearly, if the buffer size is set to b_ε then stream objects might occasionally be dropped. Given a trace of arrival times of stream objects at the buffer, we can bound the maximum number of stream objects that might be dropped. We assume that β^l, which was obtained from a set of representative multimedia streams, also holds for this trace (i.e. $\beta^l(\Delta)$ is the minimum number of stream objects that are guaranteed to be processed within any time interval of length Δ, for this stream as well).

Let $T(i)$ denote the arrival time of the i-th stream object at the buffer. Let $\xi(i,k) = T(i) - T(i-k)$ denote the length of the time interval during which the previous k consecutive stream objects adjacent to the i-th stream object arrive ($0 \le k \le i$). Then $\beta^l(\xi(i,k))$ represents the minimum number of stream objects that the PE can process during this time interval. We can have the following theorem.

Theorem 2 *The maximum backlog when the i-th stream object arrives at the buffer is equal to*

$$\sup_{0 \le k \le i} \{k - \beta^l(\xi(i,k))\}$$

Proof: Let $x(t)$ denote the number of stream objects that have arrived at the buffer within $[0,t]$, and $y(t)$ denote the number of stream objects that have been processed within $[0,t]$. From [16], the backlog at time t is

$$x(t) - y(t) \le x(t) - \inf_{0 \le s \le t} \{x(t-s) + \beta^l(s)\}$$

Thus

$$x(t) - y(t) \le \sup_{0 \le s \le t} \{x(t) - x(t-s) - \beta^l(s)\}$$

Let $i = x(t)$ and $k = x(t) - x(t-s)$, then we have $t = T(i)$ and $s = T(i) - T(i-k) = \xi(i,k)$. It follows that when the i-th stream object arrives at the buffer, the backlog is

$$x(T(i)) - y(T(i)) \le \sup_{0 \le k \le i} \{k - \beta^l(\xi(i,k))\}$$

It is equivalent to saying that the maximum backlog when the i-th stream object arrives at the buffer is equal to $\sup_{0 \le k \le i}\{k - \beta^l(\xi(i,k))$. $\square$

Hence, the i-th stream object might be dropped if

$$\sup_{0 \le k \le i} \{k - \beta^l(\xi(i,k))\} > b_\varepsilon$$

In the above inequality, the value of $\beta^l(\xi(i,k))$ is estimated to be $\gamma^{u^{-1}}(f \cdot \xi(i,k))$. This assumes that the $\beta^l(\xi(i,k))$ consecutive stream objects processed within the time interval of length $\xi(i,k)$ require the maximum possible number of processor cycles. If we instead use the approximate upper workload curve γ^u_ε, then the above inequality may be reformulated as:

$$\sup_{0 \le k \le i} \{k - \beta^l_{\varepsilon'}(\xi(i,k))\} > b_\varepsilon$$

However, unlike the previous case, in this case we can not provide deterministic guarantees on the maximum number of dropped stream objects.

7.3.2 Processor Frequency Selection

In addition to on-chip buffer size configuration, we illustrate our analytical method with the case of processor frequency configuration in this subsection. Let us consider the platform architecture shown in Figure 3.2. The fully processed stream objects are finally written out into the playout buffer B_v. This buffer is read by the real-time video output device at a pre-specified rate. One of the design constraints while configuring this platform architecture is to ensure that B_v never underflows. Clearly, the clock frequency of an on-chip PE should at least be equal to sustain the rate at which stream objects are being consumed by the output device. However, because of the variability in the execution time requirements of stream objects, computing this minimum clock frequency is not trivial. The problem becomes more complicated because of the buffering at the playout buffer. The problem of computing this frequency has become especially interesting with the advent of processor soft cores, which allow a high degree of customization. This problem has been addressed in Chapter 5 using VCCs as a means of workload characterization.

Clearly, using ε-VCCs, the computed frequency will be substantially lower compared to that obtained using VCCs. For the sake of simplicity, here we have only considered the problem of computing the minimum constant frequency at which the PE needs to be run. However, the method presented in Chapter 5 can be used in the case of frequency-scalable processors as well (to compute the different frequency levels and the frequency range that the PE should support).

From the long-term playback rate of the input streams, a designer can derive the lowest number of stream objects that a PE must process within any time interval of length Δ, i.e., the lower service curve $\beta^l(\Delta)$, for all $\Delta \geq 0$. To guarantee that a PE can process at least $\beta^l(\Delta)$ number of stream objects within any time interval of length Δ, in the worst case the PE need to provide $\gamma^u(\beta^l(\Delta))$ number of processor cycles within this time interval. Then the minimum frequency to guarantee lower service curve β^l is computed as $f = \max_{\Delta \geq 0}\{\gamma^u(\beta^l(\Delta))/\Delta\}$, assuming that these number of stream objects require the maximum possible number of processor cycles, where $\gamma^u(k)$ represents the maximum possible number of processor cycles required by any k consecutive stream objects for all $k \geq 0$.

However, we know that if one stream object cannot use up the cycles allocated to it, the redundant cycles will be used by its following stream objects. Hence, we expect that the approximate upper workload curve γ^u_ε achieves better estimation of the execution demands. Thus, in the average-case analysis the minimum frequency becomes $f_\varepsilon = \max_{\Delta \geq 0}\{\gamma^u_\varepsilon(\beta^l(\Delta))/\Delta\}$.

Unlike the analysis of buffer size, where errors will cause objects to be dropped, error in the frequency configuration will cause stream objects to miss their deadlines. In the rest of this subsection, we present how to determine if a stream object will miss its deadline. If a stream object is not available in the input buffer of a PE (or the output device) when it is time to process it, we say that this stream object has missed its deadline. Given a stream, we would like to analyze at most how many percent of stream objects miss their deadlines when the frequency f_ε is used.

When a stream object arrives at the input buffer of a PE, it gets processed immediately or waits for some time based on whether the PE is busy or not. If the PE is still processing some stream object when a new stream object arrives, the new object will *wait*. In the following, we will start from a stream object that gets processed immediately upon entering into the buffer, to analyze whether the subsequent stream objects will wait or not. In our analysis, we need to refer to the definitions of $\xi(i, k)$ and $T(i)$ in last subsection. For

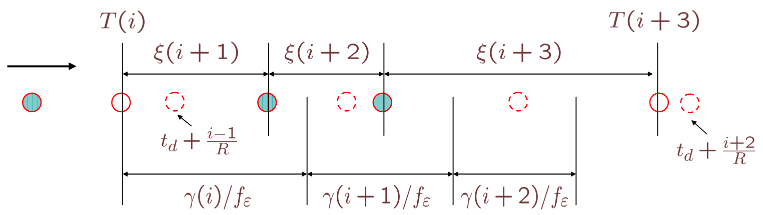


Figure 7.4: Illustration of frequency analysis

simplicity, we use $\xi(i)$ to denote $\xi(i,1)$, representing the arrival interval between i-th and $(i-1)$-th stream objects. Suppose that $\mathcal{W}$ denote the total cycles required by all the previous i stream objects, we use $\gamma(i) = \mathcal{W}(i) - \mathcal{W}(i-1)$ to represent the cycle requirement of the i-th stream object. As shown in Figure 7.4, let the i-th stream object be one of the stream objects that need not wait. The idle time for the PE is $L_i = \xi(i+1) - \gamma(i)/f_\varepsilon$ after the i-th stream object is finished and before the next one arrives into the input buffer. If $L_i < 0$, it means that the next stream object has to wait a time interval of length $|L_i|$ before it is processed. Suppose that the next k stream objects numbered $i+1, \ldots, i+k$ need to wait, then the idle time before the $(i+k+1)$-th stream object is processed is $\mathcal{L}(i, i+k) = \sum_{i \leq j \leq i+k} L_j$. If $\mathcal{L}(i, i+k) < 0$, it implies that the $(i+k+1)$-th stream object need to wait a time interval of length $|\mathcal{L}(i, i+k)|$ before it is processed. Similarly, we can continue to identify whether the following stream objects need to wait or not. When we meet the next stream object that need not wait, the influence of the wait from the stream objects following the i-th stream object is stopped. In the same way, we can analyze the subsequent stream objects after the next *non-waiting* stream object.

To identify if a stream object misses its deadline, we first assume that its next stream object is a virtual one that arrives just right at the time of its deadline and check whether its next virtual stream object needs to wait or not. As an example, we consider the last PE in the path of the stream and analyze the number of stream objects that miss the playback deadlines. Assuming that the real-time client starts to playback after a delay of time t_d, the i-th stream object will miss its deadline if it does not enter the playout buffer before $t_d + \frac{i-1}{R}$. Thus, if a virtual stream object arriving at the input buffer of the last PE at $t_d + \frac{i-1}{R}$ next to the i-th one needs to wait, then the i-th stream object will miss its deadline.

7.4 Empirical Validation

To validate our scheme for workload characterization, we experimented with the platform architecture shown in Figure 3.2 using the setup described in Section 7.4.

We experimented with multiple representative video clips chosen from a set of clips, all of which have the same long-term playback rate, i.e. the same number of macroblocks are consumed per second by the video output device. For each video clip, we first used the SimpleScalar instruction set simulator to obtain traces of execution times for the VLD/IQ and IDCT/MC tasks of the MPEG-2 decoder application. We then simulated the platform architecture shown in Figure 3.2 using a transaction-level model of the architecture written in SystemC. Traces containing the arrival times of the macroblocks at each on-chip buffer and the buffer backlogs were obtained. The VCCs and the ε-VCCs were measured from the collected execution traces. In the following, we assume that the traces of execution times have already been obtained.

7.4.1 Buffer Sizing

The results reported below only concern the buffer at the input of PE_2 (i.e. B_2). Both PE_1 and PE_2 were configured to run with their long-term average frequencies. These frequencies were computed by taking into account the long-term playback rate of the output device and the average cycle demands per macroblock for the tasks implemented on them. The system was initially simulated for all the (representative) video clips, from which we obtained the approximate lower pseudo-inverse curve $\xi^l_\varepsilon(k)$ corresponding to the arrival process of stream objects at the buffer B_2. From the simulation results we also obtained the approximate upper workload curve $\gamma^u_\varepsilon(k)$ for PE_2. We then computed the buffer size b_ε. As shown in Figure 7.5, the computed buffer size decreases as ε is increased from 0 to 20. We observed more than 20% reduction in the buffer size when ε was set to be 5.

For each video clip, we analytically estimated the upper bound on the percentage of dropped macroblocks when the size of B_2 was set to b_ε. At the same time, we simulated the execution of this clip with the size of B_2 set to be b_ε. The simulation results showed that

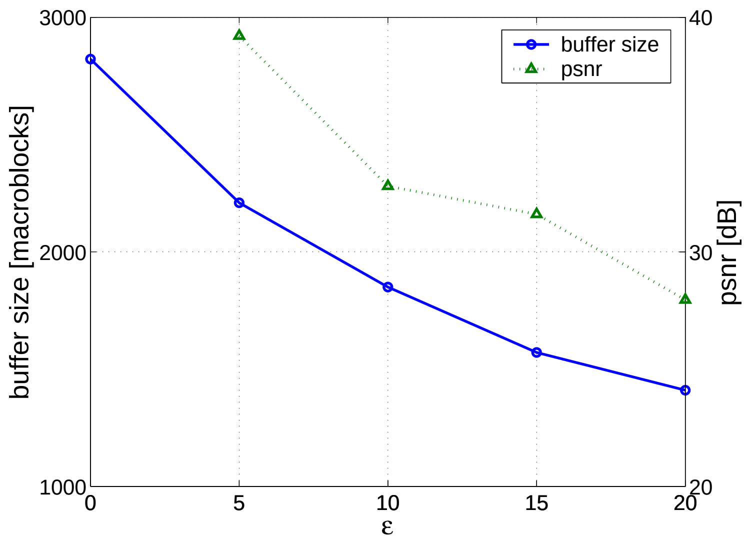


Figure 7.5: Computed buffer sizes for different values of ε.

our analytical method gives an upper bound on the percentage of dropped macroblocks for any of the clips used. Figure 7.6 shows the analytical bounds and simulation results for a representative video clip. We can observe that the drop ratio is upper bounded at about 5% with ε equal to 5. However, there is more than 20% reduction in the buffer size compared to when ε is equal to 0. As shown in Figure 7.5, we also measured the *Peak Signal-to-Noise Ratio* (PSNR) for this video clip corresponding to each buffer size. PSNR is commonly used to measure the quality of a reconstructed frame with macroblock loss, compared to the decoded frame without any loss. We defined the PSNR of a video clip as the average value of PSNRs over all those frames which suffered loss of macroblocks. Although we applied only a simple error concealment mechanism (a dropped macroblock just takes the value of the corresponding macroblock from the previous frame), Figure 7.5 shows that at $\varepsilon = 5$, the PSNR remains at 39.2 dB. PSNR values above 38 dB are generally accepted as good video quality [34].

Comparison with queuing theory analysis: Queuing theory [49] models the queueing systems using stochastic processes and can only provide stochastic performance guarantees. Our framework presented in this thesis, based on network calculus, can offer deterministic performance guarantees. We illustrate the difference between our work and queuing theory analysis, using the example of on-chip buffer sizing problem and studying

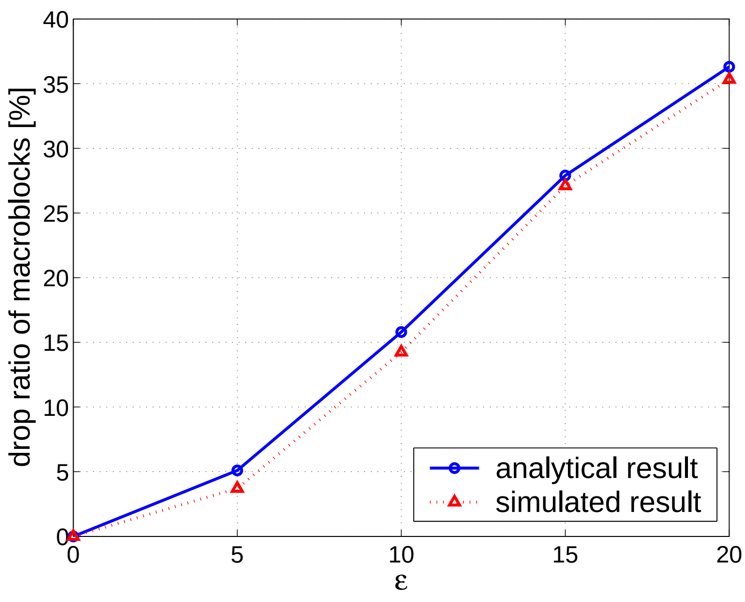


Figure 7.6: Percentage of macroblocks dropped from B_2 for different values of ε.

the buffer B_2. Firstly we show the results for queuing theory analysis. For the purpose of illustration, we choose to model the buffer B_2 as an M/M/1 queue. For a given video clip and its simulation traces, we measured the mean arrival rate λ (i.e. the mean number of macroblocks that can arrive at the buffer per second) and the mean processing rate μ of PE_2 (i.e. the mean number of macroblocks that can be processed by PE_2 per second). It is known from the queuing theory that when a macroblock arrives at the buffer, the probability that the buffer fill level is equal to n macroblocks can be expressed as:

$$P_n = (1 - \rho)\rho^n$$

where ρ is equal to λ/μ, representing the intensity of the traffic. Hence, supposing that the buffer size is configured to be N, we obtain the probability that the buffer fill level is greater than N when a macroblock arrives at the buffer (i.e. the probability that a macroblock might be dropped from the buffer):

$$P = 1 - \sum_{n \leq N} P_n$$

Using the same video clip as that in Figure 7.6, Figure 7.7 shows the probability that the backlog at B_2 is greater than 200 macroblocks is less than 1%. It is equivalent to saying that in probability-based sense, less than 1% of macroblocks might be dropped when the size of B_2 is configured to be 200 macroblocks.

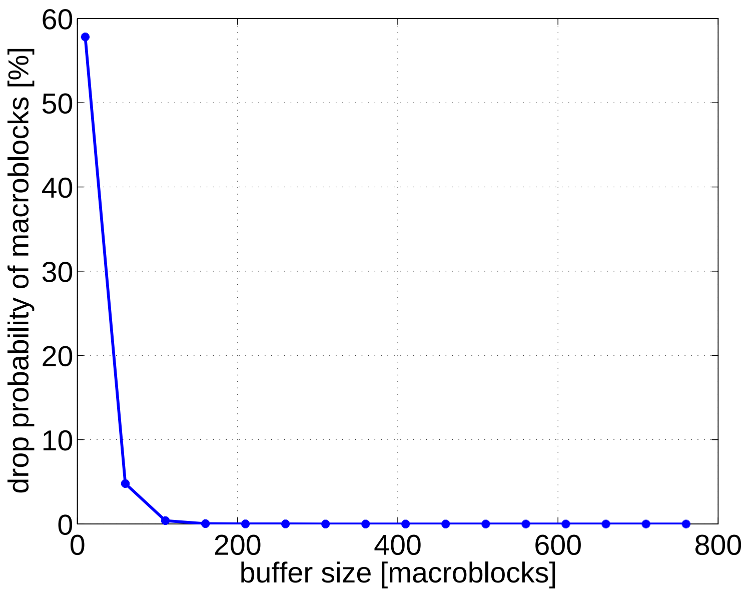


Figure 7.7: Probability of macroblocks dropped from B_2 for different values of buffer sizes.

Our framework models the multimedia workloads using the concept of ε-VCCs, which is a general model and no assumption on the probability distribution of the workloads is needed. For the on-chip buffer sizing problem, our framework offers the upper bound on the percentage of macroblocks that might be dropped from the buffer under certain buffer size value. For example, our framework gives the deterministic guarantee that at most around 35% of macroblocks might be dropped when the size of B_2 is configured to be about 1400 macroblocks and ε is set to 20, as shown in Figures 7.6 and 7.5. It is also observed that nearly 35% of macroblocks are actually dropped when the size of buffer B_2 is configured to be 1400 macroblocks, while queueing theory can only tell that less than 1% of macroblocks might be dropped. It thus shows that it is not enough to just use queueing theory to quantitatively measure the buffer overflow errors in our context of multimedia platforms. Also, our work guarantees that the maximum possible backlog at B_2 will never exceed 2822, while queuing theory analysis allows the possibility that the maximum buffer backlog can be infinite. Hence, our work has the advantage over queueing theory analysis on buffer dimensioning.

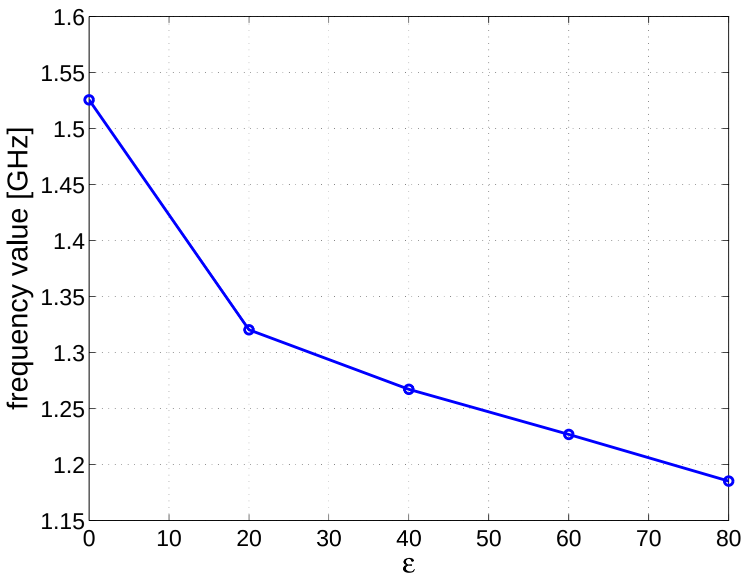


Figure 7.8: Frequency values of PE_2 for different values of ε.

7.4.2 Frequency Selection

We will use PE_2 to illustrate how the processor's clock frequency may be lowered if ε-VCCs are used. Based on the approximate upper workload curve γ_ε^u on PE_2 and the long-term playback rate, we computed the clock frequency f_ε for PE_2. As shown in Figure 7.8, considerable reduction in the frequency values were achieved when the approximate curves were used. For example, there was nearly a 20% reduction in the frequency when ε was set to be 60.

PE_1 was configured to its long-term average frequency. An initial simulation of the system was conducted for all the representative video clips, after which we had the necessary traces for the error analysis. For each video stream, we computed an upper bound on the percentage of macroblocks that can potentially miss their deadlines when PE_2 is run at different clock frequencies. When compared with simulation results, it may be seen that our analytical method gives an upper bound on the percentage of macroblocks that missed their deadlines. Table 7.1 shows the analytical bounds and the results obtained using simulation for a representative video clip with two different playback delay settings t_d. It may be noted that when the delay was set to $0.30s$, none of the macroblocks missed their deadlines, even with ε set to 60, while the required frequency was reduced by nearly 20%.

ε	% of macroblocks missing deadlines			
	$t_d = 0.28s$		$t_d = 0.30s$	
	analysis	simulation	analysis	simulation
0	3.84	3.80	0.00	0.00
20	9.73	9.63	0.00	0.00
40	16.7	16.5	0.00	0.00
60	44.0	43.7	0.00	0.00
80	97.0	97.0	80.4	69.5

Table 7.1: Analytical bounds and simulation results on the percentage of macroblocks that miss their deadlines, for different values of ε.

7.5 Summary

In this chapter we proposed a parameterized scheme for characterizing multimedia workloads, based on the novel concept of *approximate variability characterization curves* or ε-VCCs. Since most multimedia applications only require soft real-time guarantees, we demonstrated that by using ε-VCCs to design and configure platform architectures, significant resource savings may be achieved with only a negligible loss in output quality.

In our scheme, we also propose error analysis algorithms for two typical system design cases (on-chip buffer sizing and processor frequency selection), which give the bound on the error incurred by using ε-VCCs. Our scheme can be used to achieve the tradeoff between the output quality and the resource savings through an analytical way. Currently our scheme can only give the error bounds for a single stream, where the traces for this stream is needed. In the future, we would want to extend this scheme to provide guarantees for a *class* of streams. Details of this will be discussed in Chapter 8.

Chapter 8

Conclusion

In this thesis we proposed an analytical framework that can be used for the system-level design of MpSoC platform architectures for multimedia applications. According to the Y-chart scheme for the design of SoC platforms, we modeled multimedia applications using the KPN and used an system-level abstracted model of the SoC platform architectures. Based on network calculus theory, we then presented a unified framework for modeling of multimedia workloads and performance analysis of such modeled MpSoC platform architectures, which multimedia applications are partitioned and mapped onto.

8.1 Modeling of Multimedia Workloads

In our framework, we first need to model the multimedia workloads imposed on the platform architecture. Given a large library of multimedia streams that might be run on the platform, we proposed an approach that can be used for workload design in the context of MpSoC platform design, i.e. obtaining the VCCs for this library of streams. Firstly the pairwise dissimilarity between any two streams is measured, which is based on the shapes of VCCs associated with each stream. We then used a hierarchical clustering algorithm to classify the streams into different clusters. The “representative” streams can be identified from each cluster (i.e. class) to represent the workloads imposed by this cluster. The VCCs for these streams characterize the class of streams it belongs to. The VCCs associated with the set of “representative” streams resulted from all the clusters then give an accurate model of the original library.

In our approach, the VCCs are obtained only from the instruction set simulation and a simple trace-analysis algorithm. Therefore, our scheme for workload design is order of magnitude faster than using full system simulation, achieving considerable savings in the design time.

8.2 Design and Analysis

Using the obtained VCCs, which represent the workloads imposed by a class of multimedia streams, we can develop analytical approaches that can be used for system-level design and analysis of MpSoC platforms for multimedia applications, based on network calculus theory. As illustrations of our framework, this thesis proposed analytical approaches for two typical system design cases: processor frequency selection and rate analysis.

Processor Frequency Selection: We proposed an analytical approach that can help a system designer to identify the operating frequency ranges that should be supported by the different processors of a platform architecture, in order to run the target multimedia streams (that may include multiple classes). Our approach also identifies how such frequency ranges depend on the different parameters of the architecture such as on-chip buffer sizes. The service bounds on a processor for a class of streams were firstly derived, given the bounds on the arrival patterns of input streams and the playback rate. Based on the definition of service curves, we formulated the constraints that should be satisfied by the frequency values at which a processor runs. The frequency range was then identified. These theoretical results were validated by experimenting with sample MPEG-2 streams, where the on-chip processors run at the frequency schedules bounded by the computed frequency ranges.

Rate Analysis: We proposed an analytical approach to determine tight bounds on the rates at which different multimedia streams can be fed into a platform architecture. We also studied this problem of rate analysis when a scheduler (such FPS and TDM) is implemented on a processor. Our approach can aid in selecting the parameters for a scheduler, e.g. the weights associated with each stream for a TDM scheduler. Experimental results show that our approach can give valid tight bounds on the arrival rates of multimedia streams.

The design of SoC platforms for multimedia applications is especially difficult due to the various kinds of variabilities arising from multimedia processing, such as the high variability in the execution requirements and great burstiness in the on-chip traffic etc. Our framework accurately models the burstiness in these kinds of variabilities using the concept of VCCs. At the same time, the analytical approaches developed for the design space exploration and performance analysis of MpSoC platforms take fully into account the various burstiness, which we think has critical influence on platform architecture design. What is particular to our analysis is that all the operations are done for a *class* of streams. A major contribution of our analytical approaches is that it can help to greatly reduce the design time and costs and avoid the time-consuming simulation.

8.3 New Characterization of Multimedia Workloads

In the above analytical approaches, we used VCCs to capture the worst-case characteristics of multimedia workloads. In this thesis, we also proposed a new concept of *approximate variability characterization curves* or ε-VCCs to characterize the average-case characteristics of multimedia workloads. By taking into account the frequency of the occurrences of certain patterns, this new concept works in a parameterized fashion, where ε indicates how many percent of worst-case occurrences are omitted.

We then applied the concept of ε-VCCs to determine the platform parameters configured for a SoC platform, e.g. the sizes of on-chip buffer and the long-term frequency value configured for an on-chip processor. For the design case of on-chip buffer sizing, the experimental results showed that the value of buffer size computed using ε-VCCs reduces as the value of ε increases. This is due to the reason that some worst-case occurrences of certain patterns are ignored. It also showed that the value of computed buffer size decreases faster when the value of ε is smaller, while this becomes slower as the value of ε is greater. This may be explained since worst cases in the workloads happen less frequently relative to the average cases. Similar observations was also obtained for the case of configuring long-term frequency value.

We also presented analytical algorithms that provide an upper bound on the errors associated with different values of ε when ε-VCCs are applied in the design of SoC platforms. The simulation results showed that the proposed algorithms analytically give a valid upper bound on how many percent of stream objects might be dropped from the buffer when its size is set to be the values computed using ε-VCCs. These algorithms also give an upper bound on how many percent of stream objects might miss deadlines when the processor frequency is configured with the values computed using ε-VCCs.

It is known that multimedia applications exhibit various kinds of high variability and are characterized by soft real-time constraints, i.e. a small degree of degradation in the output quality is acceptable. Hence, it is desirable to design the SoC platforms for multimedia applications based on average-case characteristics of multimedia workloads, which would achieve great resource savings and thus reduce the cost. Our proposed parameterized framework provides an efficient scheme of characterizing the average-case behaviors of multimedia workloads. Through error analysis algorithms, our framework can help a designer to identify the tradeoffs between the output quality and the resource requirements (i.e. the selection of suitable value of ε) in an analytical way, which avoids the time-consuming simulation. Some related work on statistical network calculus presents probabilistic bounds on the errors. Our error analysis algorithms give deterministic bounds instead, which provides an effective way of measuring the output quality for multimedia applications and is complementary to the probability-based methods.

8.4 Future Work

We have presented the concept of ε-VCCs as a new characterization of multimedia workloads. Due to the importance of “average-case” analysis in the context of multimedia SoC platform design, in the future we would want to extend our analytical approaches for system-level design and analysis, using ε-VCCs as models of multimedia workloads. We hope that our framework will contain both “worst-case” and “average-case” analysis mechanisms, which provides a full support for SoC platform design for multimedia applications.

The extended framework will work in a parameterized fashion. Different values of ε correspond to different degree of resource savings and quality degradation. The major challenge to develop analytical approaches using ε-VCCs is how to bound the quality degradation associated with the different values of ε.

Same as VCCs, our concept of ε-VCCs is defined for a class of streams, and hence the platform parameters (such as the buffer sizes or processor frequency values) analyzed using ε-VCCs are valid for a class of streams. Note that the class is defined in the sense of *burstiness* that is shown in the behaviours of multimedia processing. Therefore, it is also expected that we can bound the quality degradation for a class of streams.

Now, we have only conducted a preliminary study of the error analysis algorithms that can bound the errors for a single stream belonging to the class. In practice, the system designer may need to analyze multiple representative streams from a class of streams in order to get an estimation of the errors associated with this class, which involves more design efforts. In the future, we would extend the existing error analysis algorithms to provide the error bounds for a class of streams. Such an extension would help to further reduce the design costs.

In the future, we would also want to study more complex architectures and applications. However, it is not trivial to develop the “average-case” analysis approaches for complicated design cases and to provide the error bounds at the same time. To bound the errors, we may need to identify the worst-case patterns in the sense of incurred errors after applying ε-VCCs (VCCs are not enough to identify such patterns). The analytical approaches may need to be developed with the error analysis algorithms in mind. We believe that there are many issues to be explored along this direction.

Bibliography

[1] Ptolemy project.
`http://ptolemy.eecs.berkeley.edu.`

[2] A. Acquaviva, L. Benini, and B. Riccó. An adaptive algorithm for low-power streaming multimedia processing. In *Conference on Design, Automation and Test in Europe (DATE)*, Munich, GERMANY, March 2001.

[3] PALM-DP-2000 AcurX configurable SoC platform.
`http://www.palmchip.com/`.

[4] Rajeev Agrawal, R. L. Cruz, Clayton Okino, and Rajendran Rajan. Performance bounds for flow control protocols. *IEEE/ACM Transactions on Networking*, 7(3):310–323, June 1999.

[5] Gang Quan an Xiaobo Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *DAC*, Las Vegas, Nevada, United States, 2001.

[6] H. V. Antwerpen, N. Dutt, R. Gupta, S. Mohapatra, C. Pereira, N. Venkatasubramanian, and R. von Vignau. Energy-aware system design for wireless multimedia. In *IEEE Design, Automation and Test in Europe (DATE)*, Paris, FRANCE, February 2004.

[7] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, 2002.

[8] S. Ayyorgun and R. L. Cruz. A composable service model with loss and a scheduling algorithm. In *INFOCOM*, Hong Kong, China, March 2004.

[9] S. Ayyorgun and R. L. Cruz. A service-curve model with loss and a multiplexing problem. In *ICDCS*, Tokyo, Japan, March 2004.

[10] F. Balarin, Y.Watanabe, H. Hsieh, L. Lavagno, and C. Passerone. Metropolis: an integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, 2003.

[11] A. C. Bavier, A. B. Montz, and L. L. Peterson. Predicting mpeg execution times. In *ACM SIGMETRICS*, Madison, Wisconsin, USA, 1998.

[12] E. Bini and M. D. Natale. Optimal task rate selection in fixed priority systems. In *RTSS*, Miami, Florida, USA, 2005.

[13] A. Bobrek, J. Pieper, J. Nelson, J. Paul, and D. Thomas. Modeling shared resource contention using a hybrid simulation/analytical approach. In *Design, Automation and Test in Europe*, February 2004.

[14] R. Boorstyn, A. Burchard, J. Leibeherr, and C. Oottamakorn. Statistical service assurances for traffic scheduling algorithms. *IEEE Journal on Selected Areas in Communications*, 18(13):2651–2664, 2000.

[15] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *IEEE Transactions on Information Theory*, 44(3):1087–1096, May 1998.

[16] J.-Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*. LNCS 2050, 2001.

[17] L.-O. Burchard and P. Altenbernd. Estimating decoding times of mpeg-2 video streams. In *International Conference on Image Processing*, Vancouver, BC, Canada, 2000.

[18] M. Buss, T. Givargis, and N. Dutt. Exploring efficient operating points for voltage scaled embedded processor cores. In *24th IEEE Real-Time Systems Symposium (RTSS)*, Cancun, Mexico, December 2003.

[19] H. Kim C. Im and S. Ha. Dynamic voltage scheduling technique for low-power multimedia applications using buffers. In *International Symposium on Low Power Electronics and Design (ISLPED)*, California, USA, August 2001.

[20] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *6th Design, Automation and Test in Europe (DATE)*, Munich, Germany, February 2003.

[21] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. Performance evaluation of network processor architectures: Combining simulation with analytical estimation. *Computer Networks*, 41(5):641–665, 2003.

[22] C.S. Chang. On deterministic traffic regulation and service guarantee: a systematic approach by filtering. *IEEE Transactions on Information Theory*, 44(3):1097–1110, May 1998.

[23] C.S. Chang. *Performance guarantees in communication networks*. Springer-Verlag, New York, 2000.

[24] W. Chase and F. Bown. *General Statistics*. John Wiley & Sons, 1997.

[25] C. Chen and M. Sarrafzadeh. Provably good algorithm for low power consumption with dual supply voltages. In *ICCAD*, San Jose, CA, United States, 1999.

[26] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *ICCAD*, San Jose, CA, USA, November 2002.

[27] F. Ciucu, A. Burchard, and J. Liebeherr. A network service curve approach for the stochastic analysis of networks. In *ACM Sigmetrics*, 2005.

[28] R. Cruz. A calculus for network delay, Parts 1 & 2. *IEEE Transactions on Information Theory*, 37(1), 1991.

[29] A. Dasdan, D. Ramanathan, and R. K. Gupta. A time-driven design and validation methodology for embedded real-time systems. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 3(4):533–553, 1998.

[30] Sandeep Dhar and Dragan Maksimovic. Low-power digital filtering using multiple voltage distribution and adaptive voltage scaling. In *ISLPED*, Rapallo, Italy, July 2000.

[31] S. Dutta, R. Jensen, and A. Rieckmann. Viper: A multiprocessor SOC for advanced set-top box and digital TV systems. *IEEE Design & Test of Computers*, 18(5):21–31, September-October 2001.

[32] L. Eeckhout, H. Vandierendonck, and K. De Bosschere. Workload design: Selecting representative program-input pairs. In *IEEE PACT*, pages 83–94, 2002.

[33] F. Balarin et al. *Hardware-Software Co-design of Embedded Systems – The POLIS approach*. Kluwer Academic Publishers, 1997.

[34] C. A. Gonzales, H. Yeo, and C. J. Kuo. Requirements for motion-estimation search range in MPEG-2 coded video. *IBM Journal of Research and Development*, 43(4), 1999.

[35] A. D. Gordon. *Classification*. Chapman & Hall/CRC, 1999.

[36] W. Hawkins and T. Abdelzaher. Towards feasible region calculus: An end-to-end schedulability analysis of real-time multistage execution. In *RTSS*, Miami, Florida, USA, 2005.

[37] D. P. Heyman, A. Tabatabai, and T. Lakshman. Statistical analysis and simulation study of video teleconference traffic in atm networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 2(1):49–59, 1992.

[38] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M.B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Trans. on Computer Aided-Design of Integrated Circuits and Systems*, 18(12), 1999.

[39] Shaoxiong Hua and Gang Qu. Approaching the maximum energy saving on embedded systems with multiple voltages. In *ICCAD*, San Jose, CA, United States, November 2003.

[40] C. Huang, M. Devetsikiotis, I. Lambadaris, and A. Kaye. Modeling and simulation of self-similar variable bit rate compressed video: a unified approach. In *ACM SIGCOMM*, 1995.

[41] C.J. Huges, J. Srinivasan, and S.V. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *34th Annual International Symposium on Microarchitecture (MICRO)*, 2001.

[42] C.J. Hughes, P. Kaul, S.V. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the execution of multimedia applications and implications for architecture. In *ISCA*, pages 254–265, 2001.

[43] Blue Logic technology, IBM.
`http://www.chips.ibm.com/bluelogic/`.

[44] M. Jersak and R. Ernst. Enabling scheduling analysis of heterogeneous systems with multi-rate data dependencies and rate intervals. In *Proc. 40th Design Automation Conference (DAC)*, 2003.

[45] G. Kahn. The semantics of a simple language for parallel programming. In *International Federation for Information Processing Congress*, North-Holland, Amsterdam, August 1974.

[46] Tero Kangas, Petri Kukkala, Heikki Orsila, and Erno Salminen et al. Uml-based multiprocessor soc design framework. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(2):281–320, 2006.

[47] K. Keutzer, S. Malik, R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System level design: Orthogonolization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design*, 19(12), 2000.

[48] Bart Kienhuis, Ed Deprettere, Kees Vissers, and Pieter van der Wolf. An approach for quantitative analysis of application-specific dataflow architectures. In *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, LA, CA, USA, 1997.

[49] Leonard Kleinrock. *Queuing Systems, Volume 1: Theory*. John Wiley and Sons, 1975.

[50] Marwan Krunz and Satish K. Tripathi. On the characterization of VBR MPEG streams. In *ACM SIGMETRICS*, Cambridge, MA, June 1997.

[51] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: a preliminary application to the data stream. In *Workload characterization of emerging computer applications*, pages 145–163. Kluwer Academic Publishers, 2001.

[52] K. Lahiri, A. Raghunathan, and S. Dey. System level performance analysis for designing on-chip communication architectures. *IEEE Trans. on Computer Aided-Design of Integrated Circuits and Systems*, 20(6):768–783, 2001.

[53] A. A. Lazar, G. Pacifici, and D. E. Pendarakis. Modeling video sources for real-time scheduling. *Multimedia Syst.*, 1(6):253–266, 1994.

[54] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *ACM/IEEE MICRO*, pages 330–335, 1997.

[55] D.S. Lee, B. Melamed, A. Reibman, and B. Sengupta. Analysis of a video multiplexer using TES as a modeling methodology. In *IEEE Global Telecommunications Conference (GLOBECOM)*, Phoenix, USA, December 1991.

[56] E.A. Lee and A. Sangiovanni-Vincentelli. A framework for comparing models of computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, 1998.

[57] B. Maglaris, D. Anastassiou, P. Sen, G. Karlsson, and J.D. Robbins. Performance models of statistical multiplexing in packet video communications. *IEEE Transactions on Communications*, 36(7):834–844, 1988.

[58] Rolf Ernst Marek Jersak, Rafik Henia. Context-aware performance analysis for efficient embedded system design. In *Proc. DATE*, Paris, France.

[59] A. Mathur, A. Dasdan, and R. K. Gupta. Rate analysis for embedded systems. *IEEE Transactions on VLSI*, 3(3):408–436, 1998.

[60] A. Maxiaguine, S. Knzli, and L. Thiele. Workload characterization model for tasks with variable execution demand. In *DATE*, Paris, France, February 2004.

[61] A. Maxiaguine, S. Künzli, S. Chakraborty, and L. Thiele. Rate analysis for streaming applications with on-chip buffer constraints. In *ASP-DAC*, Yokohama, Japan, January 2004.

[62] A. Maxiaguine, Y. Liu, S. Chakraborty, and W. T. Ooi. Identifying "representative" workloads in designing MpSoC platforms for media processing. In *ESTIMedia*, Stockholm, Sweden, September 2004.

[63] A. Maxiaguine, Y. Zhu, S. Chakraborty, and W.-F. Wong. Tuning SoC platforms for multimedia processing: Identifying limits and tradeoffs. In *CODES+ISSS*, Stockholm, Sweden, September 2004.

[64] S. Mohanty and V. Prasanna. Rapid system-level performance evaluation and optimization for application mapping onto SoC architectures. In *IEEE International ASIC/SOC Conference*, September 2002.

[65] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *ACM Multimedia (MM)*, Berkeley, CA, USA, November 2003.

[66] A. Nandi and R. Marculescu. System-level power/performance analysis for embedded systems design. In *DAC*, Las Vegas, Nevada, USA, June 2001.

[67] OMAP for 2.5G and 3G: Overview, Texas Instruments.
`http://www.ti.com/sc/omap/`.

[68] Andy D. Pimentel, Louis O. Hertzberger, Paul Lieverse, Pieter van der Wolf, and Ed F. Deprettere. Exploring embedded-systems architectures with artemis. *IEEE Computer*, 34(11):57–63, 2001.

[69] Flavio Polloni, Luca Mazzoni, and Serge Di Matteo. Fast system-level design space exploration for low power configurable multimedia systems-on-chip. In *ASIC/SOC Conference*, Rochester, New York, September 2002.

[70] P. Pop, P. Eles, and Z. Peng. Bus access optimization for distributed embedded systems based on schedulability analysis. In *Proc. Design, Automation and Test in Europe (DATE)*, 2000.

[71] PrimeXsys Platforms Overview, ARM.
`http://www.arm.com/products/solutions/PrimeXsysPlatforms.html`.

[72] Gang Qu and Miodrag Potkonjak. Techniques for energy minimization of communication pipelines. In *ICCAD*, San Jose, CA, United States, 1998.

[73] K. Richter and R. Ernst. Model interfaces for heterogeneous system analysis. In *Proc. 6th Design, Automation and Test in Europe (DATE)*, Munich, Germany, March 2002.

[74] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4):60–67, 2003.

[75] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Bottom-up performance analysis of Hw/Sw platforms. In *Proc. Distributed and Parallel Embedded Systems Conference (DIPES)*, Montreal, Canada, 2002.

[76] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model composition for scheduling analysis in platform design. In *Proc. 39th Design Automation Conference (DAC)*, New Orleans, LA, June 2002. ACM Press.

[77] M.J. Rutten, J.T.J. van Eijndhoven, E.G.T. Jaspers, P. van der Wolf, O.P. Gangwal, and A. Timmer. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Design & Test of Computers*, 19(4):39–50, July-August 2002.

[78] M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol. Design of multi-tasking coprocessor control for eclipse. In *10th International Workshop on Hardware/Software Codesign (CODES)*, Colorado, USA, May 2002.

[79] M.J. Rutten, J.T.J. van Eijndhoven, and E.-J.D. Pol. Robust media processing in a flexible and cost-effective network of multi-tasking coprocessors. In *14th Euromicro Conference on Real-Time Systems (ECRTS)*, Vienna, Austria, June 2002.

[80] Seamless Hardware/Software Co-Verification, Mentor Graphics. `http://www.mento.com/seamless/`.

[81] P. Skelly, S. Dixit, and M. Schwartz. A histogram-based model for video behavior in an atm network. In *IEEE INFOCOM*, Florence, Italy, 1992.

[82] N.T. Slingerland and A.J. Smith. Design and characterization of the Berkeley multimedia workload. *Multimedia Syst.*, 8(4):315–327, 2002.

[83] K. Sreenivasan and A. J. Kleinman. On the construction of a representative synthetic workload. *Commun. ACM*, 17(3):127–133, 1974.

[84] Open SystemC Initiative. `http://www.systemc.org`.

[85] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures. In *DAC*, New Orleans, LA, USA, June 2002.

[86] P. van der Wolf, W.M. Kruijtzer, and J.T.J. van Eijndhoven. System-level design of embedded media systems. In *Tutorial at the 15th International Conference on VLSI*

Design (VLSI) and Asia and South Pacific Design Automation Conference (ASP-DAC) (joint conference), Bangalore, India, January 2002.

[87] G. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for MPEG-2 video applications. *IEEE Transactions on VLSI*, 12(1), 2004.

[88] The Cadence virtual component co-design. `http://www.cadence.com/products/vcc.html`.

[89] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP*, NY, USA, October 2003.