Performance analysis of SoC architectures based on network calculi

Vink, J.P.

Award date:
2007

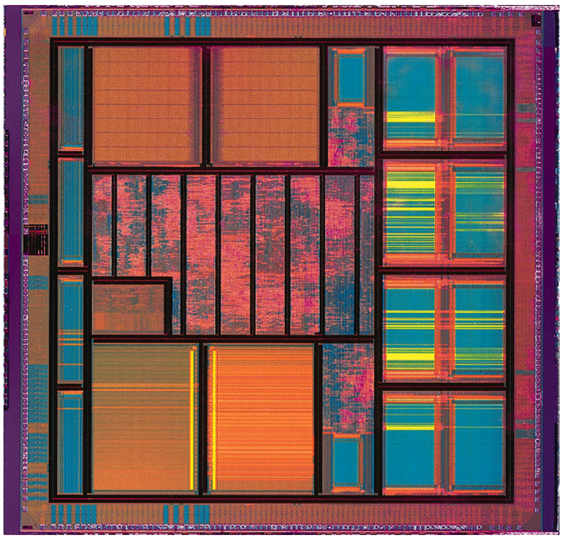
MASTER'S THESIS

Performance Analysis of SoC Architectures based on Network Calculi

by

J.P. Vink

Eindhoven, the Netherlands
version 1.1



Supervisor: Prof. Dr. Ir. C.H. van Berkel (NXP & TU/e)
Tutor: Dr. Ir. P. van der Wolf (NXP)

Abstract

This thesis describes a performance analysis method developed for obtaining upper bounds on delay and buffering requirements in a SoC architecture. These upper bounds are obtained via static analysis. Several traffic models and network calculi available in the literature are analyzed. The method of Stiliadis (Latency-Rate servers) is selected as foundation for the performance analysis method to characterize traffic and model interconnect.

The method of Stiliadis is not directly applicable for the performance analysis of SoC architectures, due to specific SoC characteristics. Therefore, it is adapted and extended to meet specific SoC requirements and to increase the accuracy. First of all, the upper bound for the delay of multiple packets is determined and a model of a memory system with suitable characteristics is added. Furthermore, the ability to express request-response streams and pipeline degrees of traffic streams are added. Several arbitration policies are added to model a variety of SoC communication infrastructures. Finally, regulators are added and the minimum latency of streams are determined to reduce the upper bound on the buffer requirements.

Complementary methods are required to obtain requirements and periodic timing schedules for the traffic. By using periodic timing schedules, dependencies between traffic streams are overcome. Backpressure is prevented by using buffers, large enough to receive all packets. Flow control is not possible in the performance analysis method at the cost of larger queues.

The applicability of the performance analysis method is illustrated by analyzing several schedule and interconnect variants for a multi-channel DVB-T set-top box case study. The influence of the frequency of the memory system and the pipeline degree of the traffic streams is shown in the multi-channel DVB-T set-top box case study. Furthermore, the influence of the packet size on the buffering requirements is shown in the results of the performance analysis method. Because the analysis method requires a small amount of execution time to analyze the performance of a SoC architecture, it is possible to analyze large numbers of SoC architecture variants.

The results of this performance analysis method give insight into the influences of several design decisions, like the frequency, the pipeline degree, the communication infrastructure and the packet size on the performance of the SoC. Therefore, this performance analysis method is useful to make architectural design decisions for SoC architectures.

Acknowledgment

This thesis concludes my study at the department of Computer Science and Engineering at Eindhoven University of Technology. This project was conducted at NXP Semiconductors Research, during the period from January 2007 to August 2007.

I would like to thank my supervisor, Kees van Berkel, for giving me the opportunity to do my Master's Thesis at NXP Semiconductors Research, for guiding me through the whole project and for all his input. I also would like to thank my tutor, Pieter van der Wolf, for all explanations, discussions, reviews and all the small talks we had.

Much appreciation goes to Özgün Paker and Tomas Henriksson, for their excellent and speedy support.

Furthermore, I also would like to thank the other members of the assessment committee, Pieter Cuijpers and Johan Lukkien, for reviewing my work.

Finally, a special word of thanks goes to my family and friends, and in particular to my parents, for all their support, encouragements and belief, especially during this project.

Jelte Peter Vink
Eindhoven, July 2007

Contents

The figure on the cover is a Core 2 X Woodcrest 65nm die, a dual-core processor of Intel [1]

Introduction

Contents

1.1 Introduction

Designing a chip becomes more and more complex. One of the reasons is that the number of transistors on a chip increases steadily. Intel's co-founder Gordon Moore predicted in 1965 that "the number of transistors on a chip doubles about every two years" [3]. Until now, his prediction has been proved to be valid [3] (see figure 1.1). Currently, we have processors with more than 1,000,000,000 transistors on a single chip. The problem is that the rate of the increase of the number of transistors per chip is higher than the rate of the increase of the number of transistors a designer can use in a design per time unit (see figure 1.2). The gap between both rates is called the "Design Productivity Gap" [7] and this gap is increasing. This means, there is a growing gap between what a process technology can offer and the ability to design in that process technology. It becomes harder to exploit all the functionality a chip may offer in newer process technologies.

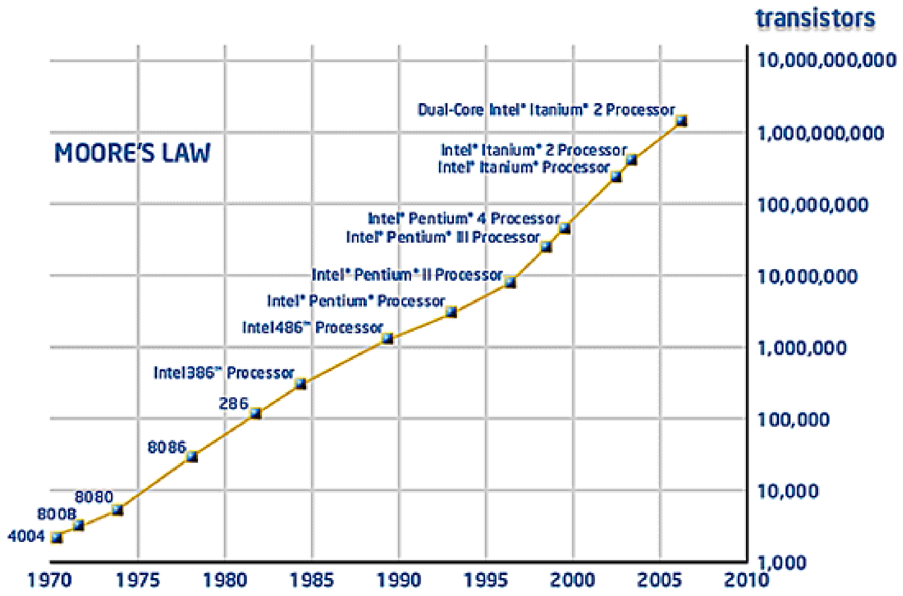


Figure 1.1: Moore's Law applied to Intel's processors [3]

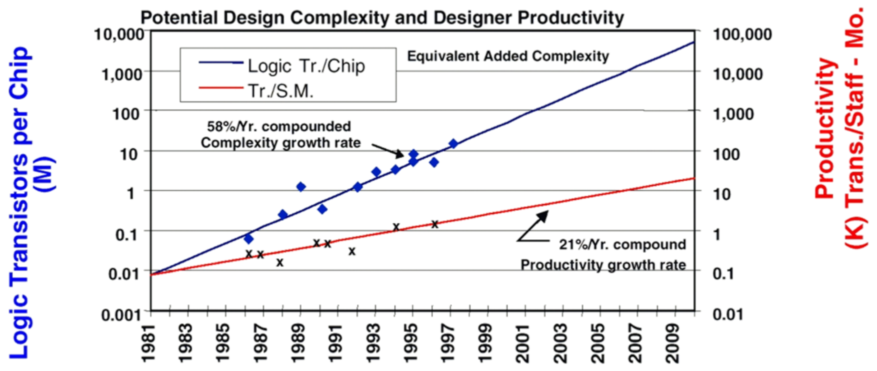


Figure 1.2: Design Productivity Gap [7]

1.2 System-on-Chip

Currently, some (embedded) systems are implemented by making use of complex chips that are typically referred to as System-on-Chips. A System-on-Chip (SoC) integrates components of an electronic system into a single chip. A SoC is composed of hardware blocks and software. A SoC can consist of hardware blocks like [5]

- one or more microcontrollers, microprocessors or Digital Signal Processors (DSP)
- one or more functional units, like a Video Processing unit
- memory blocks, like RAM, ROM or flash
- external interfaces

These hardware blocks are connected by interconnect. A block diagram of an example of a SoC is shown in figure 1.3

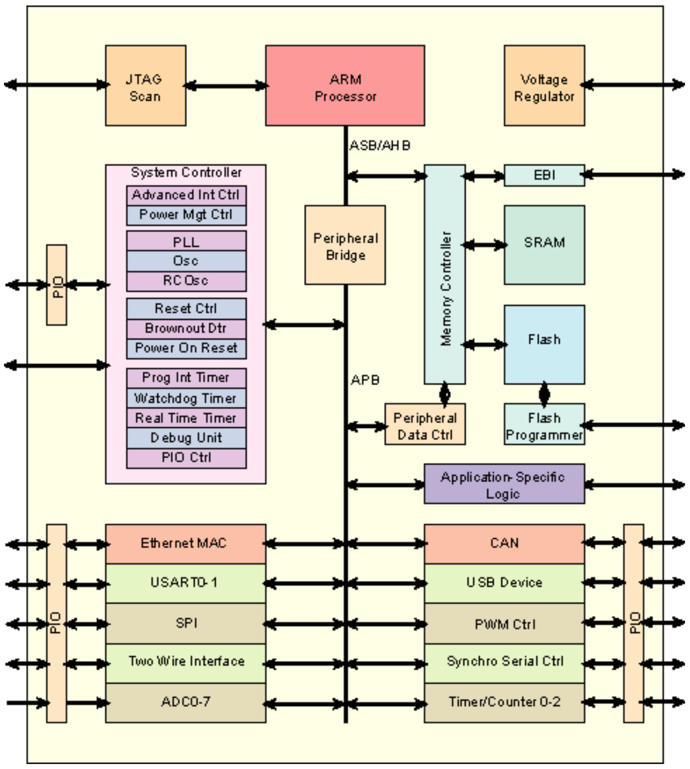


Figure 1.3: A block diagram of an example of a SoC, composed of hardware blocks [5]

Designing a SoC is a labour intensive and therefore expensive task. It requires large hardware and software design teams. The bulk of the effort of SoC design resides not in the design of the hardware blocks or the software, but in their integration of reused blocks into a working whole. Verifying that integrated hardware blocks and the software behave correctly with the required functionality and real-time performance is the bottleneck in current SoC design [12].

In order to address the complexity of SoC integration, a divide and conquer approach has been proposed [23]. Future SoCs can be built in a modular way by integrating subsystems. A subsystem is a major part of a system, which itself has the characteristics of a system. An example of a subsystem is a multi-standard modem or a media processing subsystem (see figure 1.4). Usually, these subsystems consist of several hardware blocks. These subsystems are coarse-grain, pre-integrated, pre-validated and autonomous.

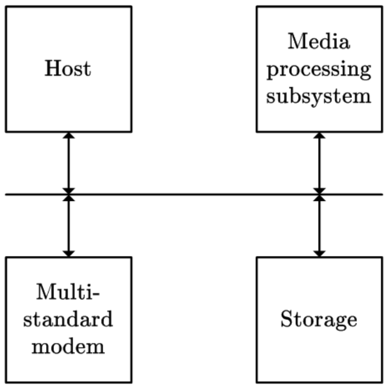


Figure 1.4: Example of a SoC, composed of subsystems [19]

1.3 SoC communication infrastructure

Subsystems interact with each other via the communication infrastructure. This communication infrastructure consists of (see figure 1.5)

- interconnect (consisting of e.g. links and arbiters)
- memory system(s)

The interconnect consists of the wires (also called the links) that connect different parts of the chip. Furthermore, in case a set of wires has to be merged, an arbiter (also called a scheduler) is used. An arbiter determines the interleaving of the data (i.e. the order of the data transfers) and is used when a resource (e.g. a memory system) is shared by multiple subsystems.

A memory system can be used to store private code of the subsystems as well as to store data. A memory system can be on-chip, as well as off-chip. In case that large amounts are required (e.g. for media processing) an off-chip memory system is cheaper than an on-chip memory system. In the case of an off-chip memory system several subsystems will probably share this resource. Then, this off-chip memory system can become a bottleneck for the performance of the SoC communication infrastructure.

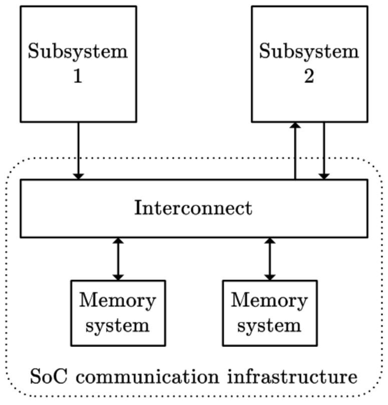


Figure 1.5: Example of a SoC communication infrastructure

In this thesis, we will focus on the traffic between the subsystems and the communication infrastructure.

1.4 Traffic

A SoC communication infrastructure is used by the subsystems connected to it to send and to receive data. This traffic can have different characteristics. One of the characteristics that determines the traffic is the rate of the traffic (i.e. the amount of data that is transferred per time unit). Also the moment in time and the length of the period of time of the data transfer determine the characteristics of the traffic. These characteristics of the traffic affect the requirements for the communication infrastructure. For example, the rate of the traffic determines the minimum rate of the interconnect.

On its turn, the performance of the SoC communication infrastructure impacts the performance of the subsystems. This is the case if a subsystem is waiting for data from the communication infrastructure. Then, the delay of the traffic (i.e. the time it takes to access data from a memory system) determines the amount of time the subsystem has to wait for its data.

Furthermore, the performance of a subsystem depends also on the actual traffic. This is the case when a resource in the communication infrastructure is shared by several subsystems (e.g. several subsystems use the same memory system). Then, the traffic of these subsystems to the shared resource has to be interleaved. This means that the traffic of one subsystem impacts the performance of another subsystem.

1.5 Problem definition

This master's thesis focuses on the integration of a set of subsystems with a communication infrastructure. Each subsystem has its own requirements. These include requirements on the SoC communication infrastructure for transferring data to and from the subsystem. As has become clear in the previous section, for a given communication infrastructure, the traffic characteristics have an impact on the performance of the subsystems.

Today, there is not enough support for a performance analysis method of SoC architectures. Nowadays, this is done by simulating a large number of scenarios. This approach is time consuming and does not give guarantees. To overcome these problems, a performance analysis method is required to give guarantees about the performance of a SoC communication infrastructure, without simulating all possible scenarios. The input of the performance analysis method (see figure 1.6) will be

- a model of the communication infrastructure
- the traffic of the subsystems

The output of the analysis method will be

- the cost of the SoC (e.g. the required area or the required amount of memory)
- the delay of the traffic
- the utilization of the SoC (e.g. of the memory or of the interconnect)

Making a performance analysis method for different types of traffic and different communication infrastructures is one of the research targets, this thesis is part of. The focus is not on the synthesis of the SoC communication infrastructure.

For the performance analysis method, the communication traffic has to be specified first. This can be done by a few characteristics, like the (minimum) rate. Then, several classes of traffic with corresponding characteristics can be defined.

In this thesis, we will address the following questions

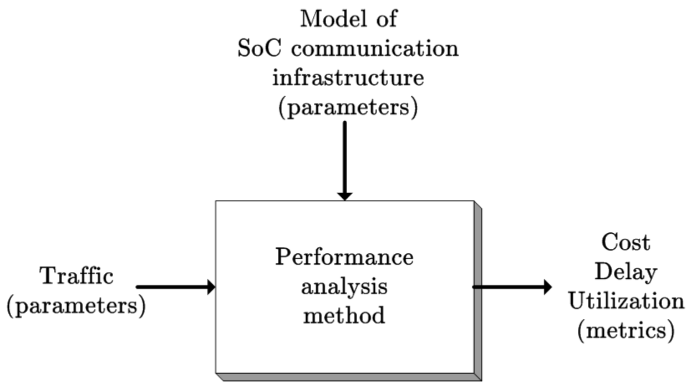


Figure 1.6: Model of the performance analysis method

1. Which characteristics determine the traffic, which are important, how can traffic (formally) be characterized and which traffic classes can be determined?
2. Which traffic classes are used by a concrete set of subsystems when executing a particular use case?
3. What are the characteristics of different communication infrastructures?
4. How can the characteristics of communication infrastructures be modeled?
5. What is computed (metrics) by the performance analysis method?
6. Which (combinations of) communication infrastructures are suitable for which traffic classes and what is the efficiency?
7. How can the performance analysis method be validated?
8. How can the system be checked whether it meets the required traffic requirements?

The final wish is to find a way to make traffic contracts between the subsystems and the communication infrastructures, such that if all parties adhere to the traffic contracts, and the combination of the traffic contracts is valid, then the systems always meet its required performance, even in the worst case. By that, simulations are no longer needed. Then, it is easier to add a new subsystem to a chip. Only the traffic contracts have to be checked. Also, subsystems can be developed (designed and validated) independently. So, the complexity of the integration of a set of subsystems with a communication infrastructure is strongly reduced.

For this thesis, the following assumptions are made

- The system and the transmission of the traffic are error-free
- There are no dependencies between the traffic of each subsystems
- The requirements of the traffic are derived by a complementary method
- The tasks, running on the subsystems, do not depend on each other (e.g. this may be the case if the tasks are periodic)

1.5.1 Application scope

The application scope of this thesis is video applications. Video applications require high rates of traffic and have tight delay constraints. Furthermore, large amounts of memory are required, caused by the large amounts of data that are transferred and need to be buffered.

1.6 Outline of Thesis

This thesis is structured as follows. In chapter 2 several traffic models available in the literature are explained. In chapter 3 two network calculi are described in more detail and one is selected as basis for the SoC performance analysis method. Then, in chapter 4 some improvements and extensions of the performance analysis method are explained. In chapter 5 a small case study is presented and the results are described and discussed. In chapter 6 a large case study is discussed. Finally, conclusions are drawn in chapter 7. In appendix A the validation of the performance analysis method is given. In appendix B several traffic classifications are discussed. Finally, in appendix C an overview is given of the Mathematica model of the performance analysis method.

1.7 Related work

In the context of SoC performance analysis, several techniques are available in the literature, but none has been found that is specific for SoC communication infrastructures. Cruz has pioneered a network calculus (see [9], [10]). He has made a mathematical framework for deriving worst-case bounds on the performance, like the delay. He assumed that the traffic can be bounded by a monotonously increasing function. His model is explained in more detail in the next chapter.

Based on the model of Cruz, Stiliadis and Varma (see [22], [21]) have created a model for "Latency-Rate servers" (i.e. an abstraction of the schedulers). They claimed that their performance bounds are tighter than the bounds of Cruz. More details can be found in the next chapters.

Chakraborty and Thiele used a task interaction approach in [8]. They defined bounds for arrival curves and service curves of the traffic. Their technique is related to the worst-case execution time of the tasks of a system. In this thesis, the focus is on the traffic between the subsystems and the communication infrastructure and not on the interaction of tasks.

Richter, Jersak and Ernst have looked in [18] to the shared resources that influence the performance. Their technique expresses the behavior of tasks (worst-case execution time) and local scheduling policies in a formalism for reasoning. Again, the focus in this thesis is on the traffic between the subsystems and the SoC communication infrastructure instead of on the behavior of tasks.

No SoC performance analysis has been found that is specific for SoC communication infrastructures.

Traffic models

Contents

Please note that all definitions and formulas in this chapter are from the literature.

2.1 General definitions and notations

In this section we define traffic models, that can be used in the SoC performance analysis method.

2.1.1 Traffic stream

First, a traffic stream is defined.

Definition *A traffic stream is the flow of related data that has the same source and the same destination*

A traffic steam is also known as a "stream" or a "session".

It is assumed that a packet is the unit of data transport. A packet is defined as a block of related data. The size of the packets can be uniform (all packets of all streams have the same size), but this is not required. It is assumed that the size of all packets of a single stream is the same.

Definition *A word is a fixed-sized group of bits that are handled together*

Definition *Let L_i be the size in words of the packets of stream i, where $L_i \geq 1$ and $L_i \in \mathbb{N}$*

Furthermore, it is assumed that if a packet is transmitted, the rate of the transmission is equal to the maximum rate, or capacity, of the communication infrastructure. This means that the communication infrastructure is transmitting data at its maximum rate, or it is not transmitting data. So, it is not possible to transfer data at a different rate than the maximum rate.

Definition *Let C be the maximum rate, or capacity, in $\frac{words}{sec}$ of the communication infrastructure, where $C \geq 0$ and $C \in \mathbb{R}$*

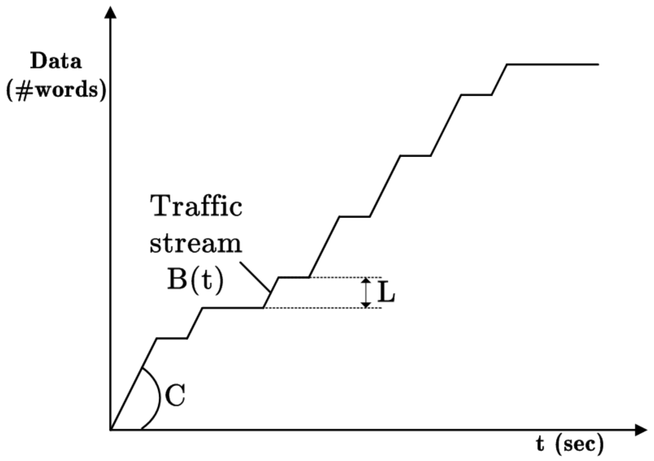


Figure 2.1: Graphical representation of a traffic stream

An example of a traffic stream is shown in figure 2.1. $B(t)$ is the amount of data transferred on a link at time $t \geq 0$, for $t \in \mathbb{R}$. $B(t)$ is a strictly non-decreasing function, because a decrease of the amount of transferred data is not possible. As is shown in the figure and explained above, the instantaneous rate of $B(t)$ is equal to either 0 or C.

Definition *A burst is a number of packets in series, that are transferred at rate C*

The packets of a traffic stream can be sent in bursts. The number of transferred *words* in figures 2.1 and 2.2 is not equal for each period of time. Also the number of *words* per burst is not uniform, as is indicated in figures 2.1 and 2.2. This means that the length of all bursts is not the same.

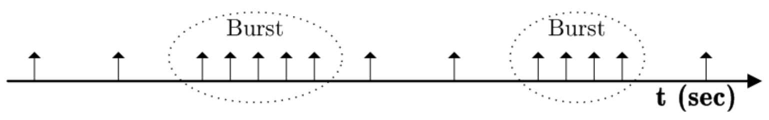


Figure 2.2: Graphical representation of bursts of a traffic stream. Each arrow represents the arrival of a packet

2.1.2 Interconnect

The above definitions are used to define properties of the interconnect. The interconnect is represented in this thesis as the connection of basic building blocks, called network elements. Most network elements have one or more input or arriving streams and one or more output or departing streams. Let $A_i(t)$ denote the number of arrived *words* of stream i at time $t \geq 0$, for $t \in \mathbb{R}$, and $B_i(t)$ denote the number of departed *words* of stream i at time $t \geq 0$, for $t \in \mathbb{R}$. Assume that no *words* are added by the network element to stream i. Then for all streams i

$$\Big(\forall t \in \mathbb{R} : t \geq 0 : A_i(t) \geq B_i(t)\Big) \tag{2.1}$$

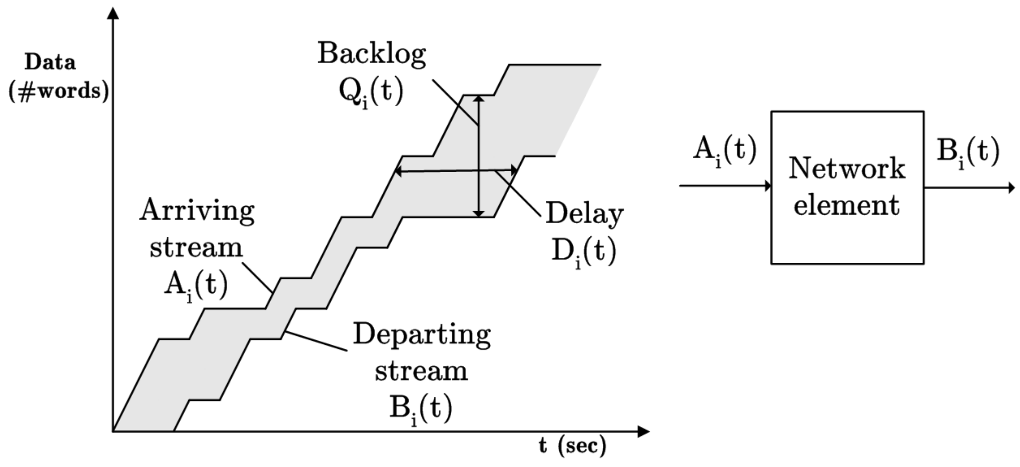


Figure 2.3: Graphical representation of $A_i(t)$, $B_i(t)$, $Q_i(t)$ and $D_i(t)$

Definition *The backlog is the amount of data that has arrived in a network element, but has not departed yet. Let backlog $Q_i(t)$ represent the number of words of stream i queued in the network element at time $t \geq 0$, $t \in \mathbb{R}$, that is*

$$Q_i(t) = A_i(t) - B_i(t) \tag{2.2}$$

Definition *The delay is the amount of time between the arriving and departing of the data of a stream in a network element. Let $D_i(t)$ denote the delay in sec of stream i in the network element at time $t \geq 0$, $t \in \mathbb{R}$, that is*

$$D_i(t) = \min_{\Delta \;:\; \Delta \geq 0 \;\wedge\; \Delta \in \mathbb{R} \;\wedge\; A_i(t) = B_i(t+\Delta)} \Big\{\Delta\Big\} \tag{2.3}$$

The above definitions are indicated in figure 2.3. The delay is the horizontal distance between the arriving and the departing stream. The vertical distance between both represents the backlog.

Please note that in this thesis, $t \geq 0$, $t \in \mathbb{R}$ is used to represent the time in *sec*. In the case that several moments of time have to be indicated, t_j, $t_j \in \mathbb{R}$ and $j \in \mathbb{N}$, will be used such that

$$t_0 = 0 \textit{ and } t_j \leq t_{j+1} \tag{2.4}$$

Using these notations and definitions, the selection criteria for the traffic model for a SoC performance analysis method are presented in the next section.

2.2 Selection criteria

Before we explain several ways of characterizing traffic, it is important to know what the selection criteria are.

Compositionality is a must for the performance analysis method. This means that a complex expression (i.e. all the traffic of a SoC transferring over the communication infrastructure) is determined by the meanings of its constituent expressions (i.e. individual streams and network elements) and formulas are used to combine them. So, it should be possible to look to individual streams and network elements. Formulas should be available to combine them. Those formulas must give upper bounds for the delay and the required amount of memory in the communication infrastructure as required for the SoC performance analysis method.

Finally, it is assumed that the traffic of a SoC will use packets and bursts. Therefore, it should be possible to express these in the traffic model.

With these requirements in mind, the literature has been studied for traffic models. Only a few traffic models meet (partially) the selection criteria. Those traffic models are explained in the next sections.

2.3 (σ, ρ)

Cruz represents a traffic stream by a nonnegative function $R(t)$ as follows [9]

Definition *$R(t)$ represents the instantaneous rate in $\frac{words}{sec}$ of traffic for the stream flowing on a link at time t. For any t_1 and t_2, $\int_{t_1}^{t_2} R(t)dt$ is the number of data in words of the stream that is transmitted on a link of the communication infrastructure in the interval $[t_1, t_2]$*

Just as before, $R(t)$ can take on only two values, namely 0 and C.

2.3.1 Traffic stream

Cruz characterizes in [9] and in [11] a traffic stream using two different parameters, namely σ and ρ, such that

Definition *Given σ in words, $\sigma \geq 0$ and $\sigma \in \mathbb{R}$, and ρ in $\frac{words}{sec}$, $\rho \geq 0$ and $\rho \in \mathbb{R}$, $R(t)$ is bounded by (σ, ρ) if and only if for all t_1 and t_2*

$$\int_{t_1}^{t_2} R(t)dt \leq \sigma + \rho \cdot (t_2 - t_1) \tag{2.5}$$

This is denoted as $R(t) \sim (\sigma, \rho)$ or $R \sim (\sigma, \rho)$

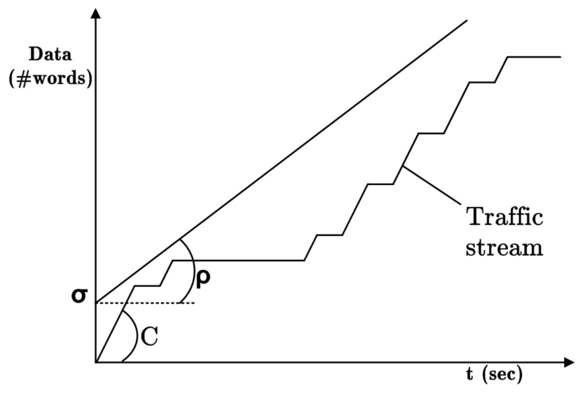


Figure 2.4: (σ, ρ) characterization of a traffic stream, the traffic of which is upper bounded by the diagonal line

Thus, if $R \sim (\sigma, \rho)$, then there is an upper bound to the amount of traffic transmitted in any interval that is equal to a constant σ plus a quantity proportional to the length of the interval (see figure 2.4).

This means that ρ is the rate at which the traffic is transmitted and σ represents the burstiness constraint of the traffic [9]. Please note that ρ is upper bounded by C i.e.

$$0 \leq \rho \leq C \tag{2.6}$$

Furthermore, σ is lower bounded by [9]

$$L \cdot \left(1 - \frac{\rho}{C}\right) \leq \sigma \tag{2.7}$$

This is the case, because the minimum burst is equal to one packet. Because the size of the packets is L and $R(t) \in \{0, C\}$ for all t, then [9]

$$\int_{t}^{t+L/C} R(t)dt \leq L \tag{2.8}$$

This means that it is possible to send a packet of length L in $\frac{L}{C}$ *sec*. In that period of time, ρ can characterize the transfer of only $\rho \cdot \frac{L}{C}$ *words* of the packet. Therefore to transfer L *words*, σ is lower bounded by $L \cdot (1 - \frac{\rho}{C})$ *words*.

A more general characterization than (σ, ρ) can be used, by making use of a non-decreasing function $b(t)$, such that [9]

$$\int_{t_1}^{t_2} R(t)dt \leq b(t_2 - t_1) \tag{2.9}$$

for all $0 \leq t_1 \leq t_2$. Now, (σ, ρ) traffic can be represented by $R \sim b$, where $b(t) = (\sigma + \rho \cdot t)$, for some $\sigma \geq 0, \rho \geq 0$, $\sigma \in \mathbb{R}$ and $\rho \in \mathbb{R}$. Then, R is (σ, ρ)-smooth or b-smooth [9].

As has described above, Cruz has defined traffic by making use of (σ, ρ). Then, it is possible to derive properties like the backlog and the delay.

2.3.2 Backlog

The above definition (see equation 2.9) can be used to determine the backlog of the network elements. Assume that a network element receives data at a rate described by $R_{in}(t)$ and sends that data at rate ρ, $\rho \geq 0$ and $\rho \in \mathbb{R}$ (see figure 2.5).

Then, according to Cruz [9], the next property is defined.

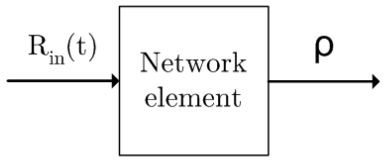


Figure 2.5: A network element receives data at a rate described by $R_{in}(t)$ and sends data at rate ρ

Property *$Q_\rho(R)(t)$ represents the backlog in words in the network element at time $t \geq 0$. Then,*

$$Q_\rho(R_{in})(t) = \max_{t_1: 0 \leq t_1 \leq t} \left\{ \int_{t_1}^{t} R_{in}(t)dt - \rho \cdot (t - t_1) \right\} \tag{2.10}$$

2.3.3 Delay

The above definition (see equation 2.9) can also be used to determine the delay of a network element. Assume that a network element receives data at a rate described by $R_{in}(t)$ and outputs its data according to (σ_{out}, ρ), for some $\sigma_{out} \geq 0, \rho \geq 0$, $\sigma_{out} \in \mathbb{R}$ and $\rho \in \mathbb{R}$.

Then, according to Cruz [11]

Property *$D(t)$ represents the delay in sec in the network element at time $t \geq 0$. Then,*

$$D(t) \leq \frac{Q_\rho(R_{in})(t) + \sigma_{out}}{\rho} \tag{2.11}$$

In case that $R_{in} \sim (\sigma_{in}, \rho)$, for $\sigma_{in} \geq 0$ and $\sigma_{in} \in \mathbb{R}$, then

$$D(t) \leq \frac{\sigma_{in} + \sigma_{out}}{\rho} \tag{2.12}$$

In [11], also a delay constraint is derived for a communication infrastructure, consisting of H arbitrary network elements in a chain, with $H \geq 1$ and $H \in \mathbb{N}$. Let R_{in} describe the input traffic of the chain and let $R_{h-1}(t)$ describe the traffic entering network element h, with $1 \leq h \leq H$, for $h \in \mathbb{N}$ and $R_{in} \sim (\sigma_{in}, \rho)$, for $\sigma_{in} \geq 0$, $\rho \geq 0$, $\sigma_{in} \in \mathbb{R}$ and $\rho \in \mathbb{R}$. Suppose that network element h outputs its data according to (σ^h_{out}, ρ), for each h, $\sigma^h_{out} \geq 0$ and $\sigma^h_{out} \in \mathbb{R}$. Then, the total delay at the output of network element H at time $t \geq 0$ is upper bounded by [11]

$$D_H(t) \leq \frac{\sigma_{in} + \sum_{h=1}^{H} \sigma^h_{out}}{\rho}, \tag{2.13}$$

for all $t \geq 0$.

So, the delay is determined by making use of σ_{in} of the input traffic and of σ^h_{out} of each network element. Moreover, ρ holds for the input and the output traffic.

2.3.4 Extra properties

Not all traffic streams will be (σ, ρ)-smooth by their own. A (σ, ρ) regulator can be used to transform an arbitrary traffic stream into a (σ, ρ)-smooth stream. The main task of a (σ, ρ) regulator is to buffer (i.e. temporarily store) the packets if needed and to output the packets such that $R_{out}(t)$ of the regulator is (σ, ρ)-smooth, for some $\sigma \geq 0$, $\rho \geq 0$, $\sigma \in \mathbb{R}$

and $\rho \in \mathbb{R}$. Assume that the regular receives data at a rate described by $R_{in}(t)$. Then, its output is according to [11]

$$R_{out}(t) = \begin{cases} C & \text{if } \Big(Q(t) > 0 \vee R_{in}(t) > 0\Big) \text{ and } Q_\rho(R_{out})(t) < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

So, a packet will depart from the (σ, ρ) regulator at time t if there is a packet available at time t (i.e. a packet is stored, $Q(t) > 0$, or a new packet just arrived, $R_{in}(t) > 0$) and if the output has not yet produced at time t a burst equal to or larger than σ.

2.3.5 Applicability of the method of Cruz

The method of Cruz, as partly described in the previous section, can be evaluated using the selection criteria of section 2.2.

First of all, it is possible to look to individual traffic streams and to get bounds for individual traffic streams. The delay and the backlog can be calculated, after the stream is characterized by (σ, ρ).

Secondly, it is also possible to determine the delay in a chain of multiple network elements. This is important for the performance analysis method to manage the complexity. Furthermore, the backlog can be determined for a single network element. For calculating the backlog in a chain of multiple network elements, there has been no general rule or definition found in the literature.

Finally, the method makes use of the length of the packets, and it is possible to express the burstiness of the traffic.

Summarizing, the method of Cruz meets the selection criteria of section 2.2. Unfortunately, no applications to practical examples of this method have been found in the literature.

2.4 Latency-Rate Servers

As an addition to the traffic model presented in the previous section, Stiliadis and Varma focused in [21] and [22] more on the latency and the rate of a system. They used a notation similar to Cruz in [9] and [11].

Please note that Stiliadis used different terms than Cruz. First of all, a “traffic stream” in the method of Cruz is called a “session” in the method of Stiliadis. Furthermore, the “amount of departed data of a stream” in the method of Cruz is called the “amount of service that stream received” in the method of Stiliadis.

In the method of Stiliadis the focus is more on schedulers. Multiple input sessions of a scheduler share a common output link of the scheduler (see figure 2.6). These sessions receive service by the scheduler. The next definition is used.

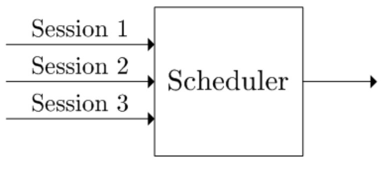


Figure 2.6: 3 sessions share a common output link of the scheduler

Definition *$A_i(t_1, t)$ and $W_i(t_1, t)$ denote the number of arrived data in words of session i by a scheduler and respectively the number of service in words session i received by a scheduler during the interval (t_1, t), where $0 \leq t_1 \leq t$*

In [22] it is assumed that V sessions, $V \geq 1$ and $V \in \mathbb{N}$, share a common output link of a scheduler. ρ_i is the (service) rate the scheduler allocate to session i, for $\rho_i \geq 0$ and $\rho_i \in \mathbb{R}$. The schedulers are noncut-through devices, meaning that a packet has to be received completely, before it can be transmitted. Finally, it is assumed in [22] that all links and schedulers have a maximum rate (i.e. capacity) of $C \ \frac{words}{sec}$, for $C \geq 0$ and $C \in \mathbb{R}$.

Then, $Q_i(t)$ represents the number of *words* of session i queued in the scheduler at time $t \geq 0$, such that [22]

$$Q_i(t) = A_i(0, t) - W_i(0, t) \tag{2.15}$$

The sessions receive service by the scheduler. If the arrival rate of a session in a period of time is not lower than the service rate of that session, a busy period occurs.

Definition *A busy period is the maximum interval of time $(t_1, t_2]$ such that at any time $t \in (t_1, t_2]$, the accumulated arrivals of session i since the beginning of the interval do not fall below the total service received during the interval at a rate of exactly ρ_i, $\rho_i \geq 0$ and $\rho_i \in \mathbb{R}$, i.e. $A_i(t_1, t) \geq \rho_i \cdot (t - t_1)$*

This definition is shown in figure 2.7. The intervals (t_1, t_2) and (t_3, t_4) are the maximum intervals of time such that the amount of arrived data (i.e. the solid line) in these intervals is more than the service received (i.e. the dashed line) in the same intervals. Please note that during a busy period, there is always data on the output link, because there is enough arriving data. Furthermore, the data on the output link has at least a rate of ρ_i.

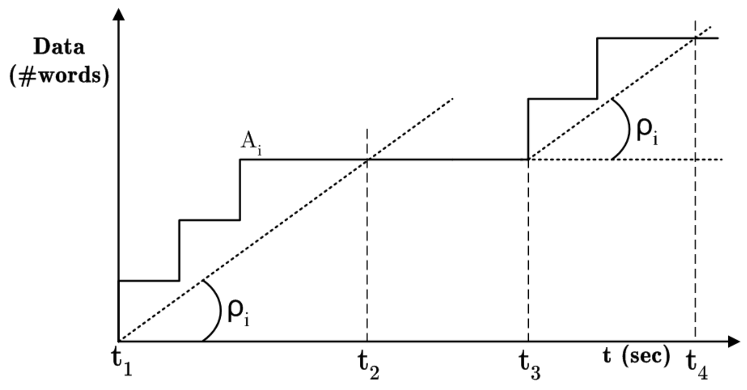


Figure 2.7: Two busy periods of session i

Please note that Stiliadis assumes in [22] that packet departures are considered as impulses (i.e. events).

Then, the total amount of received service of a session can be defined.

Definition *$W_{i,j}^S(t_1, t)$ is the total service in words provided by server S to the traffic of session i arrived during the j^{th} busy period, $j \geq 1$ and $j \in \mathbb{N}$, that started at t_1, until time t, where $0 \leq t_1 \leq t$*

Please note that a busy period has an unique starting point in time.

The above definition is used to define a general class of schedulers, called Latency-Rate servers [22]. An *LR* server makes use of the service rate of a session and the latency of the session. The latency of a session is defined as the maximum amount of time between the moment the first packet of a busy period of that session has arrived completely in the scheduler and the moment that packet has left the scheduler completely.

Definition *Let t_1 be the starting time of the j^{th} busy period, $j \geq 1$ and $j \in \mathbb{N}$, of session i in server S, t_2 be the time at which the last bit of the session arrived during the j^{th} busy period leaves the scheduler and ρ_i be the allocated rate by server S to session i, for $\rho_i \geq 0$ and $\rho_i \in \mathbb{R}$. Then a scheduler S is called an LR server (i.e. a Latency-Rate server) if and only if a nonnegative constant C_i^S can be found such that for every $t \in (t_1, t_2]$*

$$W_{i,j}^{S}(t_1, t) \geq \max\left\{0, \rho_i \cdot \left(t - t_1 - C_i^S\right)\right\} \tag{2.16}$$

The minimum nonnegative constant C_i^S satisfying the above inequality (over all busy periods) is defined as the latency in sec of session i of the scheduler S, denoted by Θ_i^S

This means that the scheduler S guarantees a service rate of at least ρ_i, a maximum of Θ_i^S *sec* later than the busy period for session i starts.

In other words, assume that at t_1 the busy period of session i starts. Then, in the time period $t_1 + \Theta_i^S$ until t, the output rate of scheduler S is at least ρ_i. Therefore, the amount of service received by session i in this period is at least $\rho_i \cdot (t - t_1 - \Theta_i^S)$.

Examples of *LR* servers are given in the next chapter.

Using this definition, an upper bound for the service provided by a network chain of m *LR* servers, $m \geq 1$ and $m \in \mathbb{N}$, is derived in [22]. Let t_1 be the starting time of the j^{th} busy period, $j \geq 1$ and $j \in \mathbb{N}$, of session i and ρ_i the minimum rate allocated to session i in the network chain. The service provided to the traffic of the j^{th} busy period of session i after the k^{th} *LR* server, $1 \leq k \leq m$ and $k \in \mathbb{N}$, during the interval $(t_1, t]$ is

$$W_{i,j}^{S_k}(t_1, t) \geq \max\left\{0, \rho_i \cdot \left(t - t_1 - \sum_{j=1}^{k} \Theta_i^{(S_j)}\right)\right\} \tag{2.17}$$

where $t_1 \leq t$ and $\Theta_i^{(S_j)}$ the latency of the j^{th} server in the network chain for session i.

2.4.1 Traffic stream

Stiliadis assumed in [22] that the input traffic of a network element is token-bucket-shaped.

Definition *The arrivals of session i at the input of a scheduler in the interval $(t_1, t]$, where $0 \leq t_1 \leq t$, are "token-bucket-shaped" if*

$$A_i(t_1, t) \leq \sigma_i + \rho_i \cdot (t - t_1) \tag{2.18}$$

for the parameters $\sigma_i \geq 0$ and $\rho_i \geq 0$, where $\sigma_i \in \mathbb{R}$ and $\rho_i \in \mathbb{R}$ denote the burstiness constraint and the average rate of session i, respectively

A regulator is used to shape the incoming traffic (see figure 2.8). Tokens are generated at a rate of ρ_i. A token represents a *word* of a packet. A packet can only be released from the buffer of the regulator after removing the required number of tokens (i.e. the length of the packet) from the token bucket. The queue for the incoming traffic is finite and the token bucket has a maximum of σ_i tokens. The output of the regulator is connected to the input of the network. The traffic that enters the network can be characterized as token-bucket-shaped.

Please note that this definition is analog to the definition of (σ, ρ) of Cruz (see equation 2.5).

As has become clear above, Stiliadis defined *LR* servers to model schedulers and defined traffic by σ and ρ. Then, it is possible to derive properties like the backlog and the delay.

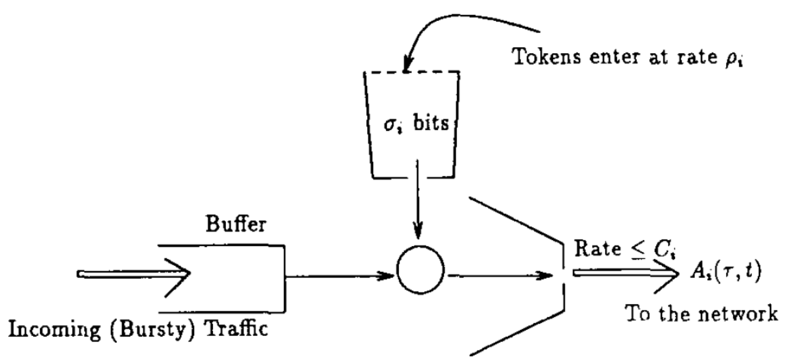


Figure 2.8: Regulator [17]

2.4.2 Backlog

Assume that input session i of a chain of m LR servers, $m \geq 1$ and $m \in \mathbb{N}$, is token-bucket-shaped with the parameters $\sigma_i \geq 0$ and $\rho_i \geq 0$, $\sigma \in \mathbb{R}$ and $\rho \in \mathbb{R}$, and $\Theta_i^{(S_j)}$ is the latency of the j^{th} LR server, $1 \leq j \leq m$ and $j \in \mathbb{N}$, for session i.

Assume that at t_1 session i sends its maximum burst (i.e. σ_i) to the first LR server and after that sends data with a rate of ρ_i. In the worst case, the first packet of session i leaves the first LR server $\Theta_i^{(S_1)}$ *sec* later. Then, this first LR server should be able to store the burst as well as the data that arrives in the period t_1 until $t_1 + \Theta_i^{(S_1)}$. Therefore, the first LR server of the chain has a backlog of at most $\sigma_i + \rho_i \cdot \Theta_i^{(S_1)}$. For the next LR server, the backlog requirement increases with $\rho_i \cdot \Theta_i^{(S_2)}$.

According to Stiliadis in [22], the next property is defined.

Property *The backlog $Q_i^{(S_k)}$ in words in the k^{th} LR server, $1 \leq k \leq m$ and $k \in \mathbb{N}$, of the chain of m LR servers of a session i is upper bounded by*

$$Q_i^{(S_k)} \leq \sigma_i + \rho_i \sum_{j=1}^{k} \Theta_i^{(S_j)} \tag{2.19}$$

This formula represents an upper bound of the required buffer size of an LR server in the chain of the LR servers.

So, the backlog is defined by making use of the characteristics of the input traffic (i.e. σ and ρ) and the latency of the session in the LR servers.

2.4.3 Delay

Assume that the input session i of LR server S is token-bucket-shaped with the parameters $\sigma_i \geq 0$ and $\rho_i \geq 0$, $\sigma_i \in \mathbb{R}$ and $\rho_i \in \mathbb{R}$, and that the latency of the server for session i is $\Theta_i^S \geq 0$. Then, according of Stiliadis in [22], the next property can be defined.

Property *The maximum delay D_i^S in sec of any packet of session i in S is upper bounded by*

$$D_i^S \leq \frac{\sigma_i}{\rho_i} + \Theta_i^S \tag{2.20}$$

Assume that at t_1 session i sends its maximum burst (i.e. σ_i) to the LR server S. In the worst case, the first packet of session i leaves this LR server Θ_i^S *sec* later. After that, the burst is serviced (i.e. output) with a rate of at least ρ_i. Therefore, the delay is upper bounded by $\frac{\sigma_i}{\rho_i} + \Theta_i^S$ *sec*.

The maximum delay D_i in *sec* of session i in a chain of m *LR* servers, $m \geq 1$ and $m \in \mathbb{N}$, is according to [22] upper bounded by

$$D_i \leq \frac{\sigma_i}{\rho_i} + \sum_{j=1}^{k} \Theta_i^{(S_j)} \tag{2.21}$$

where $\Theta_i^{(S_j)}$ is the latency of the j^{th} *LR* server, $1 \leq j \leq m$ and $j \in \mathbb{N}$, for session i.

Please note that this end-to-end delay is determined by the burstiness and the rate of the input session and the latencies of the individual servers on the path of the session.

2.4.4 Extra properties

As proved in [22], the traffic of a session i after the k^{th} node, $1 \leq k \leq m$ and $k \in \mathbb{N}$, in a chain of m *LR* servers, $m \geq 1$ and $m \in \mathbb{N}$, conforms to the token-bucket model for the parameters

$$\sigma_i + \rho_i \sum_{j=1}^{k} \Theta_i^{(S_j)} \text{ and } \rho_i \tag{2.22}$$

if the input traffic was characterized by $\sigma_i \geq 0$ and $\rho_i \geq 0$, $\sigma_i \in \mathbb{R}$ and $\rho_i \in \mathbb{R}$, and the latency for session i of the j^{th} *LR* server, $1 \leq j \leq m$ and $j \in \mathbb{N}$, is $\Theta_i^{(S_j)}$.

2.4.5 Applicability of the method of Stiliadis

The method of Stiliadis, as described in the previous section, can be evaluated again using the selection criteria of section 2.2.

First of all, it is possible to look to individual traffic sessions and to get bounds for individual traffic sessions. The delay and the backlog can be calculated per *LR* server, if the input session is token-bucket-shaped with σ and ρ as parameters.

Secondly, the method is compositional. It is possible to determine the latency for every *LR* server in isolation. Then, the delay and the backlog can be calculated for a chain of *LR* servers. This is important for the performance analysis method to manage the complexity.

Finally, since we only explained a part of the method of Stiliadis, it is not yet clear how to make use of the length of the packets. This is discussed in the next chapter. For traffic that is token-bucket-shaped, it is possible to express the burstiness constraint of the traffic.

Summarizing, also the method of Stiliadis meets the selection criteria of section 2.2. Unfortunately, no applications to practical examples have been found in the literature.

2.5 (ρ_{max}, ρ_{min})

Wang introduced in [25] a completely different way to characterize traffic. He looked to the arriving and departing traffic of a server. First we will introduce two definitions, to be used for the parameters defined by Wang to characterize traffic.

Definition *a_i and d_i represent the arrival and the departure time in sec of the i^{th} packet of a server, respectively, for $i \geq 1$ and $i \in \mathbb{N}$*

Definition *$A(t_1, t_2)$ denotes the number of packets arrived at a server in the interval $[t_1, t_2)$*

2.5.1 Traffic stream

Using the above definitions, the traffic parameters ρ_{min} and ρ_{max} can be defined [25].

Definition

$$\rho_{min} = \min\left\{X_2, ..., X_M\right\}$$

and

$$\rho_{max} = \max\left\{Y_2, ..., Y_M\right\}$$

where

$$X_m = \frac{A(a_1, a_m)}{a_m - a_1}$$

and

$$Y_m = \frac{1}{a_m - a_{m-1}}$$

for $2 \le m \le M$*,* $m \in \mathbb{N}$ *and* $M \in \mathbb{N}$

ρ_{min} is equal to the lowest rate in $\frac{\#packets}{sec}$ of arrived packets in the interval (a_1, a_M). This means, ρ_{min} represents the highest rate in $\frac{\#packets}{sec}$ at which the server is never idle in the interval (a_1, a_M). ρ_{max} is the highest rate in $\frac{\#packets}{sec}$ of the incoming traffic in the interval (a_1, a_M), see figure 2.9.

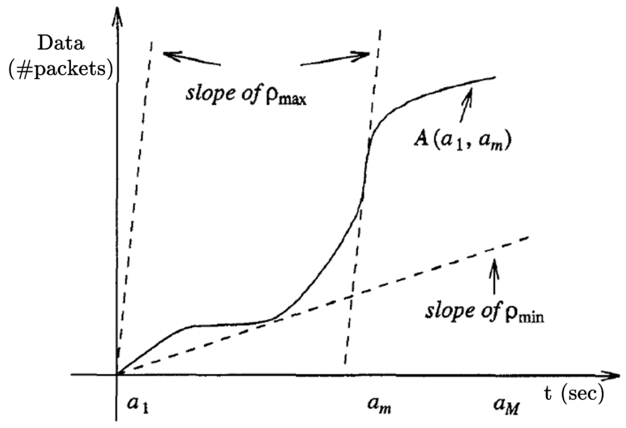


Figure 2.9: Graphical explanation of ρ_{min} and ρ_{max} [25]

To express the burstiness of the traffic, the definition of “synchronization unit” is required [25]. A “synchronization unit” G represents a group of M packets, for $M \ge 2$ and $M \in \mathbb{N}$.

Definition *A synchronization unit* G *conforms to* (ρ_{max}, ρ_{min}) *(denoted by* $G \sim (\rho_{max}, \rho_{min})$*) if* $X_m \ge \rho_{min}$ *and* $Y_m \le \rho_{max}$*, for* $2 \le m \le M$ *and* $m \in \mathbb{N}$

Using this definition, the bounds for the arrival time of a packet m of a synchronization unit G of M packets are derived in [25]. Assume that $G \sim (\rho_{max}, \rho_{min})$. Then

$$a_1 + \frac{m-1}{\rho_{max}} \le a_m \le a_1 + \frac{m-1}{\rho_{min}} \tag{2.23}$$

for $2 \le m \le M$ and m and $M \in \mathbb{N}$.

This defines the interval in which packet m will arrive. These bounds are determined by the minimum and the maximum rate.

Using these bounds, the burstiness is defined [25].

Definition *Burstiness (b_m) is the amount of time in sec between the upper bound of the m^{th} packet's arrival interval and the actual arrival time of packet m, i.e.*

$$b_m = a_1 + \frac{m-1}{\rho_{min}} - a_m \tag{2.24}$$

for packet $1 \leq m \leq M$ *and* $m \in \mathbb{N}$

This means that at time $a_1 + \frac{m-1}{\rho_{min}}$ the server starts to process the m^{th} packet, if the server has a rate of ρ_{min}. b_m can be interpreted as the amount of time that the m^{th} packet has to wait in a queue before it is served, assuming that the server works at a rate of ρ_{min}.

A bound for the burstiness of traffic is derived in [25]. If $G \sim (\rho_{max}, \rho_{min})$, $2 \leq m \leq M$, $m \in \mathbb{N}$ and $M \in \mathbb{N}$, then

$$b_m \leq \frac{m-1}{\rho_{min}} \cdot \left(1 - \frac{\rho_{min}}{\rho_{max}}\right) \tag{2.25}$$

This means that if ρ_{min} and ρ_{max} become closer to each other, the upper bound of b_m decreases.

As is described above, (ρ_{max}, ρ_{min}) is used in this method to characterize traffic.

2.5.2 Backlog

Unfortunately, no information has been found in the literature on how to determine the backlog, using this method.

2.5.3 Delay

Assume that a server is work-conserving (i.e. the server is never idle when there is backlog) and the flow is served at rate $\mu \geq \rho_{min}$ in $\frac{\#packets}{sec}$, $\mu \in \mathbb{R}$. Let d_m be the time in *sec* when the m^{th} packet departs from a server. The delay experienced by the first packet depends on the backlog of packets left over from the previous synchronization unit. Suppose D, $D \geq 0$ and $D \in \mathbb{R}$, is the time in *sec* that the first packet has to wait before being served. Then, according to [25]

$$d_1 = D + \frac{1}{\mu} + a_1 \tag{2.26}$$

Furthermore, according to [25], packet m departs from the server at

$$d_m = \max\left\{d_{m-1}, a_m\right\} + \frac{1}{\mu} \tag{2.27}$$

for $2 \leq m \leq M$, $m \in \mathbb{N}$ and $M \in \mathbb{N}$.

Unfortunately, no information has been found in the literature on how to determine D for this method.

2.5.4 Extra properties

The relation between the input traffic and the output traffic of a server is determined in [25]. Assume that a server receives traffic, that conforms to (ρ_{max}, ρ_{min}). Assume that the traffic is served at rate $\mu \geq \rho_{min}$, $\mu \in \mathbb{R}$. Then the output traffic of the server conforms to (μ, ρ_{min}).

2.5.5 Applicability of the method of Wang

The (ρ_{max}, ρ_{min}) method, as described in the previous section can be evaluated, using the same selection criteria as before.

First of all, it is possible to analyze individual traffic streams and to get bounds for individual traffic streams. Unfortunately, it is not completely clear how to calculate the delay and the backlog for a stream. This is a main disadvantage of this method.

Secondly, the method is not proved to be compositional. No information has been found on how to combine network elements.

Finally, it is not clear how to make use of the length of the packets. This is not explained in [25] and no more information has been found in the literature on how to express the length of the packets in this method.

Summarizing, the (ρ_{max}, ρ_{min}) method does not meet the selection criteria of section 2.2. Also, no applications to practical examples have been found in the literature.

2.6 Conclusion

As has become clear in the previous sections, the methods of Cruz and Stiliadis are rather similar. Until now, both methods do not have insuperable disadvantages. This, in contrast to (ρ_{max}, ρ_{min}).

For (ρ_{max}, ρ_{min}), no information has been found on how to determine the delay and the backlog. There is not sufficient evidence to classify (ρ_{max}, ρ_{min}) as a promising technique.

Looking to the properties of Cruz and Stiliadis, these methods are the best applicable for the SoC performance analysis method. Especially the compositionality and the ability to express the length of the packets and the burstiness are important.

Therefore, the method of Cruz and Stiliadis are selected to characterize traffic for the SoC performance analysis method. In the next chapter, the focus is on SoC design and the application of the methods of Cruz and Stiliadis for the SoC performance analysis method.

Network calculi for SoC design

Contents

Please note that all definitions and formulas in this chapter are from the literature.

3.1 Introduction

SoCs have specific communication behavior that needs to be captured in the models of the system components for the performance analysis method. This behavior is related to characteristics of the traffic, of the memory system and of the interconnect. After introducing these characteristics, we will investigate how well the methods of Cruz and of Stiliadis can be used to model SoC communication infrastructures and to analyze the performance.

3.1.1 Traffic characteristics

Communication in SoCs exhibits specific characteristics that need to be represented appropriately for analyzing the performance of SoCs.

First of all, the communication can use a request-only stream or a request-response stream (see figure 3.1). In the first case, the requests flow from one subsystem to the SoC communication infrastructure. No responses are sent back. In the case of a request-response stream, a subsystem sends requests to the SoC communication infrastructure and receives responses from the SoC communication infrastructure. This behavior can be used to model the traffic of a memory system. In the case data have to be loaded, first a load request, including address data, is sent to the memory system. After that, a load response is sent back to the subsystem together with the requested data.

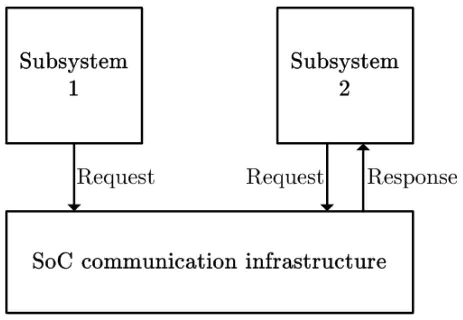


Figure 3.1: A request-only stream and a request-response stream

Secondly, for most subsystems the number of requests or packets in flight (i.e. the number of outstanding requests) is limited. This is called the “pipeline degree”. That means that if the pipeline degree has been reached, the next request can only be sent after having received a response. This is the case, because a response lowers the number the outstanding requests. So, a session is not only bounded by its σ and ρ characteristics, but also by its pipeline degree.

So, for SoCs some specific traffic characteristics influence strongly the modeling. In the next section, the characteristics of a memory system to be modeled are described.

3.1.2 Memory system characteristics

In the case a memory system is used, some specific characteristics are important for the analysis.

Firstly, the size of a load request in *words* is not equal to the size of a load response. The size of a load request is much smaller than the size of a response (see figure 3.2). This means that the packets change in size while they travel from a subsystem via a memory

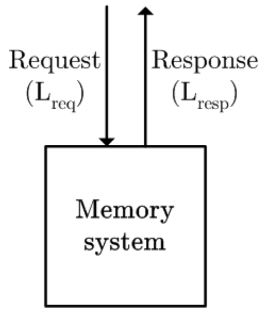


Figure 3.2: The size of a load request in *words* is much smaller than the size of a load response (i.e. $L_{req} << L_{resp}$)

system back to the subsystem, if one traffic stream is used to model the requests and the responses.

Another characteristic of a memory system is the refresh requirement. When a DRAM is used as memory system, the memory has to be refreshed frequently. Otherwise, data will be lost. Therefore, the memory subsystem needs to reserve a part of its time to refresh its content.

Finally, a memory system has a delay. It takes some time to process a request. But, this delay is not constant. It depends on the kind of the request. For example, a store request will need more time than a load request [16]. Since this difference is substantial, the analysis has to be able to model this behavior.

Not only the memory system, but also the interconnect has some specific characteristics that are relevant for the analysis.

3.1.3 Interconnect characteristics

The interconnect is modeled by several network elements, connected by wires. These network elements can be buffers, multiplexers, demultiplexers, regulators and so on. For these elements, two characteristics are important.

First of all, a multiplexer combines several input streams. The order of combining is determined by the arbitration policy of the multiplexer. These arbitration policies differ significantly in performance. Therefore, it should be possible to model this policy such that the influence of the arbitration policy of the schedulers (i.e. the multiplexers) can be taken into account.

Finally, to handle the complexity of chip design, it should be possible to combine the network elements easily. Therefore, compositionality is a requirement for the analysis.

In the next sections, the methods of Cruz and Stiliadis are explained in more detail. The above characteristics are used as selection criteria to select one method as foundation of the performance analysis method.

3.2 (σ, ρ)

First of all, an overview of the notation used by the method of Cruz [9] is given. After that, the method of Cruz is further explained and analyzed, using the characteristics described in the previous section.

3.2.1 Notation

The following notation is used:

Term	Definition
$R(t)$	The instantaneous rate in $\frac{words}{sec}$ for a stream flowing on the link at time t $R(t) \sim (\sigma, \rho)$ if and only if for all t_1 and t_2 $\int_{t_1}^{t_2} R(t)dt \leq \sigma + \rho \cdot (t_2 - t_1)$
σ	The burstiness constraint in *words* $\sigma \geq 0$ and $\sigma \in \mathbb{R}$
ρ	The rate in $\frac{words}{sec}$ at which the traffic is transmitted $\rho \geq 0$ and $\rho \in \mathbb{R}$
b	A non-decreasing function, with $b(t) = (\sigma + \rho \cdot t)$ $R(t) \sim b$ if and only if for all t_1 and t_2 $\int_{t_1}^{t_2} R(t)dt \leq b(t_2 - t_1)$
L_k	Length of packet k in *words* $L_k \in \mathbb{N}$
C	Maximum rate in $\frac{words}{sec}$ of a network element or a link $C \geq 0$ and $C \in \mathbb{R}$
$Q_\rho(R)(t)$	Represents the backlog in *words* of a network element at time t, which accepts the data at a rate of $R(t)$ and transmits the data at a rate of ρ
$D(t)$	Delay in *sec* of a network element at time t

Furthermore, the following notation is used:

Term	Definition
$I_{A(t)}$	Indicator function of the truth of statement A $I_{A(t)} = \begin{cases} 1 & \text{if } A(t) \\ 0 & \text{if } \neg A(t) \end{cases}$
D_i	Represents the maximum delay in *sec* for stream i
d_j	Represents the maximum delay in *sec* of packet j

3.2.2 Traffic characteristics

The traffic characteristics of section 3.1.1 are used to explain the method of Cruz in more detail. Unfortunately, Cruz does not describe how to model a request-response stream, nor how to express the pipeline degree of a stream.

3.2.3 Memory system characteristics

The memory system characteristics of section 3.1.2 are used to explain the method of Cruz. Unfortunately, Cruz does not describe how to model these characteristics.

3.2.4 Interconnect characteristics

In this section, the interconnect characteristics of section 3.1.3 are used to explain the method of Cruz in more detail. First, a number of relevant network elements are described. After that, we explain how to combine those network elements.

3.2.5 Network elements

In [9], Cruz describes the properties of several network elements.

FIFO

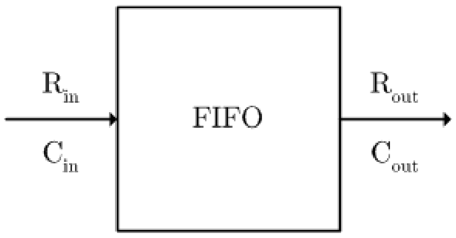


Figure 3.3: FIFO

The first relevant network element is the FIFO-buffer (see figure 3.3). Assume that $C_{in} \geq C_{out}$ and let the j^{th} packet, $j \geq 1$ and $j \in \mathbb{N}$, arrive at $s_j \geq 0$, $s_j \in \mathbb{R}$, and depart at t_j, such that $s_j < t_j$. Furthermore, assume that $R_{in}(t)$ and $R_{out}(t)$ represent the rate of the input respectively the rate of the output stream of the FIFO at time t. Then, according to [9], the rate of the output stream is

$$R_{out}(t) = C_{out} \cdot \sum_{k=1}^{\infty} I_{\{t_k \leq t < t_k + \frac{L_k}{C_{out}}\}} \tag{3.1}$$

This means that the rate of the output stream at time t is equal to C_{out} if at time t a packet is sent (i.e. there is a packet j such that $t_j \leq t < t_j + \frac{L_j}{C_{out}}$). So, at time t_j the output of the j^{th} packet starts. Then, the output rate is equal to its maximum output rate (i.e. C_{out}). $\frac{L_j}{C_{out}}$ *sec* later, the j^{th} packet has been transmitted.

The size of the backlog in the FIFO at time t is denoted by [9]

$$Q_{C_{out}}(R_{in})(t) = \max_{t_1 : 0 \leq t_1 \leq t} \left\{ \int_{t_1}^{t} R_{in}(t)dt - C_{out} \cdot (t - t_1) \right\} \tag{3.2}$$

Finally, the delay of the j^{th} packet (i.e. d_j) in the FIFO is determined in [9] as

$$d_j = \frac{1}{C_{out}} \cdot \Big(Q_{C_{out}}(R_{in})(s_j) \Big) \tag{3.3}$$

So, this means that the delay of the FIFO is modeled by the size of its backlog.

Multiplexer

The second relevant network element in the method of Cruz is the multiplexer (see figure 3.4). It is assumed that the multiplexer works according to LFCFS (Locally First-Come, First-Served). This means that for a given input stream, the packets originating from the input stream are transmitted to the output link in the same order in which they arrive.

Assume that the first input link has a maximum rate of C_1, the second one of C_2 and the output link has a maximum rate of C_{out}, such that $C_1 = C_2 = C_{out} = C$. Furthermore, assume that $R_1 \sim b_1$ characterizes the input traffic of the first link and $R_2 \sim b_2$ of the second link. R_{out} characterizes the output traffic.

Then, the output is determined in [9] by

$$R_{out} \sim b_{out} \tag{3.4}$$
$$b_{out}(t) = \min \Big\{ C_{out} \cdot t, b_1(t) + b_2(t) \Big\} \tag{3.5}$$

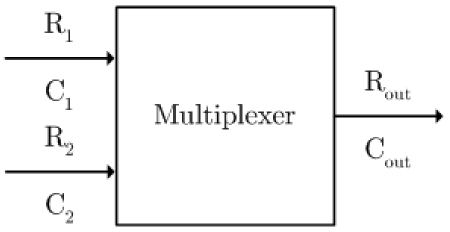


Figure 3.4: Multiplexer

Assume that $Q_i(t)$ represents the size of the backlog as a result of input stream i in the multiplexer at time t. Then, the total backlog of the multiplexer at t is according to [9]

$$Q(t) = Q_1(t) + Q_2(t) \tag{3.6}$$

Unfortunately, no information has been found on how to determine $Q_1(t)$ and $Q_2(t)$ for a locally FCFS arbitration policy. Only for a general, not locally FCFS type of arbitration policy, $Q_i(t)$ is determined in [9]

Finally, the delay of a multiplexer is determined in [9]. Assume that D_1 is the maximum delay experienced in the multiplexer by the data from input stream 1. Then

$$D_1 \leq \frac{\sigma_2}{C - \rho_2} + \frac{\sigma_1}{C - \rho_1} \cdot \frac{\rho_2}{C - \rho_2} \tag{3.7}$$

The proof of it can be found in the appendix of [9]

Arbitration policy of multiplexer

The order of combining several streams is determined by the arbitration policy of the multiplexer. Therefore, it should be possible to model this arbitration policy.

Unfortunately, no information has been found in the literature on how to express a specific arbitration policy in the method of Cruz. He only modeled classes of arbitration policies, like locally FCFS.

Demultiplexer

The next relevant network element is the demultiplexer (see figure 3.5). Assume that data is “marked” so that the demultiplexer can instantaneously determine via which output link the input stream has to be transmitted.

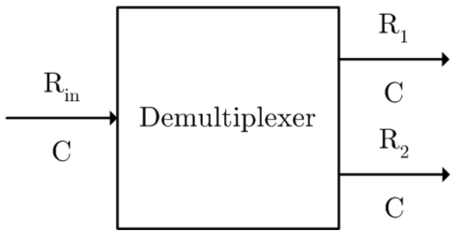


Figure 3.5: Demultiplexer

For the demultiplexer, only the output is characterized in [9]. Assume that R_{in} represents the input traffic and R_i the output streams. Furthermore, the input link and the output links have a maximum rate of C. Then,

$$R_{in} = R_1 + R_2 \tag{3.8}$$

The propagation delay and the backlog of a demultiplexer are zero, according to [9]

Regulator

The last relevant network element is the (σ, ρ) regulator (see figure 3.6). The working of a regulator has already been described in section 2.4.1. Assume that the input link and the output link have equal capacities (i.e. C). Furthermore, assume that the input (i.e. R_{in}) is an arbitrary stream. Then, according to [9] and [17], the output (i.e. R_{out}) is characterized as

$$R_{out} \sim (\sigma, \rho) \tag{3.9}$$

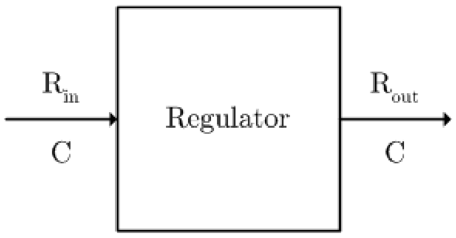


Figure 3.6: (σ, ρ) regulator

The size of the backlog of a regulator is derived in [9] as

$$Q_\rho(R_{in})(t) - \sigma = \max_{t_1 : 0 \le t_1 \le t} \left\{ \int_{t_1}^{t} R_{in}(t)dt - \rho \cdot (t - t_1) \right\} - \sigma \tag{3.10}$$

Finally, the delay of the j^{th} packet (i.e. d_j) in the regulator, that arrives at s_j, is according to [9]

$$d_j = \frac{1}{\rho} \cdot \max \Big\{ 0, \Big(Q_\rho(R_{in})(s_j) - \sigma \Big) \Big\} \tag{3.11}$$

So, this means that the delay of a regulator is determined by the size of its backlog.

As has become clear above, the properties of several relevant network elements have already been determined. These network elements are used to combine into an interconnect. This is explained in the next section.

3.2.6 Combining elements

To illustrate how to combine the network elements into an interconnect, the next example is given. Assume the next situation (see figure 3.7) [10].

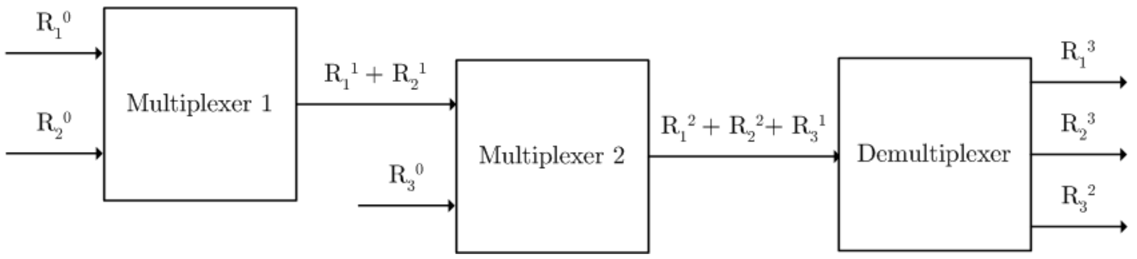


Figure 3.7: Example of an interconnect

Assume there are three input streams (i.e. R_1^0, R_2^0 and R_3^0). Traffic from stream 1 and 2 are input to multiplexer 1 (i.e. R_1^0 and R_2^0), and pass through both multiplexers before being demultiplexed and exiting the network (i.e. R_1^3 and R_2^3). Traffic from stream 3 (i.e. R_3^0) enters the network through multiplexer 2 and exits the network (i.e. R_3^2) after being demultiplexed. It is assumed that for $k = 1, 2, 3$, $R_k^0 \sim (\sigma_k, \rho_k)$. Furthermore, all links have equal capacities (i.e. C).

Then, according to [10] (also see equation 3.4)

$$R_1^1 + R_2^1 \sim \Big(\sigma_1 + \sigma_2, \rho_1 + \rho_2\Big) \tag{3.12}$$

Furthermore, let D_1 be the upper bound of the delay for stream 1 in multiplexer 1. Then according to equation 3.7

$$D_1 \leq \frac{\sigma_2}{C - \rho_2} + \frac{\sigma_1}{C - \rho_1} \cdot \frac{\rho_2}{C - \rho_2} \tag{3.13}$$

Analog, let D_2 be the upper bound of the delay for stream 1 in multiplexer 2. Then according to equation 3.7

$$D_2 \leq \frac{\sigma_3}{C - \rho_3} + \frac{\sigma_1 + \sigma_2}{C - \rho_1 - \rho_2} \cdot \frac{\rho_3}{C - \rho_3} \tag{3.14}$$

It is assumed that the demultiplexer has no propagation delay.

Then, the delay for stream 1 is upper bounded by $D_1 + D_2$. The delays for the other two streams can be calculated in a similar way.

Unfortunately, we have not found information in [10] on how to calculate the backlog of the interconnect of figure 3.7

3.2.7 Applicability of the method of Cruz

The method of Cruz, as partly described in the previous sections, is evaluated using the characteristics of section 3.1.

As has become clear in the previous sections, no information has been found on how to model a request-response stream or how to express the pipeline degree of a session.

Furthermore, no information has been found in the literature on how to express the characteristics of a memory system.

Finally, no information has been found on how to express a specific arbitration policy. It is possible to combine the network elements to model an interconnect. For a specific example, this is done in the previous section. Unfortunately, only the delays are determined in [10]. No information has been found in [10] on how to determine the backlog.

Concluding, the method of Cruz can be used for the foundation of the performance analysis method, but the method has to be adapted significantly to meet all requirements.

In the next section, the method of Stiliadis is explained in more detail.

3.3 Latency-Rate Servers

First of all, an overview of the notation used by the method of Stiliadis [22] is given. After that, the method of Stiliadis is further explained and analyzed, using the characteristics of section 3.1.

3.3.1 Notation

The following notation is used (as in section 2.4):

Term	Definition
σ_i	The burstiness constraint in *words* of session i $\sigma_i \geq 0$ and $\sigma_i \in \mathbb{R}$
ρ_i	The rate in $\frac{words}{sec}$ of session i at which the traffic is transmitted $\rho_i \geq 0$ and $\rho_i \in \mathbb{R}$
L_i	Length of packet in *words* of session i $L_i \in \mathbb{N}$
C	Maximum rate (i.e. capacity) in $\frac{words}{sec}$ of all links and *LR* servers $C \geq 0$ and $C \in \mathbb{R}$
Θ_i^S	Latency in *sec* of session i of server S
$A_i(t_1, t)$	The number of arrived data in *words* of session i in (t_1, t)
$W_i(t_1, t)$	The number of service in *words* received by session i in (t_1, t)
D_i^S	Delay in *sec* of any packet of session i in server S
$Q_i^S(t)$	Backlog in *words* of session i in server S at time t
V	The number of sessions that share an *LR* server $V \geq 1$ and $V \in \mathbb{N}$

Furthermore, the following notation is introduced:

Term	Definition
L_{max}	The maximum length in *words* of the packets an *LR* server receives

3.3.2 Traffic characteristics

The traffic characteristics of section 3.1.1 are used to explain the method of Stiliadis in more detail. Unfortunately, in the method of Stiliadis it is not clear how to model a request-response stream, nor how to express the pipeline degree of a stream.

3.3.3 Memory system characteristics

Just as before, the memory system characteristics of section 3.1.2 are used to explain the method of Stiliadis in more detail. Again, no information has been found on how Stiliadis describes these characteristics.

3.3.4 Interconnect characteristics

In this section, the interconnect characteristics of section 3.1.3 are used to explain the method of Stiliadis in more detail.

3.3.5 Network elements

First, the properties of an *LR* server are mentioned. After that, several arbitration policies are explained.

LR servers

In [22], Stiliadis uses LR servers as network elements. An LR server consists of a multiplexer that operates using a specific arbitration policy (see figure 3.8). Each input session has a queue on its input port of the multiplexer. Multiple LR servers can be connected in a chain.

As shown in the example of figure 3.8, three producers generate data, that is sent to the multiplexer. Each producer has a queue (i.e. "Q") on the input of the multiplexer. The output of the multiplexer is demultiplexed, before it is sent to one of the four consumers. Each consumer has a queue on its input to store the input packets temporarily in case the consumer cannot process the packets immediately.

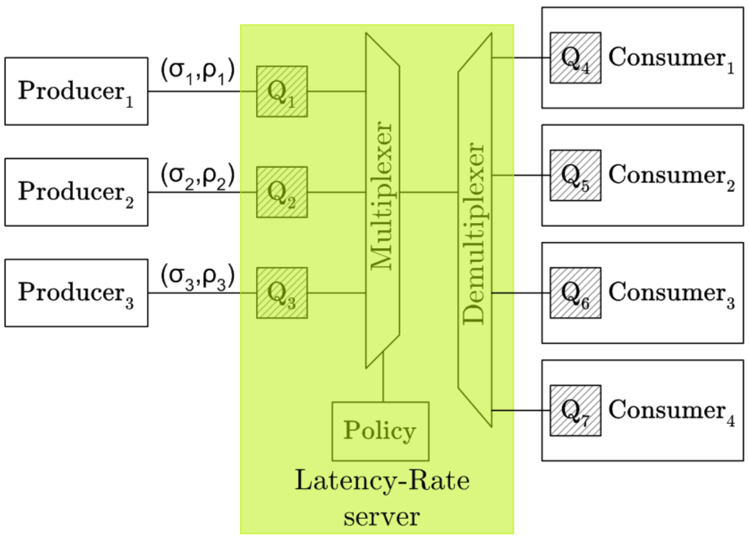


Figure 3.8: A system consisting of a Latency-Rate server, 3 producers and 4 consumers [22] ("Q" denotes a queue)

Each input session i of the LR server is represented by a ρ_i and a σ_i. Please note that the sum of the input ρ_i's cannot exceed the capacity of the LR server (i.e. C). So,

$$\sum_{i=1}^{V} \rho_i \leq C \tag{3.15}$$

The order of forwarding packets of several sessions is determined by the arbitration policy of the multiplexer. In [22] for several arbitration policies the latency has been determined. This latency is required to determine the upper bounds for the backlog and the delay. Therefore, we are looking to the worst case scenario. Please note that in the worst case, all input sessions of the LR server are transferring the maximum amount of data that is allowed by their σ and ρ. This worst case scenario is used to determine the latency of a session. Please note that Stiliadis defined the latency of a session as the maximum amount of time between the moment the first packet of a busy period of that session has arrived completely in the scheduler and the moment that packet has left the scheduler completely.

Arbitration policy of a multiplexer

For several arbitration policies the latency has been determined in [22]. These are explained in this section.

Round Robin

Round Robin is the basis of several arbitration policies explained below. Round Robin consists of rounds. In each round, a session gets an amount of service (it is called a "slot"). The total amount of service of all sessions together in one round is called a "frame". The following definitions are used (see figure 3.9).

Definition *F is the total number of service in words assigned in a round*

Definition *ϕ_i is the number of service in words assigned to session i in a round*

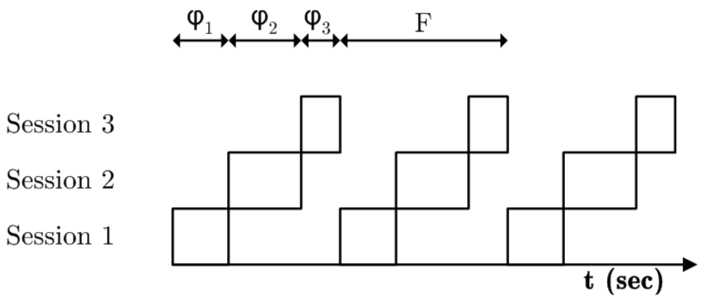


Figure 3.9: Graphical explanation of F and ϕ_i. Three sessions receive service. The size of the blocks represents the amount of service assigned to a particular session

Please note that $F = \sum_{i=1}^{V} \phi_i$.

First-Come, First-Served

First-Come, First-Served (FCFS) is a simple scheduling algorithm. Each packet that arrives at the LR server gets a time-stamp assigned. This time-stamp is the arrival time of the packet. The packets are serviced according to their arrival times. The problem of this arbitration policy is that it does not offer any isolation. No finite deterministic bounds in terms of delay or backlog, independent of the network state and the traffic characterization of the sessions can be determined [22]. Therefore, no latency can be determined.

Virtual Clock

The Virtual Clock arbitration policy [26] was inspired by Time Division Multiplexing. This arbitration policy assigns to each packet a "virtual" transmission time, based on the measured arrival rate of the previous packets of the session and the average arrival rate of that session. Let AT be the real time in *sec* a packet arrives. Then, each packet gets a time stamp (i.e. TS_i^k is the time stamp in *sec* of the k^{th} packet, $k \geq 1$ and $k \in \mathbb{N}$, of session i).

To be more specific, the time stamp of the k^{th} packet of session i is calculated using [21]

$$TS_i^k := \max\left\{AT_i^k, TS_i^{k-1}\right\} + \frac{L_i^k}{\rho_i} \tag{3.16}$$

Then, the packets are ordered according to their time stamp values. If the packet arrives later than expected (i.e. $AT > TS_i^{k-1}$), it leaves after a maximum of $\frac{L_i^k}{\rho_i}$ *sec*. If it arrives earlier (i.e. $AT < TS_i^{k-1}$), it leaves at time $TS_i^{k-1} + \frac{L_i^k}{\rho_i}$ in the worst case.

The latency of this arbitration policy is according to [22]

$$\Theta_i^{VC} = \frac{L_{max}}{C} + \frac{L_i}{\rho_i} \tag{3.17}$$

Please note that for session i a bandwidth of ρ_i is reserved by the scheduler.

Deficit Round Robin

In the scheduling algorithm Deficit Round Robin (DRR), each session has its own "deficit counter" [20]. Initially, those values are 0. When a packet arrives, it is added to the queue of its session. The nonempty queues are served in a Round Robin fashion. Each time a particular queue is being served, its deficit counter increases by some given value called the "quantum". Then, Deficit Round Robin serves a packet at the head of a nonempty queue if the deficit counter is not smaller than the size of the packet. The deficit counter decreases by the size of the packet that is served.

If the size of the packet is larger than the deficit counter, the packet is served the next round. If the queue is empty, the deficit counter is set back to 0.

In order to determine F and ϕ_i, the following formulas are used [14], [22]

$$\rho_{min} = \min_{i:1\leq i\leq V} \{\rho_i\} \tag{3.18}$$

$$w_i = \frac{\rho_i}{\rho_{min}} \tag{3.19}$$

$$\phi_i = w_i \cdot L_{max} \tag{3.20}$$

$$F = \sum_{i=1}^{V} \phi_i \tag{3.21}$$

Session i gets each round a quantum of ϕ_i *words*. ϕ_i is determined by the ratio of ρ_i and the minimum ρ of the sessions the LR server receives and by L_{max}. The sum of the ϕ's is the size of a single frame.

Please note that the size of the frames and the slots are not equal each round.

Then, as determined in [22], the latency is

$$\Theta_i^{DRR} = \frac{3 \cdot F - 2 \cdot \phi_i}{C} \tag{3.22}$$

Please note that for session i a bandwidth of $\frac{\phi_i}{F} \cdot C$ is reserved by the scheduler.

Weighted Round Robin

Weighted Round Robin (WRR) is a special case of Round Robin. In each round, session i is allowed to sent ϕ_i *words*. ϕ_i is always an integer multiple (i.e. w_i) of the size of a cell (i.e. L_c) [21]. In Weighted Round Robin a cell is a packet of fixed size. Please note that in contrast to Deficit Round Robin, $w_i \in \mathbb{N}$.

Therefore, the following formulas are used [22]

$$\phi_i = w_i \cdot L_c \tag{3.23}$$

$$F = \sum_{i=1}^{V} \phi_i \tag{3.24}$$

This means that in each round, session i is allowed to send ϕ_i *words*. The sessions are serviced in a Round Robin fashion.

Then, the latency is according to [22]

$$\Theta_i^{WRR} = \frac{F - \phi_i + L_c}{C} \tag{3.25}$$

Please note that for session i a bandwidth of $\frac{\phi_i}{F} \cdot C$ is reserved by the scheduler.

3.3.6 Combining *LR* elements

Combining the *LR* servers is done by making use of the formulas of section 2.4.

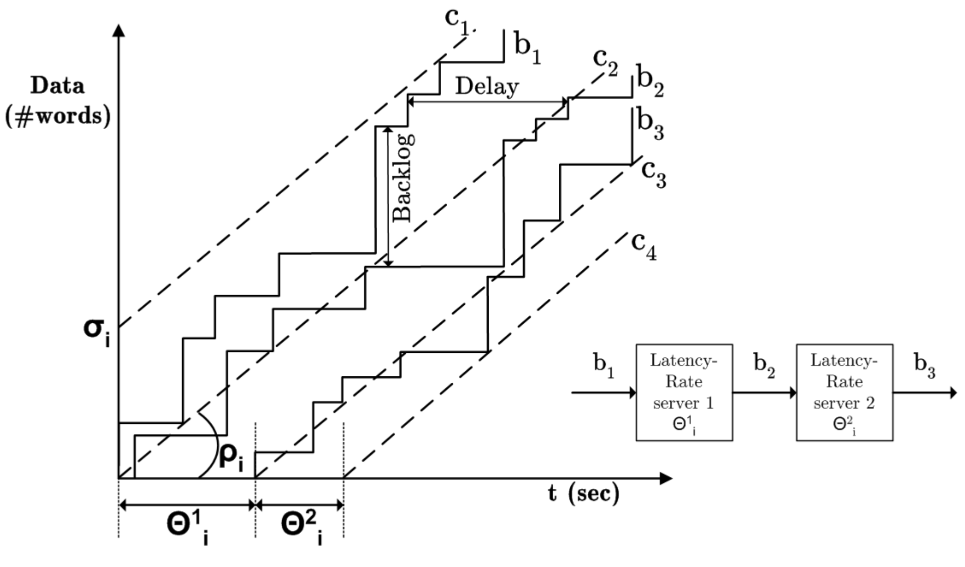


Figure 3.10: Traffic modeling of a combination of *LR* servers

Assume that b_1 is the input traffic of session i of *LR* server 1 and that this traffic is token-bucket-shaped. This means that it can be characterized by (σ_i, ρ_i), such that

$$A_i(t_1, t) \leq \sigma_i + \rho_i \cdot (t - t_1) \tag{3.26}$$

for $0 \leq t_1 \leq t$. Furthermore, assume that b_2 is the output of *LR* server 1 and the input of *LR* server 2. Finally, b_3 is the output of *LR* server 2 (see figure 3.10). The j^{th} *LR* server, $j \geq 1$ and $j \in \mathbb{N}$, has a latency of Θ_i^j that depends on the arbitration policy of the *LR* server and characteristics of the input session (see [22]). Please note that Stiliadis assumed in [22] that packet departures are considered as impulses (i.e. events).

c_1 is the upper bound of the input session b_1 (see equation 3.26). Therefore, c_1 is also the upper bound for b_2 and b_3. Furthermore, b_1 is lower bounded by c_2 because a busy period is assumed (see section 2.4). *LR* server 1 has a latency of Θ_i^1, so by definition of an *LR* server, b_2 is lower bounded by c_3. For the same reason, b_3 is lower bounded by c_4.

Then, the delay can be seen as the difference in time between the output and the input (see figure 3.10). In case of *LR* server 1, this is the horizontal difference between b_1 and b_2. The maximum delay of *LR* server 1 is the horizontal difference between the upper bound of b_1 (i.e. c_1) and the lower bound of b_2 (i.e. c_3). So, the maximum delay for session i in *LR* server 1 (i.e. D_i^1) is upper bounded by

$$\begin{aligned} D_i^1 &\leq \textit{horizontal distance between } c_1 \textit{ and } c_3 & (3.27)\\ &= \textit{horizontal distance between } c_1 \textit{ and } c_2 \\ &\quad + \textit{horizontal distance between } c_2 \textit{ and } c_3 & (3.28)\\ &= \frac{\sigma_i}{\rho_i} + \Theta_i^1 & (3.29)\end{aligned}$$

Adding another *LR* server increases the maximum delay with the latency of the extra *LR* server.

The backlog can be seen as the vertical distance between the input and the output (see figure 3.10). The maximum backlog for session i of *LR* server 1 is the vertical difference between the upper bound of b_1 (i.e. c_1) and the lower bound of b_2 (i.e. c_3). So, the maximum backlog of session i in *LR* server 1 (i.e. Q_i^1) is upper bounded by

$$\begin{aligned} Q_i^1 &\leq \textit{vertical distance between } c_1 \textit{ and } c_3 & (3.30)\\ &= \textit{vertical distance between } c_1 \textit{ and } c_2 \\ &\quad + \textit{vertical distance between } c_2 \textit{ and } c_3 & (3.31)\\ &= \sigma_i + \rho_i \cdot \Theta_i^1 & (3.32)\\ &= \rho_i \cdot D_i^1 & (3.33)\end{aligned}$$

Adding another *LR* server will increase the required amount of backlog as follows. For the extra *LR* server with latency Θ_i^2 the backlog is upper bounded by $\sigma_i + \rho_i \cdot \Theta_i^1$ plus $\rho_i \cdot \Theta_i^2$.

Finally, adding an *LR* server increases the burstiness constraint. Whereas b_1 is bounded by c_1 and c_2, b_2 is bounded by c_1 and c_3. So, the burstiness constraint of b_2 is bounded by the maximum backlog of *LR* server 1. The output of session i of *LR* server 1 can be characterized as

$$(\sigma_i + \Theta_i^1 \cdot \rho_i, \rho_i). \tag{3.34}$$

Adding another *LR* server will increase the burstiness constraint with the product of the latency of the extra *LR* server and ρ_i.

In general, for a chain of m *LR* servers, $m \geq 1$ and $m \in \mathbb{N}$, with a total latency of $\sum_{j=1}^{m} \Theta_i^{(S_j)}$, the following formulas are derived [22]

- Delay for session i in the chain of *LR* servers

$$D_i \leq \frac{\sigma_i}{\rho_i} + \sum_{j=1}^{m} \Theta_i^{(S_j)} \tag{3.35}$$

- Backlog for session i in the k^{th} *LR* server, $1 \leq k \leq m$ and $k \in \mathbb{N}$, in the chain of *LR* servers

$$Q_i^{(S_k)} \leq \sigma_i + \rho_i \cdot \sum_{j=1}^{k} \Theta_i^{(S_j)} \tag{3.36}$$

- Output of session i after the k^{th} *LR* server, $1 \leq k \leq m$ and $k \in \mathbb{N}$, in the chain of *LR* servers

$$(\sigma_i + \rho_i \cdot \sum_{j=1}^{k} \Theta_i^{(S_j)}, \rho_i) \tag{3.37}$$

In [22], Stiliadis claimed that the end-to-end delay bounds, compared to the method of Cruz, are tighter. This is claimed to be the case, because the arbitration policy of the schedulers is used to determine the delay, instead of only the property of $LFCFS$ as Cruz did. Unfortunately, no proof has been found for this claim.

As has become clear above, in the method of Stiliadis it is easy to combine several LR servers. Now, we describe some improvements and extensions by Stiliadis of the above formulas.

Improved delay bounds

Latencies of LR servers are determined by the assumption that a packet has been serviced when its last word has left the server. Because of this assumption (i.e. that a packet leaves as an impulse) it is allowed to model the arrival of the packet in the next server as an impulse as well. To compute the end-to-end delay of a session, only the time at which the last word of a packet leaves the last server is interesting. So, an improved delay bound can be determined. For a chain of m LR servers, $m \geq 1$ and $m \in \mathbb{N}$, with a total latency of $\sum_{j=1}^{m} \Theta_i^{(S_j)}$, the offered service is bounded in the first $m-1$ servers using the normal method. For the last server, the delay is calculated based on the moment a packet completes service.

Then, according to [22], the total delay of session i is

$$D_i \leq \frac{\sigma_i}{\rho_i} + \sum_{j=1}^{m} \Theta_i^{(S_j)} - \frac{L_i}{\rho_i}. \tag{3.38}$$

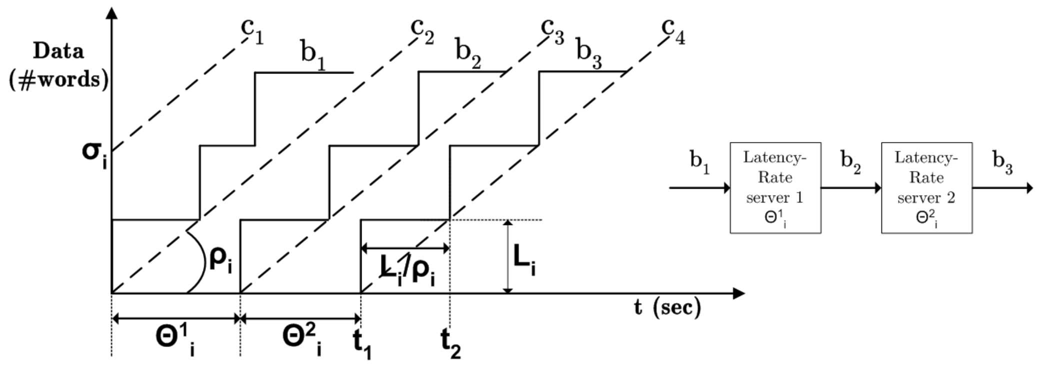


Figure 3.11: Improved delay bounds

This is indicated in figure 3.11. b_1 represents the input traffic of the first LR server, b_2 the output traffic of the first LR server and the input traffic of the second LR server. b_3 represents the output of the second LR server. c_4 indicates the lower bound of the guaranteed service of the second LR server. That means that the input, that is received between c_1 and c_2, has to be serviced (i.e. sent) before c_4. The last moment that the first packet of figure 3.11 can be serviced by the second LR server is at t_1 and not at t_2. This is because c_4 is the lower bound of the service of the LR server. Therefore, the delay bound can be reduced by $\frac{L_i}{\rho_i}$.

Propagation Delay

Until now, it is assumed that the propagation delay in the network is zero. If this is not the case, then according to [22], the total delay of session i is

$$D_i \leq \frac{\sigma_i}{\rho_i} + \sum_{j=1}^{m} \Theta_i^{(S_j)} - \frac{L_i}{\rho_i} + \sum_{j=1}^{m-1} p_i^j, \tag{3.39}$$

where p_i^j is the propagation delay between LR server j and $j+1$.

3.3.7 Applicability of the method of Stiliadis

The method of Stiliadis, as partly described in the previous sections, is evaluated using the characteristics of section 3.1.

As has become clear in the precious sections, no information has been found on how to model a request-response stream or how to express the pipeline degree of a session.

Furthermore, no information has been found in the literature on how to express the characteristics of a memory system.

Finally, several arbitration policies can be expressed in the method of Stiliadis. Using these, it is possible to combine the LR servers into an interconnect. Formulas to determine the backlog, as well as the delay and the output of the LR servers are available.

So, the method of Stiliadis can be used for the foundation of the required performance analysis method, but the method has to be adapted significantly to meets all requirements.

3.4 Conclusion

As has become clear in the previous sections, none of the methods can be used for the performance analysis method without adaptation.

Looking to the traffic characteristics, both methods do not have the ability to model a request-response stream. Furthermore, in both methods it is not clear how to express the pipeline degree of a session.

In both methods, no information is available on how to model the change of packet sizes, a memory refresh or the processing time of a request. This should be done by applying a method in a smart way or by making some changes.

Looking to combining the network elements, for both methods general formulas are available. In both methods combining elements is possible, but the method of Stiliadis is of a higher aggregation level. Stiliadis looks to the combination of a multiplexer with an arbitration policy, queues and a demultiplexer, whereas Cruz looks to individual, fine-grain network elements.

Furthermore, in the method of Stiliadis, the properties of several policies of scheduling have already been determined. By that, it is possible to model the influence of different policies of a scheduler. This is not the case in the method of Cruz.

For both methods, formulas are available how to combine the elements into an interconnect. Because Stiliadis looks at a higher aggregation level, he claims in [22] that his bounds are tighter. Because of the utilization of the interconnect, this can be very important. Unfortunately, we have not found a proof of that claim.

Finally, the method of Stiliadis gives by definition of an LR server bandwidth guarantees for the sessions. This is important for the compositionality of the analysis.

So, we have decided to use the method of Stiliadis as foundation for the required performance analysis method. Compositionality is very important for the analysis. Future interconnects of SoCs will become more and more complex and will consist of many different elements. By using Stiliadis, it is easy to combine several (different) schedulers.

Unfortunately, still some characteristics of a SoC have not been fulfilled, like

- The modeling of a request-response stream
- The use of a pipeline degree
- The modeling of memory system characteristics (the change of packet sizes, a memory refresh and the processing time of a request)

In the next chapter, some changes and extensions are made to the method of Stiliadis to fulfill the above requirements. Furthermore, several ways to improve the delay bounds and the maximum queue sizes are explained.

Network analysis for SoC design

Contents

4.1 Introduction

For the performance analysis method of SoC architectures, the method of Stiliadis has been selected as a foundation. However, this method has to be adapted and extended to meet all requirements described in chapter 3. First, some general characteristics are introduced into the method of Stiliadis. After that, the traffic characteristics, the memory system characteristics and the interconnect characteristics described in the previous chapter are introduced into the performance analysis method. Then, a number of improvements are introduced to make the performance bounds tighter. Finally, the applicability of the performance analysis method is explained (see section 3.3).

In appendix A the validation of the adaptations and extensions described in this chapter, is given. The notation and definitions of Stiliadis are used for the performance analysis method.

Furthermore, the following notation is used:

Notation	Explanation
x	The number of *words* that have to be transmitted $x \geq 1$ and $x \in \mathbb{N}$
m	The length of a chain of *LR* servers $m \geq 1$ and $m \in \mathbb{N}$
k	One of the *LR* servers of the chain of m *LR* servers $1 \leq k \leq m$ and $k \in \mathbb{N}$
m'	The length of another chain of *LR* servers $m' \geq 1$ and $m' \in \mathbb{N}$
k'	One of the *LR* servers of the chain of m' *LR* servers $1 \leq k' \leq m'$ and $k' \in \mathbb{N}$
D_{proc}	The processing delay of a request of an *LR* server $D_{proc} \geq 0$ and $D_{proc} \in \mathbb{R}$
n	The pipeline degree $n \geq 1$ and $n \in \mathbb{N}$
V	The number of sessions (producers), that share a scheduler $V \geq 1$ and $V \in \mathbb{N}$

4.2 General characteristics

To introduce the traffic characteristics, the memory system characteristics and the interconnect characteristics into our performance analysis method, some general characteristics have to be introduced. First, the calculation of the delay bound is changed. After that, the impulse assumption of Stiliadis is dropped.

4.2.1 Total delay

In the method of Stiliadis, the delay is determined for a packet entering a chain of *LR* servers. This is equivalent to the delay for σ *words*. In practice, the delay has to be calculated for more than σ *words*.

Assume that x *words* have to be transmitted. The data is sent through a chain of m *LR* servers with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$. Assume that the input session (produced by subsystem 1) of the chain is represented by b_1 and that the output session (consumed by

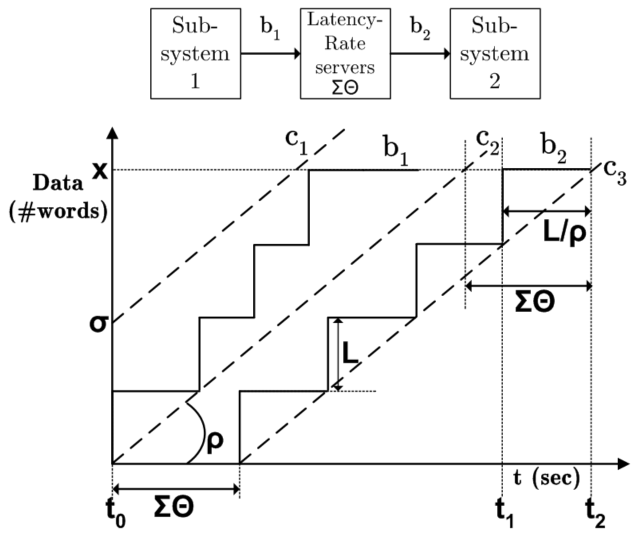


Figure 4.1: Delay for *x words*

subsystem 2) is represented by b_2 (see figure 4.1). Please note that b_1 is upper bounded by c_1 and lower bounded by c_2, and b_2 is lower bounded by c_3.

The *x words* are transmitted by making use of packets of size L. The delay of the LR servers can be calculated by

$$\begin{aligned} D &\leq t_1 - t_0 && (4.1)\\ &= t_2 - \frac{L}{\rho} && (4.2)\\ &= \frac{x}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} && (4.3)\\ &\leq \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} && (4.4) \end{aligned}$$

This is proved in lemma A.1.1 of appendix A.

The formula is similar to formula 3.38, but now $\lceil \frac{x}{L} \rceil \cdot \frac{L}{\rho}$ is used instead of $\frac{\sigma}{\rho}$, to determine the delay of multiple *words*. Please note that $\lceil \frac{x}{L} \rceil \cdot \frac{L}{\rho}$ is used for the case that *x words* is not an integer multiple of the size of the packets. Furthermore, the last packet cannot be processed after t_1. Otherwise, b_2 would cross c_3. By definition of an LR server, this is not allowed.

Also note that the backlog can still be calculated by formula 3.36.

As has become clear, to send *x words* the delay is no longer determined by σ.

4.2.2 The impulse assumption

In the method of Stiliadis, it is assumed that packet arrivals and departures are considered as impulses. His analysis starts when a packet has completely arrived in the LR server. Because the delay is required for sending packets from subsystem 1 via a chain of LR servers to subsystem 2, the arriving time of the first packet in the first LR server is part of the total delay.

Furthermore, by dropping the impulse assumption, the influence of the capacity can be shown in the graphs. By selecting a smaller capacity, the delay will increase and this should be visible in the graphs.

Therefore, the impulse assumption is dropped. The consequences are analyzed and taken care of. The result is explained below.

As explained in the previous section, the delay for *x words* is (see figure 4.1 and equation 4.4)

$$D \leq \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} \tag{4.5}$$

Stiliadis assumed in [22], in contrast to Cruz in [9], that a packet is not considered as departed the scheduler until its last bit has left the scheduler. He considered the packet arrivals and departures as impulses. To be more general, for our performance analysis method it is assumed that this is not the case. That means that it takes some time to send or receive a packet.

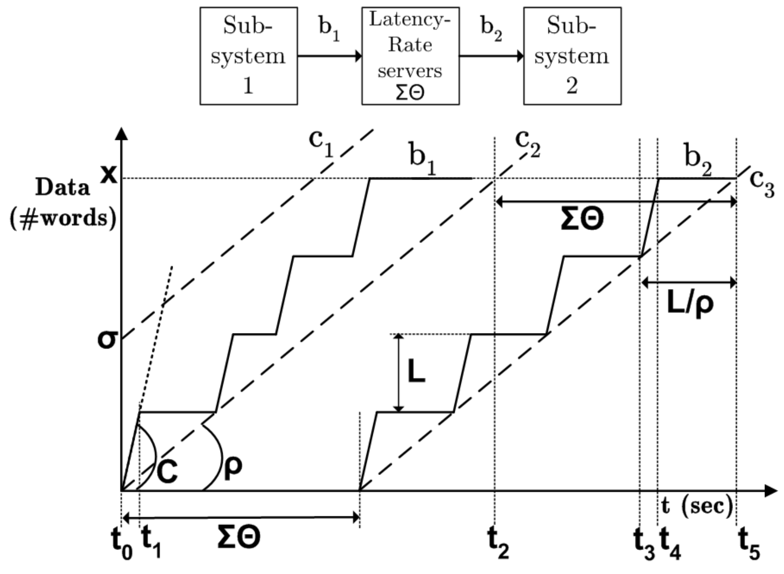


Figure 4.2: Delay for *x words*, without the impulse assumption

Then, the situation of figure 4.2 is applicable. Subsystem 1 sends packets (represented by b_1) via a chain of m *LR* servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$, to subsystem 2. b_2 represents the output traffic of the chain. Please note that $t_1 - t_0 = t_1 = \frac{L}{C}$.

Furthermore, a distinction has been made between the first bit and the last bit of a packet. This means that the arriving and departing time of a packet is included in the analysis. The starting point in the analysis of Stiliadis is the moment the first packet has arrived completely. In our performance analysis method, the arriving time of the first packet is included.

The latency of a session has not been changed, due to this adaptation. It is still the maximum amount of time between the moment the first packet of that session has arrived completely in the scheduler and the moment that packet has left the scheduler completely. Then, the arriving time of a packet has to be taken into account only once for the total delay.

Then, the total delay for the *x words* is

$$\begin{aligned}
D &\le t_4 - t_0 && (4.6)\\
&= (t_5 - t_0) - (t_5 - t_4) && (4.7)\\
&= (t_2 - t_0) + (t_5 - t_2) - (t_5 - t_4) && (4.8)\\
&= \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \left(\frac{L}{\rho} - \frac{L}{C} \right) && (4.9)\\
&= \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} && (4.10)
\end{aligned}$$

This is proved in lemma A.1.2 of appendix A.

Dropping the impulse assumption increases the delay for sending *x words* with $\frac{L}{C}$ (see equation 4.4).

Please note that the backlog of the k^{th} *LR* server in a chain of *LR* servers can still be calculated by

$$Q^k \le \sigma + \rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \qquad (4.11)$$

4.2.3 Determine σ and ρ

To use the method of Stiliadis, for each session σ and ρ have to be determined.

Assume that a session transmits periodically *x words* and that the length of the period is $\Delta t \ge 0$ *sec*. Furthermore, assume that the schedulers have a capacity of C. Finally, assume that *x words* is an integer multiple of the length of the packets. Then, the input can be as is shown in figure 4.3.

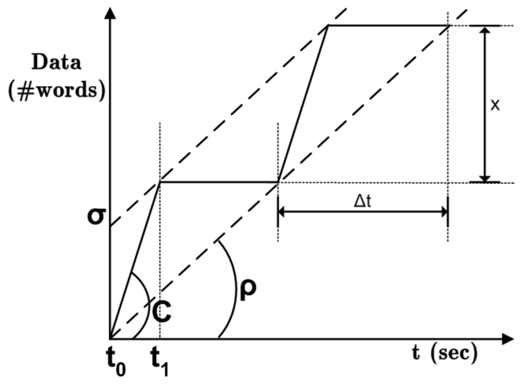


Figure 4.3: *x words* are sent in one burst

Please note that in figure 4.3, *x words* are sent in one burst. This will not always be the case. It is also possible that *x words* are sent in several smaller bursts (see figure 4.4).

Even a different σ and ρ can be used, for the same input traffic (see figure 4.5).

To determine ρ, $\frac{x}{\Delta t} \le \rho \le C$ is used. Because C is the capacity of the *LR* servers, it is not possible that $\rho > C$. Furthermore, if $\frac{x}{\Delta t} > \rho$, then σ goes to ∞ to be able to characterize the session. This is the case, because *x words* are sent every Δt *sec*.

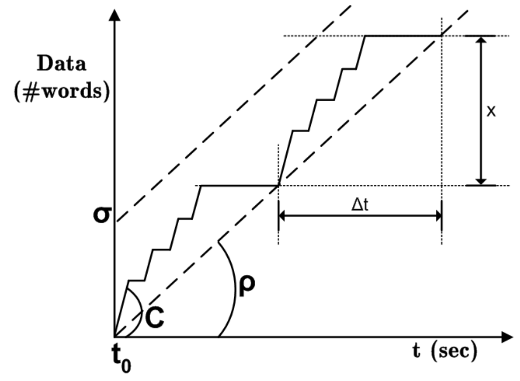


Figure 4.4: *x words* are sent in multiple small bursts

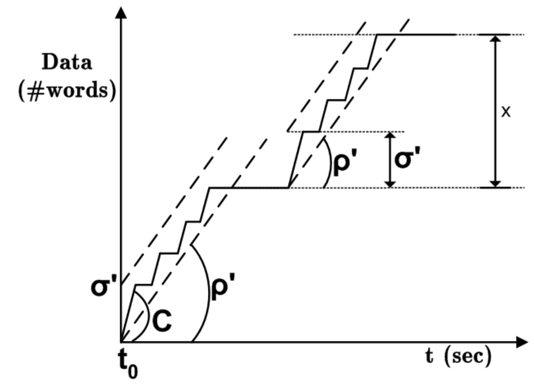


Figure 4.5: *x words* are sent in multiple small bursts, using a different σ and ρ

To determine σ, the next bounds can be used

$$L - \rho \cdot \frac{L}{C} \leq \sigma \leq x - \rho \cdot \frac{x}{C} \tag{4.12}$$

This is the case, because at t_0 (see figure 4.3), *x words* can be sent. If *x words* are sent directly in one burst, the last word is sent at time $\frac{x}{C}$ (i.e. at t_1). C is the maximum rate of the schedulers, so a higher rate is not possible. Then, σ is upper bounded (see figure 4.3)

$$\begin{aligned}
\sigma + \rho \cdot (t_1 - t_0) &\leq C \cdot (t_1 - t_0) && (4.13)\\
\sigma + \rho \cdot (t_1 - t_0) &\leq x && (4.14)\\
\sigma + \rho \cdot \frac{x}{C} &\leq x && (4.15)\\
\sigma &\leq x - \rho \cdot \frac{x}{C} && (4.16)
\end{aligned}$$

Cruz already proved in [9] that $L - \rho \cdot \frac{L}{C} \leq \sigma$ (see equation 2.7).

Please note that if $\rho \approx C$, $\sigma \approx 0$.

So, the value of σ is lower bounded by $L - \rho \cdot \frac{L}{C}$ and upper bounded by $x - \rho \cdot \frac{x}{C}$. This means that C impacts the minimum and the maximum value of σ.

4.3 Traffic characteristics

In the previous section, some general characteristics are introduced into our performance analysis method. In this section, the traffic characteristics of section 3.1 are introduced into our performance analysis method.

4.3.1 Request-response stream

Until now, all communication was restricted to request-only streams. This means that the requests flow from one subsystem (the producer), via the communication infrastructure to another subsystem (the consumer). In some cases, a subsystem sends requests via the communication infrastructure to a subsystem and receives responses from that subsystem via the communication infrastructure. An example of this are the load requests and the load responses of a memory system. Therefore, the total delay of a request-response stream has to be determined.

Assume the next situation (see figure 4.6). Subsystem 1 produces requests. These requests are sent via a chain of m LR servers, with a total latency of $\sum_{j=1}^{m}\Theta_{req}^{(S_j)}$, to subsystem 2. Subsystem 2 produces responses to the requests and sends these responses via a chain of m' LR servers, with a total latency of $\sum_{j=1}^{m'}\Theta_{resp}^{(S_j)}$, back to subsystem 1. Furthermore, assume that subsystem 2 has a delay of D_{proc} to produce a response, after a request is received. Finally, for each request exactly one response is generated.

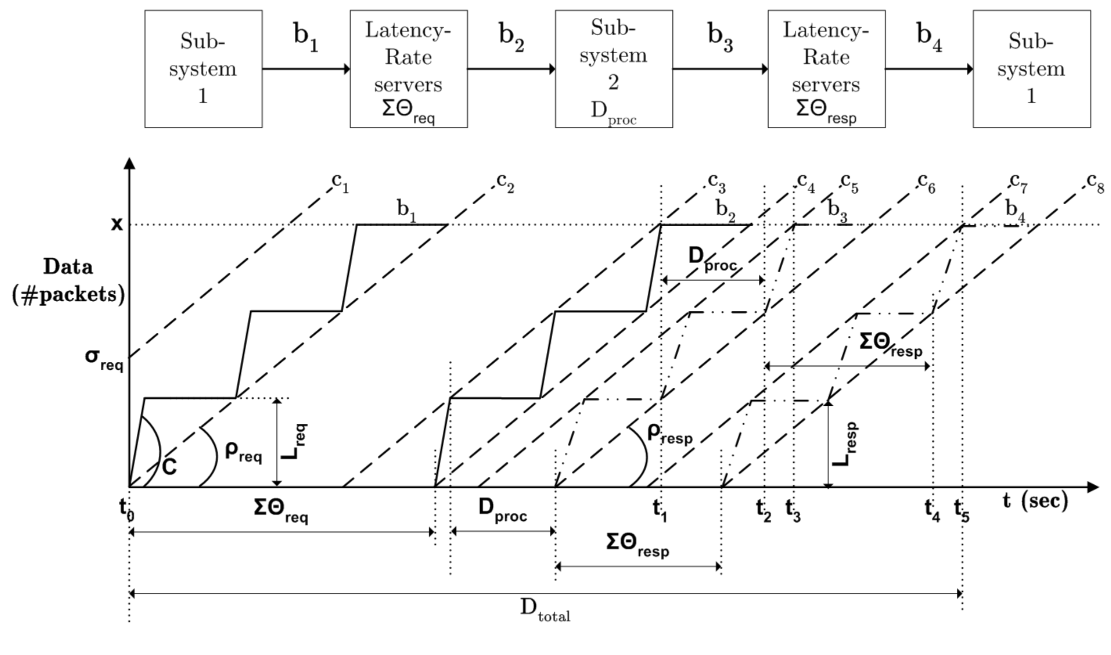


Figure 4.6: Request-response stream

Assume that the request flow (i.e. the input of the first chain of the LR servers) is characterized by b_1, σ_{req}, ρ_{req} and L_{req}. Assume that the maximum rate of the communication infrastructure is C. Furthermore, assume that the response flow (i.e. the input of the second chain of LR servers) is characterized by b_3, σ_{resp}, ρ_{resp} and L_{resp}. Finally, assume that $\frac{L_{req}}{\rho_{req}} = \frac{L_{resp}}{\rho_{resp}}$. Because L denotes the length of a packet in *words* and ρ represents the number of *words* per *sec*, ρ_{req} and ρ_{resp} represent the same number of packets per time unit.

Please note that in figure 4.6 data is expressed in number of packets.

The request flow is bounded by c_1 and c_2. By definition of LR server, these requests arrive at subsystem 2 before c_4. Then, at most D_{proc} later, the responses are sent back to subsystem 1. Therefore, the response session b_3 is lower bounded by c_6. Finally, the responses arrive at subsystem 1 before c_8, because of the definition of LR servers.

To determine the total delay, assume that x *words* of requests have to be sent. Please note that the number of requests is equal to the number of responses. Then, the total delay is the period of time between the first request is sent (i.e. t_0) and the last response is received at subsystem 1 (i.e. t_5). So,

$$\begin{aligned}
D^{total} &\leq t_5 - t_0 && (4.17)\\
&= (t_1 - t_0) + (t_5 - t_1) && (4.18)\\
&= \left\lceil \frac{x}{L_{req}} \right\rceil \cdot \frac{L_{req}}{\rho_{req}} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} - \frac{L_{req}}{\rho_{req}} + \frac{L_{req}}{C} + (t_5 - t_1) && (4.19)\\
&= \left\lceil \frac{x}{L_{req}} \right\rceil \cdot \frac{L_{req}}{\rho_{req}} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} - \frac{L_{req}}{\rho_{req}} + \frac{L_{req}}{C} + (t_2 - t_1) + (t_5 - t_2) && (4.20)\\
&= \left\lceil \frac{x}{L_{req}} \right\rceil \cdot \frac{L_{req}}{\rho_{req}} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} - \frac{L_{req}}{\rho_{req}} + \frac{L_{req}}{C} + D_{proc} + (t_5 - t_2) && (4.21)\\
&= \left\lceil \frac{x}{L_{req}} \right\rceil \cdot \frac{L_{req}}{\rho_{req}} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} - \frac{L_{req}}{\rho_{req}} + \frac{L_{req}}{C} + D_{proc}\\
&\quad + (t_5 - t_4) + (t_4 - t_2) && (4.22)\\
&= \left\lceil \frac{x}{L_{req}} \right\rceil \cdot \frac{L_{req}}{\rho_{req}} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} - \frac{L_{req}}{\rho_{req}} + \frac{L_{req}}{C} + D_{proc}\\
&\quad + \frac{L_{resp}}{C} + \sum_{j=1}^{m'} \Theta_{resp}^{(S_j)} && (4.23)
\end{aligned}$$

As has become clear, the total delay consists of three parts. First of all, a normal delay of sending x *words* (see equation 4.10). After that, the last request packet has to be processed by subsystem 2. This takes at most D_{proc}. Finally, the last response has to be sent back to subsystem 1. This is equal to the delay of sending one packet.

The backlog is determined on the standard way. For the request session, the backlog of the k^{th} LR server in the first chain of LR severs is bounded by

$$Q_{req}^{k} \leq \sigma_{req} + \rho_{req} \cdot \sum_{j=1}^{k} \Theta_{req}^{(S_j)} \qquad (4.24)$$

For the response session, the same formula can be used. The backlog of the k'^{th} LR server in the second chain of LR severs is bounded by

$$Q_{resp}^{k'} \leq \sigma_{resp} + \rho_{resp} \cdot \sum_{j=1}^{k'} \Theta_{resp}^{(S_j)} \qquad (4.25)$$

So, the total delay for a request-response stream is the combination of the delay of two sessions and the processing time (i.e. D_{proc}) of a request. The formulas for the backlog do not change.

4.3.2 Pipeline degree

Until now, there was no limitation on the number of packets in flight (i.e. the number of outstanding requests). But, for most subsystems the number of packets in flight is limited. Most subsystems are not capable of having a large number of outstanding requests. They have a maximum number of outstanding requests. This maximum number of requests or packets of a subsystem in flight is called the "pipeline degree". This is an extra characteristic of the traffic and can have a significant impact on the delay and the backlog. Please note that the requests of a particular subsystem can be distributed over a chain of LR servers. Furthermore, a higher pipeline degree can be used to lower the delay of a session. In this section, the influence of the pipeline degree for the delay and the backlog is determined.

Assume that subsystem 1 outputs a session (see figure 4.7). This session is characterized by b_1, σ, ρ and L. The packets of this session are sent via a chain of m LR servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$, to subsystem 2. Subsystem 2 processes the packets. Assume that b_2 represents the input traffic of subsystem 2. Finally, assume that no packets are sent back.

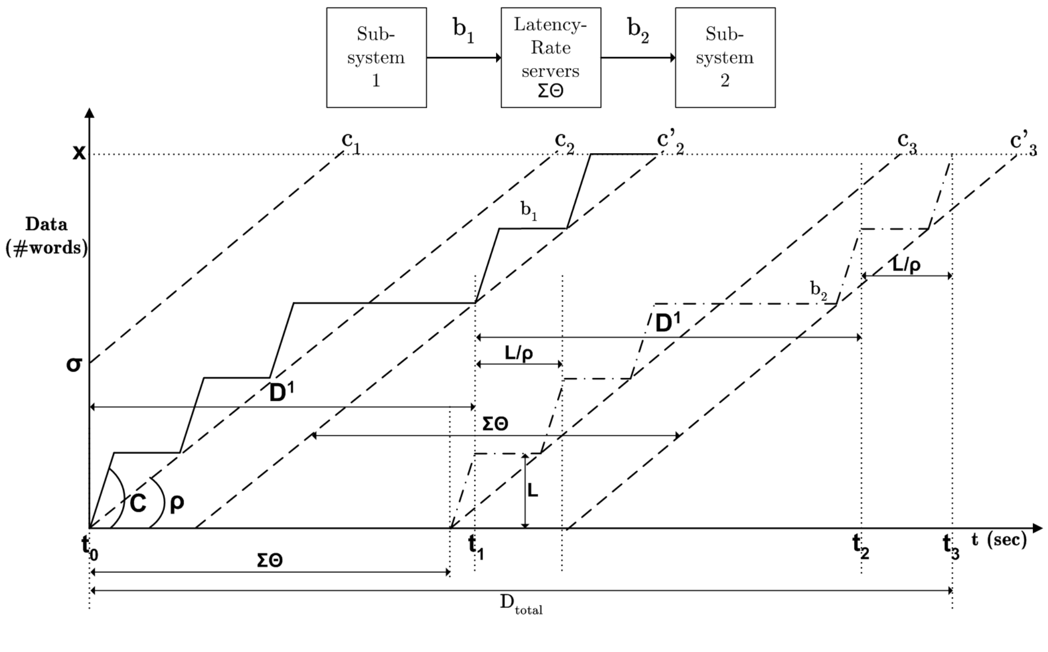


Figure 4.7: Pipeline degree for a request-only stream

For this example, assume that the pipeline degree is equal to 3. That means that when the first packet has been received by subsystem 2 (worst case at t_1), the fourth packet can be sent by subsystem 1, but not sooner.

For the first three packets, b_2 is lower bounded by c_3. By definition of LR servers, these packets cannot arrive later than c_3. When the first packet is received by subsystem 2 (i.e. worst case at t_1), the fourth packet can be sent. This process continues until all packets are sent.

Please note that the burstiness constraint of b_1 depends on the pipeline degree. In the case that the pipeline degree is equal to one, then at most one packet can be in flight (i.e. $\sigma \leq L \cdot (1 - \frac{\rho}{C})$). In general, if n is the pipeline degree, then

$$\sigma \leq n \cdot L \cdot \left(1 - \frac{\rho}{C}\right) \tag{4.26}$$

This is the case, because at most n packets can be sent in one burst (see section 4.2.3).

Assume that x *words* have to be sent, using packets of size L. Furthermore, assume that n is the pipeline degree. Then, the first packet is received at

$$\begin{aligned} D^1 &\leq t_1 - t_0 && (4.27)\\ &= \frac{L}{C} + \sum_{j=1}^{m} \Theta^{(S_j)} && (4.28) \end{aligned}$$

In case that $n = 1$, the second packet departs at subsystem 1, when the first packet arrives at subsystem 2. So, the first two packets are received at

$$D^2 \leq 2 \cdot D^1 \qquad (4.29)$$

In case that $n \geq 2$, the maximum delay can be optimized (see figure 4.7). Then, the first two packets are received at

$$\begin{aligned} D^2 &\leq (t_1 - t_0) + \frac{L}{\rho} && (4.30)\\ &= D^1 + \frac{L}{\rho} && (4.31) \end{aligned}$$

In case that $n = 3$, the fourth packet can be sent after receiving the first packet. Then, b_1 is lower bounded by c_2' and a second period of D^1 starts. So, the first five packets are received at (see figure 4.7)

$$\begin{aligned} D^5 &\leq t_3 - t_0 && (4.32)\\ &= (t_1 - t_0) + (t_3 - t_1) && (4.33)\\ &= D^1 + (t_2 - t_1) + (t_3 - t_2) && (4.34)\\ &= D^1 + D^1 + \frac{L}{\rho} && (4.35) \end{aligned}$$

In general, assume that n is the pipeline degree and x *words* have to be received. Then, the maximum delay is

$$\begin{aligned} D^{total} &\leq \left\lceil \frac{x}{n \cdot L} \right\rceil \cdot \left(\frac{L}{C} + \sum_{j=1}^{m} \Theta^{(S_j)} \right)\\ &\quad + \left(\left\lceil \frac{x}{L} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L}{\rho} && (4.36)\\ &= \left\lceil \frac{x}{n \cdot L} \right\rceil \cdot D^1 + \left(\left\lceil \frac{x}{L} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L}{\rho} && (4.37) \end{aligned}$$

This is the case, because x *words* have to be sent. A maximum of n packets of L *words* (i.e. $n \cdot L$ *words*) can be in flight. Then, there are $\lceil \frac{x}{n \cdot L} \rceil$ periods of D^1. In total, $\lceil \frac{x}{L} \rceil$ packets have to be received. After $\lceil \frac{x}{n \cdot L} \rceil$ periods of D^1, at least $n \cdot (\lceil \frac{x}{n \cdot L} \rceil - 1) + 1$ packets have been received. This is the case, because at the end of the m^{th} period of D^1, $n \cdot (m-1) + 1$ packets have already been received, otherwise the next period could not start. Then, at most $\lceil \frac{x}{L} \rceil - n \cdot (\lceil \frac{x}{n \cdot L} \rceil - 1) - 1$ packets still have to be received. This takes at most $\frac{L}{\rho}$ for each packet.

Please note that $\lceil \frac{x}{L} \rceil - n \cdot \left(\lceil \frac{x}{n \cdot L} \rceil - 1\right) - 1 < n$. This means that after $\lceil \frac{x}{n \cdot L} \rceil$ periods of D^1, at most $(n-1)$ packets have to be received. This is proved in lemma A.1.3 of appendix A.

If n is large, D^{total} converges to $\lceil \frac{x}{L} \rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C}$ (see equation 4.10). This is proved in lemma A.1.4 of appendix A.

In the case that

$$D^1 = \frac{L}{C} + \sum_{j=1}^{m} \Theta^{(S_j)} < n \cdot \frac{L}{\rho} \tag{4.38}$$

the period of D^1 is not long enough to send n packets at rate ρ. Then, formula 4.10 has to be used to calculate the delay.

Finally, the upper bound for the backlog can still be calculated by equation 4.11.

The above formulas are derived for a request-only stream. In case of a request-response stream, the same analysis can be applied.

Assume the same situation as in the previous section (see figure 4.6), only this time the number of packets in flight is limited (see figure 4.8).

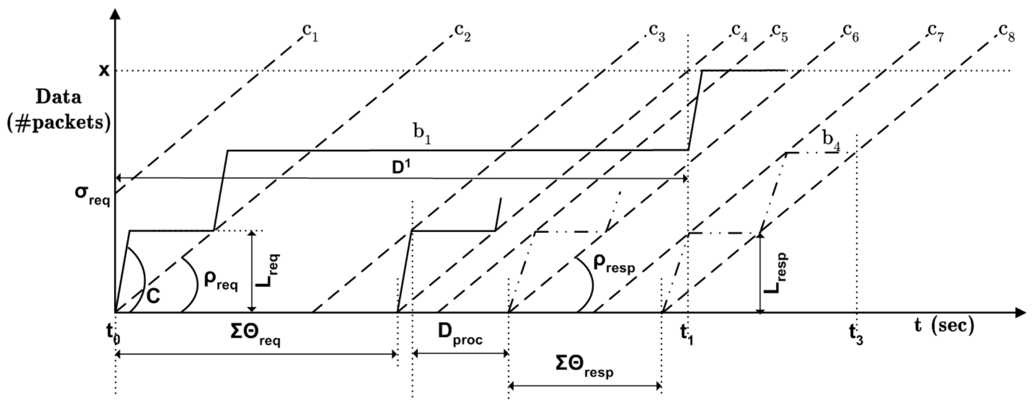


Figure 4.8: Pipeline degree for a request-response stream

Assume that $n \geq 2$ is the pipeline degree for both chains together (i.e. the number of requests that can be issued depends on the number of responses that have been received). Then, the first request and response have a delay of

$$\begin{aligned} D^1 &\leq t_1 - t_0 && (4.39) \\ &= \frac{L_{req}}{C} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} + D_{proc} + \sum_{j=1}^{m'} \Theta_{resp}^{(S_j)} + \frac{L_{resp}}{C} && (4.40) \end{aligned}$$

The first two packet requests and responses have a delay of

$$\begin{aligned} D^2 &\leq \frac{L_{req}}{C} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} + \frac{L_{req}}{\rho_{req}} + D_{proc} + \sum_{j=1}^{m'} \Theta_{resp}^{(S_j)} + \frac{L_{resp}}{C} && (4.41) \\ &= D^1 + \frac{L_{req}}{\rho_{req}} && (4.42) \end{aligned}$$

This means that at t_0, the first request is sent. The response to this request is received by subsystem 2 at t_1 or earlier. The time between t_0 and t_1 is equal to D^1. Then, the next response is received within $\frac{L_{req}}{\rho_{req}} = \frac{L_{resp}}{\rho_{resp}}$ *sec.*

Assume that n is the pipeline degree. Then, x *words* of requests and responses to the requests have a maximum delay of (see equation 4.36, using equations 4.40 and 4.42)

$$\begin{aligned} D^{total} &\leq \left\lceil \frac{x}{n \cdot L_{req}} \right\rceil \cdot \left(\frac{L_{req}}{C} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} + D_{proc} + \sum_{j=1}^{m'} \Theta_{resp}^{(S_j)} + \frac{L_{resp}}{C} \right) \\ &\quad + \left(\left\lceil \frac{x}{L_{req}} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L_{req}} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L_{req}}{\rho_{req}} \qquad (4.43) \\ &= \left\lceil \frac{x}{n \cdot L_{req}} \right\rceil \cdot D^1 + \left(\left\lceil \frac{x}{L_{req}} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L_{req}} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L_{req}}{\rho_{req}} \qquad (4.44) \end{aligned}$$

Please note that D^1 is equal to $t_1 - t_0$ in figure 4.8.

In case that

$$D^1 = \frac{L_{req}}{C} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} + D_{proc} + \sum_{j=1}^{m'} \Theta_{resp}^{(S_j)} + \frac{L_{resp}}{C} < n \cdot \frac{L_{req}}{\rho_{req}} \qquad (4.45)$$

the period of D^1 is not long enough to send n request packets at rate ρ_{req}. Then, formula 4.23 has to be used to calculate the delay.

The formulas for the backlog (see equations 4.24 and 4.25) do still apply.

As has become clear above, the delay may change significantly by introducing the pipeline degree. This in contrast to the backlog.

4.4 Memory system characteristics

In this section, the memory system characteristics of section 3.1 are introduced into our performance analysis method. A memory system is modeled as is shown in figure 4.9. First, the influence of the change of the packet size from a request packet to a response packet is discussed. After that, the refresh requirement of DRAM is discussed. Finally, the influence of the processing time of a request is discussed.

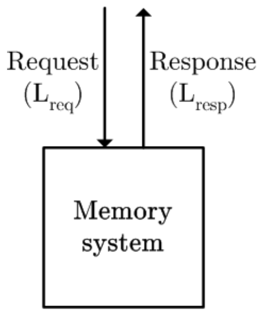


Figure 4.9: Write and read requests (of size L_{req}) enter the memory system and responses (of size L_{resp}) leave the memory system

4.4.1 Packet size change

When a load request is sent to a memory system, the memory system sends one load response back. This is a request-response stream, as explained in section 4.3.1. But, the size of a request packet is not equal to the size of a response packet. The derived delay bound of section 4.3.1 has already anticipated this situation. Therefore, the formulas of the request-response stream can be used without any adaptation.

4.4.2 Refresh requirement

In case a DRAM is used as memory system, the data stored in the DRAM has to be refreshed frequently. Therefore, time has to be reserved periodically to refresh the content of the memory system.

A DRAM memory system consists of two parts, the DRAM memory cells and the DRAM controller. The memory cells of a DRAM consist of capacitors. These capacitors leak charge. To overcome this problem, the capacitors have to be recharged periodically.

A DRAM controller is the scheduler that determines the order of the memory requests. So, the DRAM controller should periodically sent a refresh request to the DRAM, to recharge the capacitors. Therefore, an extra traffic session for the DRAM controller is introduced. Assume that a refresh request packet has a size of $L_{refresh}$ and the memory system has a maximum rate of C. Furthermore, assume that each $T_{refresh} \geq 0$ *sec* a refresh has to be executed by the DRAM. This refresh session has the following characteristics (see section 4.2.3)

$$\sigma_{refresh} \geq L_{refresh} \cdot \left(1 - \frac{\rho_{refresh}}{C}\right) \tag{4.46}$$

$$\rho_{refresh} \geq \frac{L_{refresh}}{T_{refresh}} \tag{4.47}$$

Then, periodically a refresh request is sent to the DRAM.

4.4.3 Packet stretcher

In the method of Stiliadis, the delays depend on the size of the packets. Unfortunately, the time a memory system requires to process a request does not depend on the size of the request, but on the kind of the request. A write request will take more time than a read request [16]. Also, a read request for 128 *bytes* will take more time than a read request for 32 *bytes*. Assume that the size of a request is L and the maximum rate of the DRAM memory system is C. Then, $\frac{L}{C}$ *sec* are reserved by the DRAM controller to process the request without any adaptations to the derived formulas. Assume that T_{req} *sec* are required by the DRAM to process the request. Then, $\frac{L}{C}$ *sec* should be equal to T_{req} *sec*, but $\frac{L}{C}$ *sec* is usually much smaller than T_{req} *sec*. Therefore, the DRAM memory controller is not an *LR* server. So, our performance analysis method has to be adapted.

There are several ways to overcome this problem.

One option is to use a propagation delay (see figure 4.10). Unfortunately, this is not correct. A propagation delay only represents the delay between two subsystems. The problem is that the DRAM cannot be seen as a pipeline, and therefore the DRAM is not long enough reserved by making use of a propagation delay. This means that the next request is already sent by the DRAM controller to the DRAM, although the DRAM has not processed the previous request completely. So, the next request does not wait long enough, before the DRAM is available again.

Another possibility is to change the maximum rate of the memory (i.e. C'), such that $\frac{L}{C'}$ is equal to the processing time of the request (i.e. T_{req}). This is also not a good option, because load and store requests do not scale with the same coefficient, although these have the same packet size. A store request requires more processing time than a load request [16]. Using this adaptation, the delay and queue bounds are too relaxed.

An alternative is to increase the size of a request, when it is sent. Then, the requests are enlarged, such that the size (i.e. L') represents the processing time of the memory system

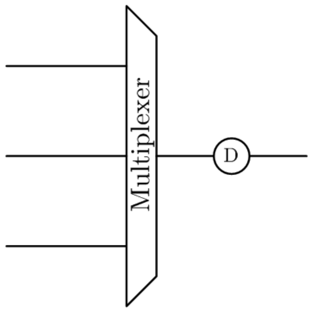


Figure 4.10: Multiplexer of the DRAM controller ("D" denotes a propagation delay)

(i.e. $\frac{L'}{C} = T_{req}$). Then, the calculated delays and queue sizes for all LR servers are not tight anymore. This is the case, because for all schedulers $\frac{L'}{C}$ *sec* are reserved to send the request, instead of $\frac{L}{C}$ *sec*. So, it should be the case that the packets are only enlarged with regard to a DRAM controller. If it is a regular scheduler, the normal packet size should be used.

So, a packet stretcher [13] is introduced. A packet stretcher temporarily changes the size of the request packets, such that the time reserved for the request is equal to the processing time of the request. Then, the output request packets of the packet stretcher have a length of $L' = C \cdot T_{req}$ instead of L. Please note that by changing the size of the request packets, the latency of this session in the DRAM controller changes. Now, enough time is reserved to process the request by the DRAM, before the DRAM controller sends the next request to the DRAM.

Please note that by changing the size of the request packets, the σ and the ρ of the request session also change (see table 4.1). This is the case because the number of packets per time unit remains constant, but the size of the packets changes. Then, ρ scales with the ratio between L and L' (this is proved in lemma A.1.5 of appendix A). For the burstiness constraint of the session, it is more complicated. First, the original maximum burst size is determined (i.e. $\frac{\sigma}{1-\frac{\rho}{C}}$). After that, the new burstiness constraint is calculated by making use of the new packet size (i.e. $\frac{L'}{L}$) and the adapted ρ (i.e. $\frac{L'}{L} \cdot \rho$). This is proved in lemma A.1.6 of appendix A.

	Stretcher in	*Stretcher out*
Packet size	L	L'
Rate	ρ	$\rho \cdot \frac{L'}{L}$
Burstiness constraint	σ	$\frac{\sigma}{1-\frac{\rho}{C}} \cdot \frac{L'}{L} \cdot \left(1 - \frac{\frac{L'}{L} \cdot \rho}{C}\right)$

Table 4.1: Influence of a packet stretcher on the L, ρ and σ of a session

To use the packet stretcher, every input session of the DRAM controller gets a stretcher. These are located between the queues and the multiplexer (see figure 4.11). The memory controller (i.e. the DRAM controller) receives the requests. Then, the arbitration policy of the multiplexer of the controller determines the order in which the requests are performed by the DRAM. The requests are forwarded to the DRAM. There, a response can be generated and sent back.

Please note that the size of the request packet in the packet stretcher is only virtually changed. Also note that by making use of a packet stretcher, only the latencies of the DRAM controller change, such that the next request is forwarded to the DRAM as soon

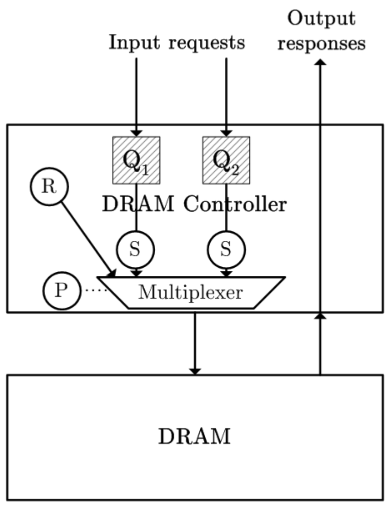


Figure 4.11: DRAM memory system ("P" denotes the arbitration policy, "S" denotes a stretcher, "Q" denotes a queue, "R" denotes the refresh session)

as the DRAM is available. This means that the output of the packet stretcher is only used for determining the latency of the DRAM controller.

So, by adding a packet stretcher, enough time is reserved for each request to be processed by the DRAM. Then, the DRAM controller can be seen as an *LR* server.

4.5 Interconnect characteristics

The interconnect consists, according to the model of Stiliadis, of *LR* servers. These *LR* servers can easily be combined in a chain. Stiliadis has introduced a number of arbitration policies for the *LR* servers. Unfortunately, some typical arbitration policies of schedulers for a SoC have not been introduced into the method of Stiliadis. Therefore, Round Robin (two variants), TDMA and Fixed Priority are introduced into our performance analysis method in this section. After that, an overview of the properties of all arbitration policies described in this thesis is given.

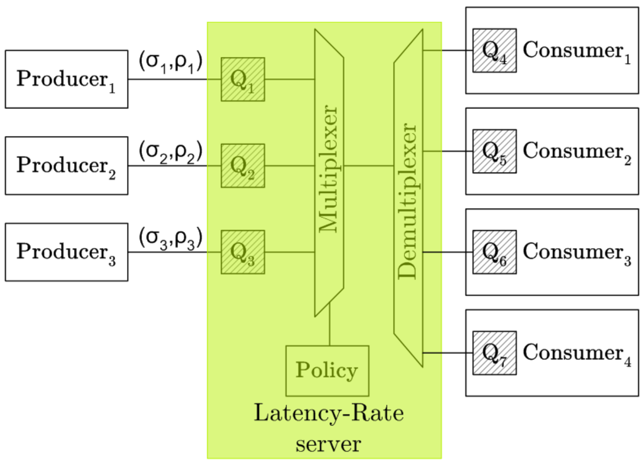


Figure 4.12: A system consisting of a Latency-Rate server, 3 producers and 4 consumers [22] ("*Q*" denotes a queue)

Please remember the working of an *LR* server (see figure 4.12). Several producers generate data, that is sent to the *LR* server. The arbitration policy of the *LR* server determines the order of the output packets. The latency of a session is the maximum amount of time between the moment the first packet of a busy period of that session has arrived completely in the scheduler and the moment on which that packet has left the scheduler completely. This latency is used to determine the upper bounds for the delay and the backlog. So, the worst case scenario is required. Please note that in the worst case scenario, all input sessions of the *LR* server are transferring the maximum amount of data that is allowed by their σ and ρ.

4.5.1 Round Robin Time based

A typical arbitration policy for SoC schedulers is Round Robin. Several variants of Round Robin are available. In this section, the latency for Round Robin Time based is determined.

A Round Robin Time based scheduler assigns equal parts of its time to each session. This means that all sessions get the same amount of service (i.e. ϕ) and no session gets priority. As defined before, the number of service in *words* assigned to session i in a round is called a "slot" (i.e. ϕ_i) and total number of service in *words* assigned in a complete round is called a "frame" (i.e. F). Please note that for Round Robin Time based $(\forall i : 1 \leq i \leq V : \phi_i = \phi)$, if V sessions (producers) share the scheduler.

The amount of time each session gets per round is determined by the largest packet. This is the case because packets cannot be split. Therefore,

$$F = V \cdot \phi \tag{4.48}$$

$$\phi \geq L_{max} \tag{4.49}$$

Then, for each session a time slot of $\frac{\phi}{C}$ *sec* per round is reserved. This means that a session with smaller packets than ϕ can send one or more packets per round.

Let assume that V sessions share the scheduler and the scheduler has a capacity of C. Then, each session gets a maximum service rate of $\frac{1}{V} \cdot C$. By definition of *LR* servers, the rate of each session (i.e. ρ_i) should not be larger than its maximum assigned service rate by the scheduler, i.e.

$$\left(\forall i : 1 \leq i \leq V : \rho_i \leq \frac{1}{V} \cdot C\right) \tag{4.50}$$

Assume that a particular packet of session i is at the head of its queue. Because Round Robin is used, in the worst case scenario the packet is sent the next round. This means that the packet has to wait at most $\frac{F-\phi}{C}$ *sec* before the session is serviced again. Then, the packet at the head of the queue is sent. So, the latency is

$$\Theta_i^{RRTB} = \frac{F - \phi + L_i}{C} \tag{4.51}$$

More formally, a server with Round Robin Time based as arbitration policy is an *LR* server, because the first packet of session i has a maximum delay equal to Θ_i^{RRTB} and after the first packet of that session has left the *LR* server, a service rate of ρ_i is guaranteed. Therefore, requirement 4.50 is necessary. This is proved in lemma A.1.7 of appendix A. Furthermore, the latency is as tight as possible (see corollary A.1.8 of appendix A).

4.5.2 Round Robin Packet based

An alternative of the previous scheduling policy is a Round Robin scheduler that allows each session to send at most one packet per round (i.e. Round Robin Packet based).

Assume that V sessions share the scheduler and the scheduler has a capacity of C. Then, a session i gets a slot equal to the size of its packet (i.e. $\phi_i = L_i$). Then, the size of a frame is equal to

$$F = \sum_{i=1}^{V} L_i \tag{4.52}$$

Assume that a particular packet of session i is at the head of its queue. Because Round Robin is used, in the worst case scenario the packet is sent the next round. This means that the packet has to wait at most $\frac{F-\phi_i}{C}$ *sec* before the session is serviced again. Then, the packet at the head of the queue is sent. So, the latency for session i is

$$\Theta_i^{RRPB} = \frac{F - \phi_i + L_i}{C} \tag{4.53}$$

Because $\phi_i = L_i$, this can be simplified to

$$\Theta^{RRPB} = \frac{F}{C} \tag{4.54}$$

Please note that the LR server allocates a maximum service rate of $\frac{L_i}{F} \cdot C$ to session i. Therefore, the following requirement is needed.

$$\left(\forall i : 1 \leq i \leq V : \rho_i \leq \frac{L_i}{F} \cdot C\right) \tag{4.55}$$

More formally, a server with Round Robin Packet based as arbitration policy is an LR server, because the first packet of session i has a maximum delay equal to Θ_i^{RRPB} and after the first packet of that session has left the LR server, a service rate of ρ_i is guaranteed. Therefore, requirement 4.55 is necessary. This proof goes similar as lemma A.1.7 of appendix A.

4.5.3 TDMA

A TDMA[1] scheduler is similar to Round Robin. In TDMA, a session can send multiple packets per round.

Assume that V sessions share the scheduler and the scheduler has a capacity of C. A session gets a fixed amount of time per round to process packets. This amount of time is determined in advance. So, in advance the number of packets (i.e. w_i, $w_i \geq 1$ and $w_i \in \mathbb{N}$) session i is allowed to send each round is determined. Then, the size of a slot (i.e. ϕ_i) and a frame (i.e. F) are calculated by

$$\begin{aligned} \phi_i &= w_i \cdot L_i && (4.56) \\ F &= \sum_{i=1}^{V} \phi_i && (4.57) \end{aligned}$$

1. Time Division Multiple Access

Assume that a particular packet of session i is at the head of its queue. Because Round Robin is used, in the worst case scenario the packet has to wait at most $\frac{F-\phi_i}{C}$ *sec* before the session is serviced again. Then, the packet at the head of the queue is sent. So, the latency is

$$\Theta_i^{TDMA} = \frac{F - \phi_i + L_i}{C} \tag{4.58}$$

Just as in the previous section, the *LR* server allocates a maximum service rate of $\frac{\phi_i}{F} \cdot C$ to session i. Therefore, the rate of session i (i.e. ρ_i) should not be larger than its maximum assigned service rate by the scheduler, i.e.

$$\left(\forall i : 1 \leq i \leq V : \rho_i \leq \frac{\phi_i}{F} \cdot C\right) \tag{4.59}$$

Please note that in TDMA, in contrast to Round Robin, unused slots are not skipped.

More formally, a server with TDMA as arbitration policy is an *LR* server, because the first packet of session i has a maximum delay equal to Θ_i^{TDMA} and after the first packet of that session has left the *LR* server, a service rate of ρ_i is guaranteed. Therefore, requirement 4.59 is necessary. This is proved in lemma A.1.9 of appendix A. Furthermore, the latency is as tight as possible (see corollary A.1.10 of appendix A).

4.5.4 Fixed Priority

A completely different arbitration policy is Fixed Priority. Using this arbitration policy, the priorities of the sessions determine the order of the output.

For a Fixed Priority scheduler, priorities are assigned to the sessions. That means that a session with a higher priority has a smaller latency, than a session with a lower priority. Assume that non-preemptive schedulers are used. This means that it is not possible that a packet that is sent, is interrupted. Furthermore, assume that V sessions share a scheduler with a capacity of C and that session 1 has the highest priority, session 2 has the next highest priority and so on.

Then, the first packet of session 1 has a latency of

$$\begin{aligned}
\Theta_1^{FP} &= effective\ processing\ time\ of\ current\ packet \\
&\quad + effective\ processing\ time\ of\ first\ packet\ of\ session\ 1 && (4.60)\\
&= \frac{L_{max}}{C} + \frac{L_1}{C} && (4.61)
\end{aligned}$$

Please note that L_{max} is the maximum length of a packet the *LR* server can receive.

So, it takes at most $\frac{L_{max}}{C}$ *sec* to send the current packet, because a non-preemptive scheduler is used. After that, because session 1 has the highest priority, the first packet of session 1 is sent.

For session 2, it is always possible that a packet of session 1 arrives. Therefore, a service rate of ρ_1 has to be reserved for session 1. Then, session 2 has a latency of

$$\begin{aligned}
\Theta_2^{FP} &= effective\ processing\ time\ of\ current\ packet \\
&\quad + effective\ processing\ time\ of\ burst\ of\ session\ 1 \\
&\quad + effective\ processing\ time\ of\ first\ packet\ of\ session\ 2 && (4.62) \\
&= \frac{L_{max}}{C-\rho_1} + \frac{\sigma_1}{C-\rho_1} + \frac{L_2}{C} && (4.63) \\
&= \frac{L_{max}+\sigma_1}{C-\rho_1} + \frac{L_2}{C} && (4.64)
\end{aligned}$$

The current packet has a maximum size of L_{max}. After sending that packet, in the worst case scenario, the maximum burst of session 1 (i.e. σ_1) has to be sent. The maximum guaranteed rate to send the current packet and this burst is $C - \rho_1$. This is the case, because ρ_1 has to be reserved for new packets of session 1, that arrive during sending the current packet and the burst of session 1. After that, the first packet of session 2 can be sent. Then, it takes $\frac{L_2}{C}$ *sec* to send the first packet of session 2.

So in general, the latency for session i is

$$\begin{aligned}
\Theta_i^{FP} &= effective\ processing\ time\ ofcurrent\ packet \\
&\quad + effective\ processing\ time\ of\ bursts \\
&\quad + effective\ processing\ time\ of\ first\ packet\ of\ session\ i && (4.65) \\
&= \frac{L_{max}}{C-\sum_{j=1}^{i-1}\rho_j} + \frac{\sum_{j=1}^{i-1}\sigma_j}{C-\sum_{j=1}^{i-1}\rho_j} + \frac{L_i}{C} && (4.66) \\
&= \frac{L_{max}+\sum_{j=1}^{i-1}\sigma_j}{C-\sum_{j=1}^{i-1}\rho_j} + \frac{L_i}{C} && (4.67)
\end{aligned}$$

It takes at most $\frac{L_{max}}{C}$ *sec* to process the current packet. But, in the meanwhile, packets of sessions with a higher priority than session i can arrive. In that case, these packets have also to be processed before a packet of session i can be processed. So a rate of $\sum_{j=1}^{i-1}\rho_j$ is reserved. Then, the maximum guaranteed rate to send the current packet is $C - \sum_{j=1}^{i-1}\rho_j$. In the worst case scenario, all sessions with a higher priority than session i send their maximum bursts. That means that those packets have to be processed, before packets of session i can be processed. During that period, only a rate of $C - \sum_{j=1}^{i-1}\rho_j$ can be used, because a rate of $\sum_{j=1}^{i-1}\rho_j$ has to be reserved for the new packets of the sessions with a higher priority than session i. Finally, the first packet of session i can be sent. This takes $\frac{L_i}{C}$ *sec.*

Fixed Priority is a valid LR-server, because after Θ_i^{FP}, the complete bursts of all sessions with a higher priority than i and the first packet of session i are sent. Then, a rate of ρ_i is guaranteed, because $\sum_{j=1}^{i-1}\rho_j$ is reserved for the sessions with a higher priority and $\sum_{j=1}^{i}\rho_j \leq C$. This is proved in lemma A.1.11 of appendix A. Furthermore, the latency is as tight as possible (see corollary A.1.12 of appendix A).

4.5.5 Overview schedulers

In this section, an overview of the properties of the schedulers of sections 3.3.5 and 4.5 is given.

In table 4.2, the properties of the *LR* servers are mentioned. The *LR* servers are characterized by the following five categories:

- Size of slots, meaning if all slots are equal or not
- Unused slots, meaning if the unused slots are skipped or not
- Order per round, meaning if the order of the slots is fixed or dynamic
- Size per round, meaning if the amount of data to be processed per round is fixed or dynamic
- Number of packets per round, meaning if one or more packets per session can be sent in a single round

Policy	Equal size slots	Non equal size slots	Skip unused slots	Do nothing during unused slots	Fixed order per round	Dynamic order per round	Static size per round	Dynamic size per round	Single packet per round	Multiple packet per round	Remark
FCFS		x	x			x		x		x	Not an LR server
VC		x	x			x		x		x	
DRR		x	x		x			x		x	Extra constraint
WRR		x	x		x		x			x	Extra constraint
RRTB	x		x		x		x			x	Extra constraint
RRPB		x	x		x		x		x		Extra constraint
TDMA		x		x	x		x			x	Extra constraint
FP		x	x			x		x		x	

Table 4.2: Overview of the properties of the arbitration policies of *LR* servers

In the sections above, some typical arbitration policies of schedulers for SoCs are introduced into our performance analysis method. In the next section, some improvements for the performance bounds are discussed.

4.6 Improvements

In this section, two improvements for the backlog bounds are discussed, namely the use of a regulator and the use of a minimum latency.

4.6.1 Regulator

The size of the backlog of an *LR* server depends on the burstiness constraint of the input session of the *LR* server. To reduce the size of the queues needed, the maximum burst should be as small as possible. Unfortunately, the input session of a chain of *LR* servers determines the value of σ. A regulator can be used to enforce a lower σ value and thereby reduce the queues in the *LR* servers.

Assume that subsystem 1 produces packets (i.e. b_1) of size L_i, with parameters σ and ρ. These packets are sent via a chain of m *LR* servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$, to subsystem 2 (see figure 4.13). Note that b_1 is bounded by c_1 and c_2 and that b_2 (the output of the chain of *LR* servers) is bounded by c_1 and c_3. Furthermore, assume that no pipeline degree or a very high pipeline degree is used. Then, according to section 4.2.2, the delay and the backlog are calculated by the following formulas.

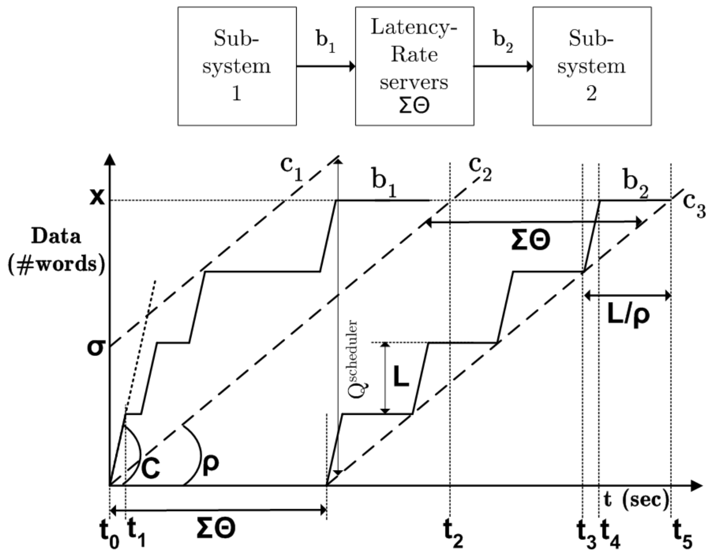


Figure 4.13: Chain of LR servers, without a regulator

The total delay for the x $words$ is (see equation 4.10)

$$D \leq \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} \tag{4.68}$$

The backlog of the k^{th} LR server in the chain of m LR servers is (see equation 4.11)

$$Q^k \leq \sigma + \rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \tag{4.69}$$

From the above formula of the backlog can be concluded that every LR server has to be able to store the maximum burst of the input session. The backlog of the first scheduler is

$$Q^1 \leq \sigma + \rho \cdot \Theta^{(S_1)} \tag{4.70}$$

and the backlog for the second scheduler is

$$Q^2 \leq \sigma + \rho \cdot (\Theta^{(S_1)} + \Theta^{(S_2)}) \tag{4.71}$$

By adding a regulator (see figure 4.14) before the traffic enters the chain of LR servers, the burstiness constraint of the input session of the chain of LR servers can be reduced. The regulator works as described in section 2.4.1. So, the input traffic of the regulator has a burstiness constraint of σ and a rate of ρ. Then, the regulator can output the packets with a minimum burstiness constraint (see section 4.2.3)

$$\sigma_{reg} = L \cdot \left(1 - \frac{\rho}{C}\right) \tag{4.72}$$

Then, b_1 is bounded by c_1' and c_2 and b_2 by c_1' and c_3 (see figure 4.14).

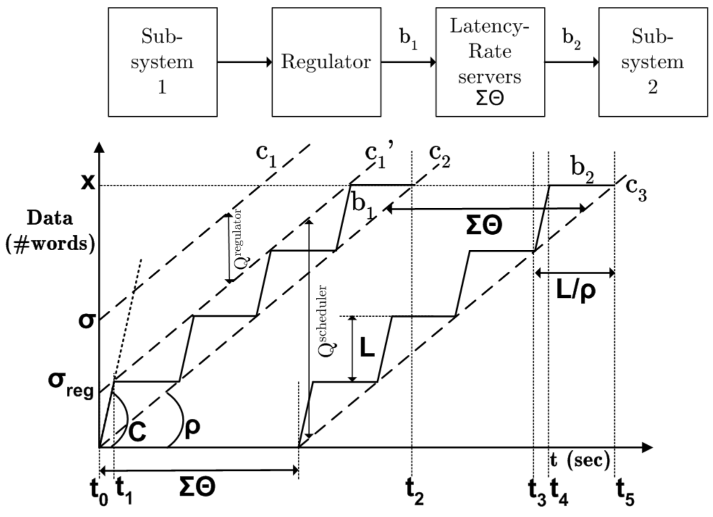


Figure 4.14: Chain of LR servers, with a regulator

Using a regulator, the backlog of the k^{th} LR server in the chain of LR servers is reduced to

$$Q^k \leq \sigma_{reg} + \rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \tag{4.73}$$

Please note that this backlog is the vertical distance between c_1' and c_3.

The price to pay is the regulator. The regulator requires a queue. The size of the backlog of a regulator is upper bounded in [9] by the difference between the burstiness constraint of the input traffic and the burstiness constraint of the output traffic of the regulator. To be precise, the backlog of the regulator (i.e. $Q^{regulator}$) is

$$\begin{aligned} Q^{regulator} &\leq \sigma_{in} - \sigma_{out} && (4.74) \\ &= \sigma - \sigma_{reg} && (4.75) \end{aligned}$$

Please note that the size of the backlog of the regulator is the vertical distance between c_1 and c_1' in figure 4.14. This is proved [9].

So, the total required queue size in a chain of m LR servers (Q^{total}) without a regulator is

$$\begin{aligned} Q^{total} &\leq Q^{schedulers\ without\ regulator} && (4.76) \\ &= \sum_{k=1}^{m} \left(\sigma + \rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \right) && (4.77) \\ &= m \cdot \sigma + \sum_{k=1}^{m} \left(\rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \right) && (4.78) \end{aligned}$$

and with regulator

$$\begin{aligned}
Q^{total} &\leq Q^{regulator} + Q^{schedulers\ with\ regulator} && (4.79)\\
&= (\sigma - \sigma_{reg}) + \sum_{k=1}^{m} \left(\sigma_{reg} + \rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \right) && (4.80)\\
&= (\sigma - \sigma_{reg}) + m \cdot \sigma_{reg} + \sum_{k=1}^{m} \left(\rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \right) && (4.81)\\
&= \sigma + (m-1) \cdot \sigma_{reg} + \sum_{k=1}^{m} \left(\rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} \right) && (4.82)
\end{aligned}$$

Therefore, by adding a regulator, the total reduction in required queue size in a chain of m LR servers is $(m-1) \cdot (\sigma - \sigma_{reg})$. So, the required queue size of the LR servers can be reduced by adding a regulator.

The formula for the delay (see equation 4.10) does still apply. This is proved in lemma A.1.13 of appendix A.

4.6.2 Minimum latency

The latency of a session is defined as the maximum amount of time between the moment the first packet of that session has arrived completely in the scheduler and the moment on which that packet has left the scheduler completely. A stricter bound on the backlog can be given, if the minimum delay of a packet in a scheduler is taken into account.

The latency of a scheduler is determined by two components: the arbitration time (i.e. Θ^{arb}) and the processing time (i.e. Θ^{proc}). The arbitration time represents the time between the moment the first packet of that session has arrived completely in the LR server and the moment the first packet of that session is the next packet to be serviced. The processing time is used to process and send the particular packet. Therefore, the processing time is lower bounded by $\frac{L}{C}$, if C is the capacity of the LR server and L the length of the particular packet.

In the case that all queues of the LR server are empty, the first packet of a session that arrives is sent immediately. In that case, the arbitration time is 0. So, the minimum delay of the session is

$$\begin{aligned}
D &\geq \min\left\{\Theta^{arb}\right\} + \min\left\{\Theta^{proc}\right\} && (4.83)\\
&= 0 + \frac{L}{C} && (4.84)\\
&= \frac{L}{C} && (4.85)
\end{aligned}$$

The minimum amount of time between the moment the first packet of a session has arrived completely in the LR server and the moment on which that packet has left the scheduler completely is called the minimum latency (indicated by Θ_{min}).

This is shown in figure 4.15. Assume that b_1 (with σ_{in} and ρ as parameters) represents the input session of the first LR server, with latency Θ. Furthermore, assume that the first packet of this session arrives at t_0. So, this packet can be serviced as soon it is received completely (i.e. at t_1). It is not possible that this packet is serviced earlier. Then, the packet can be serviced completely at t_2. This means that the LR server cannot service between c_1 and c_1'. Therefore, the LR server transmits between c_1' and c_3. So, the burstiness constraint of the session the next LR server can receive is not equal to the vertical distance between c_1 and c_3, but between c_1' and c_3. So, this is a reduction of $\rho \cdot (t_1 - t_0) = \rho \cdot \frac{L}{C}$.

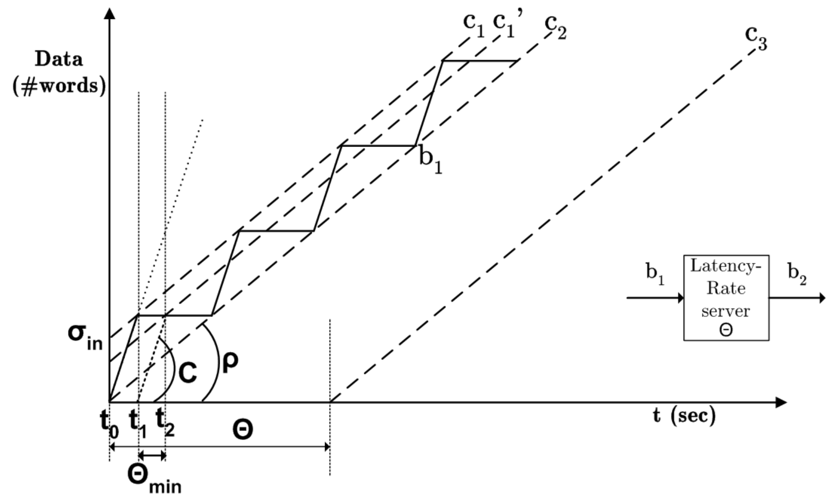


Figure 4.15: Reduction of the burstiness constraint of the output of the LR server by making use of the minimum latency

More general, assume that an LR server receives a session with the parameters σ_{in} and ρ. Furthermore, assume that Θ and Θ_{min} represents the latency respectively the minimum latency of the LR server for this session. Then, the burstiness constraint of the output session of the LR server (i.e. σ_{out}) is reduced to (see formula 3.34)

$$\sigma_{out} \leq \sigma_{in} + \rho \cdot \Theta - \rho \cdot \Theta_{min} \tag{4.86}$$

This is proved in lemma A.1.14 of appendix A.

Assume a chain of m LR servers. Furthermore, assume that the k^{th} LR server has a latency of $\Theta^{(S_k)}$ and a minimum latency of $\Theta_{min}^{(S_k)}$ for this session. Then, the backlog of the first LR server has not changed.

$$Q^{(S_1)} \leq \sigma_{in} + \rho \cdot \Theta^{(S_1)} \tag{4.87}$$

The burstiness constraint of the output session of the first LR server is upper bounded by

$$\sigma_{in} + \rho \cdot \Theta^{(S_1)} - \rho \cdot \Theta_{min}^{(S_1)} \tag{4.88}$$

Then, the backlog of the second LR server is

$$Q^{(S_2)} \leq \sigma_{in} + \rho \cdot \Theta^{(S_1)} - \rho \cdot \Theta_{min}^{(S_1)} + \rho \cdot \Theta^{(S_2)} \tag{4.89}$$

and the burstiness constraint of the output session of the second LR server is upper bounded by

$$\sigma_{in} + \rho \cdot \Theta^{(S_1)} - \rho \cdot \Theta_{min}^{(S_1)} + \rho \cdot \Theta^{(S_2)} - \rho \cdot \Theta_{min}^{(S_2)} \tag{4.90}$$

In general, the backlog of the k^{th} LR server in the chain of LR servers is bounded by

$$\begin{aligned} Q^{(S_k)} &\leq \sigma_{in} + \rho \cdot \sum_{j=1}^{k} \Theta^{(S_j)} - \rho \cdot \sum_{j=1}^{k-1} \Theta_{min}^{(S_j)} && (4.91) \\ &= \sigma_{in} + \rho \cdot \sum_{j=1}^{k-1} \left(\Theta^{(S_j)} - \Theta_{min}^{(S_j)} \right) + \rho \cdot \Theta^{(S_k)} && (4.92) \end{aligned}$$

Please note that the burstiness constraint of the output session of the k^{th} LR server of the chain of LR servers (i.e. σ_{out}) is bounded by

$$\sigma_{out} \leq \sigma_{in} + \rho \cdot \sum_{j=1}^{k} \left(\Theta^{(S_j)} - \Theta_{min}^{(S_j)} \right) \tag{4.93}$$

In general, the total required queue size for the chain of m LR servers ($Q^{schedulers}$) is

$$\begin{aligned} Q^{schedulers} &\leq \sum_{k=1}^{m} \left(\sigma_{in} + \rho \cdot \sum_{j=1}^{k-1} \left(\Theta^{(S_j)} - \Theta_{min}^{(S_j)} \right) + \rho \cdot \Theta^{(S_k)} \right) && (4.94) \\ &= m \cdot \sigma_{in} + \sum_{k=1}^{m} \left(\rho \cdot \sum_{j=1}^{k-1} \left(\Theta^{(S_j)} - \Theta_{min}^{(S_j)} \right) + \rho \cdot \Theta^{(S_k)} \right) && (4.95) \\ &= m \cdot \sigma_{in} + \rho \cdot \sum_{k=1}^{m} \left(\sum_{j=1}^{k-1} \left(\Theta^{(S_j)} - \Theta_{min}^{(S_j)} \right) + \Theta^{(S_k)} \right) && (4.96) \end{aligned}$$

As has become clear, a stricter bound on the required queue size can be given, if the minimum latency of a session in a scheduler is taken into account.

4.7 Application to SoC performance analysis

A SoC consists of subsystems, connected to a communication infrastructure. These subsystems communicate directly, or by making use of a memory system. Three different types of communication can be distinguished, namely

- Sending data from one subsystem via the interconnect to another subsystem
- Sending data from a subsystem via the interconnect to a memory system
- Sending data from a memory system via the interconnect to a subsystem

The first one is direct communication between two subsystems. The second one and the third one are communications between a subsystem and a memory system. In this section, the performance analysis method as described in this chapter is used for these types of SoC communication.

4.7.1 Direct communication between subsystems

The first type of communication is direct communication between two subsystems. One subsystem (the producer) sends packets via the interconnect to another subsystem (the consumer). Our performance analysis method is used for this type of traffic.

Assume a producer outputs a session, characterized by (σ_p, ρ_p) and L (see figure 4.16). This data is sent to a regulator. The regulator outputs the session with the parameters (σ_{reg}, ρ_p), such that $\sigma_{reg} \leq \sigma_p$. Then, the data is sent via a chain of m LR servers with capacity C, to the consumer with a rate of ρ_p. Assume that the k^{th} LR server has a latency of $\Theta^{(S_k)}$ and a minimum latency of $\Theta^{(S_k)}_{min}$ for this session. Furthermore, assume that n is the pipeline degree of the session and x *words* have to be sent to the consumer. Finally, assume that the x *words* have to be at the consumer side within $D^{max} \geq 0$ *sec*.

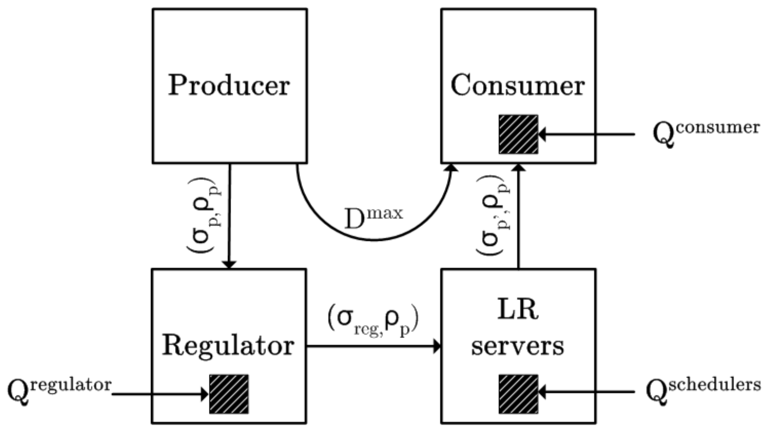


Figure 4.16: Direct communication

Please note σ_p is upper bounded by (see section 4.3.2)

$$\sigma_p \leq n \cdot L \cdot \left(1 - \frac{\rho_p}{C}\right) \tag{4.97}$$

Then, the total delay for x *words* (i.e. D^{total}) is by applying formula 4.36

$$\begin{aligned} D^{total} \leq & \left\lceil \frac{x}{n \cdot L} \right\rceil \cdot \left(\frac{L}{C} + \sum_{j=1}^{m} \Theta^{(S_j)} \right) \\ & + \left(\left\lceil \frac{x}{L} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L}{\rho_p} \end{aligned} \tag{4.98}$$

The deadline is met if $D^{total} \leq D^{max}$.

The required queue size of the regulator (i.e. $Q^{regulator}$) and all LR servers together (i.e. $Q^{schedulers}$) are (see formulas 4.74 and 4.96)

$$Q^{regulator} \leq \sigma_p - \sigma_{reg} \tag{4.99}$$

$$Q^{schedulers} \leq m \cdot \sigma_{reg} + \rho_p \cdot \sum_{k=1}^{m} \left(\sum_{j=1}^{k-1} \left(\Theta^{(S_j)} - \Theta^{(S_j)}_{min} \right) + \Theta^{(S_k)} \right) \tag{4.100}$$

The consumer is not always ready to process the received packets. Therefore, the consumer needs a queue (i.e. $Q^{consumer}$) to temporarily store the bursts of the input session of the consumer. The size of this queue is determined by the burstiness constraint of the input session of the consumer (i.e. $\sigma_{p'}$). According to formula 4.93, $\sigma_{p'}$ is upper bounded by

$$\sigma_{p'} \leq \sigma_{reg} + \rho_p \cdot \sum_{j=1}^{m} \left(\Theta^{(S_j)} - \Theta^{(S_j)}_{min} \right) \tag{4.101}$$

The consumer has to be able to store the maximum burst of its input session. Therefore,

$$Q^{consumer} \leq \sigma_{reg} + \rho_p \cdot \sum_{j=1}^{m} \left(\Theta^{(S_j)} - \Theta_{min}^{(S_j)} \right) \tag{4.102}$$

So, the total required queue size (i.e. Q^{total}) is bounded by

$$Q^{total} \leq Q^{regulator} + Q^{schedulers} + Q^{consumer} \tag{4.103}$$

As has become clear above, our performance analysis method can be used for direct communication between two subsystems.

4.7.2 Indirect communication between subsystems

In the second type of communication, data is stored in a memory system. Therefore, data is sent from one subsystem via the interconnect to a memory system. There, the data is temporarily stored. After that, a subsystem can request for data stored in the memory system. Then, the data is sent from the memory system via the interconnect to that subsystem. Please note that this can be the same subsystem, i.e. the subsystem that has stored the data in the memory system. The first part of the communication is called the "store" and the second part the "load". These data transfers are discussed in the section below.

Store

The first part of indirect communication is the store communication. One subsystem (the producer) sends packets via the interconnect to a memory system to temporarily store this data. Our performance analysis method is used for this type of traffic.

The same assumptions are made as in the previous section, only since the DRAM controller is an LR server itself, the m^{th} LR server is the DRAM controller (see figure 4.17). As explained in section 4.4.3, a packet stretcher is placed before the multiplexer of the DRAM controller. Therefore, the latency of the DRAM controller (i.e. Θ^{mem}) is calculated by making use of the adapted packet size.

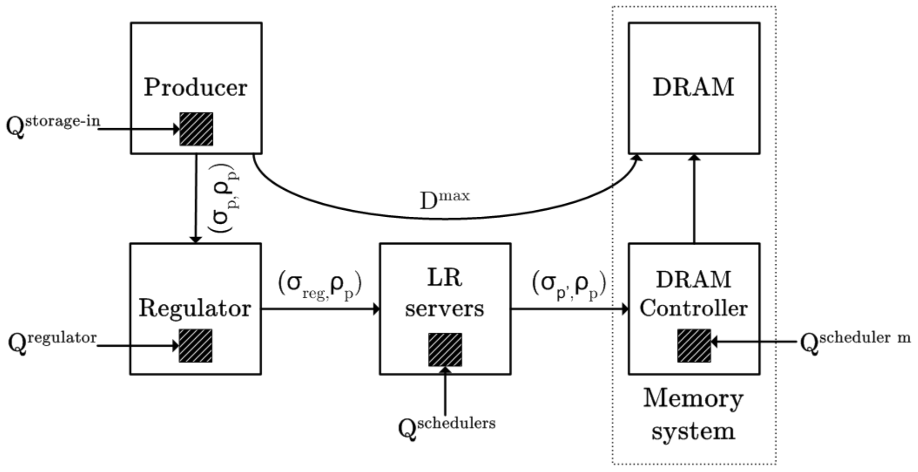


Figure 4.17: Indirect communication - store

Assume that no acknowledgment is sent back via the interconnect. Then, this type of traffic is a request-only stream. Furthermore, assume that $T_{req} \geq 0$ *sec* is required to process a store request in the DRAM. Then Θ^{mem} can be calculated by making use of a packet stretcher, as explained in section 4.4.3.

After that, the total delay and the required queue size can be calculated in the same way as in section 4.7.1. There are only two differences, namely

- $\Theta^{(S_m)} = \Theta^{mem}$. This is the case, because the m^{th} *LR* server is the DRAM controller.
- $Q^{consumer} = 0$. This is the case, because the DRAM controller does not forward the next request to the DRAM before the DRAM is available. Therefore, no queues in the DRAM are required.

So, the same formulas as in section 4.7.1 can be used.

Load

The second part of indirect communication is the load communication. One subsystem sends request packets via the interconnect to a memory system to request for data. The memory system sends response packets back via the interconnect to the subsystem. Our performance analysis method is used for this type of traffic.

Assume that a subsystem (the producer) outputs a session, characterized by (σ_p, ρ_p) and L_{req} (see figure 4.18), such that $\rho_p = \rho_{req}$. This data is sent to a regulator. The regulator outputs the session with the parameters (σ_{reg}, ρ_p), such that $\sigma_{reg} \leq \sigma_p$. Then, the data is sent via a chain of $(m-1)$ *LR* servers, with capacity C, to the memory system with a rate of ρ_p. Assume that the k^{th} *LR* server has a latency of $\Theta_{req}^{(S_k)}$ and a minimum latency of $\Theta_{min,req}^{(S_k)}$. The m^{th} *LR* server is the DRAM controller, with a latency of Θ^{mem}. A packet stretcher is used with $T_{req} \geq 0$ *sec* as processing time of the request. Then the memory system sends responses back, characterized by (σ_c, ρ_c) and L_{resp}, such that $\rho_c = \rho_{resp}$. These responses are sent to the second regulator. This regulator outputs the responses with the parameters (σ'_{reg}, ρ_c), such that $\sigma'_{reg} \leq \sigma_c$. Then, these responses are sent via a chain of m' *LR* servers, with capacity C, to the subsystem (the consumer) with a rate of ρ_c. Assume that the k^{th} *LR* server has a latency of $\Theta_{resp}^{(S_k)}$ and a minimum latency of $\Theta_{min,resp}^{(S_k)}$. Furthermore, assume that n is the pipeline degree for both chains together. Finally, assume that responses on the x *words* of requests have to be at the consumer side within $D^{max} \geq 0$ *sec*, which needs to be determined by complementary methods.

Please note that σ_p is according to formula 4.26 bounded by

$$\sigma_p \leq n \cdot L_{req} \cdot \left(1 - \frac{\rho_{req}}{C}\right) \tag{4.104}$$

Then, σ_c is bounded by

$$\sigma_c \leq n \cdot L_{resp} \cdot \left(1 - \frac{\rho_{resp}}{C}\right) \tag{4.105}$$

Furthermore, the request and the response session are related. The number of request packets per time unit is equal to the number of response packets per time unit. This is the case, because for each request exactly one response is generated. Then,

$$\rho_{resp} = \frac{L_{resp}}{L_{req}} \cdot \rho_{req} \tag{4.106}$$

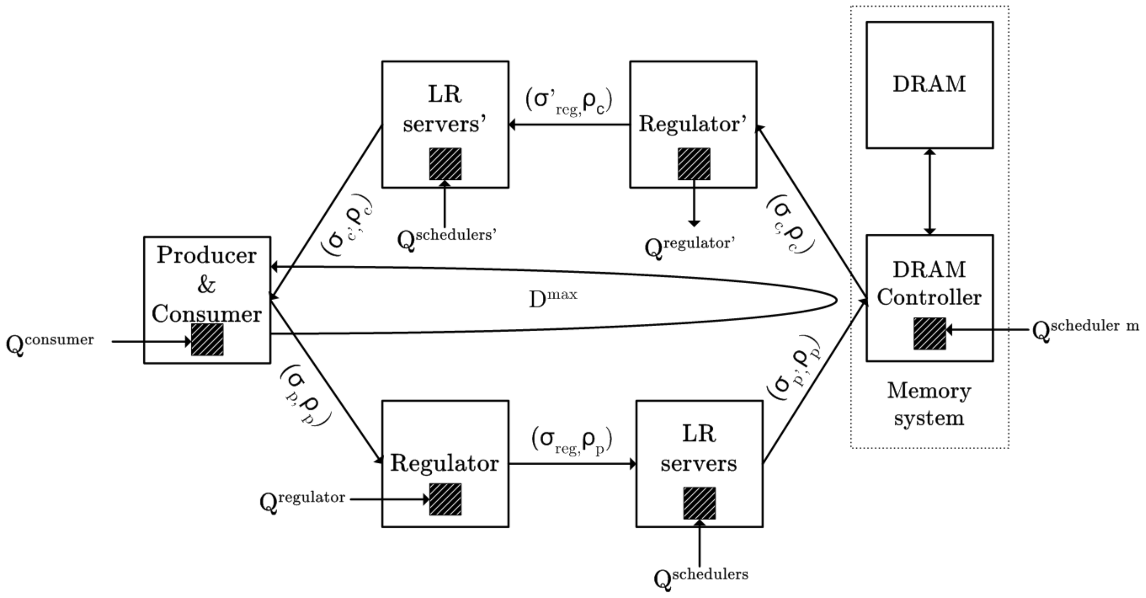


Figure 4.18: Indirect communication - load

Please note that this type of traffic is a request-response stream, so formula 4.43 can be used to calculate the total delay for *x words* of requests and responses to the requests.

$$\begin{aligned} D^{total} \leq\ & \left\lceil \frac{x}{n \cdot L_{req}} \right\rceil \cdot \left(\frac{L_{req}}{C} + \sum_{j=1}^{m} \Theta_{req}^{(S_j)} + D_{proc} + \sum_{j=1}^{m'} \Theta_{resp}^{(S_j)} + \frac{L_{resp}}{C} \right) \\ & + \left(\left\lceil \frac{x}{L_{req}} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L_{req}} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L_{req}}{\rho_{req}} \end{aligned} \tag{4.107}$$

$D_{proc} = 0$ because by making use of a packet stretcher, enough time is reserved by the DRAM controller to process a request by the DRAM, before the next request to the DRAM is sent. So,

$$\begin{aligned} D^{total} \leq\ & \left\lceil \frac{x}{n \cdot L_{req}} \right\rceil \cdot \left(\frac{L_{req}}{C} + \sum_{k=1}^{m} \Theta_{req}^{(S_k)} + \sum_{k=1}^{m'} \Theta_{resp}^{(S_k)} + \frac{L_{resp}}{C} \right) \\ & + \left(\left\lceil \frac{x}{L_{req}} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L_{req}} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L_{req}}{\rho_{req}} \end{aligned} \tag{4.108}$$

with $\Theta_{req}^{(S_m)} = \Theta^{mem}$.

Then the required queue size can be calculated. The required queue size of the first regulator and the first chain of *LR* servers are (see formulas 4.74 and 4.96)

$$Q^{regulator} \leq \sigma_p - \sigma_{reg} \tag{4.109}$$

$$Q^{schedulers} \leq m \cdot \sigma_{reg} + \rho_{req} \cdot \sum_{k=1}^{m} \left(\sum_{j=1}^{k-1} \left(\Theta_{req}^{(S_j)} - \Theta_{min,req}^{(S_j)} \right) + \Theta_{req}^{(S_k)} \right) \tag{4.110}$$

with $\Theta_{req}^{(S_m)} = \Theta^{mem}$ for the same reason as above.

Furthermore, the required queue size of the second regulator and the second chain of *LR* servers can be calculated by the same formulas.

$$Q^{regulator'} \leq \sigma_c - \sigma'_{reg} \tag{4.111}$$

$$Q^{schedulers'} \leq m' \cdot \sigma'_{reg} + \rho_{resp} \cdot \sum_{k=1}^{m'} \left(\sum_{j=1}^{k-1} \left(\Theta_{resp}^{(S_j)} - \Theta_{min,resp}^{(S_j)} \right) + \Theta_{resp}^{(S_k)} \right) \tag{4.112}$$

The subsystem (the producer & consumer) is not always ready to process the received responses. Therefore, the subsystem needs a queue (i.e. $Q^{consumer}$) to temporarily store the bursts of the input session of the consumer. The size of this queue is determined by the burstiness constraint of the input session of the consumer (i.e. $\sigma_{c'}$). According to formula 4.93, $\sigma_{c'}$ is upper bounded by

$$\sigma_{c'} \leq \sigma'_{reg} + \rho_{resp} \cdot \sum_{j=1}^{m'} \left(\Theta_{resp}^{(S_j)} - \Theta_{min,resp}^{(S_j)} \right) \tag{4.113}$$

The consumer has to be able to store the maximum burst of its input session. Therefore,

$$Q^{consumer} \leq \sigma'_{reg} + \rho_{resp} \cdot \sum_{j=1}^{m'} \left(\Theta_{resp}^{(S_j)} - \Theta_{min,resp}^{(S_j)} \right) \tag{4.114}$$

So, the total required queue size (i.e. Q^{total}) is bounded by

$$Q^{total} \leq Q^{regulator} + Q^{schedulers} + Q^{regulator'} + Q^{schedulers'} + Q^{consumer} \tag{4.115}$$

We have derived the formulas to determine the total delay and the total required queue size for accessing data in a memory system. Therefore, our performance analysis method can be used for direct communication as well as for indirect communication between two subsystems.

In the next chapters, we will use these formulas for the case studies.

Case study: Video Playback

Contents

5.1 Introduction

In [13], a Video Playback case study has been defined. In this chapter, we use this case study to compare the arbitration policies described in sections 3.3.5 and 4.5.

5.2 Description

The Video Playback case study is defined as follows. There are four processing elements (see figure 5.1). All communication between the processing elements goes via the shared DRAM memory system. The ARM reads a multimedia bitstream from a flash memory via a dedicated link (not shown), demultiplexes it and writes the video part into the DRAM. The TriMedia runs a video decoder. It reads the video part of the multimedia bitstream from the DRAM, decodes it and writes the raw video data back into the DRAM. Then, the Scaler reads the raw video data from the DRAM, scales it and writes the video data back into the DRAM. Finally, the Display Controller reads the video data from the DRAM and sends it to the display via a dedicated link (not shown) [13].

Please note that a refresh session is required to model the refresh of the DRAM. Furthermore, the responses of the DRAM are sent via a shared bus. No arbitration is required for this bus, because the DRAM controller is the only master.

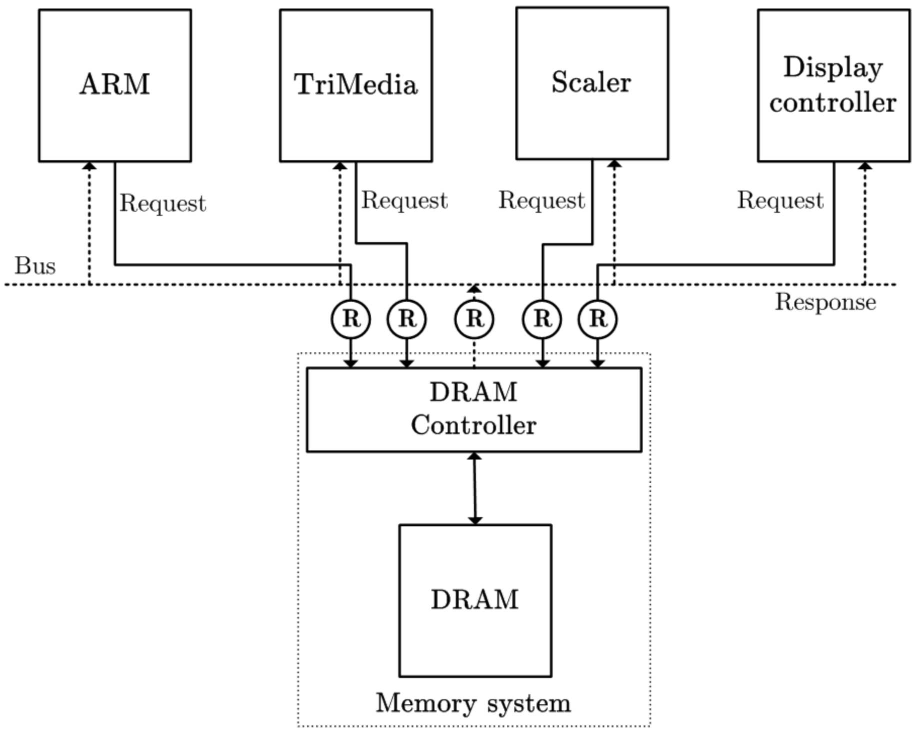


Figure 5.1: Block diagram of Video Playback case study [13]

A request denotes either the address information for the DRAM for reading data or the address and data information for writing data. A response denotes the data read from the DRAM. Please note that no acknowledgment is sent for a write request, because an error-free system is assumed.

On the inputs and the outputs of the DRAM controller, regulators (i.e. "R") are placed to shape the traffic, if necessary.

5.3 Traffic characteristics

In [13], the traffic characteristics are determined. For the ARM and the TriMedia, the characteristics are estimated based on the clock frequency, the cache miss rates and the basic characteristics of bitstream demultiplexing and video decoding. The characteristics of the traffic from the scaler and the display controller are determined based on the frame rate, the resolution and the pixel formats. The traffic characteristics are determined in [13] as listed in table 5.1. We assume a packet size of 8 *bytes* for a memory read request.

Session	Type	Max burst length ($\#packets$)	Rate ($\frac{\#packets}{msec}$)	L_{req} (B)	L_{resp} (B)	T_{req} (cc)
1	Read (ARM)	4	190	8	32	10
2	Write (ARM)	2	31.3	32	-	13
3	Read (TriMedia)	4	320	8	128	22
4	Write (TriMedia)	18.4	243	128	-	25
5	Read (Scaler)	1	243	8	128	22
6	Write (Scaler)	1	750	128	-	25
7	Read (DC)	1	750	8	128	22
8	Refresh	1	128	8	-	10

Table 5.1: Traffic characteristics of the Video Playback case study

In table 5.1, the columns denote:

- Number of the session
- Type of the session
- Maximum number of packets in one burst
- Rate of the session, expressed in number of request or response packets per *msec*
- The size of a request packet
- The size of a response packet
- The worst case required processing time in the DRAM, expressed in clock cycli

Please note that for reading from the DRAM, first a read request is sent to the DRAM controller (of length L_{req}). After that, the DRAM responses by sending the requested data (of length L_{resp}).

Assume that the frequency of the DRAM is 100 MHz with a bus width of 8 *bytes*. Then, the maximum rate of the DRAM controller is

$$C = 100 \cdot 8 = 800\ MBps \tag{5.1}$$

Please note that this maximum rate is the raw capacity. Because of all the overhead in the DRAM, this rate is never reached.

Using the traffic characteristics of table 5.1, the session characteristics for our performance analysis method are determined. The session characteristics are calculated using (see sections 4.2.3 and 4.4.3)

$$\begin{aligned} \rho_i &= rate_i \cdot L_i & (5.2)\\ \sigma_i &= (max\ burst\ length)_i \cdot L_i \cdot \left(1 - \frac{\rho_i}{C}\right) & (5.3)\\ L'_i &= T_{req} \cdot bus\ width & (5.4) \end{aligned}$$

Please note that in our performance analysis method, there is a distinction between a read request and a read response. Furthermore, L' represents the length of the output packets of the packet stretcher in the model of the DRAM controller (see section 4.4.3).

Table 5.2 gives the session characteristics for the performance analysis method.

Session	Type	σ (B)	ρ $(\frac{MB}{s})$	L (B)	L' (B)
1a	Read request (ARM)	31.9	1.52	8	80
1b	Read response (ARM)	127	6.08	32	-
2	Write (ARM)	63.9	1.00	32	104
3a	Read request (TriMedia)	31.9	2.56	8	176
3b	Read response (TriMedia)	486	41.0	128	-
4	Write (TriMedia)	2264	31.1	128	200
5a	Read request (Scaler)	7.98	1.94	8	176
5b	Read response (Scaler)	123	31.1	128	-
6	Write (Scaler)	113	96.0	128	200
7a	Read request (DC)	7.94	6.00	8	176
7b	Read response (DC)	113	96.0	128	-
8	Refresh	7.99	1.02	8	80

Table 5.2: Session characteristics of the Video Playback case study

After characterizing the sessions, the performance analysis method is performed. Therefore, the SoC communication infrastructure as indicated in figure 5.1 is used. The DRAM controller is modeled as explained in section 4.4.3. Please note that for each session a packet stretcher is used in the DRAM controller to calculate the latency of the DRAM memory system.

5.4 Mathematica

To perform the performance analysis method, we have created a Mathematica model. In this model, we use the formulas derived in section 4.7 to determine the upper bounds for the delay of the sessions and queue size. Mathematica is used, because of its capability to calculate with vectors. The bounds for a single session can be determined by the same set of formulas as for multiple sessions. Therefore, we have used Mathematica, instead of a standard programming language. More details can be found in appendix C.

5.5 Results

Using the above characteristics, the delays are calculated using different schedulers.

For each session, the delay is calculated for only the first packet. The following arbitration policies are used (see table 5.3).

- Virtual Clock (VC) (see section 3.3.5)
- Deficit Round Robin (DRR) (see section 3.3.5)
- Weighted Round Robin (WRR) (see section 3.3.5, using $L_c = 200\ B$)
- Round Robin Time Based (RRTB) (see section 4.5)
- Round Robin Packet Based (RRPB) (see section 4.5)
- TDMA (see section 4.5), such that in each round one packet per session is serviced

- Fixed Priority (FP) (see section 4.5), using the following order of priority (first one has highest priority)
  - Write (ARM)
  - Refresh
  - Read (ARM)
  - Read (Scaler)
  - Write (TriMedia)
  - Read (TriMedia)
  - Read (DC)
  - Write (Scaler)

Please note that all used arbitration policies are non-preemptive.

Session	**VC**	**DRR**	**WRR**	**RRTB**	**RRPB**	**TDMA**	**FP**
Read (ARM)	5.56	103	34.2	1.90	1.54	1.54	0.64
Write (ARM)	32.3	105	34.2	1.92	1.53	1.53	0.42
Read (TM)	3.55	97.3	31.3	2.14	1.66	1.66	1.59
Write (TM)	4.53	98.5	31.9	2.16	1.65	1.65	1.27
Read (Scaler)	4.54	99.4	32.4	2.14	1.66	1.66	0.99
Write (Scaler)	1.74	82.9	24.1	2.16	1.65	1.65	2.70
Read (DC)	1.75	85.6	25.5	2.14	1.66	1.66	1.96
Refresh	8.08	104	34.6	1.86	1.50	1.50	0.49

Table 5.3: Delay in μsec of first packet of the sessions

The total required queue size of the regulators and of the scheduler in the DRAM controller are calculated and shown in table 5.4.

Policy	Q^{total} (B)
VC	3352
DRR	15309
WRR	6695
RRTB	3270
RRPB	3199
TDMA	3199
FP	3285

Table 5.4: Total required queue size

For the above results, the following conclusions can be drawn.

First of all, there is a big difference in delay among the arbitration policies.

Virtual Clock is much better than Deficit Round Robin and Weighted Round Robin in this example, although the difference with TDMA, RRTB, RRPB and Fixed Priority is significant.

Deficit Round Robin and Weighted Round Robin are very poor compared to the other arbitration policies. The problem of these arbitration policies is the way the latency is calculated. Deficit Round Robin makes use of the ratio between ρ_i and ρ_{min} (see section 3.3.5). Furthermore, the smallest slot has a size equal to the largest packet (i.e. 200 B). Therefore, too much time is reserved for some sessions per round. Weighted Round Robin also makes use of the largest packet to determine the size of the slots. For sessions with small packets (like Read (ARM)) too much time is reserved per round.

Round Robin Time Based has a very good delay performance in this case study compared to the other arbitration policies. But, each session gets the same amount of service per round. Therefore, still too much time is reserved for the sessions with small packets.

Round Robin Packet Based and TDMA even have a better delay performance than Round Robin Time Based. Because at most one packet per session is serviced during a round, these arbitration policies have a small delay for the first packet. Please note that TDMA is a generalization of Round Robin Packet Based, but in TDMA unused slots are not skipped. In the worst case scenario, all slots are used. Because in this case study in each round one packet is served per session for TDMA, the results of TDMA and RRPB are identical.

Finally, Fixed Priority has the best delay performance in this case study. Unfortunately, the spread in delays for different sessions is much bigger, compared to TDMA.

More or less the same remarks are applicable for the buffer requirements, although the differences among the arbitration policies are not very large. Again, Round Robin Time Based, TDMA and Fixed Priority require the less buffers in this case study.

So, TDMA and Fixed Priority are the best arbitration policies in this case study, if we look to the delay bound and the buffer requirements.

In the next chapter, a larger case study is discussed.

Case study: Multi-channel DVB-T set-top box

Contents

6.1 Introduction

This chapter reports about a case study we used to assess the usefulness of our performance analysis method. Therefore, a sufficiently diverse and complex case study is required. We have selected a multi-channel DVB-T set-top box case study with several different traffic characteristics. First, we explain the case study. After that, we apply the performance analysis method to the case study, to obtain results. Finally, we present these results, to indicate the usefulness of our performance analysis method.

6.2 Description

DVB-T is the abbreviation of "Digital Video Broadcasting - Terrestrial". With this technology, digital video signals are broadcast through the air. Since DVB-T proceeds largely in the digital domain, error correction is possible. If the signal is strong enough, the quality of the video can be good. Another advantage is that less energy is needed, compared to analog broadcasting [2].

In our case study, the signals are received by an antenna (see picture 6.1). RF is the "Radio Frequency Front-End", which includes the tuner to select the data channel. The resulting analog signal is transformed into a digital signal. This digital signal is sent to the DVB-T Channel Decoders (DVB-T CDs).

The digital signal is encoded. The functionality of the DVB-T CDs is to decode this signal. Therefore, $OFDM$-symbols[1] are used. $OFDM$-symbols are blocks of data of 20 kB, used during the decoding. After decoding the signal, the DVB-T CD also performs error correction.

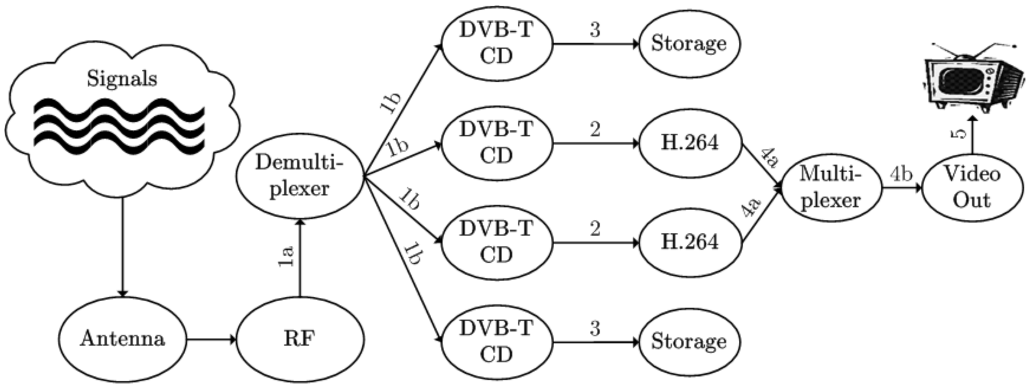


Figure 6.1: Multi-channel DVB-T set-top box functional overview, the numbers indicate the link numbers

In this case study, four different data channels can be distilled from the input signals. So, the signals of the RF are demultiplexed and divided among four DVB-T CDs. Each DVB-T CD needs memory for the decoding of the channel. The four output streams of the DVB-T CDs are MPEG-2 multiplex transport streams. An MPEG-2 multiplex transport stream is a combination of audio and video. Audio is outside the scope of this thesis. Video can be an H.264 bitstream. Two of these streams are sent to Storage. The other two signals are sent to H.264 decoders. The latter two signals are decoded, such that they can be displayed. These signals can be used for dual screen or dual window functionality.

1. Orthogonal Frequency-Division Multiplexing symbols

Please note that for the video decoding, memory is required because of the spatial and temporal dependencies among the different frames of the video stream.

The implementation architecture and the mapping of the tasks to subsystems is according to figure 6.2. As you can see, there is one central (off-chip) memory system. Small buffers can be located on-chip in each subsystem. Furthermore, the four DVB-T CDs are combined into one subsystem. The two H.264 decoders are sharing a single subsystem. The Video Out system is responsible for displaying the union of the two output streams of the 2xH.264 decoder on the display(s). Finally, the traffic of all connections of figure 6.1 is implemented by a single interconnect. Details of the interconnect are mentioned later.

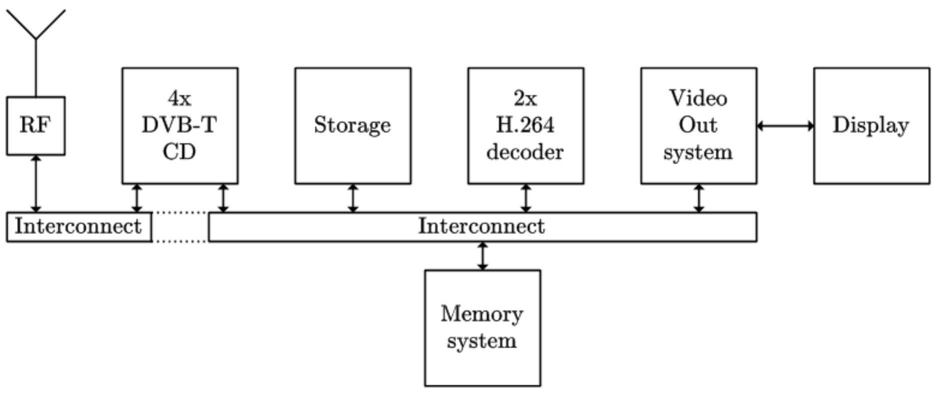


Figure 6.2: Multi-channel DVB-T set-top box block diagram

Now the case study has been defined, the traffic among the components can be defined.

6.3 Traffic

In this section, the traffic between the different components is defined. This is done in the order indicated by the numbering of figure 6.1.

6.3.1 From RF to the 4xDVB-T CD

See the links 1a and 1b of figure 6.1.

The digital version of the signals received by the antenna are sent from the RF to the 4xDVB-T CD. For each channel 20 $MBps$ is needed [6]. Hence, for four channels, 80 $MBps$ in total is required. The delay is not really important, as long as the rate is high enough. Therefore, the main constraint is related to the rate of the link.

The DVB-T CD performs the decoding. Therefore, a control link between the DVB-T CD and the RF is needed to adjust the settings of the RF. To reduce the complexity of this case study a bit, the traffic between the RF and the 4xDVB-T CD is left out of this case study.

6.3.2 Between the 4xDVB-T CD and the Memory System

The four Channel Decoders are implemented on a single processor. This means that the shared processor divides its time among the four Channel Decoders. Time is divided in time slots of $OFDM$-periods[2]. Then, each $OFDM$-period is split into four equal parts. In each $\frac{1}{4}$ $OFDM$-period, one Channel Decoder is active.

To perform the decoding, a Channel Decoder needs $OFDM$-symbols. Each $OFDM$-symbol has a size of 20 kB. For each Channel Decoder, the four most recent $OFDM$-symbols have to be stored, but not all symbols are used all the time. Because the $OFDM$-symbols are large, it is cheaper to store the four $OFDM$-symbols per Channel Decoder in an off-chip memory system, than in an on-chip memory system. When a symbol needs to be processed, it is copied to the local on-chip memory. Each $\frac{1}{4}$ $OFDM$-period only two $OFDM$-symbols are required. Furthermore, in each $\frac{1}{4}$ $OFDM$-period one $OFDM$-symbol is produced. This means that in each $\frac{1}{4}$ $OFDM$-period two reads from and one write to the off-chip memory have to be performed.

The flow graph of one DVB-T Channel Decoder is shown in figure 6.3. Because we are only interested in the communication behavior, only a part of the complete flow graph is shown. Each circle denotes an operation. The arrows indicate the data dependencies. Furthermore, a “8192” represents a buffer for an $OFDM$-symbol.

Please note that the three dotted squares indicate the same buffers. This means that in the implementation, these buffers will be shared. A corresponding number indicates the same buffer.

Number 0 is the output of the FFT (i.e. Fast Fourier Transform) (see figure 6.3), using the digital signal of the RF. The FFT will take 13,540 cycles to perform an $8K$ FFT. So, it needs 45.1 μsec to produce an $OFDM$-symbol, if a 300 MHz EVP[3] is used.

Furthermore, number 0 is used by (see figure 6.3)

- the *Mult* of the *Scat sync* (i.e. A)
- the *Mult* of the $Coarse\ freq$ (i.e. C)
- the *Continual pilots extraction* of the $Fine\ freq$ & *Sampling* (i.e. B)
- the *Scat pilots extraction* of the $Demod\ +\ eq$ (i.e. D)

Number 1 is only used by the *Mult* of the $Coarse\ freq$ (i.e. C).

Number 2 is not used this $\frac{1}{4}$ $OFDM$-period.

Number 3 is used by

- the *Mult* of the *Scat sync* (i.e. A)
- the *Equalize* of the $Demod\ +\ eq$ (i.e. D)

The two *Mult*'s perform $8K$ multiplications. With an EVP that can do $\frac{16\ multiplications}{cycle}$, for a *Mult* 512 *cycles* are required. So, it takes 1.7 μsec to execute each *Mult*. The *Continual pilots extraction* and the *Scat pilots extraction* produce output instantaneously. Finally, the *Equalize* needs 26 μsec per $OFDM$-symbol, if the same EVP is used.

The $OFDM$-symbols are stored in a cyclic off-chip buffer (see figure 6.4). First, number 1 and 3 of a specific Channel Decoder, stored in the off-chip memory, are needed for the coarse frequency, scattered synchronization, fine frequency and sampling, demodulation and the equalization. Therefore, the two read request address registers (i.e. d_1 and d_2, see figure 6.4) are filled with addresses. The requests are sent to the memory system. The memory system responses by sending the requested data back. The requested data is temporarily stored in a_1 and b_1 of the on-chip memory. After that, all tasks of the

2. An $OFDM$-period takes 896 μsec
3. An EVP is an Embedded Vector Processor developed by NXP and specialized to operate on vectors

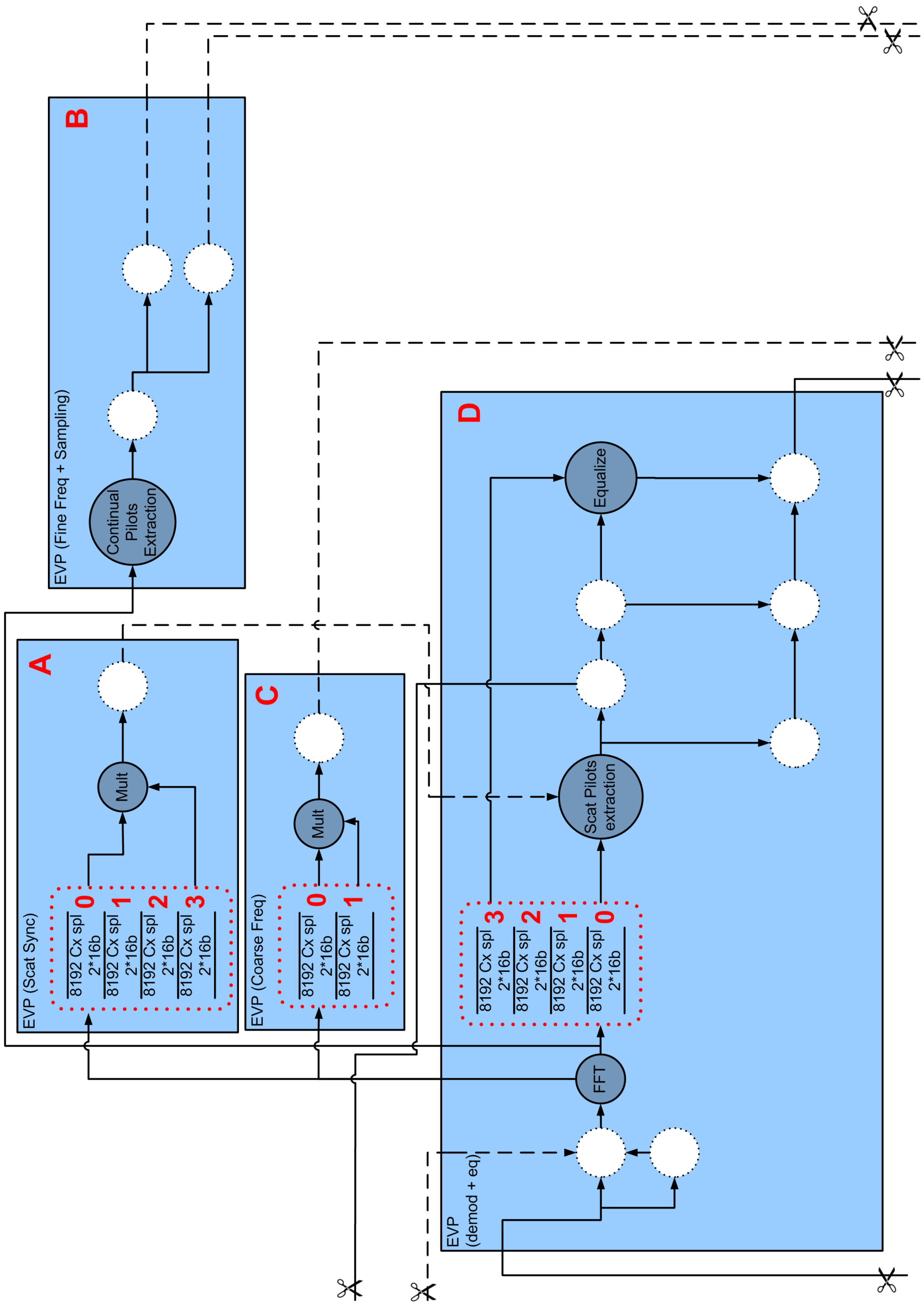


Figure 6.3: Part of the DVB-T CD Flow Graph

Channel Decoder are performed. Then, buffer number 0 of the off-chip memory has to be replaced by the output of the FFT. This data is first temporarily stored in c_1 of the on-chip memory. Therefore, a write action transports this data to the off-chip memory (to number 0 of the selected Channel Decoder).

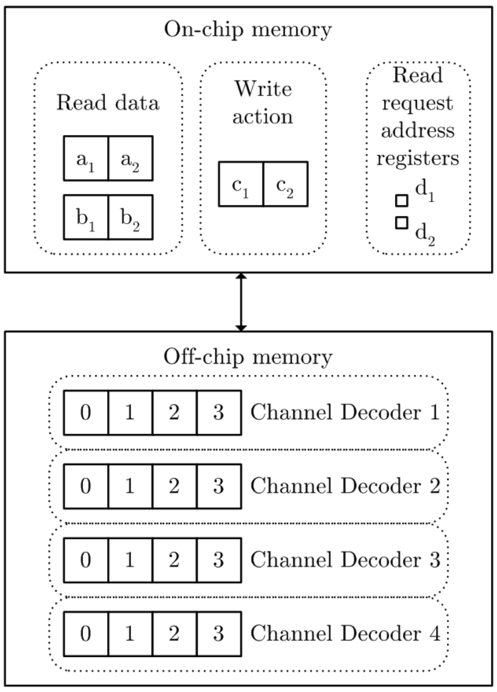

On-chip memory
Read data
a1
a2
b1
b2
Write action
c1
c2
Read request address registers
d1
d2
Off-chip memory
0 1 2 3 Channel Decoder 1
0 1 2 3 Channel Decoder 2
0 1 2 3 Channel Decoder 3
0 1 2 3 Channel Decoder 4

Figure 6.4: Memory for decoding, a square represents a buffer

Then, the next Channel Decoder gets active. This means that its required data (i.e. its number 1 and 3) is loaded to the on-chip memory. And again, at the end of his period its number 0 is replaced.

After one round, all $OFDM$-symbols are shifted one place, such that number 0 (just produced) becomes number 1 and so on (the number represents the age, so the $OFDM$-symbols become older). Number 3 is no longer needed. Because a circular buffer is used, this happens only virtually, i.e. by moving the references rather than by physically moving the $OFDM$-symbols.

When the first Channel Decoder gets active again, number 1 and 3 stored in the off-chip memory are required. At the end of his period, number 0 has to be replaced by the output of the FFT. This process continues.

To optimize the performance, ping pong buffers are used. When a_1 and b_1 are used, a_2 and b_2 can already be filled for the next Channel Decoder. Then, the next Channel Decoder does not have to wait for its data after it gets active. The write buffers (i.e c_1 and c_2) are used in the same way. First, c_1 is filled. When the next Channel Decoder gets active, the content of c_1 is sent to the off-chip memory. At the same time, c_2 can be filled by the output of the FFT of the current Channel Decoder. This process is repeated constantly.

In the next sections, three schedule variants are described that are applicable to the traffic between the on-chip and the off-chip memory. Please note that we make use of periodic schedules in which four complete $OFDM$-symbols are processed. Therefore, the decoding takes place on complete $OFDM$-symbols, and not on parts of symbols. That means that a complete $OFDM$-symbol has to be produced, before it can be sent to the memory system.

By using periodic schedules, we do not have to synchronize between tasks. The schedules guarantee when the data is available.

Schedule 1

In the first schedule variant, the read requests for the two $OFDM$-symbols are sent since the beginning of the previous $\frac{1}{4}$ $OFDM$-period (see the “S”’s in figure 6.5). The deadline (i.e. “D”) is just before the Channel Decoder gets active. So, for Channel Decoder 2, the requests are sent when Channel Decoder 1 is active. The read operations have a time span of a $\frac{1}{4}$ $OFDM$-period to be processed. This means that when Channel Decoder 2 gets active, the requested data has to be available. During the period that Channel Decoder 2 is active, the received data is used (i.e. “Use”) and the data for the next Channel Decoder is read. When the next Channel Decoder gets active, this requested data has to be available.

When Channel Decoder 1 gets active, the read request address registers (i.e. d_1 and d_2 of figure 6.4) are filled with the addresses of the $OFDM$-symbols needed for Channel Decoder 2. Then, a_1 and b_1 are filled with the requested data. When Channel Decoder 2 gets active, a_1 and b_1 should contain $OFDM$-symbol 1 and 3, such that they can be used. Meanwhile, d_1 and d_2 can be filled with the addresses of $OFDM$-symbol 1 and 3 for the next Channel Decoder, to be loaded into a_2 and b_2.

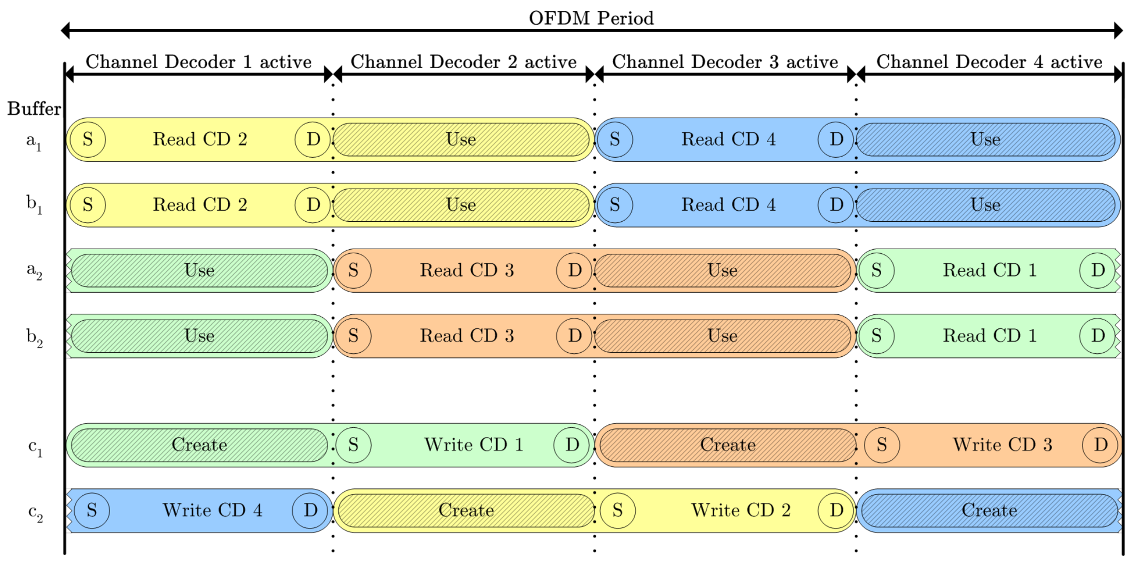


Figure 6.5: Repeating pattern of schedule 1, “S” represents sent read or write request, “D” represents deadline of request, the read buffers are used during “Use” and the write buffers are filled during “Create”

The write action is processed when the particular Channel Decoder gets inactive. So, the data produced when Channel Decoder 2 is active, begins to send to the off-chip memory when Channel Decoder 2 gets inactive. Again, there is a deadline after $\frac{1}{4}$ $OFDM$-period. During the period the Channel Decoder is active, the data is created (i.e. “Create”). This means that for Channel Decoder 2, buffer c_2 is filled with $OFDM$-symbol 0 by the FFT when Channel Decoder 2 is active. When Channel Decoder 3 gets active, the content of c_2 is sent to the off-chip memory system. This has to be completed, before Channel Decoder 4 gets active. Meanwhile, c_1 is used by Channel Decoder 3 for its output.

So, the number of buffers needed is

- 2 registers for the read request addresses (as you can see in figure 6.5, these registers can be reused)
- 4 buffers for the read data (there are 2 read actions per $\frac{1}{4}$ $OFDM$-period, but two buffers are already in use when the read actions are processed)
- 2 buffers for the write action (there is 1 write action per $\frac{1}{4}$ $OFDM$-period, but one buffer is already filled when the write action is processed)

Looking to the amount of time, one $OFDM$-period takes by definition 896 μsec. This means that within 224 μsec, one Channel Decoder has to do all its work. All the blocks of figure 6.3 are performed by one EVP. So, this means that the $Mult$ of the $Scat\ sync$ (i.e. 1.7 μsec), the $Mult$ of the $Coarse\ freq$ (i.e. 1.7 μsec), the FFT (i.e. 45.1 μsec) and the $Equalize$ of the $Demod + eq$ (i.e. 26 μsec) will need $2 \cdot 1.7 + 45.1 + 26 = 74.5$ μsec. Then, $224 - 74.5 = 149.5$ μsec are left for the rest of the flow graph. This is sufficient, because the $Mult$'s and the FFT are the most time consuming tasks of the Channel Decoder.

Schedule 2

In the second schedule variant, the period a Channel Decoder is active is split into two parts (see figure 6.6). In the first part, the data for the write action is generated. This means that in this part, the FFT has to produce its output. In the second part, the requested data is used. Now, more buffers can be reused than in the previous variant.

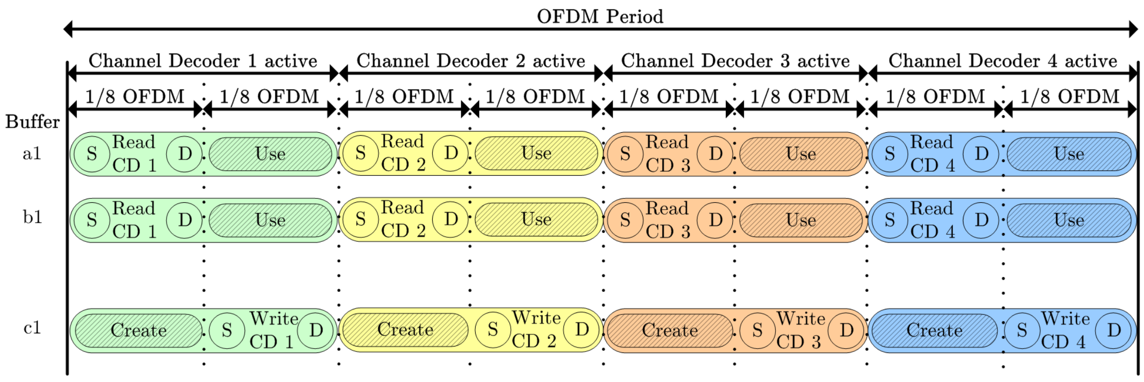


Figure 6.6: Repeating pattern of schedule 2, "S" represents sent read or write request, "D" represents deadline of request, the read buffers are used during "Use" and the write buffer is filled during "Create"

At the beginning of the period a Channel Decoder is active, its read requests are sent. So, for channel 1, the requests are sent at the beginning of the period of Channel Decoder 1. The read actions have a deadline of $\frac{1}{8}$ $OFDM$-period. Furthermore, the requested data is only used during the second half the particular Channel Decoder is active.

The write action is executed halfway the period, a Channel Decoder is active. A $\frac{1}{8}$ $OFDM$-period is reserved to generate the data to be sent. The deadline is a $\frac{1}{8}$ $OFDM$-period later (see figure 6.6).

To be more precise, when Channel Decoder 1 gets active, d_1 and d_2 (see figure 6.4) are filled with addresses. Then, $\frac{1}{8}$ $OFDM$-period later, a_1 and b_1 should be filled with the requested data (i.e. $OFDM$-symbol 1 and 3). In the meanwhile, FFT produces its outputs (i.e. $OFDM$-symbol 0) and stores it at c_1. FFT only needs 45.1 μsec, so that should not be a problem. After $\frac{1}{8}$ $OFDM$-period, the read buffers are filled. So, $OFDM$-symbol number 0, 1 and 3 are available for Channel Decoder 1. Because the $Mult$'s and the $Equalize$ need less time than $\frac{1}{8}$ $OFDM$-period, it should be possible to schedule everything.

When Channel Decoder 2 gets active, a_1 and b_1 are no longer used by Channel Decoder 1. This means that these buffers can be reused by Channel Decoder 2. The same holds for c_1. The content of c_1 should have been sent to the memory system within $\frac{1}{8}$ $OFDM$-period. Then, c_1 can be reused by the next Channel Decoder.

Using this schedule, the number of buffers needed is

- 2 registers for the read requests addresses (see figure 6.6)
- 2 buffers for the read data, because these can be reused
- 1 buffer for the write action, because it can be reused

Schedule 3

In the last schedule variant, there is more time reserved to execute the read and the write actions. Again, the period a Channel Decoder is active is split into two parts. In the second part, the requested data is used (see figure 6.7). The read requests are sent $\frac{3}{8}$ $OFDM$-period before the data is required.

The data for the write action is determined in the first half of the period a specific Channel Decoder is active. It starts sending halfway the period the Channel Decoder is active. Then, it has $\frac{3}{8}$ $OFDM$-period to perform the write action.

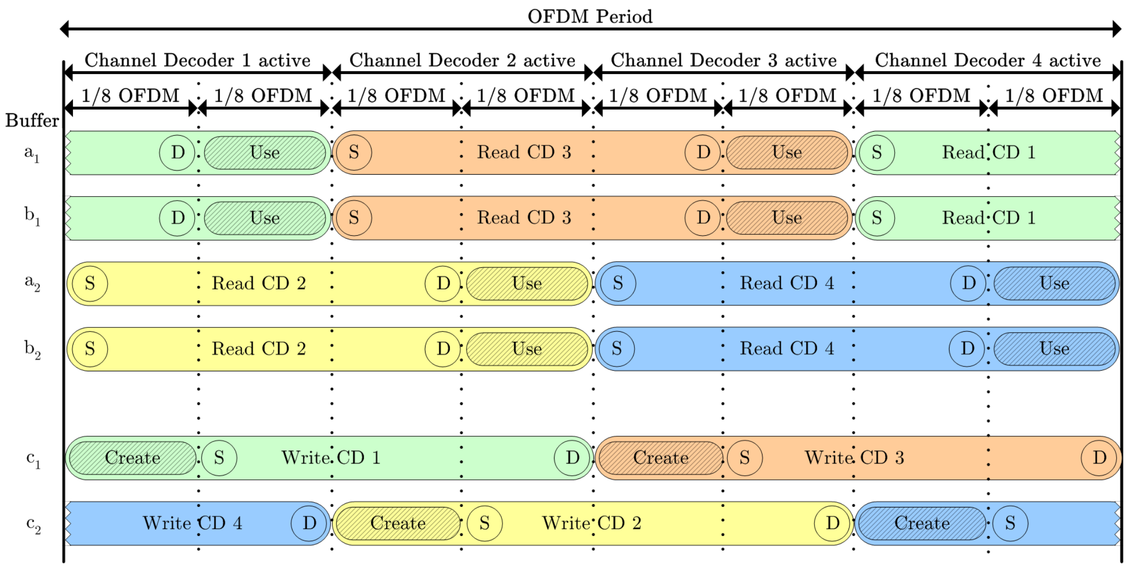


Figure 6.7: Repeating pattern of variant 3, "S" represents sent read or write request, "D" represents deadline of request, the read buffers are used during "Use" and the write buffers are filled during "Create"

Using this schedule, the number of buffers needed is

- 4 registers for the read request addresses (see figure 6.7)
- 4 buffers for the read data
- 2 buffers for the write action

Schedule 3 has longer deadlines than schedule 1, although the buffer requirements are almost the same.

By making use of our performance analysis method, the results of the performance analysis method should show which delays can worst case be expected, and hence which schedule should be used. Please note that the schedules have different deadline constraints for the read and write actions.

Traffic characterization

Now, we characterize the traffic.

A read request contains the address information for the off-chip memory to read data. The off-chip memory responses by sending the requested data back.

Let b_{1a} (i.e. $b_{1a} \sim (\sigma_{1a}, \rho_{1a})$ with pipeline degree n_1) represent the read requests of the $OFDM$-symbols, b_{1b} the responses on b_{1a}, and b_2 represent the write actions.

We assume that all read request packets have a size of 8 *bytes*, and all read responses and write actions have a size of 128 *bytes*. Furthermore, we assume that the address information for a write action is sent via separate wires. Finally, for each read request, one read response is sent back by the memory system.

Please remember that an $OFDM$-period takes 896 μsec.

Then, for the first schedule we have (see section 4.2.3 and equation 4.26)

$$\begin{aligned}
\rho_{1a} &\geq requests\ for\ 2\ streams\ of\ 20\ kB\ per\ \frac{1}{4}\ OFDM\ period && (6.1)\\
&= requests\ for\ 313\ responses\ of\ 128\ B\ per\ \frac{1}{4}\ OFDM\ period && (6.2)\\
&= 313\ requests\ of\ 8\ B\ per\ \frac{1}{4}\ OFDM\ period && (6.3)\\
&= 11.2\ MBps && (6.4)\\
\sigma_{1a} &= n_1 \cdot 8 \cdot \left(1 - \frac{\rho_{1a}}{C}\right) && (6.5)
\end{aligned}$$

So, this means that 2 times 20 kB has to be read within $\frac{1}{4}$ $OFDM$-period. 2 times 20 kB is equivalent to 313 requests of 128 *bytes*. Because each read request has a size of 8 *bytes*, this session requires at least 11.2 $MBps$.

$$\begin{aligned}
\rho_{1b} &\geq 313\ responses\ of\ 128\ B\ per\ \frac{1}{4}\ OFDM\ period && (6.6)\\
&= 179\ MBps && (6.7)\\
\sigma_{1b} &= n_1 \cdot 128 \cdot \left(1 - \frac{\rho_{1b}}{C}\right) && (6.8)\\
\rho_2 &\geq write\ actions\ for\ 1\ stream\ of\ 20\ kB\ per\ \frac{1}{4}\ OFDM\ period && (6.9)\\
&= 157\ write\ actions\ of\ 128\ B\ per\ \frac{1}{4}\ OFDM\ period && (6.10)\\
&= 89.7\ MBps && (6.11)\\
\sigma_2 &= n_2 \cdot 128 \cdot \left(1 - \frac{\rho_2}{C}\right) && (6.12)
\end{aligned}$$

- The deadline is $\frac{1}{4}$ $OFDM$-period for 313 requests and responses
- The deadline is $\frac{1}{4}$ $OFDM$-period for 157 write actions

For the second schedule, we will have

$$\begin{aligned}
\rho_{1a} &\geq requests\ for\ 2\ streams\ of\ 20\ kB\ per\ \frac{1}{8}\ OFDM\ period && (6.13)\\
&= requests\ for\ 313\ responses\ of\ 128\ B\ per\ \frac{1}{8}\ OFDM\ period && (6.14)\\
&= 22.4\ MBps && (6.15)\\
\rho_{1b} &\geq 358\ MBps && (6.16)\\
\rho_{2} &\geq write\ actions\ for\ 1\ stream\ of\ 20\ kB\ per\ \frac{1}{8}\ OFDM\ period && (6.17)\\
&= 179\ MBps && (6.18)
\end{aligned}$$

- The deadline is $\frac{1}{8}$ $OFDM$-period for 313 requests and responses
- The deadline is $\frac{1}{8}$ $OFDM$-period for 157 write actions

Please note that σ_{1a}, σ_{1b} and σ_2 do not change by selecting a different schedule.

For the third schedule, we have

$$\begin{aligned}
\rho_{1a} &\geq requests\ for\ 2\ streams\ of\ 20\ kB\ per\ \frac{3}{8}\ OFDM\ period && (6.19)\\
&= requests\ for\ 313\ responses\ of\ 128\ B\ per\ \frac{3}{8}\ OFDM\ period && (6.20)\\
&= 7.45\ MBps && (6.21)\\
\rho_{1b} &\geq 119\ MBps && (6.22)\\
\rho_{2} &\geq write\ actions\ for\ 1\ stream\ of\ 20\ kB\ per\ \frac{3}{8}\ OFDM\ period && (6.23)\\
&= 59.8\ MBps && (6.24)
\end{aligned}$$

- The deadline is $\frac{3}{8}$ $OFDM$-period for 313 requests and responses
- The deadline is $\frac{3}{8}$ $OFDM$-period for 157 write actions

Please note that for the third schedule, b_{1a}, b_{1b} and b_2 cannot be used by all Channel Decoders. This is the case, because the communication parts of the four Channel Decoders overlap. It is possible that more than three sessions are active simultaneously. Therefore, another three sessions have to be added with the same characteristics to model the sessions for the overlapping communication traffic. The deadlines for these sessions are the same.

6.3.3 From the 4xDVB-T CD to the 2xH.264 decoder

See link 2 of figure 6.1.

The output of a DVB-T CD is an MPEG-2 multiplex transport stream, with an H.264 video bitstream. This stream has a bitrate of 31.67 $Mbps$ per channel[4], according to [6]. Please note that because of the large amount of data to be transferred and the variations in the consumption rate of the 2xH.264 decoder, the communication takes place via the (off-chip) memory system.

4. Using 8 MHz channels and 64-QAM Modulation

Traffic characterization

Assume that b_3 represents the write actions of the Channel Decoders, b_{4a} the read requests of the H.264 decoders and b_{4b} the read responses on b_{4a}. Then,

$$\begin{aligned}
\rho_3 &\geq 31.67\ Mbps\ per\ channel && (6.25)\\
&= 3,547\ B\ per\ channel\ per\ OFDM\ period && (6.26)\\
&= 28\ write\ actions\ of\ 128\ B\ per\ channel\ per\ OFDM\ period && (6.27)\\
&= 28 \cdot 128\ B\ per\ channel\ per\ OFDM\ period && (6.28)\\
&= 4 \cdot 28 \cdot 128\ B\ per\ OFDM\ period && (6.29)\\
&= 16.0\ MBps && (6.30)\\
\sigma_3 &= n_3 \cdot 128 \cdot \left(1 - \frac{\rho_3}{C}\right) && (6.31)\\
\rho_{4a} &\geq requests\ for\ 28\ responses\ of\ 128\ B\ per\ channel\\
&\quad per\ OFDM\ period && (6.32)\\
&= 28 \cdot 8\ B\ per\ channel\ per\ OFDM\ period && (6.33)\\
&= 2 \cdot 28 \cdot 8\ B\ per\ OFDM\ period && (6.34)\\
&= 0.500\ MBps && (6.35)\\
\sigma_{4a} &= n_4 \cdot 8 \cdot \left(1 - \frac{\rho_{4a}}{C}\right) && (6.36)\\
\rho_{4b} &\geq 28 \cdot 8\ B\ per\ channel\ per\ OFDM\ period && (6.37)\\
&= 2 \cdot 28 \cdot 128\ B\ per\ OFDM\ period && (6.38)\\
&= 8.00\ MBps && (6.39)\\
\sigma_{4b} &= n_4 \cdot 128 \cdot \left(1 - \frac{\rho_{4b}}{C}\right) && (6.40)
\end{aligned}$$

- The deadline is $\frac{1}{4}$ $OFDM$-period for 28 write actions
- The deadline is $\frac{1}{2}$ $OFDM$-period for 28 read requests and responses

6.3.4 From the 4xDVB-T CD to the Storage

See link 3 of figure 6.1.

The output of a DVB-T CD is an MPEG-2 multiplex transport stream, with an H.264 video bitstream. Each MPEG-2 multiplex transport stream can be stored. In this case study, we assume that two streams are stored.

Traffic characterization

The write actions of the Channel Decoders are already characterized by b_3. Therefore, only the read requests (b_{5a}) and responses (b_{5b}) of the Storage have to be characterized.

$$\begin{aligned}
\rho_{5a} &\geq requests\ for\ 28\ requests\ of\ 128\ B\ per\ channel \\
&\quad per\ OFDM\ period && (6.41) \\
&= 28 \cdot 8\ B\ per\ channel\ per\ OFDM\ period && (6.42) \\
&= 2 \cdot 28 \cdot 8\ B\ per\ OFDM\ period && (6.43) \\
&= 0.500\ MBps && (6.44) \\
\sigma_{5a} &= n_5 \cdot 8 \cdot \left(1 - \frac{\rho_{5a}}{C}\right) && (6.45) \\
\rho_{5b} &\geq 28 \cdot 128\ B\ per\ channel\ per\ OFDM\ period && (6.46) \\
&= 2 \cdot 28 \cdot 128\ B\ per\ OFDM\ period && (6.47) \\
&= 8.00\ MBps && (6.48) \\
\sigma_{5b} &= n_5 \cdot 128 \cdot \left(1 - \frac{\rho_{5b}}{C}\right) && (6.49)
\end{aligned}$$

- The deadline is $\frac{1}{2}$ $OFDM$-period for 28 read requests and responses

6.3.5 Between the 2xH.264 decoder and the Memory System

H.264 decoding consists of several tasks. One of the tasks with high traffic requirements is motion compensation. In motion compensation, the frames are partitioned into blocks of pixels (e.g. macroblocks of 16x16 pixels). Each block is predicted from a block of the same size in a reference frame. The blocks are not transformed in any way apart from being translated to the position of the predicted block. This translation is represented by a motion vector [4].

We assume a resolution of 720x576 with decoding at a rate of 30 $\frac{frames}{sec}$ (upscaling to a higher frame rate may occur after the H.264 decoder). Then, we have to perform motion compensation for $45 \cdot 36 \cdot 30 = 48,600$ $\frac{macroblocks}{sec}$.

In YUV 4:2:0 color resolution, each macroblock contains 16x16 *bytes* Y data, 8x8 *bytes* U data, and 8x8 *bytes* V data. We assume Y data to be stored in a line-based frame memory. Furthermore, U and V data are stored together in an interleaved fashion in a line-based frame memory of size 720x288 *bytes*.

H.264 employs different block sizes. We assume that for each macroblock, we have 16 motion vectors: e.g. 8 forward predictions to 8 8x4 or 4x8 blocks and 8 backward predictions to 8 8x4 or 4x8 blocks. Furthermore, we assume that in accessing the unaligned blocks we have an overhead of a factor 4. Then, per macroblock we have to fetch $8 \cdot 8 \cdot 4 \cdot 2 \cdot 4$ *bytes* of Y data (i.e. 2048 *bytes*). UV data is half the size. For simplicity we assume the same efficiency in accessing the unaligned data. Then, we have to fetch 1024 *bytes* of UV data per macroblock.

So, the total required rate is

$$\begin{aligned}
rate &= \frac{macroblocks}{sec} \cdot \frac{bytes}{macroblok}\ per\ video\ decoder && (6.50) \\
&= 48,600 \cdot (2048 + 1024)\ Bps\ per\ video\ decoder && (6.51) \\
&= 149\ MBps\ per\ video\ decoder && (6.52)
\end{aligned}$$

Traffic characterization

Assume that b_{6a} represents the read requests of the 2xH.264 decoders and b_{6b} the read responses on b_{6a}.

$$\begin{aligned}
\rho_{6a} &\geq 149\ MBps\ per\ video\ decoder && (6.53)\\
&= 4.976\ MB\ per\ frame\ per\ video\ decoder && (6.54)\\
&= 38,880\ requests\ for\ response\ of\ 128\ B\\
&\quad per\ frame\ per\ video\ decoder && (6.55)\\
&= 38,880\ requests\ of\ 8\ B\ per\ frame\ per\ video\ decoder && (6.56)\\
&= 18.7\ MBps && (6.57)\\
\sigma_{6a} &= n_6 \cdot 8 \cdot \left(1 - \frac{\rho_{6a}}{C}\right) && (6.58)\\
\rho_{6b} &\geq 38,880\ responses\ of\ 128\ B\ per\ frame\ per\ video\ decoder && (6.59)\\
&= 299\ MBps && (6.60)\\
\sigma_{6b} &= n_6 \cdot 128 \cdot \left(1 - \frac{\rho_{6b}}{C}\right) && (6.61)
\end{aligned}$$

- The deadline is $\frac{1}{60}$ *sec* for 38,880 requests and responses

Please note that for the 2xH.264 decoders we have made the following assumptions

- Both H.264 decoders are executed by the same processor core. We assume that this is possible by a (future) TriMedia core
- We assume task switching on frame boundaries. By that, the data cache miss rate is low
- We assume that both decoders have the same code. Then, the instruction cache miss rate is low
- The upscaling from 30 Hz to 50 Hz is not modeled

6.3.6 From the 2xH.264 decoder to the Video Out system

See links 4a and 4b of figure 6.1.

To calculate the required rate, we use the following formula

$$rate = \frac{bits}{pixel} \cdot \frac{pixels}{frame} \cdot \frac{frames}{sec} \tag{6.62}$$

We assume an SD output resolution of 720x576 pixels, using a frequency of 50 Hz (progressive) and 8 bits per pixel. Therefore, we will need a rate of

$$\begin{aligned}
rate &= 8 \cdot (720 \cdot 576) \cdot 50\ bps && (6.63)\\
&= 20.7\ MBps. && (6.64)
\end{aligned}$$

This traffic is implemented in the following manner. First, the output of the 2xH.264 decoder is written to the memory system. After that, the Video Out system reads this data from the memory system.

We assume that the two streams are used for dual screen or dual window functionality. So, the two streams are interleaved. Therefore, the total required rate is $2 \cdot 20.7 = 41.5\ MBps$.

Traffic characterization

Assume that b_7 represents the write actions of the 2xH.264 decoder and b_{8b} the read responses on the read request b_{8a} of the Video Out system.

$$\begin{aligned}
\rho_7 &\geq 2\ frames\ of\ 720 \cdot 576\ B\ per\ \frac{1}{50}\ sec && (6.65)\\
&= 2 \cdot 720 \cdot 576\ B\ per\ \frac{1}{50}\ sec && (6.66)\\
&= 6,480\ write\ actions\ of\ 128\ B\ \frac{1}{50}\ sec && (6.67)\\
&= 6,480 \cdot 128 \cdot 50\ B\ per\ sec && (6.68)\\
&= 41.5\ MBps && (6.69)\\
\sigma_7 &= n_7 \cdot 128 \cdot \left(1 - \frac{\rho_7}{C}\right) && (6.70)\\
\rho_{8a} &\geq requests\ for\ 2 \cdot 720 \cdot 576\ B\ per\ \frac{1}{50}\ sec && (6.71)\\
&= requests\ for\ 6,480\ responses\ of\ 128\ B\ per\ \frac{1}{50}\ sec && (6.72)\\
&= 6,480 \cdot 8 \cdot 50\ B\ per\ sec && (6.73)\\
&= 2.59\ MBps && (6.74)\\
\sigma_{8a} &= n_8 \cdot 8 \cdot \left(1 - \frac{\rho_{8a}}{C}\right) && (6.75)\\
\rho_{8b} &\geq 6,480 \cdot 128 \cdot 50\ B\ per\ sec && (6.76)\\
&= 41.5\ MBps && (6.77)\\
\sigma_{8b} &= n_b \cdot 128 \cdot \left(1 - \frac{\rho_{8b}}{C}\right) && (6.78)
\end{aligned}$$

- The deadline is $\frac{1}{50}$ *sec* for 6,480 write actions
- The deadline is $\frac{1}{50}$ *sec* for 6,480 requests and responses

6.3.7 From the Video Out system to the Display

See link 5 of figure 6.1.

This traffic is not mapped on the interconnect, but a dedicated link is used.

6.3.8 Refresh

According to [16], every 7.8125 μsec a refresh has to be processed by the Memory System. We assume that a refresh request will have a packet size of 8 *bytes*.

Traffic characterization

Assume that b_9 represents the refresh requests.

$$\begin{aligned}
\rho_9 &\geq 1 \cdot 8\ B\ per\ 7.8125\ \mu sec && (6.79)\\
&= 1.02\ MBps && (6.80)\\
\sigma_9 &= n_9 \cdot 8 \cdot \left(1 - \frac{\rho_9}{C}\right) && (6.81)
\end{aligned}$$

- The deadline is 7.8125 μsec for 1 refresh

6.3.9 Overview

The traffic characteristics of all sessions are summarized in table 6.1.

Session	**Timing variant**	**From**	**To**	σ (B)	ρ $(MBps)$	**x per D^{max} period** $(\#requests)$	D^{max} (sec)
1_a	1	DVB-T CD	Mem	$n_1 \cdot 8 \cdot (1 - \frac{\rho_{1a}}{C})$	11.2		
1_b	1	Mem	DVB-T CD	$n_1 \cdot 128 \cdot (1 - \frac{\rho_{1b}}{C})$	179	313	$\frac{1}{4} \cdot OFDM$
2	1	DVB-T CD	Mem	$n_2 \cdot 128 \cdot (1 - \frac{\rho_2}{C})$	89.7	157	$\frac{1}{4} \cdot OFDM$
1_a	2	DVB-T CD	Mem	$n_1 \cdot 8 \cdot (1 - \frac{\rho_{1a}}{C})$	22.4		
1_b	2	Mem	DVB-T CD	$n_1 \cdot 128 \cdot (1 - \frac{\rho_{1b}}{C})$	358	313	$\frac{1}{8} \cdot OFDM$
2	2	DVB-T CD	Mem	$n_2 \cdot 128 \cdot (1 - \frac{\rho_2}{C})$	179	157	$\frac{1}{8} \cdot OFDM$
1_a	3	DVB-T CD	Mem	$n_1 \cdot 8 \cdot (1 - \frac{\rho_{1a}}{C})$	7.45		
1_b	3	Mem	DVB-T CD	$n_1 \cdot 128 \cdot (1 - \frac{\rho_{1b}}{C})$	119	313	$\frac{3}{8} \cdot OFDM$
2	3	DVB-T CD	Mem	$n_2 \cdot 128 \cdot (1 - \frac{\rho_2}{C})$	59.8	157	$\frac{3}{8} \cdot OFDM$
3	-	DVB-T CD	Mem	$n_3 \cdot 128 \cdot (1 - \frac{\rho_3}{C})$	16.0	28	$\frac{1}{4} \cdot OFDM$
4_a	-	H.264	Mem	$n_4 \cdot 8 \cdot (1 - \frac{\rho_{4a}}{C})$	0.500		
4_b	-	Mem	H.264	$n_4 \cdot 128 \cdot (1 - \frac{\rho_{4b}}{C})$	8.00	28	$\frac{1}{2} \cdot OFDM$
5_a	-	Storage	Mem	$n_5 \cdot 8 \cdot (1 - \frac{\rho_{5a}}{C})$	0.500		
5_b	-	Mem	Storage	$n_5 \cdot 128 \cdot (1 - \frac{\rho_{5b}}{C})$	8.00	28	$\frac{1}{2} \cdot OFDM$
6_a	-	H.264	Mem	$n_6 \cdot 8 \cdot (1 - \frac{\rho_{6a}}{C})$	18.7		
6_b	-	Mem	H.264	$n_6 \cdot 128 \cdot (1 - \frac{\rho_{6b}}{C})$	299	38880	$\frac{1}{60}$
7	-	H.264	Mem	$n_7 \cdot 128 \cdot (1 - \frac{\rho_7}{C})$	41.5	6480	$\frac{1}{50}$
8_a	-	Video	Mem	$n_8 \cdot 8 \cdot (1 - \frac{\rho_{8a}}{C})$	2.59		
8_b	-	Mem	Video	$n_8 \cdot 128 \cdot (1 - \frac{\rho_{8b}}{C})$	41.5	6480	$\frac{1}{50}$
9	-	Mem	Mem	$n_9 \cdot 8 \cdot (1 - \frac{\rho_9}{C})$	1.02	1	$7.81 \cdot 10^{-6}$

Table 6.1: Overview of the characteristics of all sessions

Please remember that an $OFDM$-period takes 896 μsec.

For the performance analysis method, we have made the following general assumptions:

- All read requests have a size of 8 *bytes*
- All read responses and write actions have a size of 128 *bytes*
- A packet does not have any overhead (no packet header)
- No acknowledgment of a write action is sent
- The interconnect propagation delay is 0 *sec*
- The arbitration policy is TDMA

Furthermore, we have made the following assumptions for the Memory System (see [13])

- The size of a refresh request is 8 *bytes*
- A read request for 128 *bytes* will take 22 cycles to process (including all overhead)
- A write action of 128 *bytes* will take 25 cycles to process (including all overhead)
- A refresh action will take 10 cycles to process (including all overhead)

Using these traffic characteristics and assumptions, our performance analysis method can be performed.

6.4 Mathematica

To obtain results, we use the performance analysis method as described in section 4.7. Therefore, we have created a Mathematica model.

To use the Mathematica model, we first set the frequency of the DRAM. This is done, because the DRAM is the shared resource and therefore the bottleneck. After that, we search for solutions by changing the TDMA wheel (i.e. the set of w_i, see section 4.5). Please note that the service rates provided by the DRAM controller should be at least equal to the ρ's of the sessions (see section 4.5). If a correct setting of the TDMA wheel is found, all pipeline degrees are set to 1 and the delays are determined by executing the Mathematica model. Then, the degree of a session is increased, until its deadline is met. Each time, the ρ's are increased to the maximum service rate provided by the DRAM controller. When all deadlines are met, the solution is stored. Increasing the pipeline degree more is not useful, because the deadlines are already met.

Then, the next solutions are determined by changing the TDMA wheel. Finally, the DRAM frequency is changed and the whole process is repeated.

Only the best solutions (i.e. the less queue size required for each maximum pipeline degree and for each frequency) are output. More details can be found in appendix C.

6.5 Results

We compare several schedule and interconnect variants using the Mathematica model as described above.

6.5.1 Schedule variants

In this section, we compare the three schedules defined in section 6.3.2. We use the following SoC communication infrastructure (see figure 6.8). All subsystems have a private link to the DRAM controller. The responses of the DRAM controller are sent via the bus. Because only one master is connected to this bus, no scheduler is required.

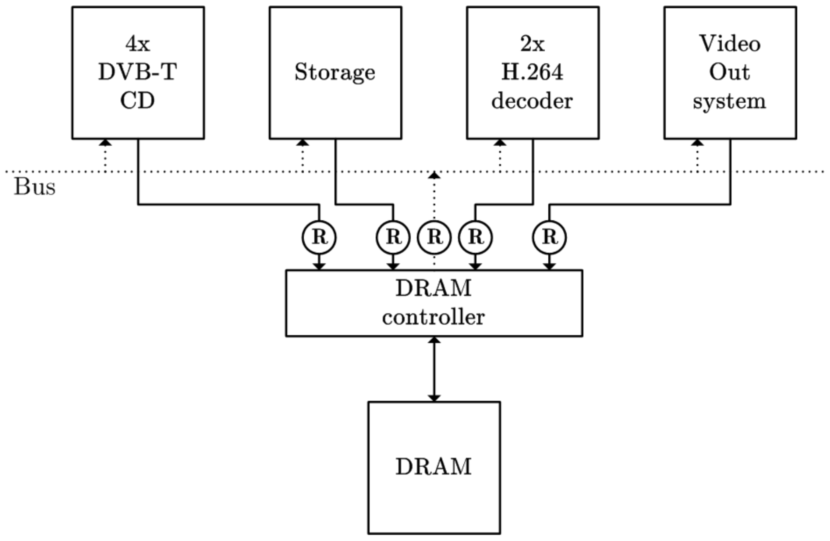


Figure 6.8: Private links interconnect

On the inputs and the outputs of the DRAM controller, regulators (i.e. "R") are placed to shape the traffic, if necessary.

Using schedule 1, we get the results of figure 6.9.

The dimensions of the graph are

- The pipeline degree, which represents the maximum number of allowed outstanding requests per session
- The frequency of the scheduler(s) (i.e. the DRAM controller)
- The total required queue size of the scheduler(s) and the regulator(s) together

Please note that we have not found a solution with a frequency less than 150 MHz. This means, we are not able to guarantee that all deadlines were met.

Using schedule 2, we get the results of figure 6.10.

Using schedule 3, we get the results of figure 6.11.

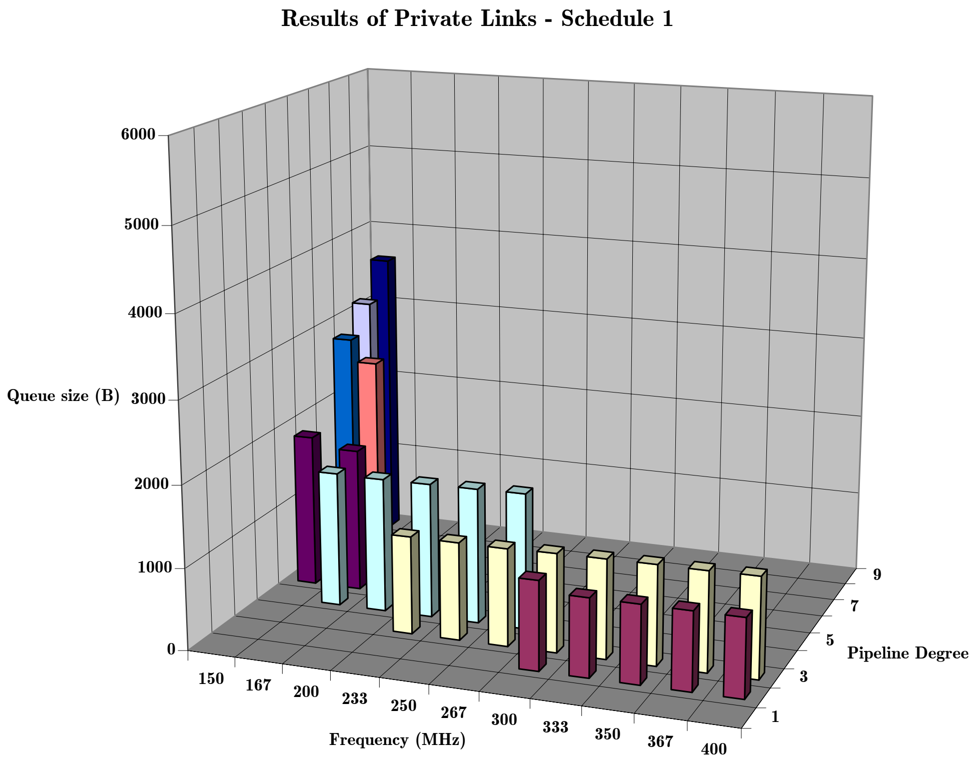


Figure 6.9: Results of Private Links analysis using schedule 1

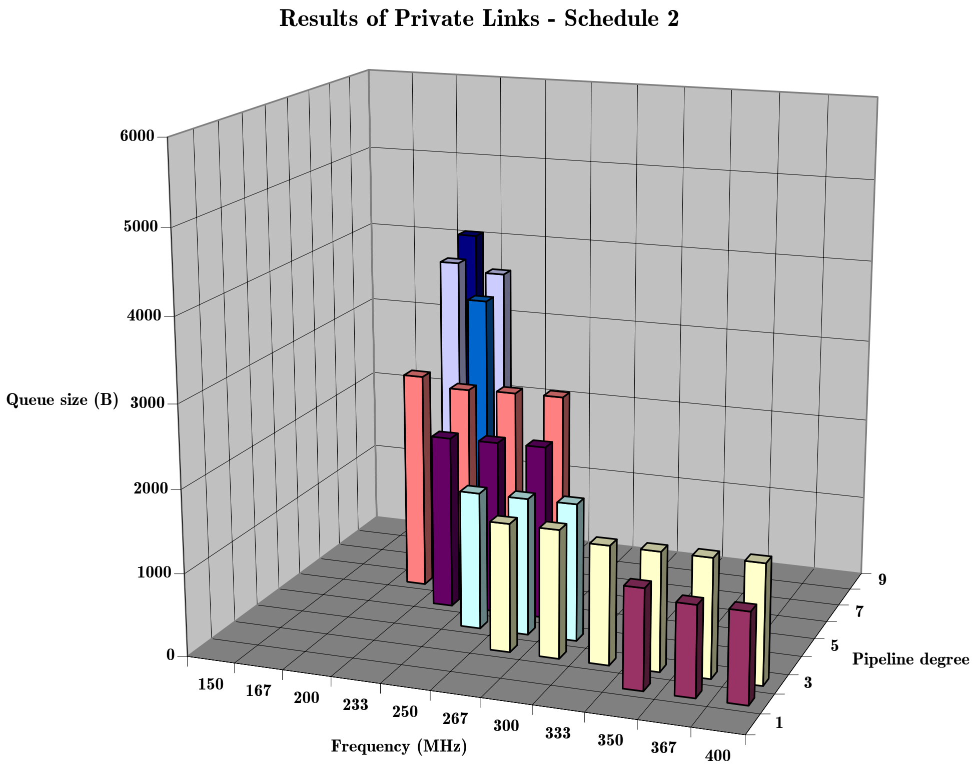


Figure 6.10: Results of Private Links analysis using schedule 2

Comparison

Analyzing the results of the three schedules, we observe the following general trends.

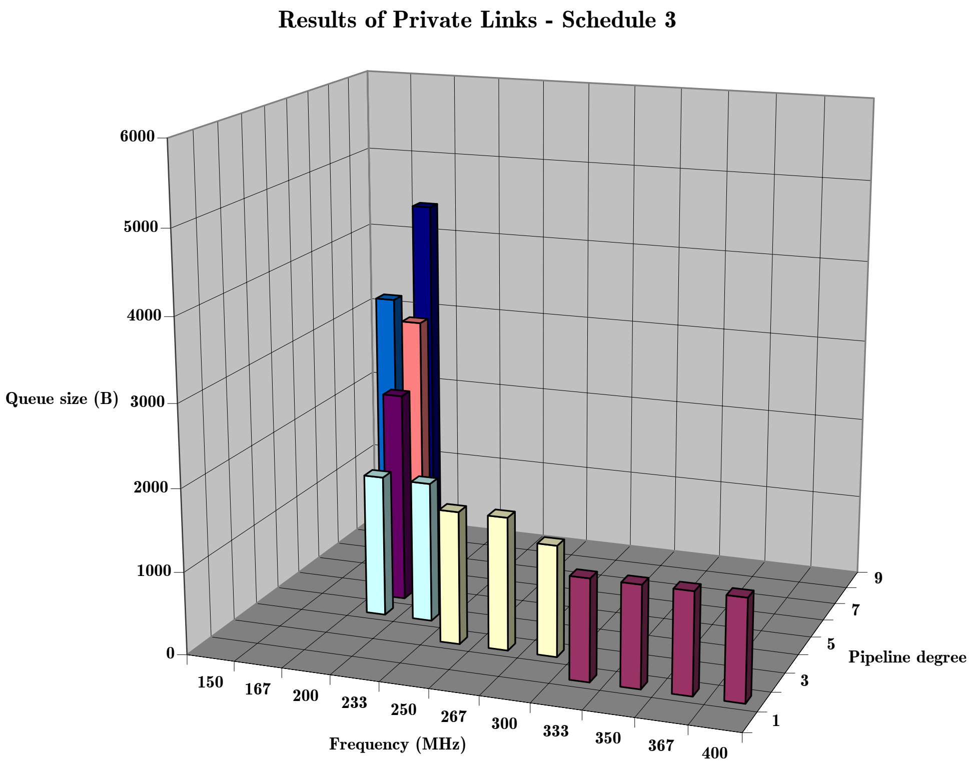


Figure 6.11: Results of Private Links analysis using schedule 3

By increasing the frequency, it is possible to meet the deadlines with a lower pipeline degree and smaller queues. The reason for this is simple. By increasing the frequency, the packets are processed faster. The latencies decrease and the service rates of the sessions can be higher. Please note that there is no need to increase the number of outstanding requests, if all deadlines are already met. The lower the pipeline degree is, the smaller the size of the queues required can be and the less complex the subsystems can be. All three graphs show monotonicity. If the pipeline degree decreases, or the frequency increases, the size of the queues required decreases.

Furthermore, all schedules have a low queue size limit. Increasing the frequency starting at a low frequency results in a significant amount of reduced queue space. The higher the frequency gets, the less benefit from reducing the pipeline degree and the queue size.

When we look into more detail, it becomes clear that schedule 2 has a higher required frequency compared to the other two schedules. For a frequency less than 200 MHz, no solution has been found for schedule 2. The deadlines for the $OFDM$-symbols of schedule 2 are more strict, so a higher service rate has to be reserved for these sessions. Therefore, a high frequency or a large TDMA wheel is required. The disadvantage of a large TDMA wheel is the high latency. The higher the latency, the larger the queues. Therefore, schedule 2 requires significantly more queues compared to the other two schedules. But, the higher the frequency gets, the smaller the difference in queue size among the schedules is.

Schedule 1 outperforms schedule 2. By using a frequency of 200 MHz, the pipeline degree and the required queue size of schedule 1 are much less. A higher frequency results in a further decrease of the pipeline degree and the queue size.

So, comparing schedule 1 and 2, the choice is easy. Schedule 1 is in all aspects (queue size, frequency and pipeline degree) superior. Please note that we only look to the queues in the regulators and the scheduler. The queues required to store the $OFDM$-symbols in the DVB-T CDs are not measured. If these queues are taken into account, schedule 2 has a much better performance because less queues are required to store the $OFDM$-symbols, compared to the other two schedules.

When we look at schedule 3, it becomes clear that schedule 3 is not as good as schedule 1. It requires larger queues. Although the deadlines of the $OFDM$-symbols are less strict, in the worst case scenario it is possible that 4 $OFDM$-symbols are read at the same moment. Then, the service rates have to be able to handle this worst case scenario. Therefore, the frequency and the queue size are larger compared to schedule 1.

So, we conclude that schedule 1 is the most interesting one. It is superior in all aspects, when we compare it with schedules 2 and 3. In the next section, we look at several interconnects and we use schedule 1 to compare different solutions.

6.5.2 Interconnect variants

In this section, we compare three interconnect variants using the same traffic as defined in section 6.3. As already indicated, schedule 1 will be used.

Private links

With Private Links we have the same analysis as in the previous section. The results are shown in figure 6.9

Bus

The second interconnect variant is the bus based interconnect as is shown in figure 6.12. The traffic to and from the memory system goes via a bus. There is a single point of arbitration in the bus arbiter. The subsystems and the DRAM controller first send a request to the bus arbiter. Then, the bus arbiter determines the order of the data transfers on the bus.

Please note that the DRAM controller does not reschedule the requests it receives. Furthermore, each connection to the bus has a regulator ("R") to shape the traffic if necessary.

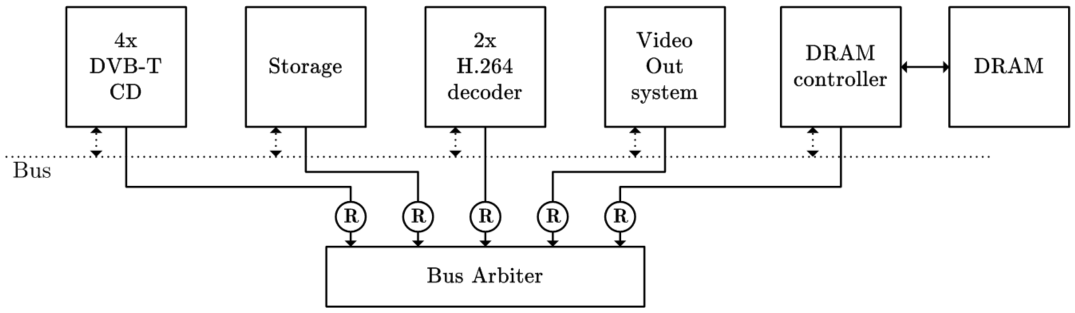


Figure 6.12: Bus interconnect

Using this interconnect, we obtain the results of figure 6.13.

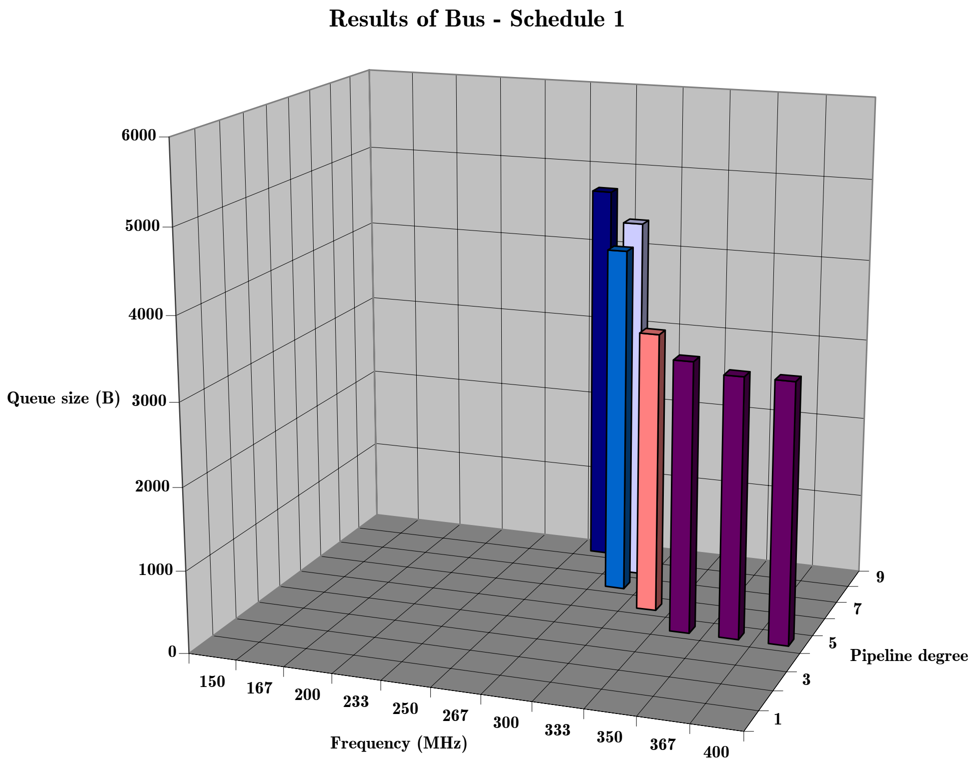


Figure 6.13: Results of Bus analysis using schedule 1

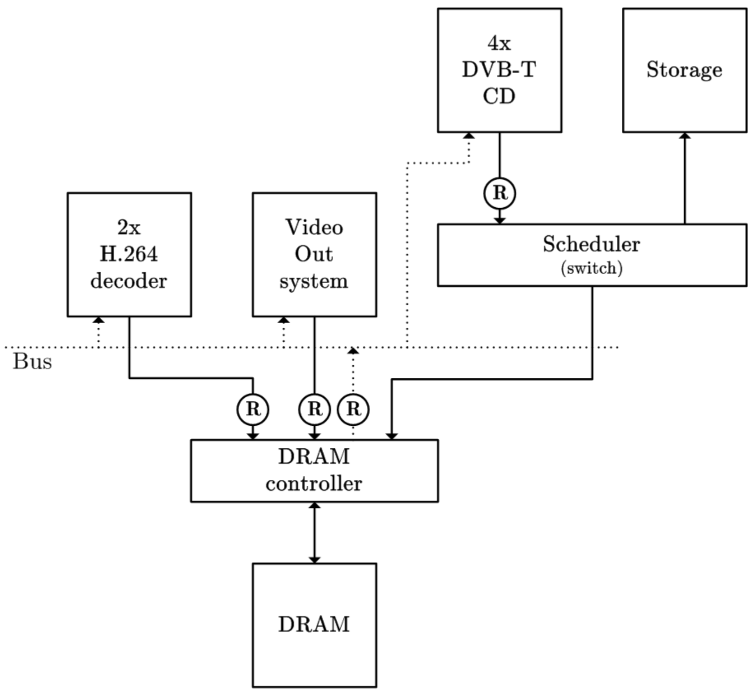


Figure 6.14: NoC interconnect

Network-on-Chip

The third interconnect variant is a Network-on-Chip (NoC), as is shown in figure 6.14. Compared to the first variant, an extra scheduler (or switch) is added, to send the traffic from the 4xDVB-T Channel Decoders to Storage directly.

Please note that only in this interconnect variant, session 5 is not used. This is the case, because the data is sent directly from the 4xDVB-T Channel Decoder to the Storage, instead of via the memory system.

Using this interconnect, we obtain the results of figure 6.15.

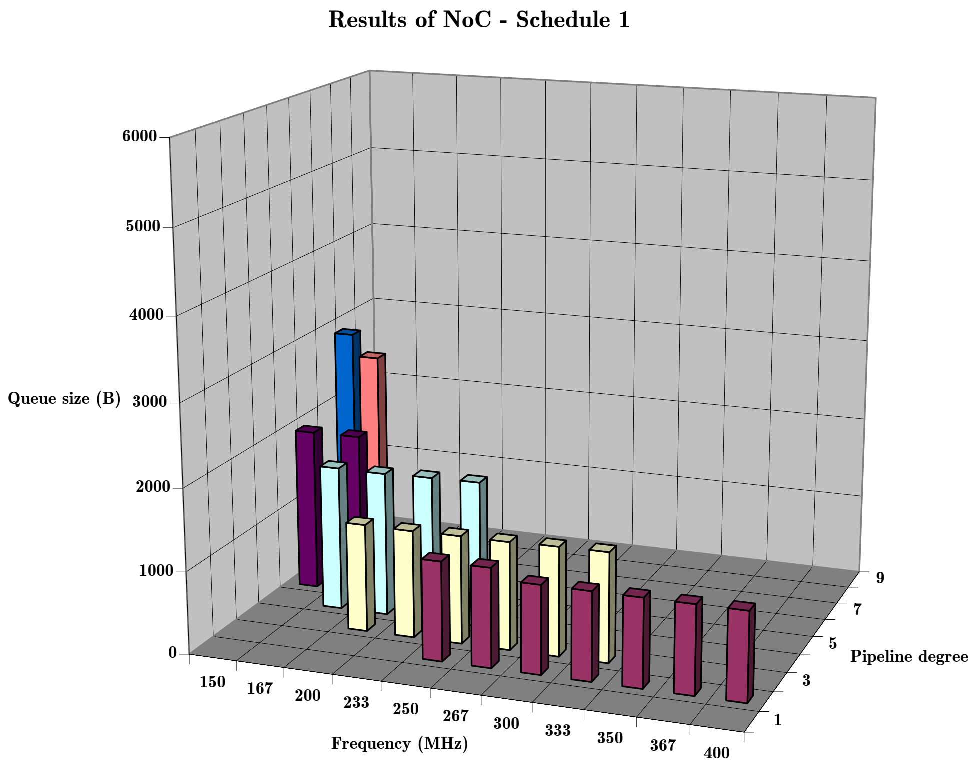


Figure 6.15: Results of NoC analysis using schedule 1

Comparison

Analyzing the results of the three interconnect variants, we observed the following.

The same general remarks as in the previous comparison can be made. Again, it is clear that by increasing the frequency, the pipeline degree and the queue size can decrease. Only this time, the differences among the several interconnect variants are larger than the differences among the schedules. Once again, there is a lower limit on the size of the queues. By increasing the frequency more and more, the profit of reducing the queues becomes smaller.

Furthermore, the bus interconnect sends all packets via the same wires. The other two interconnects make a distinction between the traffic to and from the memory. The bus interconnect does not make this distinction. Therefore, the frequency has to be higher for the bus interconnect, compared to the other two interconnects, to guarantee that all deadlines are met.

Because more traffic is handled via the same wires, more queues are required. The latencies for a selected frequency are higher (a larger TDMA wheel is required, because more sessions share the same scheduler), so even more queues are required. Therefore, the size of the queues required for the bus based interconnect is significantly higher than for the other two interconnect variants.

Although the bus based architecture has low cost (only one scheduler and one bus are required), the frequency is high compared to the other two solutions. So the bus architecture is not an option for this case study. For a low frequency (around 200 MHz), the deadlines are not guaranteed. For a frequency between 300 MHz and 400 MHz, the queue size and the pipeline degree are significantly higher compared to the situation that Private Links or a NoC is used.

The difference between Private Links and NoC is not significant. The NoC needs more queues for a given frequency and pipeline degree. The main reason is that an extra scheduler is required and schedulers simply require queues. Therefore, the NoC requires more queues than the Private Links variant.

So, for this specific case study, the Private Links are a bit better than the NoC. Using the same frequency, smaller queues are required. But, the NoC is able to decrease its pipeline degree at a lower frequency than Private Links. Sometimes it is possible that by using the same frequency, the NoC can have a smaller pipeline degree than the Private Links. Still, a NoC requires more queues for the same frequency in this case study.

We expect that if a different case study is used with more direct communication, the NoC will become better than the Private Links. In this case study, the extra scheduler used by the NoC has a maximum utilization of only 8%. If more point-to-point communication is available, this utilization will increase. Then, the DRAM controller can be relieved a bit. Right now, the cost of the extra scheduler is higher than its benefits.

So, the Private Links interconnect is the best of the three interconnects variants. It requires significant less queues, and it is cheaper to produce than the NoC interconnect. There is a break point around a frequency of 333 MHZ. By increasing the frequency, the queue sizes do not change significantly.

6.5.3 Influence on an individual session

As we saw in the previous sections, the smaller the pipeline degree is or the higher the frequency is, the smaller the queues can be. In this section, we look into more detail to the influence of the pipeline degree and the frequency.

We took the results of section 6.5.1, using Private Links and schedule 1. One of the most interesting traffic sessions is session 1 (DVB-T CDs reading the $OFDM$-symbols). This is the case, because the rate of this session is high and the deadline is very strict. Therefore, special attention has to be given to this session such that its deadline is met.

The influence of the pipeline degree and the frequency are shown in figure 6.16. For each solution of figure 6.9, we calculated the amount of time slack per $\frac{1}{4}$ $OFDM$-period for session 1 (see section 6.3). This means

$$time\ slack\ of\ session\ i = D_i^{max} - D_i^{total} \tag{6.82}$$

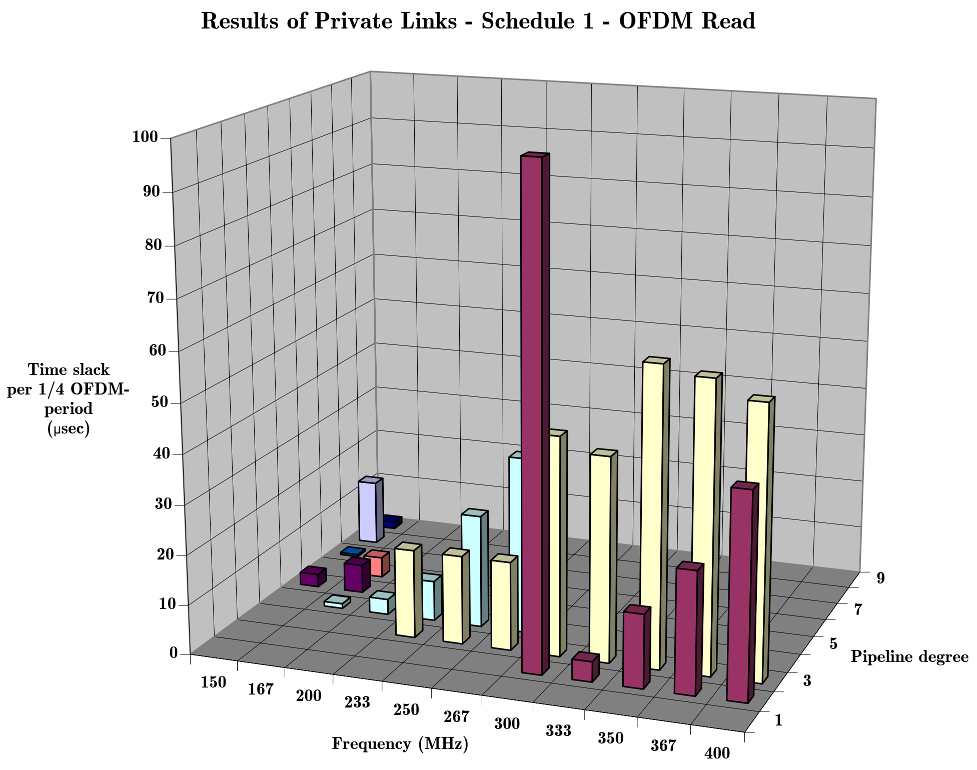


Figure 6.16: Results of Private Links analysis using schedule 1

Then, the following general trends can be observed.

There is a general trend that if the frequency or the pipeline degree increases, the time slack also increases. A higher frequency results in faster servicing of the packets. Using a higher pipeline degree causes more packets in flight to be serviced. For a frequency of 200 MHz or less, the time slack is almost zero. This means that the deadline is just met. Using a higher frequency, the session has ample time.

Some results attract attention.

Firstly, at a frequency of 300 MHz and using a pipeline degree of 2, the slack time is enormous. This is caused by the TDMA wheel. In this situation, session 1 is allowed to send two packets per round. If this number is decreased to one, the deadline is not met. Looking to the surrounding results, less time per round is reserved for this session. Therefore, this result is not according to the general trend.

Furthermore, at a frequency between 300 and 400 MHz and using a pipeline degree of 3, the results are significantly different than the general trend. This is also caused by the TDMA wheel. Each time a different TDMA wheel is selected. Because the solutions with the lowest queue requirements are selected, different TDMA wheels are selected. Therefore, the general trend is not applicable here.

It is also possible to use a fixed TDMA wheel for all solutions. Assume that all sessions get a service of one packet per round. Then, the results are shown in figure 6.17.

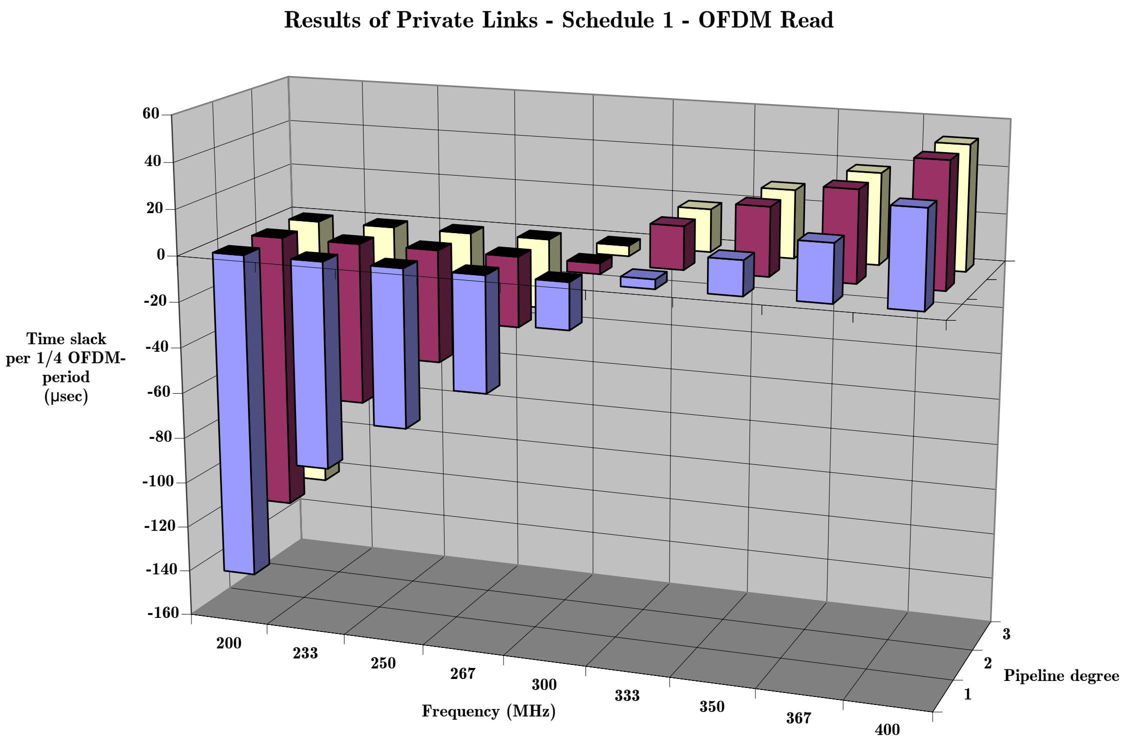


Figure 6.17: Results of Private Links analysis using schedule 1 and a fixed TDMA wheel

Please note that a negative time slack means that in the worst case scenario the deadline is not met.

As is shown in figure 6.17, the amount of time slack is monotonically increasing if the pipeline degree is increased, or the frequency is increased. This result is as expected.

6.5.4 Influence of packet size

Until now, we assumed that the size of all read responses and write actions are 128 *bytes*. Assume that the size of theses packets decreases to 64 *bytes*. Then, the characteristics of the sessions change a bit, but can still be determined in the same way as is done in section 6.3.

Furthermore, we assume that for the Memory System (see [13])

- A read request for 64 *bytes* will take 14 cycles to process (including all overhead)
- A write action of 64 *bytes* will take 17 cycles to process (including all overhead)

Using Private Links and schedule 1, we obtain the results of figure 6.18.

Comparing these results with the results of figure 6.9, several general trends can be observed.

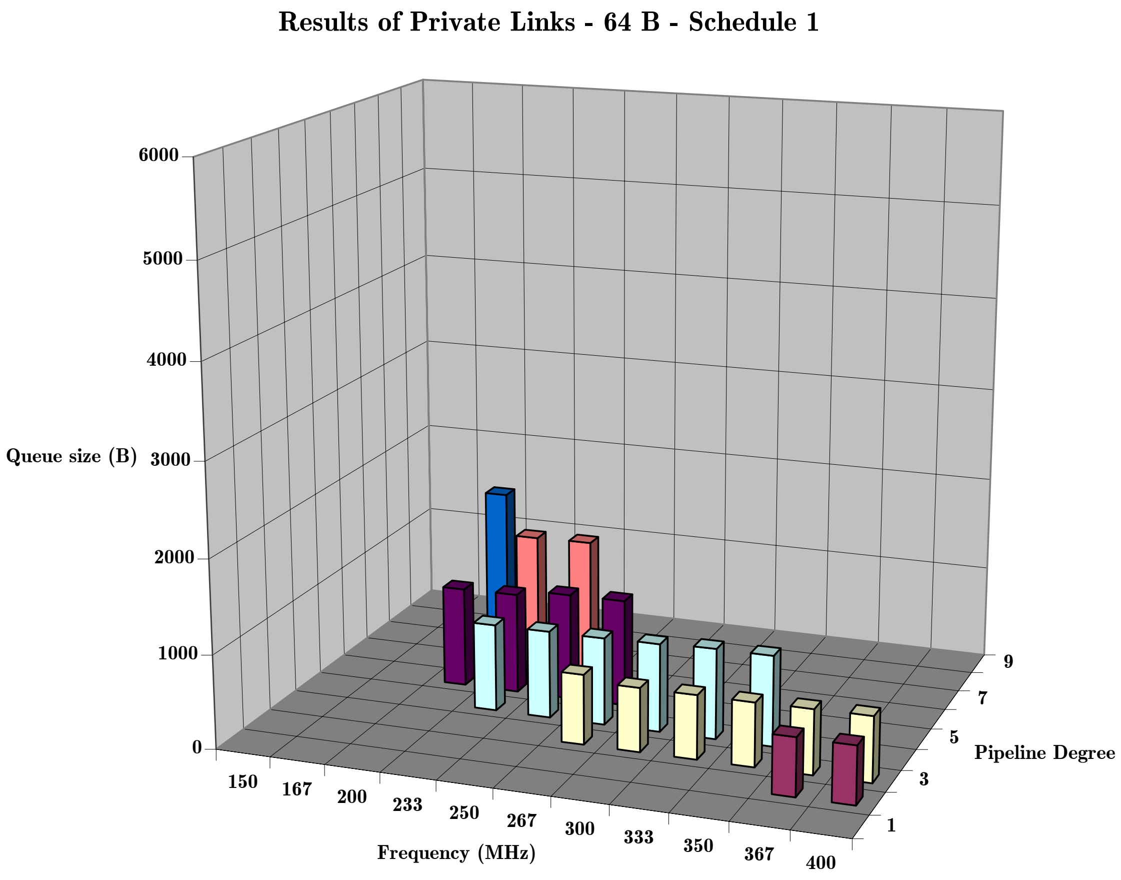


Figure 6.18: Results of Private Links analysis using schedule 1 and packets of 64 *bytes*

First all all, using packets of 64 *bytes*, a higher frequency or a higher pipeline degree is required. This is the case, because the read requests and write actions have more overhead. It takes more time per byte to be processed by the DRAM, compared to packets of 128 *bytes*.

Furthermore, in the case that all deadlines are guaranteed for a particular pipeline degree and frequency, the queues can be smaller. The same number of packets can be in flight, but the packets are smaller. Therefore, the size of the queues can be reduced.

So, if it is possible to use packets of 64 *bytes* (i.e. all deadlines are met), it is better to use the smaller packets, instead of packets of 128 *bytes*.

6.6 Conclusion

In this case study several interesting aspects have become visible.

First of all, the case study itself is diverse. The traffic sessions have significantly different characteristics. This means that not all sessions belong to the same traffic class (see appendix B). Some sessions (e.g. reading $OFDM$-symbols) have very strict deadlines, other sessions have a low priority (e.g. load traffic of Storage). Nevertheless, all sessions can be characterized and analyzed using the performance analysis method of chapter 4.

Furthermore, the various schedules have quite diverse influences. By only changing the characteristics of one or two sessions, the results change significantly. Using the performance analysis method it is showed that although the deadlines of schedule 3 are less strict compared to schedule 1, in the worst case scenario more queues are required. This is caused by the fact that extra service has to be reserved, because the read and write sessions of the Channel Decoders are not mutually exclusive anymore. Please note that in this schedule it is possible that four $OFDM$-symbols are read at the same time.

Also the influence of different interconnects can be analyzed with the performance analysis method. By selecting a cheap interconnect like the bus based interconnect, a higher price has to be paid for the queue sizes and the frequency. Adding an extra scheduler can only be profitable if the extra scheduler has enough packets to service. Unfortunately, in this case study not enough direct communication is available to achieve a high utilization of the extra scheduler.

Furthermore, also the influence of the pipeline degree and the frequency on individual sessions can be shown by making use of the performance analysis method. Especially the monotonicity of the slack time with respect to the pipeline degree and the frequency is important for defining the architecture of a system.

Finally, the influence of the packet size can be shown by making use of the performance analysis method. In the case that all deadlines are met for a particular pipeline degree and frequency, it is better to use packets of size 64 $bytes$ instead of packets of size 128 $bytes$.

Therefore, we can conclude that this case study has interesting architectural aspects that become visible by our performance analysis method. Furthermore, the value of the performance analysis method has been demonstrated by showing an easy analysis of different solutions. By applying the method, insight is gained in the performance and the bottlenecks of the architecture.

In the next chapter, we conclude this thesis.

Conclusion

Conclusion

The literature describes several techniques to characterize traffic (see chapter 2). One technique to characterize traffic (i.e. the method of Cruz) makes use of two parameters, namely ρ (i.e. the rate of the traffic) and σ (i.e. the burstiness constraint of the traffic). With these parameters, an upper bound for traffic can be represented.

As an extension to σ and ρ, Stiliadis introduced an abstract type of schedulers, called Latency-Rate (LR) servers. The arbitration policy of the scheduler is used to characterize the schedulers of the communication infrastructure. These schedulers can easily be connected in a network to model interconnect. By making use of arbitration policies, the method of Stiliadis determines an upper bound for the size of the queues required and the delay of the packets in the schedulers. Because of the high aggregation level of the method of Stiliadis, the method of Stiliadis has been selected as foundation for the performance analysis method of SoC architectures in chapter 3.

The method of Stiliadis is not directly applicable to analyze the performance of a SoC architecture, due to specific SoC characteristics. It does not support the calculation of the delay of multiple packets of a traffic stream. Also, only request-only traffic streams are expressible in the method of Stiliadis. Finally, Stiliadis does not take the number of outstanding requests of a traffic stream into account. Therefore, several extensions and changes had to be made to the performance analysis method to meet all requirements (see chapter 4).

A memory system is an important part of a SoC architecture with specific characteristics. Therefore, we have deployed a model of a memory system with suitable characteristics. Furthermore, we have added specific arbitration policies to the performance analysis method that are used in the schedulers of SoCs. Finally, we have made two improvements to the performance analysis method to tighten the bounds of the size of the queues required. All improvements and additions to the method of Stiliadis are proved to be correct in appendix A.

Our performance analysis method can handle traffic streams of several traffic classes, as is shown in a multi-channel DVB-T set-top box case study. All traffic streams are characterized by the same set of parameters. This flexibility is a valuable property of the performance analysis method.

Furthermore, it appears to be easy to characterize a traffic stream, to model interconnects or to set the size of the packets in the performance analysis method. The maximum delay of the traffic steams and the size of the queues required are calculated by the performance analysis method within a second. By that, large numbers of variants can be tested in small amount of time. In the multi-channel DVB-T set-top box case study, several schedule and interconnect variants are analyzed. The influence of the pipeline degree and the maximum rate of the communication infrastructure has become visible in the required queue size and the time slack of the traffic streams. Therefore, the performance analysis method can be used to select a suitable SoC communication infrastructure for selected use cases.

In future work, the performance analysis method can be made more precise by taking propagation delays into account. Also the assumptions made in the performance analysis method can be made more realistic. For example, all schedulers are assumed to be able to make decisions without delay. In the implementation, this will not be possible.

The performance analysis method also has some limitations. It is not possible to express backpressure. Backpressure is prevented by using queues large enough to receive all packets. It is also not possible to express traffic dependencies in the performance analysis method. Therefore, all schedules have to be periodic. Complementary methods should be used to obtain the traffic requirements and schedules. Then, SoC architectures can be checked by using these traffic requirements. Furthermore, flow control is also not possible in our performance analysis method at the cost of larger queues.

Finally, it should be possible to extend the performance analysis method to output data for more metrics. Possible extensions could be generating first order estimations for the required area size and the power consumption. Furthermore, preemptive arbitration policies could possibly be added to the performance analysis method.

Bibliography

[1] "The balusc server - core 2." [Online]. Available: http://balusc.xs4all.nl/srv/har-cpu-int-c2.php

[2] "Dvb-t." [Online]. Available: http://nl.wikipedia.org/wiki/DVB-T

[3] "Moore's law." [Online]. Available: http://www.intel.com/technology/mooreslaw/

[4] "Motion compensation." [Online]. Available: http://en.wikipedia.org/wiki/Motion_compensation

[5] "system-on-a-chip." [Online]. Available: http://www.answers.com/SoC

[6] E. E. . 744, *Digital Video Broadcasting (DVB); Framing structure, channel coding, and modulation for digital terrestrial television*, Nov. 2004.

[7] S. I. Association, "International technology roadmap for semiconductors: 1999 edition." [Online]. Available: ftp://ftp.cordis.europa.eu/pub/ist/docs/ka4/pdesign_gap.pdf

[8] S. Chakraborty and L. Thiele, "A new task model for streaming applications and its schedulability analysis," in *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 486–491.

[9] R. L. Cruz, "A calculus for network delay, part i: Network elements in isolation." *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 114–131, 1991.

[10] ——, "A calculus for network delay, part ii: Network analysis." *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 132–141, 1991.

[11] ——, "Quality of service guarantees in virtual circuit switched networks," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1048–1056, 1995. [Online]. Available: citeseer.ist.psu.edu/cruz95quality.html

[12] K. Goossens, S. González Pestana, J. Dielissen, O. P. Gangwal, J. van Meerbergen, A. Rădulescu, E. Rijpkema, and P. Wielage, "Service-based design of systems on chip and networks on chip," in *Dynamic and Robust Streaming In And Between Connected Consumer-Electronics Devices*, ser. Philips Research Book Series, P. van der Stok, Ed. Springer, 2005, vol. 3, ch. 2, pp. 37–60.

[13] T. Henriksson, P. van der Wolf, A. Jantsch, and A. Bruce, "Network calculus applied to verification of memory access performance in socs," *Manuscript*, 2007.

[14] S. Kanhere and H. Sethu, "The latency bound of deficit round robin," 2002. [Online]. Available: citeseer.ist.psu.edu/article/kanhere02latency.html

[15] T. management and V. The, "Technical committee," 1996. [Online]. Available: http://www.mfaforum.org/ftp/pub/approved-specs/af-tm-0056.000.pdf

[16] I. Micron Technology, "Mobile ddr sdram mt46h32m16lf," 2004. [Online]. Available: http://download.micron.com/pdf/datasheets/dram/mobile/MT46H32M16LF.pdf

[17] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, 1993.

[18] K. Richter, M. Jersak, and R. Ernst, "A formal approach to mpsoc performance verification," 2003. [Online]. Available: citeseer.ist.psu.edu/richter03formal.html

[19] C. Rowen, "Pushing the envelope of system-on-chip integration." [Online]. Available: http://asia.stanford.edu/events/Spring05/slides/050407-Rowen.pdf

[20] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *SIGCOMM '95: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication.* New York, NY, USA: ACM Press, 1995, pp. 231–242.

[21] D. Stiliadis, "Traffic scheduling in packet-switched networks: analysis, design, and implementation," Ph.D. dissertation, 1996.

[22] D. Stiliadis and A. Varma, "Latency-rate servers: a general model for analysis of traffic scheduling algorithms," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, pp. 611–624, 1998.

[23] P. van der Wolf, "Memory sharing - in modular network-centric architectures," in *M&P Architecture Board*, 2007.

[24] P. van der Wolf et al, "Definition of communication and configuration services with qos support," in *SPRINT*, 2007.

[25] Z. Wang and J. Crowcroft, "Analysis of burstiness and jitter in real-time communications," *Conference proceedings on Communications architectures, protocols and applications, September 13 - 17, 1993, San Francisco, CA USA*, pp. 13–19, 1993. [Online]. Available: citeseer.ist.psu.edu/article/wang93analysis.html

[26] L. Zhang, "Virtualclock: a new traffic control algorithm for packet-switched networks," *ACM Trans. Comput. Syst.*, vol. 9, no. 2, pp. 101–124, 1991.

Appendices

Validation

A.1 Lemmas

Lemma A.1.1 *Assume a session with parameters (σ, ρ) and packet size L. Then, the delay for x words in a chain of m LR servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$ and a capacity of C, is upper bounded by*

$$\left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho}$$

Proof Assume a busy period. Then (see section 2.4)

$$\rho \cdot (t - t_1) \leq A(t_1, t) \leq \sigma + \rho \cdot (t - t_1) \tag{A.1}$$

According to [22] (see equation 3.38), the delay for σ *words* is upper bounded by

$$\frac{\sigma}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} \tag{A.2}$$

So, the delay for x *words* is upper bounded by

$$\begin{aligned} & \frac{x}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} && \text{(A.3)} \\ \leq \quad & \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} && \text{(A.4)} \end{aligned}$$

This is the case, because it is assumed there is enough input. $\square$

Lemma A.1.2 *Assume a session with parameters (σ, ρ) and packet size L. Then, the delay for x words in a chain of m LR servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$ and a capacity of C, including receiving the first packet is upper bounded by*

$$\left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C}$$

Proof According to lemma A.1.1, the delay for x *words* is upper bounded by

$$\left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho}$$

But, the time to receive the first packet is not included. So, the delay for x *words* including receiving the first packet is upper bounded by

$$\left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} \tag{A.5}$$

□

Lemma A.1.3 *Assume $x \geq 1$, $x \in \mathbb{N}$, $L \geq 1$, $L \in \mathbb{N}$, $n \geq 1$ and $n \in \mathbb{N}$, then*

$$\left(\left\lceil \frac{x}{L} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L} \right\rceil - 1 \right) - 1 \right) < n$$

Proof

$$\begin{aligned}
\left\lceil \frac{x}{L} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L} \right\rceil - 1 \right) - 1 &= \left\lceil \frac{x}{L} \right\rceil - n \cdot \left\lceil \frac{x}{n \cdot L} \right\rceil + n - 1 && \text{(A.6)} \\
&< \frac{x}{L} + 1 - n \cdot \left\lceil \frac{x}{n \cdot L} \right\rceil + n - 1 && \text{(A.7)} \\
&= \frac{x}{L} - n \cdot \left\lceil \frac{x}{n \cdot L} \right\rceil + n && \text{(A.8)} \\
&\leq \frac{x}{L} - n \cdot \frac{x}{n \cdot L} + n && \text{(A.9)} \\
&= n && \text{(A.10)}
\end{aligned}$$

□

Lemma A.1.4 *Assume a session with parameters (σ, ρ) and packet size L. Further assume that the delay (i.e. D) for x words in a chain of m LR servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$, a capacity of C and a pipeline degree of n, including receiving the first packet is upper bounded by*

$$\begin{aligned}
&\left\lceil \frac{x}{n \cdot L} \right\rceil \cdot \left(\frac{L}{C} + \sum_{j=1}^{m} \Theta^{(S_j)} \right) \\
&+ \left(\left\lceil \frac{x}{L} \right\rceil - n \cdot \left(\left\lceil \frac{x}{n \cdot L} \right\rceil - 1 \right) - 1 \right) \cdot \frac{L}{\rho}
\end{aligned} \tag{A.11}$$

If the pipeline degree is large, D converges to (see lemma A.1.2)

$$\left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} \tag{A.12}$$

Proof

$$\begin{aligned}
\lim_{n\to\infty} D &= \lim_{n\to\infty}\Bigg(\left\lceil\frac{x}{n\cdot L}\right\rceil\cdot\left(\frac{L}{C}+\sum_{j=1}^{m}\Theta^{(S_j)}\right) \\
&\quad+\left(\left\lceil\frac{x}{L}\right\rceil-n\cdot\left(\left\lceil\frac{x}{n\cdot L}\right\rceil-1\right)-1\right)\cdot\frac{L}{\rho}\Bigg) && \text{(A.13)}\\
&= 1\cdot\left(\frac{L}{C}+\sum_{j=1}^{m}\Theta^{(S_j)}\right)+\left(\left\lceil\frac{x}{L}\right\rceil-1\right)\cdot\frac{L}{\rho} && \text{(A.14)}\\
&= \frac{L}{C}+\sum_{j=1}^{m}\Theta^{(S_j)}+\left(\left\lceil\frac{x}{L}\right\rceil-1\right)\cdot\frac{L}{\rho} && \text{(A.15)}\\
&= \left\lceil\frac{x}{L}\right\rceil\cdot\frac{L}{\rho}+\sum_{j=1}^{m}\Theta^{(S_j)}-\frac{L}{\rho}+\frac{L}{C} && \text{(A.16)}
\end{aligned}$$

□

Lemma A.1.5 *Assume that the input packets of the packet stretcher have a size of L and a rate of ρ. Furthermore, assume that the output packets have a length of L'. Then, the output traffic has a rate of*

$$\rho' = \rho\cdot\frac{L'}{L} \tag{A.17}$$

Proof Assume that x *words* are received in $\Delta t \geq 0$ *sec*. Then

$$\rho = \frac{x}{\Delta t} \tag{A.18}$$

and

$$\begin{aligned}
x \textit{ input words} &\equiv \frac{x}{L} \textit{ input packets} && \text{(A.19)}\\
&\equiv \frac{x}{L}\cdot L' \textit{ output words} && \text{(A.20)}\\
&\equiv x' \textit{ output words} && \text{(A.21)}
\end{aligned}$$

So, x' *words* are sent in Δt *sec*. Then,

$$\begin{aligned}
\rho' &= \frac{x'}{\Delta t} && \text{(A.22)}\\
&= \frac{\frac{x}{L}\cdot L'}{\Delta t} && \text{(A.23)}\\
&= \frac{x}{\Delta t}\cdot\frac{L'}{L} && \text{(A.24)}\\
&= \rho\cdot\frac{L'}{L} && \text{(A.25)}
\end{aligned}$$

□

Lemma A.1.6 *Assume that the input packets of the packet stretcher have a size of L, a rate of ρ and a burstiness constraint of σ. Furthermore, assume that the output packets have a length of L' and a rate of $\rho' = \rho\cdot\frac{L'}{L}$. Finally, assume a capacity of C. Then, the output traffic has a burstiness constraint of*

$$\sigma' = \frac{\sigma}{1-\frac{\rho}{C}}\cdot\frac{L'}{L}\cdot\left(1-\frac{\frac{L'}{L}\cdot\rho}{C}\right) \tag{A.26}$$

Proof Assume $n \geq 0$ and $n \in \mathbb{R}$, then σ can be written as

$$\sigma = n \cdot L \cdot \left(1 - \frac{\rho}{C}\right) \tag{A.27}$$

Because the number of packets in the burstiness constraint on the input of the packet stretcher is equal to the number of packets in the burstiness constraint on the output of the packet stretcher, σ' can be written as

$$\sigma' = n \cdot L' \cdot \left(1 - \frac{\rho'}{C}\right) \tag{A.28}$$

Then,

$$\begin{aligned}
\sigma' &= n \cdot L' \cdot \left(1 - \frac{\rho'}{C}\right) && \text{(A.29)}\\
&= \frac{\sigma}{L \cdot (1 - \frac{\rho}{C})} \cdot L' \cdot \left(1 - \frac{\rho'}{C}\right) && \text{(A.30)}\\
&= \frac{\sigma}{1 - \frac{\rho}{C}} \cdot \frac{L'}{L} \cdot \left(1 - \frac{\frac{L'}{L} \cdot \rho}{C}\right) && \text{(A.31)}
\end{aligned}$$

□

Lemma A.1.7 *Assume an LR server with a capacity of C. Then, Round Robin Time based is an LR server with latency*

$$\frac{F - \phi + L_i}{C}$$

Proof Let $W_i(t_1, t)$ denote the number of service in *words* session i receives by the *LR* server during the interval (t_1, t), for $t_1 \leq t$. A backlogged period for session i is any period of time during which traffic belonging to that session is continuously queued in the server. In the worst case, a packet of session i is transmitted at the end of the round. Let $t_1, ..., t_k$ denote the ending times of the k rounds, $k \geq 1$ and $k \in \mathbb{N}$, after the beginning of a backlogged period at time t_0. Let t be the time during the k^{th} round after the beginning of a backlogged period for session i that a packet of session i is being serviced. Assume there are V sessions sharing the *LR* server. Then,

$$\begin{aligned}
W_i(t_0, t) &= W_i(t_0, t_{k-1}) + W_i(t_{k-1}, t) && \text{(A.32)}\\
&= \max\left\{0, (k-1) \cdot \phi\right\} + 0 && \text{(A.33)}\\
&= \max\left\{0, (k-1) \cdot \phi\right\} && \text{(A.34)}
\end{aligned}$$

Since session i is continuously backlogged,

$$t_{k-1} - t_0 \leq (k-1) \cdot \frac{F}{C} \tag{A.35}$$

From equations A.34 and A.35, and because $\phi = \frac{F}{V}$ and $\rho_i \leq \frac{C}{V}$

$$\begin{aligned} W_i(t_0,t) &\geq \max\left\{0, (t_{k-1}-t_0)\cdot\phi\cdot\frac{C}{F}\right\} && \text{(A.36)} \\ &= \max\left\{0, (t_{k-1}-t_0)\cdot\frac{F}{V}\cdot\frac{C}{F}\right\} && \text{(A.37)} \\ &= \max\left\{0, (t_{k-1}-t_0)\cdot\frac{C}{V}\right\} && \text{(A.38)} \\ &\geq \max\left\{0, \rho_i\cdot(t_{k-1}-t_0)\right\} && \text{(A.39)} \end{aligned}$$

Furthermore, at time t

$$\begin{aligned} t &\leq t_{k-1} + \frac{\sum_{n=1,n\neq i}^{V}\phi_n + L_i}{C} && \text{(A.40)} \\ t &\leq t_{k-1} + \frac{F-\phi+L_i}{C} && \text{(A.41)} \\ t_{k-1} &\geq t - \frac{F-\phi+L_i}{C} && \text{(A.42)} \end{aligned}$$

Combining with equation A.39,

$$W_i(t_0,t) \geq \max\left\{0, \rho_i\cdot\left(t-t_0-\frac{F-\phi+L_i}{C}\right)\right\} \quad \text{(A.43)}$$

See equation 2.16.

□

Corollary A.1.8 *Round Robin Time based is an LR server and its latency is equal to*

$$\frac{F-\phi+L_i}{C} \quad \text{(A.44)}$$

It is required to show that the calculated latency is as tight as possible. It is sufficient to give an example. Assume that all sessions become backlogged. Furthermore, assume that all packets have size L_{max}. Session i is the last session that is serviced in the round. Because Round Robin Time based is used, session i gets serviced after

$$\frac{F-\phi}{C} \quad \text{(A.45)}$$

So, the first packet of session i is serviced after

$$\frac{F-\phi}{C} + \frac{L_i}{C} = \frac{F-\phi+L_i}{C} \quad \text{(A.46)}$$

Lemma A.1.9 *Assume an LR server with a capacity of C. Then, TDMA is an LR server with latency*

$$\frac{F-\phi_i+L_i}{C}$$

Proof Let $W_i(t_1, t)$ denote the number of service in *words* session i receives by the LR server during the interval (t_1, t), for $t_1 \leq t$. A backlogged period for session i is any period of time during which traffic belonging to that session is continuously queued in the server. In the worst case, the packets of session i are transmitted at the end of the round. Let $t_1, ..., t_k$ denote the ending times of the k rounds, $k \geq 1$ and $k \in \mathbb{N}$, after the beginning of a backlogged period at time t_0. Let t be the time during the k^{th} round after the beginning of a backlogged period for session i that the j^{th} packet, $j \geq 1$ and $j \in \mathbb{N}$, of session i is being serviced. Assume there are V sessions sharing the LR server. Then,

$$\begin{aligned} W_i(t_0, t) &= W_i(t_0, t_{k-1}) + W_i(t_{k-1}, t) && \text{(A.47)} \\ &= \max\left\{0, (k-1) \cdot \phi_i\right\} + (j-1) \cdot L_i && \text{(A.48)} \end{aligned}$$

Since session i is continuously backlogged,

$$t_{k-1} - t_0 \leq (k-1) \cdot \frac{F}{C} \tag{A.49}$$

From equations A.48 and A.49, and because $\rho_i \leq \frac{\phi_i}{F} \cdot C$

$$\begin{aligned} W_i(t_0, t) &\geq \max\left\{0, (t_{k-1} - t_0) \cdot \phi_i \cdot \frac{C}{F}\right\} + (j-1) \cdot L_i && \text{(A.50)} \\ &\geq \max\left\{0, \rho_i \cdot (t_{k-1} - t_0)\right\} + (j-1) \cdot L_i && \text{(A.51)} \end{aligned}$$

Then, at time t

$$\begin{aligned} t &\leq t_{k-1} + \frac{\sum_{n=1, n \neq i}^{V} \phi_n + j \cdot L_i}{C} && \text{(A.52)} \\ t &\leq t_{k-1} + \frac{F - \phi_i + j \cdot L_i}{C} && \text{(A.53)} \\ t_{k-1} &\geq t - \frac{F - \phi_i + j \cdot L_i}{C} && \text{(A.54)} \end{aligned}$$

Combining with equation A.51,

$$W_i(t_0, t) \geq \max\left\{0, \rho_i \cdot \left(t - t_0 - \frac{F - \phi_i + j \cdot L_i}{C}\right)\right\} + (j-1) \cdot L_i \tag{A.55}$$

The minimum value of the right-hand side of the above inequality occurs when $j = 1$.

$$W_i(t_0, t) \geq \max\left\{0, \rho_i \cdot \left(t - t_0 - \frac{F - \phi_i + L_i}{C}\right)\right\} \tag{A.56}$$

See equation 2.16.

$\square$

Corollary A.1.10 *TDMA is an LR server and its latency is equal to*

$$\frac{F - \phi_i + L_i}{C} \tag{A.57}$$

It is required to show that the calculated latency is as tight as possible. It is sufficient to give an example. Assume that all sessions become backlogged. Session i is the last session that is serviced in a round. Because TDMA is used, session i gets serviced after

$$\frac{F-\phi_i}{C} \tag{A.58}$$

Then, the first packet of session i is serviced after

$$\frac{F-\phi_i}{C}+\frac{L_i}{C}=\frac{F-\phi_i+L_i}{C} \tag{A.59}$$

Lemma A.1.11 *Assume an LR server with a capacity of C. Fixed Priority is an LR server with latency*

$$\frac{L_{max}+\sum_{j=1}^{i-1}\sigma_j}{C-\sum_{j=1}^{i-1}\rho_j}+\frac{L_i}{C} \tag{A.60}$$

Proof Let $W_i(t_1,t)$ denote the number of service in *words* session i receives by the *LR* server during the interval (t_1,t), for $t_1 \leq t$. A backlogged period for session i is any period of time during which traffic belonging to that session is continuously queued in the server. Assume that session 1 has the highest priority, session 2 the next highest priority and so on. At time t_0 the backlogged period of session i starts, $i \geq 1$ and $i \in \mathbb{N}$. Assume that it is started by the arrival of the k^{th} packet, $k \geq 1$ and $k \in \mathbb{N}$, of session i. Now, consider the n^{th} packet, $n \geq 1$ and $n \in \mathbb{N}$, of the same backlogged period (i.e. $(k+n-1)^{th}$ packet). Then, this packet starts being serviced at time D_i^{k+n-1}. Assume there are V sessions sharing the *LR* server. Please note that in the worst case the $(k+n-1)^{th}$ packet start transmission after

- the current packet is processed
- all packets of session i that arrived before the $(k+n-1)^{th}$ packet are processed
- the bursts of all sessions with a higher priority are processed
- all packets of sessions with a higher priority that arrive during this period are processed

Then,

$$D_i^{k+n-1} \leq t_0+t \tag{A.61}$$

$$t = \frac{1}{C}\cdot\left(L_{max}+\sum_{j=k}^{k+n-1}L_i+\sum_{j=1}^{i-1}\sigma_j+\sum_{j=1}^{i-1}t\cdot\rho_j\right) \tag{A.62}$$

$$t\cdot\left(1-\frac{\sum_{j=1}^{i-1}\rho_j}{C}\right) = \frac{1}{C}\cdot\left(L_{max}+n\cdot L_i+\sum_{j=1}^{i-1}\sigma_j\right) \tag{A.63}$$

$$t\cdot\left(C-\sum_{j=1}^{i-1}\rho_j\right) = L_{max}+n\cdot L_i+\sum_{j=1}^{i-1}\sigma_j \tag{A.64}$$

$$t = \frac{L_{max}+n\cdot L_i+\sum_{j=1}^{i-1}\sigma_j}{C-\sum_{j=1}^{i-1}\rho_j} \tag{A.65}$$

Consider any time $t^* \geq t_0$ during the session backlogged period. Assume that the marked packet (the $(k+n-1)^{th}$ packet) was the last packet of session i processed during the period (t_0,t^*). Then, the total service received by session i during (t_0,t^*) is

$$W_i(t_0,t^*) = n\cdot L_i \tag{A.66}$$

But, t^* is upper bounded by the time at which the $(k+n-1)^{th}$ packet of the session leaves the LR server.

$$\begin{aligned} t^* &\leq t_0 + t + \frac{L_i}{C} && \text{(A.67)} \\ t^* &\leq t_0 + \frac{L_{max} + n \cdot L_i + \sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} + \frac{L_i}{C} && \text{(A.68)} \\ \frac{n \cdot L_i}{C - \sum_{j=1}^{i-1} \rho_j} &\geq t^* - t_0 - \frac{L_{max}}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{\sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{L_i}{C} && \text{(A.69)} \end{aligned}$$

Then, because $\rho_i \leq C - \sum_{j=1}^{i-1} \rho_j$ because $C \geq \sum_{i=1}^{V} \rho_i$

$$\frac{n \cdot L_i}{\rho_i} \geq t^* - t_0 - \frac{L_{max}}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{\sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{L_i}{C} \quad \text{(A.70)}$$

Combining equation A.66 and A.70, we get

$$\begin{aligned} W_i(t_0, t^*) &= n \cdot L_i && \text{(A.71)} \\ &\geq \rho_i \cdot \left(t^* - t_0 - \frac{L_{max}}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{\sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{L_i}{C} \right) && \text{(A.72)} \\ &= \rho_i \cdot \left(t^* - t_0 - \frac{L_{max} + \sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{L_i}{C} \right) && \text{(A.73)} \\ &= \max \left\{ 0, \rho_i \cdot \left(t^* - t_0 - \frac{L_{max} + \sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} - \frac{L_i}{C} \right) \right\} && \text{(A.74)} \end{aligned}$$

See equation 2.16.

□

Corollary A.1.12 *Fixed Priority is an LR server and its latency is equal to*

$$\frac{L_{max} + \sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} + \frac{L_i}{C} \quad \text{(A.75)}$$

It is required to show that the calculated latency is as tight as possible. It is sufficient to give an example. Assume that session 1 has the highest priority, session 2 the next highest priority and so on. Assume the LR server is processing a packet of size L_{max} and that all sessions with a higher priority than session i send their bursts. Furthermore, assume that currently a packet of lower priority than session i is processed. A rate of $\sum_{j=1}^{i-1} \rho_j$ has to be reserved for packets that arrive during processing the current packet and the bursts. Then, the first packet of session i is processed after

$$\frac{L_{max} + \sum_{j=1}^{i-1} \sigma_j}{C - \sum_{j=1}^{i-1} \rho_j} + \frac{L_i}{C} \quad \text{(A.76)}$$

Lemma A.1.13 *Assume a session with parameters (σ, ρ) and packet size L. Then, the delay for x words in a chain of m LR servers, with a total latency of $\sum_{j=1}^{m} \Theta^{(S_j)}$ and a capacity of C, including receiving the first packet with regulator is equal to the delay without a regulator.*

Proof The input can be characterized by $R(t)$ (see equation 2.5)

$$\int_{t_1}^{t_2} R(t)dt \leq \sigma + \rho \cdot (t_2 - t_1) \tag{A.77}$$

Then, without a regulator, the total delay for x *words* including receiving the first packet is upper bounded by (see lemma A.1.2)

$$\left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} \tag{A.78}$$

Assume that a regulator is added and the output of the regulator has a burstiness constraint of σ'. Then, according to section 3.2.5 and equation 2.10

$$\begin{aligned} d_j &= \frac{1}{\rho} \cdot \max\left\{0, \left(Q_\rho(R)(s_j) - \sigma'\right)\right\} && \text{(A.79)} \\ Q_\rho(R)(t) &= \max_{t_1: 0 \leq t_1 \leq t} \left\{ \int_{t_1}^{t} R(t)dt - \rho \cdot (t - t_1) \right\} && \text{(A.80)} \end{aligned}$$

Please note that d_j represents the delay of the j^{th} packet in the regulator, that arrives at s_j. Then,

$$\begin{aligned} Q_\rho(R)(t) &= \max_{t_1: 0 \leq t_1 \leq t} \left\{ \int_{t_1}^{t} R(t)dt - \rho \cdot (t - t_1) \right\} && \text{(A.81)} \\ &\leq \max_{t_1: 0 \leq t_1 \leq t} \left\{ \sigma + \rho \cdot (t - t_1) - \rho \cdot (t - t_1) \right\} && \text{(A.82)} \\ &= \sigma && \text{(A.83)} \\ d_j &= \frac{1}{\rho} \cdot \max\left\{0, \left(Q_\rho(R)(s_j) - \sigma'\right)\right\} && \text{(A.84)} \\ &\leq \frac{1}{\rho} \cdot \max\left\{0, \left(\sigma - \sigma'\right)\right\} && \text{(A.85)} \\ &= \frac{\sigma - \sigma'}{\rho} && \text{(A.86)} \end{aligned}$$

At most $\sigma - \sigma'$ *words* are in the regulator. Then, the total delay for $x - (\sigma - \sigma')$ *words* including receiving the first packet is upper bounded by

$$D' \leq \left\lceil \frac{x - (\sigma - \sigma')}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} \tag{A.87}$$

Then, the total delay with regulator is

$$\begin{aligned} D &= d_j + D' && \text{(A.88)} \\ &\leq \frac{\sigma - \sigma'}{\rho} + \left\lceil \frac{x - (\sigma - \sigma')}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} && \text{(A.89)} \\ &\leq \left\lceil \frac{x}{L} \right\rceil \cdot \frac{L}{\rho} + \sum_{j=1}^{m} \Theta^{(S_j)} - \frac{L}{\rho} + \frac{L}{C} && \text{(A.90)} \end{aligned}$$

So, the regulator does not change the total delay. □

Lemma A.1.14 *Assume an LR server, with a capacity of C, receives a session with parameters (σ_{in}, ρ) and packet size L. Then, the burstiness constraint of the output (i.e. σ_{out}) of the LR server, with a latency of Θ and a minimum latency of Θ_{min}, is*

$$\sigma_{out} \leq \sigma_{in} + \rho \cdot \Theta - \rho \cdot \Theta_{min} \tag{A.91}$$

Proof The input is token-bucket-shaped. Let $c(t)$ be the number of tokens at time t in the token bucket shaping the incoming traffic. Let $Q(t)$ be the amount of backlogged data in the *LR* server at time t. Then, the maximum possible backlog is

$$c(t) + Q(t) \tag{A.92}$$

But, the maximum backlog is upper bounded by

$$\sigma_{in} + \rho \cdot \Theta \tag{A.93}$$

So,

$$c(t) + Q(t) \leq \sigma_{in} + \rho \cdot \Theta \tag{A.94}$$

The arrivals $A(t_1, t)$, for $t_1 \leq t$, of the session cannot exceed the number of tokens in the token bucket at time t_1, plus the amount of data arrived in the interval (t_1, t), minus the number of tokens remaining in the bucket at time t.

$$A(t_1, t) \leq c(t_1) + \rho \cdot (t - t_1) - c(t) \tag{A.95}$$

A packet requires at least Θ_{min} *sec* processing time. Therefore, the service offered to the session in the interval (t_1, t) is

$$\begin{aligned}
W(t_1, t) &= Q(t_1) + A\Big(t_1, t - \Theta_{min}\Big) - Q(t - \Theta_{min}) && \text{(A.96)}\\
&\leq Q(t_1) + A\Big(t_1, t - \Theta_{min}\Big) && \text{(A.97)}\\
&\leq Q(t_1) + c(t_1) + \rho \cdot \Big(t - \Theta_{min} - t_1\Big) - c(t - \Theta_{min}) && \text{(A.98)}\\
&= Q(t_1) + c(t_1) - \rho \cdot \Theta_{min} + \rho \cdot (t - t_1) - c(t - \Theta_{min}) && \text{(A.99)}\\
&\leq \sigma_{in} + \rho \cdot \Theta - \rho \cdot \Theta_{min} + \rho \cdot (t - t_1) - c(t - \Theta_{min}) && \text{(A.100)}\\
&\leq \Big(\sigma_{in} + \rho \cdot \Theta - \rho \cdot \Theta_{min}\Big) + \rho \cdot (t - t_1) && \text{(A.101)}\\
\sigma_{out} &\leq \sigma_{in} + \rho \cdot \Theta - \rho \cdot \Theta_{min} && \text{(A.102)}
\end{aligned}$$

Therefore

$$\sigma_{out} \leq \sigma_{in} + \rho \cdot \Theta - \rho \cdot \Theta_{min} \tag{A.103}$$

□

Traffic classes

B.1 Introduction of traffic classes

In chapter 4, we described how to calculate the maximum delay and the required buffer sizes for a communication infrastructure and a set of traffic streams. The parameters that we use to characterize traffic are

- rate (ρ)
- burstiness constraint (σ)
- packet size (L)
- pipeline degree (n)

Using these parameters, the traffic streams can be characterized. Furthermore, we use D^{max} to express the amount of time available to send x *words*.

The literature gives several traffic classifications. In this appendix, we discuss two traffic classifications and explain the applicability of the traffic parameters of our performance analysis method.

B.2 *ATM*-classes

One way to classify traffic is to look at the characteristics of the traffic. For ATM[1] several classes are defined (see [15] and [24]):

- Constant Bit Rate (CBR)
- Variable Bit Rate - Real-Time (VBR-RT)
- Variable Bit Rate - Non-Real-Time (VBR-NRT)
- Available Bit Rate (ABR)
- Unspecified Bit Rate (UBR)

The first category (CBR) is defined for predictable traffic with a static rate. CBR is intended to support Real-Time applications requiring tightly constrained delays (like video applications). So, for this traffic class we have to select at least the static rate.

The Real-Time VBR service category is intended for traffic with tightly constrained delays. Low latencies and small delays are required. This class is used by Real-Time applications. The subsystems send their data at a rate which varies over time. Therefore, the traffic can be seen as bursty.

The Non-Real-Time VBR category is intended for traffic with a variable rate. Furthermore, it has weaker requirements on latencies and on delays than the previous two traffic classes. This class is intended for Non-Real-Time applications which have bursty traffic characteristics.

1. Asynchronous Transfer Mode

ABR is intended for traffic characteristics that may change over time. It does not require a bounding on the delays experienced by a given traffic session. ABR is not intended to support Real-Time applications. This traffic has loose requirements on the latencies.

Finally, the Unspecified Bit Rate service category is intended for traffic sessions that do not require tightly constrained delays. UBR does not specify traffic related service guarantees. It is intended for Non-Real-Time applications. Therefore, it can be seen as traffic with the lowest priority.

B.2.1 Applicability of *ATM*-classes

The *ATM*-classes described above are applied to the traffic parameters of the performance analysis method described in chapter 4.

In table B.1 an overview is given of the relation between the *ATM*-classes and the traffic parameters of the performance analysis method.

	σ	ρ	L	n	D^{max}
CBR	low	-	-	-	low
VBR-RT	high	-	-	-	low
VBR-NRT	high	-	-	-	average
ABR	-	low	-	-	high
UBR	-	low	-	-	high

Table B.1: Overview of the relation between the *ATM*-classes and the traffic parameters of the performance analysis method

CBR is defined for traffic with a constant bit rate. It is predictable traffic (i.e. the traffic is not bursty), so σ can be small. Because it is intended to support Real-Time applications requiring tightly constrained delays, D^{max} will be small.

VBR-RT has a variable rate. Because the traffic is bursty, we expect that σ is larger than for CBR. VBR-RT is intended for traffic with tightly constrained delays, so D^{max} will be small.

VBR for Non-Real-Time applications also represents bursty traffic, so we expect a high value for σ. Furthermore, because the deadlines are less tight than for CBR and VBR-RT, D^{max} will be average.

ABR does not require constrained delays. Therefore, ρ will be smaller than for the traffic classes above. For the same reason, D^{max} will get a high value.

Finally, UBR also does not require constrained delays. It has no traffic related guarantees. Therefore, ρ will get a low value and D^{max} will be high.

As has become clear, all traffic classes can be expressed in our performance analysis method using the same set of parameters. Often, no information is available for a few parameters.

B.3 Traffic classes

In [24], traffic classes for programmable processors have been defined according to a set of traffic attributes.

The first attribute is "cacheability". This attribute is used to specify whether it is possible to cache the data. It indicates whether it has benefits to cache. The second attribute is "spatial distribution". This attribute indicates whether the addressing is predictable. If it is predictable, prefetching is possible. "Temporal distribution" is related to the burstiness of the traffic. Finally, the "latency sensitivity" indicates whether the deadline is hard or soft. It the deadline is hard, it also indicates whether the deadline is for a single transaction or for a set of transactions.

Then, the traffic attributes are defined as follows [24]:

- Cacheability
  - 0: Should not be cached
  - 1: Permitted to be cached
  - 2: Benefits from being cached
- Spatial distribution (addressing)
  - 0: Unpredictable
  - 1: Predictable (can pre-fetch)
  - 2: Regular
- Temporal distribution
  - 0: Unpredictable
  - 1: Predictable
  - 2: Regular (non-bursty)
- Latency sensitivity
  - 0: Insensitive (best effort)
  - 1: Hard constraint for set of transactions
  - 2: Hard constraint for individual transactions

Using these attributes, traffic classes for programmable processors have been defined in [24] (see table B.2).

	Characteristics			Latency sensitivity	
	Cacheability	Spatial distribution	Temporal distribution	Request & response	Only request
Instruction fetch (cached)	2	1	0	1	N/A
Scratch data (local variables)	2	0	0	1	0
Streaming data	2	2	1	1	0
Register I/O (device programming)	0	0	0	1	0
Coherency traffic (messages)	0	0	0	1	0
Interrupts	0	0	0	N/A	2

Table B.2: Traffic classes for programmable processors [24]

B.3.1 Applicability of traffic classes

The traffic attributes as described above are applied to the traffic parameters of the performance analysis method of chapter 4.

In the performance analysis method, it is not possible to express the possibility of caching. The reason for this is simple. The performance analysis method determines an upper bound for the delays. Caches are ignored, because the worst case scenario is based upon cache misses for all memory accesses.

Spatial distribution is not directly related to a traffic parameter of the performance analysis method. But for various situations, there is an indirect relation with the pipeline degree. In case the addressing is known in advance, the pipeline degree can be higher. This is the case, because if some addresses are known in advance, those packets can already be sent.

The burstiness constraint σ is related to the temporal distribution. If the traffic is regular, the value for σ can be low. If the traffic is unpredictable, the value for σ should be higher.

Latency sensitivity is related to ρ, n and D^{max}. In case of best effort traffic, no traffic related service guarantees are specified. Therefore, the delay of the traffic is not important. So, ρ will be small and D^{max} will be large. In case of hard constraints on the latency, ρ will be larger, to reduce the delay. Furthermore, D^{max} will be smaller. The difference between a set of transactions and individual transactions can be expressed using n. In case of an individual transaction, n is small. Otherwise, n will have a higher value.

In table B.3 an overview is given of the relation between the traffic classes for programmable processors and the traffic parameters of the performance analysis method.

	σ	ρ	L	n	D^{max}
Instruction fetch (cached)	high	-	-	low	low
Scratch data (local variables)	high	-	-	low	low
Streaming data	average	-	-	average	low
Register I/O (device programming)	high	-	-	low	low
Coherency traffic (messages)	high	-	-	low	low
Interrupts	high	-	-	low	low

Table B.3: Overview of the relation between the traffic classes for programmable processors and the traffic parameters of the performance analysis method

For programmable processors, the number of outstanding requests is limited. Therefore, for all traffic classes n will have a low value, except for streaming data. This is the case, because streaming data has a regular spatial distribution. Therefore, the pipeline degree can have an average value.

Instruction fetch has an unpredictable temporal distribution. Therefore, σ will be high. Furthermore, because the deadlines are strict, D^{max} will be small.

For scratch data we expect irregular traffic, because of the unpredictable temporal distribution. So, σ should be high. Because of the strict latency requirements, a low value for D^{max} is expected.

Streaming data has a predictable temporal distribution. Therefore, σ will be smaller than for the other traffic classes. Because of the strict latency requirements, a low value for D^{max} is expected.

Register I/O, coherency traffic and interrupts have an unpredictable temporal distribution. Therefore, σ will be high. Furthermore, because the deadlines are strict, D^{max} will be small.

As indicated above, it is hard to make a clear distinction among the different traffic classes in this section. Often, no information is available for several parameters. Please note that still the same set of parameters is used and that all traffic classes can be expressed by the traffic parameters of the performance analysis method.

B.4 Conclusion

As has become clear in the previous sections, there is generally not a direct relation between the different traffic classifications of sections B.2 and B.3 and the traffic parameters of our performance analysis method. The different traffic classifications only define some coarse bounds on the values of ρ, σ, D^{max} and n. But, all traffic classes defined are expressible by the traffic parameters of our performance analysis method.

Mathematica model

C.1 Model

To perform the performance analysis method, a Mathematica model has been created. In this appendix, we will explain how it works.

First, the input has to be defined. The input consists of the following items (see table C.1).

Variable	Explanation
RhoIn	The rate per session
SigmaIn	The burstiness constraint per session
DegreeIn	The pipeline degree per session
LPacket	The packet size per session
Regulator	The option whether a regulator is used per session
Selection	The option whether a particular session is used
DPropagation	The propagation delay per session
Xtotal	The number of *words* to determine the total delay per session
DelayTogether	A number per session, to combine sessions (identical number indicates a combination, needed for a request-response stream)
Deadline	The deadline per (combined) session
ReadMem	The option whether a session is a read request session
Subsystems	The name per subsystem
From	The source per session (number refers to *Subsystems*)
Targets	The destination per session per scheduler (number refers to *Subsystems*)
R	The maximum rate of all links and schedulers
SchedulerType	The arbitration policy per scheduler, the following policies are used: 0: Virtual Clock 3: Deficit Round Robin 4: Weighted Round Robin 5: Round Robin Time Based 6: Round Robin Packet Based 7: Fixed Priority 8: TDMA 9: No policy (direct connection)
TDMA	The number of packets per session in the TDMA wheel per scheduler
PriorityList	The priority per session, only used when Fixed Priority is the arbitration policy ("1" indicates the highest priority, "2" indicates the next highest priority and so on)

S	The worst case number of memory cycles needed for a request per session (including e.g. precharge and activate)
Se	The effective number of memory cycles needed for a request per session
$MemControl$	The option whether a scheduler is a normal scheduler or a memory controller
Lc	The size of the fixed packet (cell) for Weighted Round Robin (see section 3.3.5)

Table C.1: Input overview of the Mathematica model

Using this information, the Mathematica model can start calculating the delays and the queue sizes.

First of all, the length of the input is checked. After that, it is checked if the maximum rate of the schedulers is high enough to process the sessions (i.e. $\sum_{i=1}^{V} \rho_i \leq C$) and all constraints are satisfied. This is done for each scheduler individually. In case that the scheduler is a memory controller, the ρ's are adapted using a packet stretcher (see section 4.4.3). If the maximum rate is not high enough, the calculation will stop. Otherwise, the model will start calculating the delay and the queue sizes.

Finally, the output of the Mathematica model is generated. The generated report consists of a few parts. First of all, an overview of the input is given (see table C.2).

Name	Explanation
$\#sessions$	The number of sessions
$\#schedulers$	The number of schedulers
$Rate$	The maximum rate of all links and schedulers
$Schedulers$	The sum of the ρ's per scheduler and the maximum utilization per scheduler
$Memory$	The sum of the ρ's per memory controller and the maximum utilization per memory controller

Table C.2: Output overview of the Mathematica model

Then, an overview of the size of the regulators is given. After that, for each scheduler the following information is output (see table C.3).

Name	Explanation
$Overview$	An overview of the characteristics per input session
$Policy$	The selected arbitration policy of the scheduler
$Phis$	ϕ_i per session (see sections 3.3.5 and 4.5)
$Frame$	F (see sections 3.3.5 and 4.5)
$Theta$	Θ_i, $\sum_{j=1}^{k} \Theta_{min,i}^{(S_j)}$ and $\sum_{j=1}^{k} \Theta_i^{(S_j)}$ up to this scheduler per session
$Max\ delay$	D^{total} up to this scheduler per session
$Max\ queue$	$Q^{scheduler}$ per session and the total required queue size

Table C.3: Output overview per scheduler

Finally, a total overview is generated, using the information of all schedulers.

- Overview of the arbitration policies
- Overview of the input sessions
- Overview of the sessions (e.g. $\sum_{j=1}^{m} \Theta^{(S_j)}$, $\sum_{j=1}^{m} \Theta_{min}^{(S_j)}$ and pipeline degree per session)
- Overview of the delay (D^{total}, D^{max} and whether the deadlines are met)
- Overview of the required queues

C.2 Graphical interface

To improve the usability of the Mathematica model, a "graphical interface" has been created (see figure C.1). The interface can be started by executing `StartConsole`. Then, the following buttons are created (see table C.4).

Button	Explanation
New notebook	To generate a new notebook sheet
Reset Rhos	To set all ρ's to $RhoIn$ (see table C.1)
Reset Degree	To set all pipeline degrees to one
Calculate Auto	To start the performance analysis method (a report is generated)
Set Rate	To set the maximum rate of all links and schedulers
Increase Rhos TDMA	To set the ρ's to the service rates of a scheduler
Increase Degrees	To increase the pipeline degree if the deadline is not met
Save	To save the output
Save Auto	To save the output automatically
Calculate	To start the performance analysis method, after setting a few parameters
Find Next	To determine the next TDMA-wheel, after setting a few parameters
Find Next Auto	To determine the next TDMA-wheel, using standard values for the parameters
Find All	To determine solutions for several rates and TDMA-wheels, after setting a few parameters
Find All Auto	To determine solutions for several rates and TDMA-wheels, using standard values for the parameters

Table C.4: Graphical interface overview of the Mathematica model

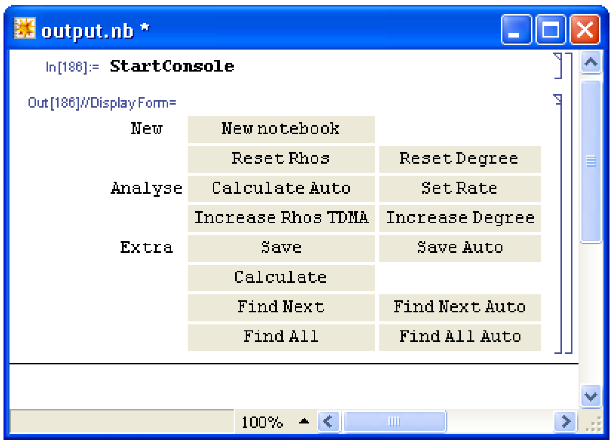


Figure C.1: Graphical interface of the Mathematica model

The pseudo code for the buttons can be found in listing C.1. The following functions are defined:

- `Calculate(b)` starts the performance analysis method and generates a report if b is true
- `ResetRho()` sets all ρ's to the original $RhoIn$ (see table C.1)
- `ResetDegree()` sets all n_i's to 1
- `ResetTDMAwheel()` sets all w_i's to 1
- `DetermineRhoTDMA(j)` sets all ρ's to the maximum service rates (i.e. $\frac{\phi_i}{F} \cdot C$) of scheduler j (see section 4.5)
- `DetermineDegree(m)` increases n_i until the deadline of session i is met or $n_i = m$
- `DetermineNextTDMA(maxn, maxDegree, nrScheduler)` determines the next TDMA wheel by
  - increasing w_i if session i has the largest pipeline degree of all sessions
  - increasing w_i until the service rate of session i is at least equal to ρ_i or w_i reaches the maximum allowed value

  this is repeated until a solution has been found with a maximum of $maxn$ attempts
- `FindSolutions(maxn, maxDegree, nrScheduler)` searches for solutions by setting the maximum rate of the schedulers and calling `DetermineNextTDMA` a number of times. This is repeated for a list of maximum rates. After that, the best solution per maximum rate and per pipeline degree is output

Please note that the optimal solution does not have to be found.

Listing C.1: Pseudocode of the Mathematica model

```
void Calculate (bool b)
#Execute performance analysis method
#if b, then print output

void ResetRho ()
#Set Rho[] to RhoIn[]

void ResetDegree ()
#Set n[] to 1

void ResetTDMAwheel ()
#Set w[] to 1

void DetermineRhoTDMA (int j)
#Set Rho[] to max service rate of scheduler j

void DetermineDegree (int m)
#Increase n[i] until session i meets deadline or n[i] = m

void DetermineNextTDMA (int maxn, int maxDegree, int nrScheduler) {
#Find next TDMA wheel
#Max allowed w[i] is set to 20
  nr := 0; continue := true;
  while (nr < maxn & continue) {
    increase w[i] if n[i] = max[n];
    ResetRho();ResetDegree();
    increase w[i] until (Rho[i] <= Phi[i]/F*C or w[i] > 20);
    DetermineRhoTDMA(nrScheduler);
    DetermineDegree(maxDegree);
    Calculate(false);
    if (Deadlines & Constraints satisfied)
        {continue := false;}
    else {nr++;}
  }
}

void FindSolutions (int maxn, int maxDegree, int nrScheduler) {
#Find solutions for range of capacities
#Number of iterations is 10
  RateList = {150, 167, ..., 400};
  nr := 1;
  while (nr <= Length[RateList])
    Rate := RateList[nr];
    ResetTDMAwheel();
    nr2 := 1;
    ResetRho();ResetDegree();DetermineRhoTDMA(nrScheduler);
    DetermineDegree(maxDegree);
    Calculate(false);
    if (Deadlines & Constraints satisfied){Store solution;}
    while (nr2 <= 10){
        DetermineNextTDMA(maxn, maxDegree, nrScheduler);
        if (Deadlines & Constraints satisfied){Store solution;}
        nr2++;
    }
    nr++;
  }
  output best solution for each maximum rate and for each pipeline degree;
}
```