Data-flow based temporal analysis for TDM arbitration

Lele, A.

Award date:
2011

Data-flow based Temporal Analysis for TDM Arbitration

Alok Lele, B.E.
0755956

Master Thesis

Eindhoven University of Technology
Department of Mathematics and Computer Science
Chair of Systems Architecture and Networking

Supervisor:
Dr. ir. P. J. L. Cuijpers
Associate Professor
P.J.L.Cuijpers@tue.nl

Tutor:
Dr. ir. O. Moreira
Principal DSP Engineer
ST-Ericsson B.V.
orlando.moreira@stericsson.com

Abstract

An increasing number of embedded applications is being implemented on heterogeneous Multi Processor Systems on Chip [MPSoC]. Embedded streaming applications, such as multi-radio modems, are a specific category of embedded applications that typically perform a sequence of transformations (data processing) on one or more streams of input data. These applications often need to satisfy hard real-time requirements such as guaranteed minimum throughput and maximum latency. To establish hard real-time guarantees for each application independently on a shared resource MPSoC, we need to bound the influence of resource sharing among the jobs in these applications. This can be done by scheduling the jobs of multiple applications such that each job has a separate execution budget. Real-time schedulers based on budget scheduling, such as Time Division Multiplexing (TDM), bound the influence of resource sharing independently of the execution times and execution rates of the jobs of different applications. One can establish hard real-time guarantees of individual applications based on the worst-case temporal analysis of the application under the effect of resource arbitration.

TDM arbitration allows resource sharing amongst jobs of different applications where each application may have independent hard real-time requirements. The analysis of temporal behavior of an entire application, based on the worst-case temporal behavior of its jobs, is used to establish hard real-time guarantees such as minimum throughput and maximum latency. The graph obtained by representing each job in an application with a response model that represents the worst-case temporal behavior of the modeled job, we obtain the worst-case temporal behavior for the entire application.

Data-flow modeling is extensively used in literature to model the temporal behavior of embedded applications. The data-flow paradigms fit well with these application domains, as they can represent the inherent concurrency, the pipelined behavior, and data-oriented style of radio-streaming algorithms, while at the same time allowing analysis and synthesis. The existing data-flow based modeling techniques for TDM arbitration over-estimate the worst-case temporal behavior of jobs in an application. Since each job is pessimistically represented using the current modeling scheme, the resultant graph obtained by replacing each job in the application by its response model, compositionally leads to a very high over-estimation of the temporal behavior of the entire application. The

analysis of this over-estimated representation of the temporal behavior of an application will, again, provide rather pessimistic guarantees on throughput and latency which may not satisfy the application's requirement. To establish guarantees that satisfy the hard real-time requirements of an application results over-allocation of resources for the jobs in the application. Pessimism in resource allocation may lead to under-utilization of the system resources and unnecessary rejection of applications that could have been accommodated in the system. We observe that there is a need for a accurately modeling the worst-case temporal behavior of TDM arbitration and enable optimized resource allocation. Optimized resource allocation provides accurate real-time guarantees and avoids unnecessary rejection of applications that can be accommodated within the system.

We propose three models in this thesis, (a) the Latency-Phased-Rate (LPR) model, the Latency-Cyclic-Rate (LCR) model and the Multi-Rate Data-Flow (MRDF) model.

- **LPR-model:** TDM can be shown to have a cyclic pattern over a fixed number of replenishment periods. We model this pattern using single-rate data-flow by capturing the execution behavior for each replenishment period separately.

- **LCR-model:** Instead of defining the cyclic patters over a fixed number of periods, the LCR-model is used to define the cyclic patterns in terms of the job iterations. The execution rate of each iteration is individually modeled.

- **MRDF-model:** The MRDF-model is a multi-rate model that describes the actual provisioning and consumption of the allocated resource by each individual iterations of the job.

We demonstrate that each of the proposed models are conservative in defining the worst-case temporal behavior of jobs scheduled using TDM arbitration. We show this by demonstrating that, even in the worst-case, the modeled finish time of each job iteration using the proposed models, is an upper-bound to the actual finish time of the job iterations. Unlike some existing models, we do not make any assumptions on the characteristics of TDM arbitration, such as on the size of the slice allocated to a job, making our models more generic. Furthermore, we enumerate the model characteristics per proposed model with respect to the existing state-of-the-art models, namely the LR-model [1], and Staschulat's LR-model [2]. With respect to the existing state-of-the-art models, we demonstrate that the models proposed in this thesis are more accurate in defining the worst-case temporal behavior of TDM arbitration.

We also address various other issues regarding each of the models. We address the complexity of our models in terms of size and analysis time. At the end of our dissertation, we present several challenging topics for future work. We address the issue of extending the proposed models, for a broader class of budget schedulers and also provide an illustrative example. Lastly, we address the issue of modeling a special case of TDM arbitration, i.e. TDM with static-ordering.

Acknowledgements

This thesis is written as part of the master project that I have performed at the chair of System Architecture and Networking at Eindhoven University of Technology, the Netherlands, in cooperation with ST-Ericsson BV, High-Tech Campus, Eindhoven, the Netherlands. Upon successfully finishing the master project comes the conclusion of my study Computer Science and Engineering at Eindhoven University of Technology.

I have enjoyed working on this project, mainly because of the the freedom I was given to explore into data-flow to design response models that perform better than the state-of-the-art. Although this was not completely in my initial project description, my inclination towards this field was encouraged by my supervisors. Therefore, I express my gratitude to Orlando Moreira for initiating this project and giving magnificent support along the route. Furthermore, I thank all the folks at ST-Ericsson BV, for the pleasant working and social environment.

I would like to thank Pieter Cuijpers for supervising my work from as the Eindhoven University of Technology , for providing great advice and giving numerous amounts of new insights. I especially want to thank both Pieter and Orlando for teaching me how to formalize different models and also the various aspects of writing a technical paper. I would also like to thank Reinder J. Bril for his helpful discussions and suggestions. Finally, a big thanks goes to my friends and fellow colleagues Shreya Adyanthaya and Ashwini Moily for proofreading my thesis.

As there is time for studying and working, a student should also spend time on social activities. I would like to thank my friends from Manipal University for providing the unforgettable moments and a great environment for social development.

Last, but absolutely not the least, I would like to thank my parents Dileep and Anjali, my sister Deepti, and my fiance Shruti for their great support and encouragement throughout my study years.

Contents

Chapter 1

Introduction

An increasing number of embedded applications is being implemented on heterogeneous Multi Processor Systems on Chip [MPSoC]. These applications often need to satisfy hard real-time requirements such as guaranteed minimum throughput and maximum latency. For cost efficiency reasons, multiple applications share resources on a single MPSoC. For instance, a multi-radio modem has to process multiple streams of input independently with each stream having its own latency and throughput requirements [3],[4]. The independent analysis of the worst-case temporal behavior of individual applications, for a given shared resource environment, allows us to establish guarantees on the minimum throughput and the maximum latency for each application. This project focuses on developing modeling techniques that accurately capture the worst-case temporal behavior of an application in a resource-shared environment.

1.1 Context

The use of embedded systems has become widespread in todays world, with devices ranging from mobile phones and cameras to control systems in nuclear power plants. An embedded system is designed to perform one or a few dedicated functions often with real-time computing constraints. Nowadays there is a visible shift from the use of simple single processor systems to complex heterogeneous multi-processor system on chip [MPSoC] devices. For cost effectiveness reasons and for greater functionality, multiple applications are mapped on a single MPSoC platform. This requires a proper resource sharing mechanism to ensure that the processing requirements of individual applications are satisfied.

Embedded streaming applications are a specific category of embedded applications that typically perform a sequence of transformations (data processing) on one or more streams

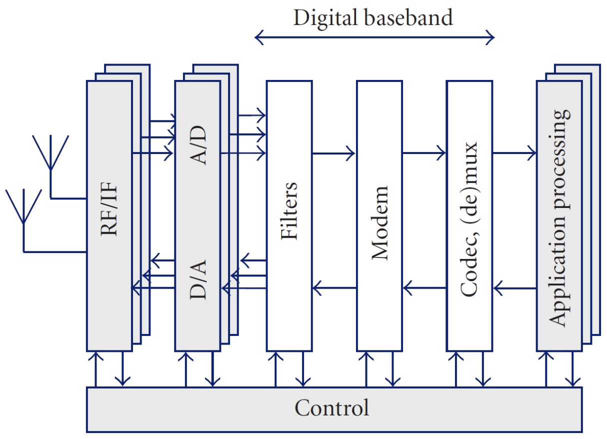

Figure 1.1: Multiple radio streams with independent hart real-time requirements running on an MPSoC (figure taken from [5])

of input data. For example, Figure 1.1 shows a crude architecture of a multi-radio modem [5] describing a sequence of jobs (data transformation processes) for the multi-radio application. A radio application, for instance, can be broadly divided into three stages, namely the filter stage, the modem stage, and the codec stage. Each stage performs a specific processing on the received input signal and produces a corresponding output.

Such applications often need to satisfy hard real-time requirements such as guaranteed minimum throughput and maximum latency. For instance, multi-radio modems have to process multiple streams of input independently with each stream having its own latency and throughput requirements [3, 4]. In Figure 1.1, we can see that there are multiple input radio streams received by the RF/IF transceiver and sent to the A/D converter. Similarly at the output side there is a processed output stream per input stream sent by the D/A converter to the RF/IF transceiver. Each input-output stream may have its own throughput and latency requirement.

A streaming application is composed of multiple jobs that communicate with each other. Each job performs a specific transformation (data processing) on the stream of input data, either from an external source or from the output stream of a previous job. Each job is mapped to a specific resource of the MPSoC, and the jobs communicate with each other through FIFO channels or buffers. Jobs mapped to the same resource require some form of resource sharing mechanism to ensure that the processing requirements of individual jobs are satisfied.

To establish hard real-time guarantees for each application independently on a shared resource MPSoC, we need to bound the influence of resource sharing among the jobs in these applications. This can be done by scheduling the jobs of multiple applications such that each job has a separate execution budget. A *Budget Scheduler* can guarantee a minimum amount of time to every scheduled job in a maximum interval. Real-time schedulers based on budget scheduling bound the influence of resource sharing independently of the execution times and execution rates of the jobs of different applications. Time Division Multiplexing [TDM] is a budget scheduler that divides a fixed time frame, referred to as replenishment period, into various slots and each slot is assigned to a single job. All iterations of a job can execute only in the time slots, referred to as slice or budget, assigned to it. A TDM scheduler is simple to implement with negligible runtime overhead, since it allows only time-driven context switches. Time-driven context switching enables the analysis of the temporal behavior of a single job independently of all the other jobs sharing the same resource [6]. Figure 1.2 illustrates TDM arbitration as the service time (shown as the pie slice) received in a fixed interval of time (visualized as one rotation of the time wheel).

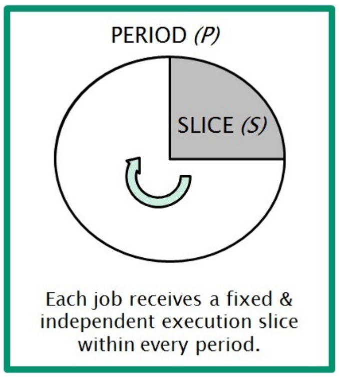
PERIOD (P)
SLICE (S)
Each job receives a fixed & independent execution slice within every period.

Figure 1.2: Time Wheel of a TDM Arbitration Scheme. Each job receives a fixed and independent execution slice per replenishment period

To establish real-time guarantees, we require some modeling mechanism that isolates the temporal behavior of applications. A streaming application can be represented as a graph of communicating single-threaded jobs [6, 7]. The vertices of the graph represent individual jobs while the edges of the graph describe the communication channels between jobs. To isolate the temporal properties of an application we transform the application graph into a *temporal analysis graph* such that each job actor in the application graph is replaced by a graph component, called response model. A response model of a job represents the worst-case temporal behavior of that job under the effect of resource arbitration. The graph obtained by replacing each job for its response model can be used for conservative temporal analysis of each application separately. That is, if we compositionally represent the worst-case temporal behavior of each job in an application, we can derive the worst-case temporal behavior of the entire application.

Data-flow models have been extensively used to represent applications graphs and response models for embedded streaming applications. Data-flow modeling paradigms fit well with these application domains, as they can represent the inherent concurrency, the pipelined behavior, and data-oriented style of radio-streaming algorithms, while at the same time allowing analysis and synthesis [9]. Figure 1.3 shows a data-flow model for a

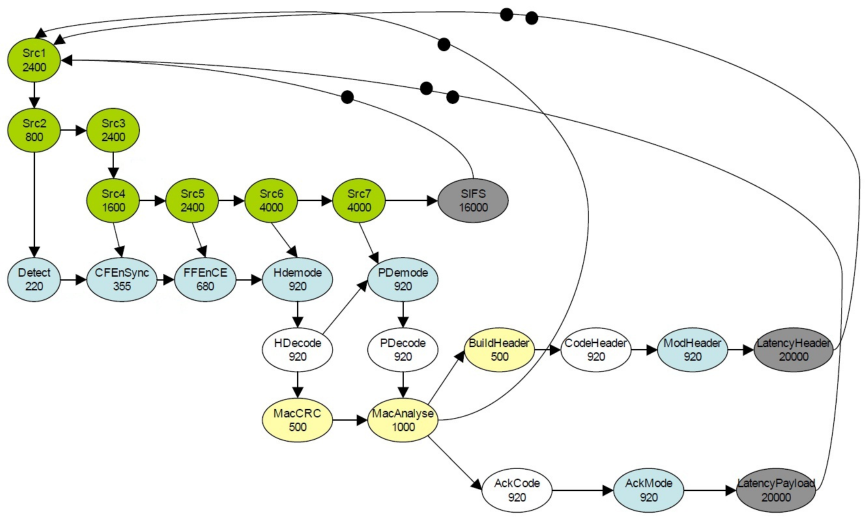

Figure 1.3: A data-flow based representation of a sample embedded streaming application (WLAN) (figure taken from [8])

wireless LAN application. In a data-flow based representation, the worst-case response time of each job in an application can, for instance, be represented as a separate actor (vertex). An application abstracted as a data-flow based temporal analysis graph where the worst-case temporal behavior of each job is represented as a separate actor, can compositionally describe the worst-case temporal behavior of the entire application. From the analysis of this derived worst-case temporal behavior of an application, we can establish some hard real-time guarantees on the application such as its guaranteed maximum latency or minimum throughput.

1.2 Problem Description

Data-flow has been extensively used in literature for modeling and analysis of applications, as it can easily isolate the temporal behavior the applications from their implementation detail. The worst-case response time of a job scheduled using TDM is the total time taken to execute a single iteration of the job after the effects of TDM arbitration (i.e. scheduling, pre-emption, etc) are taken into account. *Bekooij et al* [6] proposed a single actor response model in which each actor represents the worst-case response time of a job scheduled using TDM arbitration. But these different effects of resource arbitration need not collectively affect each iteration of a job in an application. Thus this collective worst-case response time for a single job is a highly over-estimated bound.

Wiggers et al [1] also show that the temporal behavior of a restricted class of budget schedulers can be bounded by modeling the effect of latency and rate of execution. They use a two-actor SRDF model called Latency-Rate [LR] model. The budget schedulers whose behavior can be captured using the LR-model are called *Latency-Rate Servers* [10]. It is shown in [1] that TDM arbitration also belongs to the class of latency-rate servers. The LR-model, claims to be a tight conservative approach to describe the worst-case temporal behavior of TDM arbitration. Worst-case behavior of TDM arbitration implies the worst-case temporal behavior of jobs scheduled using TDM arbitration, and we use these phrases throughout our work interchangeably. It is conservative because the model always provides an upper-bound to the worst-case temporal behavior of a job. The model claims to be tight because there are instances when the modeled bound is equal to actual worst-case temporal behavior of job execution instead of being an over-estimation of the worst-case. However, we observe that such instances are rather rare, and more often than not, the LR-model significantly over-estimates the worst-case temporal behavior of TDM arbitration.

Staschulat et al [2] proposes a LR-based model for a memory arbiter that considers the history of memory accesses. However, the model assumes that the allotted slice within one replenishment period is an integral multiple of the worst-case execution time of that job. This poses restrictions on the size of the slice allotted to the job. Such restrictions may give an accurate worst-case analysis for the obtained schedule, but the restricted slice size may itself produce poorer arbitration performance.

The state-of-the-art models [6, 8, 1] provide pessimistic estimates for worst-case temporal behavior of jobs scheduled using TDM arbitration. Since each job is pessimistically represented using the current modeling scheme, the resultant graph obtained by replacing each job in the application by its response model, compositionally leads to a very high over-estimation of the temporal behavior of the entire application. The analysis of this over-estimated representation of the temporal behavior of an application will, again, provide rather pessimistic guarantees on throughput and latency which may not satisfy the application's requirement. To establish guarantees such that they satisfy the hard real-time requirements of an application will inevitably require over-allocation of resources (larger slice sizes) for the jobs in the application. The over-estimation causes pessimistic allocation of resources to guarantee the throughput and latency requirements of the application. Pessimism in resource allocation may lead to under-utilization of the system resources and unnecessary rejection of applications that could have been accommodated in the system.

1.3 Project Goals

The focus of this project is to improve upon the existing approaches used for modeling the worst-case temporal behavior of a single job scheduled using TDM arbitration. The intention is that if the worst-case temporal behavior of each job in an application is modeled more accurately, then we can derive a more accurate estimate of the worst-case temporal behavior of the entire application. More accurate estimations enable optimized resource allocation for TDM arbitration. This in turn improves the overall system utilization and avoids unnecessary rejection of applications that can be accommodated in the system. It is also important that the modeling technique should be generic i.e. it should not make unnecessary assumptions on the characteristics of TDM arbitration. The goal of this project can be stated as "To design a data-flow based modeling technique that captures the worst-case behavior of jobs scheduled using TDM arbitration, such that:

- it provides more accurate estimation of the worst-case behavior of a TDM scheduled job as compared to the existing state-of-the-art models.
- it is conservative i.e. it will define an upper-bound to the worst-case temporal behavior of jobs scheduled using TDM arbitration.
- it is generic i.e. it should not make unnecessary assumptions on the characteristics of TDM arbitration, such as slice sizes".

1.4 Approach

The execution of a continuous sequence of consecutive job iterations on a TDM scheduled resource, can be shown to have a cyclic pattern over a fixed number of iterations. We propose to model this cyclic pattern using data-flow. This enables us to accurately specify worst-case behavior of TDM arbitration. We do not pose restrictions on the size of the slice allocated to a job. This allows us to model jobs having arbitrary executions and allotted an arbitrary TDM slice, enabling optimized resource allocation for TDM arbitration.

An alternative approach is to have a general characterization of slice consumption per replenishment period. This approach appears to be well suited for multi-rate environment (discussed in chapter 7). It models the actual consumption of each slice by individual job iterations.

1.5 Contribution

In this project, we propose three different data-flow based models to capture the worst-case behavior of TDM arbitration. We propose two single-rate data-flow models and one multi-rate data-flow model. A brief description of each model is given below:

1. The first model is a single-rate data-flow based model with an initial assumption that the execution time of a single job iteration is smaller that the size of a single slice. For this setup of TDM arbitration, we demonstrate that the model provides a conservative estimation that is tighter than the existing state-of-the-art models i.e. the LR-model [1]. We then show that we can extend this model to capture the effect of TDM arbitration for jobs with arbitrary execution times. However, when we extend this model to larger execution times, we demonstrate that the modeled estimations are comparable to the LR-modeled estimations but not always better.

2. To overcome the disadvantages of the previous model, we propose a new approach to model the cyclic pattern of execution. We define a new single-rate model that is an improvement of the previous one such that it does not make any assumption on the execution time of a single job iteration. We demonstrate that this model provides an accurate estimation of the worst-case temporal behavior of an arbitrary TDM setup.

3. Our last model is a multi-rate data-flow model that is a more generic characterization of slice consumption per replenishment period. We show that this model is well suited of multi-rate environments but can also be expanded to adapt to single-rate environments. This model intuitively provides an accurate estimate of the worst-case temporal behavior of TDM arbitration.

1.6 Outline

The remainder of this thesis is organized as follows. In chapter 2 we describe the basic concepts of data-flow modeling and data-flow based representation of embedded streaming applications. In chapter 3 we describe the concept of resource sharing via TDM arbitration. We formalize the worst-case temporal behavior of TDM arbitration and show how the existing state-of-the-art approaches conservatively model this behavior. Chapter 4 describes the inadequacy of the state-of-the-art models in accurately capturing the worst-case temporal behavior of TDM and the resulting pessimism in the overall system utilization. We then propose three different approaches in chapters 5, 6, and 7 and demonstrate the obtained improvement in accuracy. The discussion and potential future scope of the proposed approaches is presented in chapter 8. Finally, we conclude in chapter 9

Chapter 2

Single-Rate Data-Flow

In this chapter, we introduce the concepts of single-rate data flow and data-flow based modeling and analysis, which are required to understand the issues addressed in this project. The concepts presented in this chapter do not provide a complete description of the data-flow formalism, but only the aspects that are relevant to the use of data-flow for temporal analysis of resource arbitration. A detailed account of data-flow formalisms is provided in *Sriram et al* [11].

There are many flavors of data-flow formalisms used to model and analyze signal processing and multimedia streaming applications. Data-flow paradigms fit well with these application domains, as they can represent the inherent concurrency, the pipelined behavior, and data-oriented style of radio-streaming algorithms, while at the same time allowing analysis and synthesis [9]. We first provide a brief introduction to data-flow. We then explain a flavor of data-flow formalisms, namely Single-Rate Data-Flow and how it can be analyzed. We also give a brief description of some other flavors of data-flow formalism. Next, we describe how an application can be modeled as a singe-rate data-flow graph and how these application graphs can be extended to capture the effects of resource arbitration. Finally, we give a brief description of the various techniques used to analyze the obtained graphs and establish hard real-time guarantees such as the minimum throughput and maximum latency.

2.1 Introduction to Data-Flow Graphs

To start with, we define a *directed graph* as an ordered pair (V, E), where V is a set of vertices and E is a set of edges. A single edge is expressed as an ordered pair (v_1, v_2) with $v_1, v_2 \in V$. We say that an edge $e = (v_1, v_2) \in E$ is directed from v_1 to v_2. We may also say that v_1 is the source of the edge e while v_2 is the sink of the edge e. In a directed graph there may not be more than one edge with the same source and sink vertices.

A *data flow graph* is a directed graph, where the vertices (actors) represent functionally deterministic computations and edges represent FIFO queues that direct data values from the output of one computation to the input of another. The data values are represented by containers called *tokens*. The actors have the capability of performing a computation (or *firing*) by consuming tokens from the incoming edges and producing tokens on the outgoing edges. There is an initial distribution, called *delay*, of tokens defined for all the edges in a data flow graph

The *firing rule* of an actor defines what happens upon a single firing of that actor. A firing rule is defined in terms of the number of input token consumed per incoming edge, the number of outgoing tokens produced per outgoing edge, and the time (called firing time) required for a single firing of a particular actor. The firing time is defined as the time difference between consuming input tokens and producing output tokens in a single computation or firing of an actor. The number of input tokens per incoming edge, required for an actor to fire, is called the *firing condition* of that actor. Throughout this thesis, we consider the firing of data-flow actors to be *self-timed*, i.e. an actor fires as soon as its firing conditions are met.

2.2 Single-Rate Data-Flow Graphs

2.2.1 Overview

A Single Rate Data-Flow (SRDF) model, also referred to as Homogeneous Synchronous Data-Flow [12], is one of the simplest data-flow models that can be effectively used to express the temporal behavior of concurrent jobs [9]. The SRDF formalism has the advantage that it can be used for static analysis by converting the graphs into *max-plus* equations [13, 14]. Almost all the work in this project uses the SRDF formalism.

2.2.2 Formal Definition of Single-Rate Data-Flow Graphs

SRDF graphs are expressed as a 4-tuple $G = (V, E, d, t)$. Vertices of the graph are a finite set of actors V and represent deterministic computational functions (jobs). The directed edges $E = \{(v_i, v_j)|v_i, v_j \in A\}$ represent first-in-first-out communication channels. Data is transported in discrete containers (tokens). There is an initial placement of tokens across the edges, called *delay*, and defined as $d : E \to \mathbb{N}$. Every firing (execution) of an SRDF actor consumes (removes) one token from every incoming edge and produces (places) one token on every outgoing edge. This is called the *firing rule* of SRDF actors.

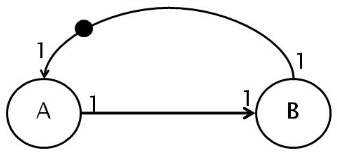

Figure 2.1: Single-Rate Data-Flow Graphs

We assume that the firing of actors is self-timed, i.e. an actor fires as soon as there is at least one token on every incoming edge of that actor. This is called the enabling condition of the actor, i.e. an actor is enabled and can fire if there is at least one token present on each incoming edge of that actor. The time between consumption and production of tokens, i.e. the start time and finish time of firing for a single actor is defined as the firing time $t(v_i)$ of actor v_i such that $t : V \to \mathbb{R}$.

2.2.3 Temporal Analysis of Single-Rate Data-Flow Graphs

The temporal behavior of single-rate data-flow actors is defined in terms of the start time and finish time of a firing of that actor. If $s(v_i, j)$ and $f(v_i, j)$ are the start time and finish time, respectively, of the j-th firing of an actor $v_i \in V$, we define the finish time $f(v_i, j)$ as $f(v_i, j) = s(v_i, j) + t(v_i)$ where $t(v_i)$ is the firing time of the actor v_i.

Since SRDF-actor firings are considered to be self-timed, the start time of an actor firing is when its firing conditions are satisfied, i.e. there is at least one token present on each incoming edge of that actor. Let us understand how the firing conditions are satisfied by considering the examples shown in Figure 2.2.

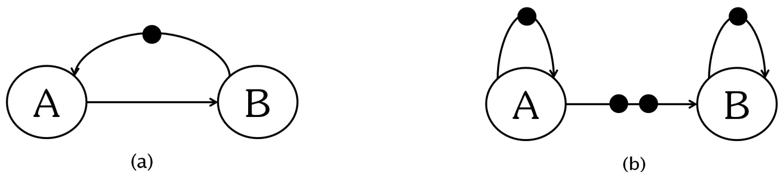

Figure 2.2: Sample SRDF graphs

Figure 2.2(a) shows a simple SRDF graph with two actors A and B and edges (A, B) and (B, A). The delay of the graph is defined such that there is a single token on the edge (B, A) i.e. $d(B, A) = 1$, while there are no tokens present on the edge (A, B) i.e. $d(A, B) = 0$. Let us now simulate the actor firings to establish a formal definition on the temporal behavior of the graph. At the start of the system, only actor A can fire, as its firing condition has been satisfied by the initial delay. The finish time of its first firing is given as $f(A, 1) = s(A, 1) + t(A) = t(A)$, as the start of the system is also the start time of the first firing of A, i.e. $s(A, 1) = 0$. As soon as actor A finishes its first firing it produces a token on the edge (A, B), thus satisfying the firing condition for actor B.

Actor B can fire as soon as there is token on its incoming edge (A, B), i.e. start time of B is expressed as $s(B, 1) = f(A, 1)$. The finish time of the first firing of actor B can now be defined as $f(B, 1) = s(B, 1) + t(B) = f(A, 1) + t(B)$. As there was only a single token on edge (B, A) at the start of the system, A can no longer fire after its first firing until a token is produced by B on the edge (A, B). Therefore, the second firing of A can only start the first firing of B has finished, i.e. $s(A, 2) = f(B, 1)$ and $f(A, 2) = f(B, 2)$. Notice that each subsequent firing of A waits for the firing of B to produce a token an vice-versa. The firing dependency of A and B is defined by the delay present on the edges connecting the two actors. We can formally define the temporal behavior for the given graph as $f(A, j) = f(B, j - 1) + t(A)$ and $f(B, j) = f(A, j) + t(B)$. We can generalize this behavior for any two actors v_1 and v_2 in an SRDF graph (V,E,d,t), where $v_1, v_2 \in V$ and $(v_1, v_2) \in E$ as:

$$f(v_2, j) = f(v_1, j - d(v_1, v_2)) + t(v_2). \tag{2.1}$$

Figure 2.2(b) shows another instance of an SRDF graph, again with two actors A and B, and three edges (A, A), (A, B), and (B, B). The delays are defined as $d(A, A) = 1$, $d(A, B) = 2$, and $d(B, B) = 1$. The firing condition for actor A is that consecutive firings are non-overlapped. In other words, A begins a firing as soon as its previous firing finishes i.e. $f(A, j) = f(A, j-1)+t(A)$. Meanwhile, actor B has two incoming edges (B, B) and (A, B) which collectively define its firing condition. The firing constraint for edge (B, B) is that consecutive firings of B have to be non-overlapped i.e. $f(B, j) \geq f(B, j-1)+t(B)$. We express the constraint as an inequality since there may be other constraints on the firing of B due to the other incoming edge (A, B). There are two tokens present on the edge (A, B) at the start of the system. This allows actor B to fire twice without any constraint due to this edge. The third firing however will be constrained such that it will require the first iteration of the A to finish and produce a token on the edge (A, B). We can generalize this as $f(B, j) \geq f(A, j - 2) + t(B)$. The firing condition of B is that it can fire as soon as all the incoming edges of B have at least one token present on them. Therefore, we can express the firing condition as a *max*-expression:

$$f(B, j) = \max(f(B, j - 1), f(A, j - 2)) + t(B) \tag{2.2}$$

The above two examples, give a brief description of the analysis of single-rate data-flow. The readers should note that, although we start by describing the temporal behavior of SRDF graphs in terms of start time and finish time of actor firings, we actually define the behavior in terms of just the finish time. The start time of an actor firing can be derived from the corresponding finish time due to the relation $f(v_i, j)$ as $f(v_i, j) = s(v_i, j)+t(v_i)$, where $v_i \in V$. Although, the temporal behavior of data-flow graphs is predominantly defined is literature using start times, we do not follow the same approach. This thesis performs analysis of the temporal behavior of graph components (i.e. our proposed models) representing the worst-case temporal behavior of a job scheduled under the effect of resource arbitration. The graph component typically has a single start actor whose incoming edge represents the arrival of a job iteration, and a single finish actor, whose firing will produce a token representing the output of the job and indicating that the current job iteration has finished. In other words, we define the temporal behavior of a the graph components in terms of the arrival time of the input tokens to the graph component and the finish times of the output tokens of the graph component. This has been explained in detail in chapter 3.

2.3 Other Flavors of Data-Flow

Till now we have talked about the single-rate data-flow formalism and its analysis. We will now give a brief description of some other flavors of data-flow. Our work does not primarily rely on these formalisms, although we later discuss the use of our proposed analysis models for resource arbitration in these data-flow formalisms as well.

2.3.1 Multi-Rate Data-Flow Graphs

Multi-Rate Data-Flow [MRDF], also called Synchronous Data-Flow [12], is a data-flow formalism that is a generalization of the SRDF formalism. It poses the restriction on the firing of actors, that the number of tokens consumed and produced for a single firing of an actor is fixed and known apriori. The number of tokens produced or consumed by each MRDF actor on each of its edges is annotated in a MRDF graph by numbers at the edge source and edge sink respectively, as shown in Figure 2.3.

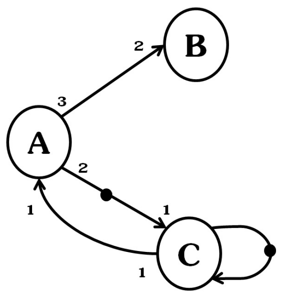

Figure 2.3: Multi-Rate Data-Flow Graphs

MRDF graphs are expressed as a 5-tuple (V, E, f, d, t). V represents the set of actors of the graph. E defines the set of directed edges in the graph expresses as $\{e = (v_1, v_2)|v_1, v_2 \in V\}$. The firing rules are defined as $f : E \rightarrow \mathbb{N} \times \mathbb{N}$, such that $f(e) = (p, c)$ defines that for each firing of the source actor of the edge e will produce p tokens on this edge, while each firing of the sink actor of the edge e will consume c tokens from this edge. There is an initial placement of tokens across the edges (delay) defined as $d : E \rightarrow \mathbb{N}$. The time between consumption and production of tokens, i.e. the start time and finish time of firing for a single actor is defined as the firing time $t(v_i)$ of actor v_i such that $t : V \rightarrow \mathbb{R}$.

2.3.2 Cyclo-Static Data-Flow Graphs

Cyclo-Static Data-Flow [CSDF] [15] is an extension to Multi-Rate Data-Flow, such that the CSDF actors can have a static-ordered list of consumption and production rates per incoming edges and outgoing edges respectively. CSDF actors may also have a static-ordered list of firing times. These static-ordered lists define a repeating sequence of the firing behavior of a CSDF actor. The work presented in this thesis does not focus on or use cyclo-static data-flow. Therefore, only a brief description of CSDF graphs is provided. Formalism on cyclo-static data-flow and its uses can be found in [15, 16, 17].

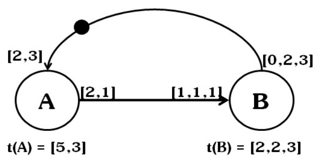

Figure 2.4: Cyclo-Static Data-Flow Graphs

2.3.3 Converting MRDF and CSDF graphs into SRDF graphs:

There are many algorithms existing in literature [11, 12, 18], to convert multi-rate and cyclo-static data-flow graphs into single-rate data-flow graph. Let us illustrate the underlying concept of converting MRDF graphs to SRDF graphs. We use this conversion technique to analyze the proposed MRDF-model in chapter 7.

To understand the conversion of MRDF graphs into SRDF graphs, we first describe the concept of a *repetition vector*. The repetition vector of a data-flow graph defines the number of firings for each actor in the graph after which the number of tokens on each edge of the graph remain unchanged [11]. In other words, the repetition vector of a graph with s actors is described using a column vector $\vec{q}$ of length s, such that if each actor i is fired a number of times equal to the i-th entry of q, then the number of tokens on each edge is the same as the initial delay defined for that edge.

Figure 2.5(a) shows a MRDF graph with two actors A and B such that for every two firing of the actor A there are three firings of the actor B. The repetition vector for this graph can be expressed as:

$$\vec{q} = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \tag{2.3}$$

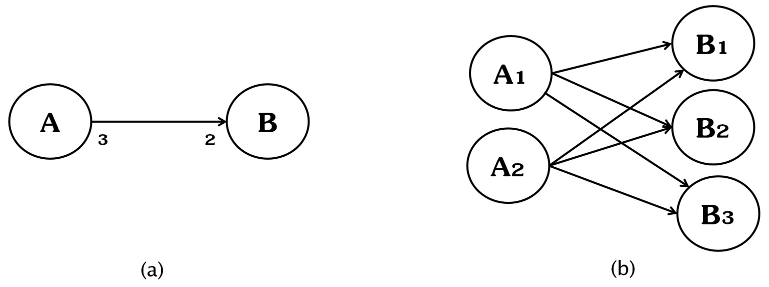

Figure 2.5: Converting a multi-rate data-flow graph to a single-rate data-flow graph

To convert this MRDF graph into an SRDF graph we replicate each actor by the number of firings specified in the repetition vector, i.e. $q(A) = 2$ and $q(B) = 3$ are the number of firings of A and B, respectively, as defined in $\vec{q}$. We now have two SRDF actors A_1, and A_2 for the single MRDF actor A such that each SRDF actor of A represents a separate firing in the repetition vector for A. Similarly, B_1, B_2, and B_3 are the SRDF actors for B.

To complete the MRDF to SRDF conversion, we must connect the single-rate actors such that the tokens produced and consumed by every firing of each actor in the SRDF graph, remains identical to the firings in the original MRDF graph [11]. Figure 2.5(b) shows the SRDF equivalent of the MRDF graph shown inf Figure 2.5(a).

2.4 Using Single-Rate Data-Flow to represent Embedded Streaming Applications

SRDF Graphs have been extensively used in literature to represent streaming applications running on an MPSoC platform [3, 4, 6, 7, 1, 8, 11]. A streaming application is made up of multiple single-threaded jobs communicating with each other. We can represent such an application as an *application graph* within the constructs of data-flow. Each job of the application is represented as a separate actor, while the communication between the jobs is represented via the edges connecting these actors. The firing time of actors describes the worst-case execution times of the corresponding jobs. For instance, Figure 2.6 shows a wireless LAN application modeled as a data-flow graph.

An application graph can be further extended to capture a variety of application details, such as constraints on buffer sizes for the communicating jobs [9], [19], [20], and representing the effect of resource arbitration [6, 7, 1, 2, 8, 11, 21, 22, 23].

This thesis focuses on modeling only the effect of resource arbitration, particularly arbitration via Time Division Multiplexing [TDM]. Chapter 3 provides a detailed description of modeling and analysis of TDM arbitration using data-flow. Let us consider, for now,

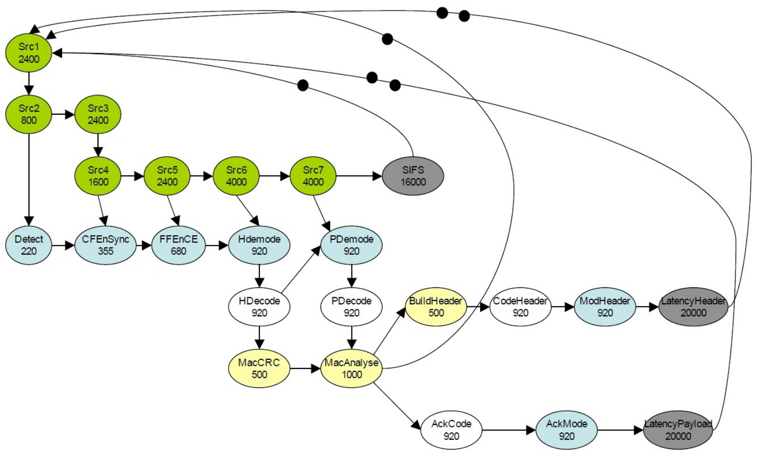

Figure 2.6: Representing a WLAN application using a data-flow graph (figure taken from [8]. Same as figure 1.3)

that the effect of TDM arbitration on a job can be represented via a SRDF-based response model of that job. This response model defines a upper-bound on the worst-case temporal behavior of a job scheduled using TDM. Figure 2.7 shows an arbitrary job x in an application, represented as a single-actor, being replaced by a response model consisting of two actors x_L and x_R used to describe the worst-case temporal behavior of resource arbitration on the job x. This response model in commonly called the Latency-Rate [LR] model [10]. The features of this model have been discussed in detail in chapter 3.

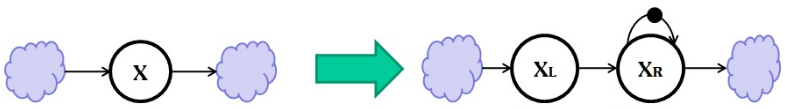

Figure 2.7: Replacing a job actor in an application by its response model to capture the effect of resource arbitration

We can compositionally model the worst-case temporal behavior of communicating jobs by representing each job by its response model. The graph obtained by replacing each job for its response model can be used for conservative temporal analysis of each application separately. Figure 2.8 shows how to extend an application graph to capture the effect of resource arbitration by replacing each job actor in the graph by its corresponding LR-model.

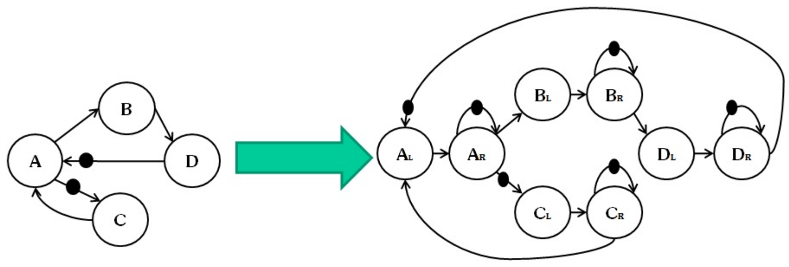

Figure 2.8: Extending an application graph to capture the effect of resource arbitration

2.5 Temporal Analysis of Embedded Streaming Application

The graph obtained by replacing each job for its response model can be used for conservative temporal analysis of each application separately. This section gives a brief overview of some of the techniques used to perform temporal analysis of the application graphs. The modeling approaches presented in this thesis can be used by these analysis techniques to establish real-time guarantees for an application such as guaranteed minimum throughput and guaranteed maximum latency. The focus of this thesis is not to improve the analysis techniques themselves, but to improve the modeling of worst-case temporal behavior of applications such that it facilitates more accurate analysis.

2.5.1 Simulation based Analysis

Application graphs can be analyzed by simulating the self-timed execution of the application graph to get a finite state space. The state-space consists of a finite sequence of state transitions followed by a sequence that is periodically repeated ad infinitum [24].

Since the application graph is considered to be strongly-connected, every actor depends on tokens from every other actor in the graph. This guarantees that there is a bound on the difference in the number of firings of actors, relative to the corresponding entries in the repetition vector. Therefore, the number of tokens that may accumulate on any edge is bounded. Also the amount of auto-concurrency is bounded such that only a finite number of actors can fire simultaneously. Since both the number of simultaneous actor firings and the number of tokens on any edge in the application graph are bounded, the number of states of an SDFG in self-timed execution is finite. The continuous self-timed execution will now have to re-visit some states in some periodic fashion, signifying that the execution has some periodic regime.

We can compute the throughput of an actor a during a simulation run σ such as the average number of firings of the actor a per time unit in σ Since the simulations can run infinitely, we compute this average as:

$$Th(a, \sigma) = \lim_{t \to \infty} \frac{|\sigma|_a^t}{t} \tag{2.4}$$

Simulation based tools have been extensively used for throughput calculation in [19, 21, 22, 23, 24]. Some approaches [21, 22] make use of response modeling of each job in the application such that each job is represented by its worst-case response model. Others [19, 23, 24], use simulation tools that do not use a unified model for representing an application and the worst-case effect of resource arbitration within the same graph. Instead the application graph is constructed in which each actor represents the worst-case execution time of the corresponding job. An additional independent binding aware time function is used to keeps track of the simulation time wheel, in the case of TDM arbitration, to capture the effect of resource arbitration.

Practically, most of the simulations for application graphs take a very short time to establish throughput guarantees. But theoretically, although the state space of the simulation is finite, it cannot be bounded in polynomial space. In [24], it is shown that the periodic behavior (number of state transitions) of a graph is a multiple of the repetition vector of a graph. Furthermore, the throughput of a graph computed using simulation based analysis gives the throughput for that simulation run only. The justification provided is that since data-flow is monotonic, simulating the worst-case arrival times of job iterations will provide the worst-case throughput of the graph [23].

2.5.2 Static Analysis

Analysis of the temporal behavior of the obtained response model via data-flow analysis enables us to define the minimum guaranteed throughput and maximum latency of the system. In [25], it is shown how latency constraints can be converted into throughput constraints as long as the best-case temporal behavior of the source of the system is characterized. To compute throughput of the given system we first compute the *Maximum Cycle Mean* (MCM) of the obtained response model. We stick to the MCM computation as already described in [8]. The cycle mean of a cycle c in a timed SRDF graph is defined as

$$\mu_c = \frac{\sum_{a_i \epsilon V(c)} t_{a_i}}{\sum_{e \epsilon E(c)} d(e)} \tag{2.5}$$

where $V(c)$ is the set of all nodes traversed by cycle c and $E(c)$ is the set of all edges traversed by cycle c.

The MCM μ of a timed SRDF graph G is defined as:

$$\mu(G) = \max_{c \epsilon C(G)} \frac{\sum_{a_i \epsilon V(c)} t_{a_i}}{\sum_{e \epsilon E(c)} d(e)} \tag{2.6}$$

where $C(G)$ is the set of cycles in graph G. The inverse of the MCM of a timed SRDF graph provides a fundamental upper bound to its minimum throughput [26]. We say that an application has a guaranteed minimum throughput described by this bound for the current setup of TDM arbitration. In practice the static analysis of an application takes longer than simulation based analysis. However, many algorithms used to compute the MCM of data-flow graphs have polynomial complexity [27]. Also, static MCM analysis provides worst-case guarantees of throughput and latency irrespective of the arrival times of job iterations [26].

Chapter 3

Resource Arbitration via Time Division Multiplexing

Till now we have presented the use of data-flow in generating application graphs for streaming applications. We will now look in detail to the use of data-flow to capture the effect of resource arbitration, particularly Time Division Multiplexing [TDM], of a single job. We first briefly present the mapping of embedded streaming application to the resources on an MPSoC platform and the need of resource arbitration. We then describe the use of TDM arbitration as a means of facilitating effective resource sharing among jobs mapped to the same resource. We then formalize the wost-case temporal behavior of job iterations scheduled using TDM arbitration. Finally we present the existing state-of-the-art data-flow models used for capturing this worst-case behavior.

3.1 Overview

An embedded streaming application for an MPSoC platform constitutes of multiple communicating jobs such that each job is mapped to a specific resource on the MPSoC platform. Figure 3.1 shows a sample embedded streaming application mapped to a MPSoC. The jobs are mapped to three different resources, such that jobs B and C are mapped to the same resource $R2$, while jobs A and D are mapped to resources $R1$ and $R3$ respectively.

For cost-effective reasons, multiple applications are deployed on a single MPSoC platform such that resources on the MPSoC are shared by jobs that may belong to different applications. All the jobs mapped to the same resource run on that resource via some arbitration mechanism. *Time Division Multiplexing* is a simple resource arbitration mechanism in which each job is allotted a fixed amount of time, called *slice*, within a

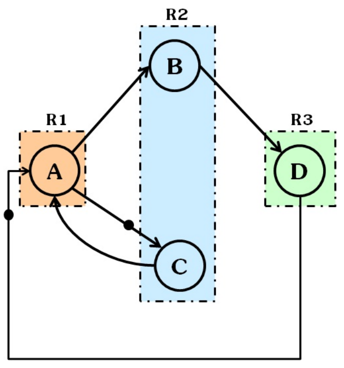

Figure 3.1: Application as a SRDF graph and mapped to MPSoC

fixed time frame, called *replenishment period*. This replenishment period is the same for all jobs, but the slice sizes may be different for different jobs. There may also be slices in a replenishment period in which no job is executing. Figure 3.2 depicts a simple TDM scheduling scheme, called a time wheel, for the resource $R2$ (refer Figure 3.1). Figure 3.2 depicts resource allocation for jobs B and C in their respective slices, while the rest of the replenishment period may be assigned to a job of another application or may remain idle. In a streaming application, all jobs have to be iteratively performed for each data item in the input stream. Therefore each job executes for multiple iterations (once for each input data item). In TDM arbitration, all iterations of a job must execute only during the slice allocated to that job within a single replenishment period. At the end of the slice, the execution of the current iteration of that job is pre-empted, and is only allowed to resume execution at the start of the next slice in the next replenishment period.

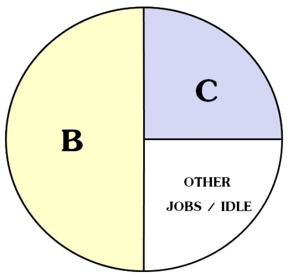

Figure 3.2: A sample TDM time wheel

3.2 Formalizing the behavior of TDM arbitration

We formalize the temporal behavior of TDM arbitration by expressing the finish time of an iteration in relation to the the finish times or start times of the previous iterations. A sequence of consecutive job iterations is where every iteration, except the first iteration in the sequence, arrives before its previous iteration finishes executing. The execution of each iteration will have to wait until all previous iterations have finished executing.

Consider that for a sequence of n consecutive job iterations, s_n is the arrival time of the first iteration in the sequence and f_n is the finish time of the last iteration in the sequence. If each job iteration takes t_A time to execute, $n \cdot t_A$ is the total execution time required. This amount of time can be expressed in terms of the slice size as $\lfloor \frac{n \cdot t_A}{S} \rfloor \cdot S + (n \cdot t_A \ \% \ S)$ i.e. we will require $\lfloor \frac{n \cdot t_A}{S} \rfloor$ complete slices and and additional $(n \cdot t_A \ \% \ S)$ time for the remaining execution. We require $\lfloor \frac{n \cdot t_A}{S} \rfloor$ periods for each slice that is consumed completely, and if $n \cdot t_A \ \% \ S \neq 0$ then an additional waiting time of $P - S$ is required before the job is allotted a slice to complete the remaining $(n \cdot t_A \ \% \ S)$ of the execution time. We define the execution of a sequence of n consecutive iterations as.

$$f_n = \begin{cases} s_n + \lfloor \frac{n \cdot t_A}{S} \rfloor \cdot P & \text{if } n \cdot t_A \ \% \ S = 0 \\ s_n + \lfloor \frac{n \cdot t_A}{S} \rfloor \cdot P + (P - S) + (n \cdot t_A \ \% \ S) & \text{if } n \cdot t_A \ \% \ S \neq 0 \end{cases} \tag{3.1}$$

Equation 3.1 gives a formalization for the worst-case finish time of each iteration of a continuous sequence of consecutive job iterations by expressing the finish time of that iteration as the finish time of the sequence and updating the length of the sequence accordingly. Although this formalization seems straight-forward, to the best of our knowledge, the worst-case temporal behavior of TDM arbitration defined via Equation 3.1 cannot be found in literature.

3.3 Response Modeling of TDM Arbitration using data-flow

We will now discuss the existing approaches to model the worst-case effect of TDM arbitration. There have been many response models proposed to capture the worst-case temporal behavior of jobs scheduled using TDM arbitration. Let us look at some of them one by one.

3.3.1 Single-actor Response Model

The most simplistic approach is to model the temporal behavior of the job as the worst case response time [6]. Consider an arbitrary job with execution time t_x for a single iteration of the job. The job is alloted a slice of size S per replenishment period P. We can represent the execution time in terms of the slice size as $t_x = \lfloor \frac{t_x}{S} \rfloor \cdot S + [t_x \ \% \ S]$. We express that the maximum time required for completing a single iteration of the task is $\lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S]$ from the time the iteration starts executing. It is also possible that a job iteration arrives (marked by the arrival of the corresponding input data item) outside the allotted slice for that job. It will then have to wait for the next slice assigned for that job to start executing. In the maximum initial waiting time is $P - S$, i.e. when the job iteration arrives just after the job slice has ended. Therefore we can express the total response time of this model as $(P - S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S]$. *Bekooij et al* [6] uses this value to model the worst-case temporal behavior of a job scheduled using TDM by representing a job x as a single actor v_x with firing time:

$$t(v_x) = (P - S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S] \tag{3.2}$$

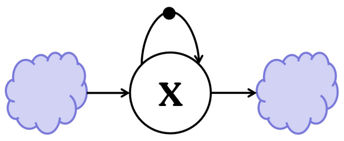

Figure 3.3: A job in an application represented using a single-actor response-model

Since a job iteration can execute only after all previous iterations have finished, [6] adds a self-edge to a job actor with a single token delay. Figure 3.3, shows a single job x of an embedded streaming application represented using a single-actor response-model v_x.

The temporal behavior of this response model defines the upper bound on the worst-case temporal behavior of the job. Consider, that the availability or arrival of data input for the job is represented by arrival of tokens on the incoming edge of actor v_x. There may be multiple inputs from different sources for a job, but let us consider for now, that this can be represented a single edge. The arrival time $\hat{a}(i)$ for the i=th input token on this edge, is the time at which the firing condition posed by all incoming edges from all external sources is satisfied. Similarly, there may be multiple outputs produced by the job meant for multiple jobs. This can also be represented by a single outgoing edge, such that the the i-th job iteration finishes at time $\hat{f}(i)$ and produces of the i-th output token on this outgoing edge. We can now describe the temporal behavior of this model according to the firing constraints posed by the two incoming edges of the single actor. The self-edge restricts a firing of the actor, i.e. the execution of a single iteration of the job, to start only after the previous iteration has finished. This is expressed as $\hat{f}(i) \geq \hat{f}(i-1) + t(v_x)$. The other firing constraint is due to the edge on which data-inputs arrive. This constraint can be expressed as $\hat{f}(i) \geq \hat{a}(i) + t(v_x)$. The combination of these two constraints define the overall behavior of this model as:

$$\begin{aligned} \hat{f}(i) &= \max(\hat{a}(i), \hat{f}(i-1)) + t(v_x) \\ &= \max(\hat{a}(i), \hat{f}(i-1)) + (P - S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S] \end{aligned} \tag{3.3}$$

Bekooij et al [6], illustrate that the single-actor response model is an upper bound to the worst-case temporal behavior of TDM arbitration and argue that it is conservative. In section 3.4, we discuss how we can formally prove the conservativity of a model. The full proof of the conservativity of the single-actor response model is provided in Appendix A.2.

3.3.2 Latency-Rate model:

In [8, 1], it is shown that jobs represented by a single actor response model have very pessimistic estimates. Instead, they describe the worst-case response time on the basis of two different effects of TDM arbitration, i.e. latency and rate. The latency defines the maximum time (including waiting time for allocating a slice) before which a job iteration will finish execution. The rate defines the response time of consecutive job iterations once it has started execution. Latency and rate have been modeled as a combination of two actors in [1] and used to capture a more accurate worst-case influence of a TDM scheduler.

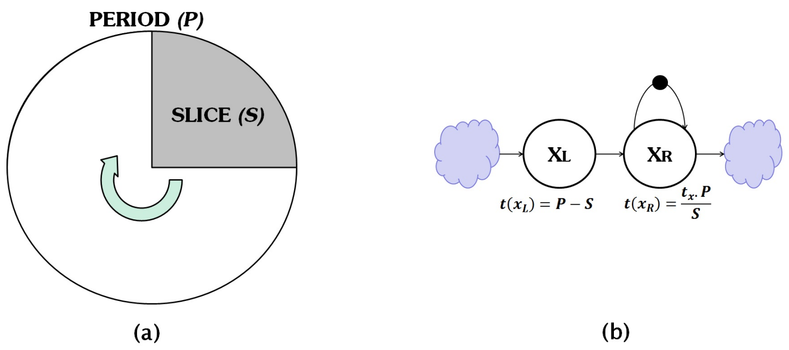

Figure 3.4: (a)Sample TDM Schedule. (b) LR-model for a TDM scheduled job

Consider a job x that has been allocated a slice S on a resource with replenishment period P. In Figure 3.4(b), the temporal behavior of the job on a TDM scheduler is modeled using two data-flow actors x_L, and x_R. The production and consumption of data containers by job x is modeled by the production and consumption of tokens in the constructed data-flow model. The enabling condition of job x is equal to the firing rule of actor x_L. The firing times of the two actors are defined as $t(x_L) = P - S$ and $t(x_R) = \frac{t_x \cdot P}{S}$ where t_x is the execution time for a single job iteration, and P and S are the replenishment period and allotted slice respectively.

Similar to the single-actor response model, the arrival time of a job iteration is modeled as the arrival time of an input token on the incoming edge of the actor x_L, and the finish time of a job iteration is modeled as the finish time of the actor x_R. As the execution of the data-flow actors is self-timed, actor x_L can fire as soon as a token arrives on its incoming edge. Therefore, we can represent the start time of the i-th firing of x_L as the arrival time $\hat{a}(i)$ of the i-th input token, i.e. $f(x_L, i) = \hat{a}(i) + t(x_A)$. Similarly when the actor x_R finishes its i-th firing, it produces the i-th output token on the outgoing edge of x_R and represents the finish of the i-th job iteration. Therefore the modeled finish time $\hat{f}(i)$ of the i-th iteration is also the finish time of the i-th firing of x_R, i.e. $f(x_R, i) = \hat{f}(i)$. If we observe the firing constraints of the actors, they all can be represented in terms of either the arrival time or finish time of some job iteration. We can, therefore, express the modeled temporal behavior of the job only in terms of the arrival and finish times of job iterations, and ignore the start times and finish times of individual actors in the model.

The bound defined by the LR-model can be attributed to two firing constraints. The first is the self-edge on the actor x_R giving us the constraint $\hat{f}(i) \geq \hat{f}(i-1) + t(x_R)$. The second firing constraint can be viewed as a chained constraint in which x_R fires as soon as a token is produced by x_L, and x_L in turn will fire only when a token arrives from its incoming edge. The firing constraint due to this chain can be expressed as the sum of the firing times of both these actors, i.e. $\hat{f}(i) \geq \hat{a}(i) + t(x_L) + t(x_R)$. The temporal behavior of this model can now be defined as:

$$\begin{aligned} \hat{f}(i) &= \max(\hat{a}(i) + t(x_L) + t(x_R), \hat{f}(i-1) + t(x_R)) \\ &= \max(\hat{a}(i) + (P - S) + \frac{P \cdot t_x}{S}, \hat{f}(i-1) + \frac{P \cdot t_x}{S}) \end{aligned} \tag{3.4}$$

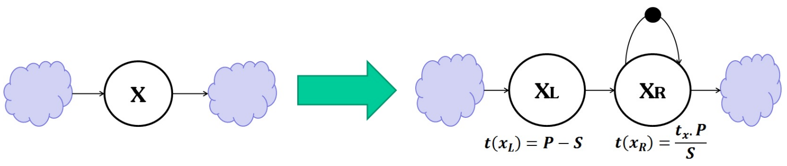

Figure 3.5: Representing a single job actor as an LR-model to capture the worst-case effect of TDM arbitration

The LR-model, provides a tight conservative estimate of the worst-case temporal behavior of TDM arbitration. The full proof of the conservativity of the LR-model is provided in Appendix A.3. However, in section 4.1 we illustrate the existing pessimism in the LR-model and the need for more accurate analysis of the worst-case temporal behavior of TDM arbitration.

3.3.3 Staschulat's LR-Model

Based on the latency-rate model [1], *Staschulat et al* [2] proposed a model that accurately models the response time of a memory arbiter. The general assumption he makes is that the time required for a single memory access is fixed such that a fixed number of memory accesses can occur within a single slice. That is to say that the size of the slice allotted for a memory access job is a integer multiple of the time taken by a single iteration of the job. Under this consideration, the modeling of the rate of service for a job can be further split into actual slice consumption and inter-slice waiting time.

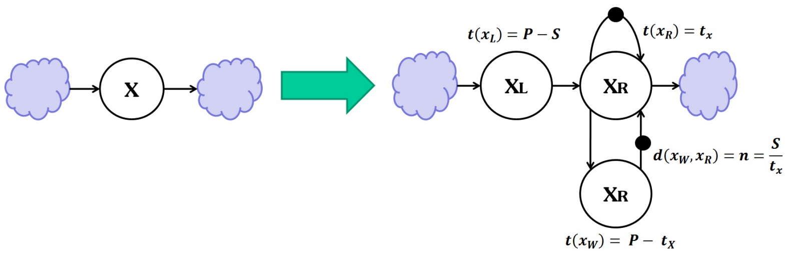

Figure 3.6: Representing a single job actor as a Staschulat's LR-model to capture the worst-case effect of TDM-based memory arbiter

Figure 3.6 shows an instance of Staschulat's model for a memory arbiter. Consider a memory access job x that has been allocated a slice S on a resource with replenishment period P. The temporal behavior of the job on a TDM scheduler is modeled using three data-flow actors x_L, x_W, and x_R. x_W and x_R together represent the rate of service provided by TDM arbitration. The delay n specified on the edge (x_W, x_R) defines the number of job iterations that can be completed within a single slice. This relation is expressed as $n \cdot t_x = S$ where t_x is the execution time for a single iteration and S is slice size. The firing times of the actors are defined as $t(x_L) = P - S$, $t(x_R) = \frac{t_x \cdot P}{S}$ and $t(x_W) = P - t_x$, where t_x is the execution time for a single job iteration, and P and S are the replenishment period and allotted slice respectively. The temporal behavior of a LR-model for a TDM scheduled job is illustrated in Section 4.1.1.

The bound defined by this model is a combination of three firing constraints. Similar to the previous models, there is a constraint due to the the self-edge (i.e. $\hat{f}(i) \geq \hat{f}(i-1) + t(x_R))$ and another constraint based on the arrival of incoming tokens (i.e. $\hat{f}(i) \geq \hat{a}(i) + t(x_L) + t(x_R))$. A third constraint is defined by the loop between actors x_R and x_W. As there are n tokens initially placed on the edge (x_W, x_R), the first n firings of x_R are not constrained by this loop. The $n+1$-th firing of x_R will, however, be constrained to fire only after the first firing of the actor x_W produces a token, which in turn is constraint by the first firing of actor x_R. We can express this constraint as $\hat{f}(i) \geq \hat{f}(i-n) + t(x_W) + t(x_R)$. The temporal behavior thus modeled is given as

$$\hat{f}(i) = \max \begin{cases} \hat{a}(i) + (P - S) + t_x \\ \hat{f}(i-1) + t_x \\ \hat{f}(i-n) + P \end{cases} \tag{3.5}$$

Staschulat's LR-model is an accurate representation of the worst-case temporal behavior of the modeled TDM arbitration setup. The full proof of the conservativity of Staschulat's LR-model is provided in Appendix A.4. In section 4.2, we show that Staschulat's LR-model has very restrictive applicability such that it can only model exceptional cases of TDM arbitration. This model is, thus, suitable for very few applications such as for modeling memory arbiters.

3.4 Conservativity of data-flow based models for TDM arbitration

The models explained in section 3.3 claim to capture the worst-case temporal behavior of a job scheduled using TDM. The temporal behavior of a job iteration is defined in terms of the arrival time, i.e. the time at which the job iteration is ready to execute, and the finish time, i.e. the time at which the job iteration completes its execution. By definition, modeling the worst-case temporal behavior of a job scheduled using TDM arbitration implies that, given the worst-case arrival time of a job iteration, the model defines the worst-case finish time bound of that iteration. In other words, the actual finish time of a job iterations is, in the worst-case, equal to the finish time obtained using these models. This property of modeling an upper-bound for the finish times of job iterations is called *conservativity* and is defined as:

Definition 1. *A data-flow model is* conservative *if, whenever the modeled arrival times $\hat{a}$ are an upper bound to the actual arrival times a, the modeled finish times $\hat{f}$ are an upper bound to the actual finish times f. Formally, this means that:*

$$\forall_{i \geq 0}\ a(i) \leq \hat{a}(i) \ \Rightarrow\ \forall_{i \geq 0}\ f(i) \leq \hat{f}(i) \tag{3.6}$$

In section 3.2, we have formalized the worst-case temporal behavior of the actual execution on job iterations. To claim that them models proposed in [6], [1], and [2] are conservative, we need to show that the temporal behavior defined by these models are an upper bound to the worst-case temporal behavior defined in section 3.2 such that, Equation 3.6 holds. In Appendix A, we provide the complete proof for the conservativity of each of these models.

3.5 TDM arbitration and Static-Ordering

Until now, we have discussed the resource sharing of jobs using TDM arbitration such that each job is allocated an independent slice to execute in irrespective of whether the jobs mapped to the same resource belong to the same or different applications. The main problem with such a strategy is that the bounds on the worst-case response times of actors executing on independent TDM slices completely overlook the fact that, within a application, we have more information about the interdependence of jobs [8]. For instances, a set of jobs may be mutually exclusive. In an SRDF application graph, this can be observed when the job actors belong to the same single-delay cycle.

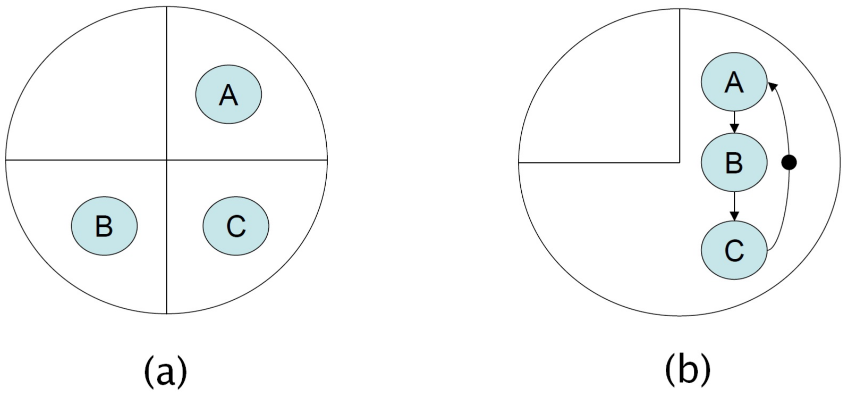

Figure 3.7: (a) Jobs are assigned independent slices. (b) Jobs are in a static-order and are assigned the same slice (figure taken from [8])

Allocating a different slice to each of these jobs wastes resources. Instead, if all share the same slice, each job can use the whole slice when enabled [8]. As we assume static processor allocation, we know that actors allocated to the same processor are already forced to execute in mutual exclusion. This is called *static-ordering* of jobs in an application. Mixing TDM with static-ordering assigns a TDM slice for a group of mutually exclusive jobs in a static-order instead of assigning each job with a separate slice. Figure 3.7 shows an instance where three jobs mapped to the same resource are *(a)* assigned independent slices; and *(b)* are grouped is a static order and assigned a single slice of size equal to the three independent slices.

Moreira et al [8] show the advantages of incorporating static-order in TDM arbitration and modeling this resource arbitration mechanism using the LR-model. In chapter 8, we address the issue of modeling TDM with static ordering using the data-flow models proposed in this thesis.

Chapter 4

Problem Description

This chapter gives an account of the problems with the state-of-the-art models in defining the worst-case temporal behavior of a job executing on a TDM scheduled resource. We illustrate how the existing models are inadequate in accurately capturing the worst-case temporal behavior of TDM arbitration. The existing approaches are either too pessimistic such as the LR-model [1], or they can only model exceptional cases of TDM arbitration such as Staschulat's LR-model [2].

We first illustrate the pessimism in defining the worst-case temporal behavior of TDM arbitration using the LR-model. The worst-case temporal behavior of TDM arbitration is defined as the worst-case finish time of job iterations scheduled using TDM arbitration. We show that the LR-model significantly over-estimates the worst-case finish time of the iterations of an arbitrary job. Compositionally, if each job in an application is represented by a pessimistic model, the overall worst-case temporal behavior obtained has a high degree of pessimism. This pessimism affects the analysis of the hard real-time guarantees of the application, thereby requiring over-allocation of resources to satisfy these requirements. Over-allocation of resources will lead to low system utilization and also unnecessary rejection of applications that could have been accommodated in the system.

Following this, we show that Staschulat's LR-model is restricted to exceptional cases of TDM arbitration. This model makes the basic assumption that the size of slice allocated to a job is an integer multiple of the execution time of a single job iteration. This model, can therefore be used to model only a confined class of applications such as memory arbiters. Although the model accurately defines the worst-case temporal behavior of a job satisfying this assumption, we cannot use this model for a broader class of TDM arbitration.

4.1 Pessimism in the LR-model

In this section we will first illustrate some simple cases that highlight the pessimistic estimation of the worst-case temporal behavior of TDM arbitration using the LR-model. We then identify and generalize the cause of this pessimism and show that it can potentially lead to very high degree of over-estimation.

4.1.1 Illustration of the problem

Consider a TDM scheduler with a replenishment period $P = 100$ that schedules a job with execution time $t_A = 4$ in a slice $S = 10$ per replenishment period. To make the analysis simple we assume that each job iteration can execute as soon as the previous iteration finishes, i.e. all iterations are assumed to be ready to execute at the start of the system. The slice allocated in every period is just before the start of the next period, as shown in Figure 4.1. We consider this particular slice allocation as it has the maximum initial waiting time before a slice is allocated and , thus, is identified as the worst-case.

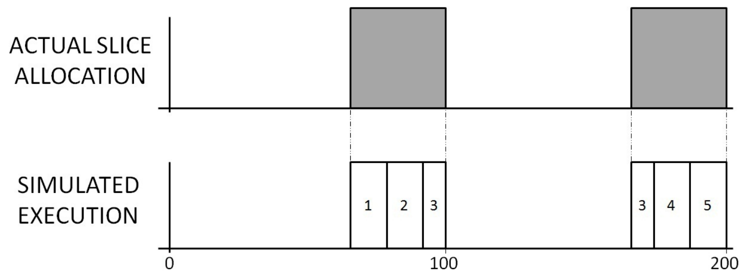

Figure 4.1: Actual temporal behavior for the given TDM schedule

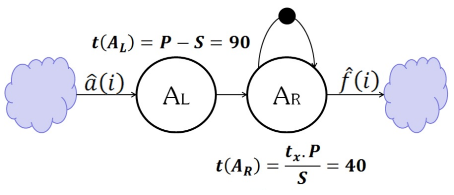

Figure 4.2: Modeling the temporal behavior for the given TDM schedule

Figure 4.2 shows the LR model for this example. To capture the effect of initial waiting time as part of the latency, we use actor A_L with firing time $t(A_L) = P - S = 90$. The rate is modeled as the actor A_R with firing time $t(A_R) = \frac{t_A \cdot P}{S} = 40$. Table 4.1 compares the actual finish time of each iteration with the finish time of the corresponding output token defined by the LR-model, assuming that all the input tokens are available at the input FIFO of the actor A_L, at the start of the system. This assumption is equivalent to all job iterations being ready at the start of the system.

Iterations $\rightarrow$	**1**	**2**	**3**	**4**	**5**	**6 ...**
Actual Finish Time	94	98	192	196	200	294 ...
LR-Modeled Finish Time	130	170	210	250	290	330 ...

Table 4.1: Actual and LR-modeled finish times for TDM arbitration

Now consider the same TDM period and slice size setup for another job with execution time $t_B = 15$. Figure 4.3 shows the time line of the execution of consecutive job iterations. Figure 4.4 shows the LR-model for this example. Table 4.2 compares the actual finish time of each iteration with the finish time modeled using the LR-model for this job.

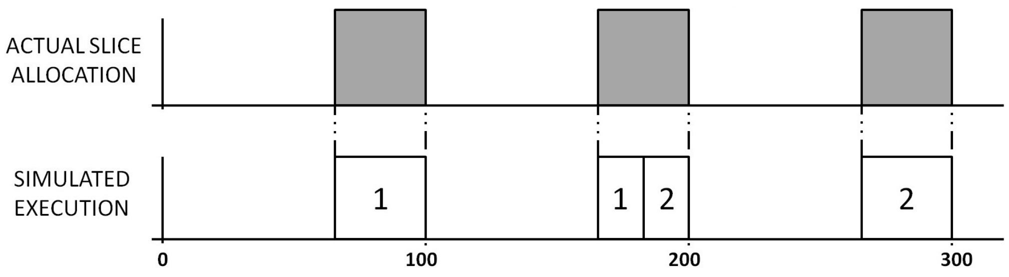

Figure 4.3: Actual temporal behavior for the given TDM schedule

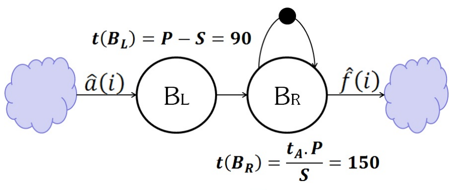

Figure 4.4: Modeling the temporal behavior for the given TDM schedule

4.1.2 Analyzing the pessimism in the LR-model

As can be seen from both the examples, the LR-model highly over-estimates the finish time of each iteration. The finish time bounds for the LR-model, are governed by the firing time of the rate actor, given by $\frac{t_A}{S} \cdot P$, which generalizes the overall rate of execution of consecutive iterations. Since the replenishment period is always greater than the allotted slice(i.e. $P \geq S$), we can observe that the modeled bound overestimates the required time for execution by a factor of $\frac{P}{S}$. This factor becomes more significant for larger differences between the slice and the replenishment period, leading to pessimistic estimation of the temporal behavior. If each job in an application is represented using the LR-model, the overall pessimism gives a high over-estimation of the temporal behavior of the entire application. This causes over-allocation of resources to satisfy the real-time requirements of the application.

Iterations $\rightarrow$	**1**	**2**	**3**	**4**	**5**	**6 ...**
Actual Finish Time	195	300	495	600	795	900 ...
LR-Modeled Finish Time	240	390	540	690	842	990 ...

Table 4.2: Actual and LR-modeled finish times for TDM arbitration

4.2 Restricted Applicability of Staschulat's LR-Model:

Staschulat's LR model makes the basic assumption on the characteristics of TDM arbitration setup that the slice allocated to the job is just enough to complete a fixed number of task iterations. The number of iterations accommodated in a single slice is defined by the delay on the edge (x_W, x_R) expressed as $d(x_W, x_R) = n$. For a slice size S allocated to a job with execution time t_x, we compute the value of n as $\frac{S}{t_x}$. As the delay on an edge has to be an integer, the value of $n = \frac{S}{t_x}$ is an integer, i.e. S is an integer multiple of t_x.

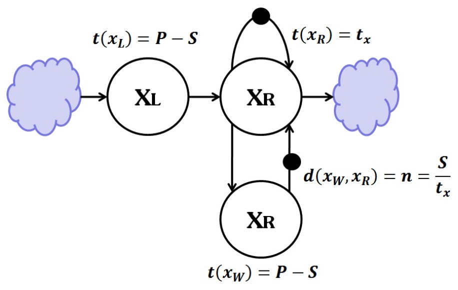

Figure 4.5: Staschulat's LR-model

Due to this restriction, Staschulat's LR-model is only applicable when the TDM setup satisfies the above criteria. Under this assumption, however, it must be noted that Staschulat's model provides accurate estimates of the worst-case temporal behavior of a job scheduled using TDM arbitration. The results and proof of this accuracy is provided in [2].

The initial assumption made by Staschulat's LR-model limits its use to a small number of applications that satisfy this assumed criterion, for example, a memory arbiter. In most applications, however, execution of a complete iteration of a job cannot be accommodated in a single slice, to have an efficient resource sharing mechanism among the jobs sharing that resource. The execution of a job iteration is spread across multiple slices and this cannot be modeled using Staschulat's LR-model. Although some jobs in an application may be accommodated is a single slice, the restriction on the size of the slice allocated may result in sub-optimal performance of TDM arbitration and may possibly not satisfy the required real-time guarantees.

Chapter 5

Modeling TDM Arbitration: Latency-Phased-Rate Model

In this chapter, we present our first approach to improve the estimation of the worst-case temporal behavior of a job scheduled using TDM arbitration. We first give a brief overview of our approach. We then illustrate an instance of our model and observe the modeled temporal behavior for a job. This illustration gives an intuitive idea of our approach. Next, we formalize the construction of this model. We then analyze this construction to derive the bound given by the model on the worst-case temporal behavior of TDM arbitration.

5.1 Overview

We initially assume that the execution time of a single job iteration is smaller that the size of a single slice. The execution of continuous iterations of a job using TDM can be shown to have a cyclic pattern over a fixed number of replenishment periods. Our approach is to model this pattern using single-rate data-flow by capturing the execution behavior for each replenishment period separately. This enables us to accurately specify worst-case behavior for each period rather than generalizing over all periods, as done in [6, 8, 1]. The obtained result shows a tighter approximation of resource requirements enabling improved resource utilization. Unlike some existing models [2], we do not pose restrictions on the size of the slice allocated to a job. This enables optimized resource allocation for TDM arbitration. We then show that we can extend this model to capture the cyclic pattern for jobs with arbitrary execution times and slice allocation. As the job execution in each replenishment period of the cyclic pattern can be considered as a separate phase, having its own rate of execution, we call our model the *Latency-Phased-*

Rate (LPR) model.

5.2 Sketch of the proposed approach

Consider the same example as depicted in section 4.1.1. Figure 4.1 shows a time line of the simulated execution. The first iteration executes after waiting for its slice and executes completely from time 90 to 94. The second iteration starts immediately at time 94 and finishes at time 98. Now the remaining slice (98 to 100) is not enough to execute the third iteration completely. The third iteration has to halt execution at the end of slice at time 100 and resumes execution only after it receives the next slice at time 190 and finishes execution at time 192. The fourth iteration has enough slice time left and finishes execution at time 196. The fifth iteration has just enough slice remaining to execute completely within the same slice and finish execution by time 200. This pattern repeats itself from the sixth iteration onwards.

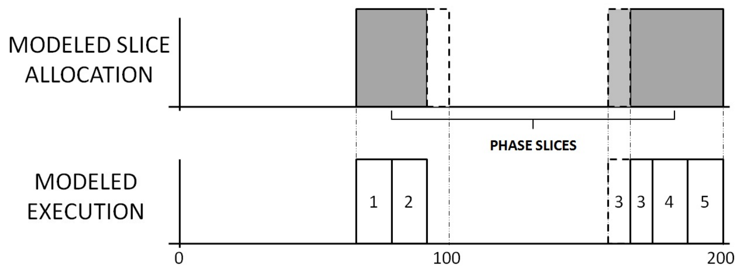

Figure 5.1: Modeled temporal behavior for the given TDM schedule

The arrival time and finish time of a job iteration are the only relevant aspects of the temporal behavior, irrespective of when the execution actually happens. This allows us to shift the partial execution of the iteration from the slice in which it started executing, and join it to the slice in which it completes. For instance, the execution of the third iteration that starts in the first slice, can be modeled as an extended part of the second slice, as shown in Figure 5.1. We now model the execution pattern as the number of iterations that finish execution in a particular period within the modified slices. Each modified slice, which we refer to as *phase slice*, represents a different phase in the execution pattern. Each phase slice can be derived based on the amount of time remaining in the previous slice after executing the last iteration that finishes execution in that slice.

The LPR-model for the current example is shown in Figure 5.2. We model each phase as a separate actor sequentially linked to each other forming a cycle. The number of iterations that can execute in a phase is given by the initial tokens present on each link. There is an initial waiting time for the first job iteration of the entire cyclic pattern, and is modeled using a separate actor. The execution of an iteration is modeled with a self-looped actor such that only a single iteration can execute at a time. Table 5.1 shows

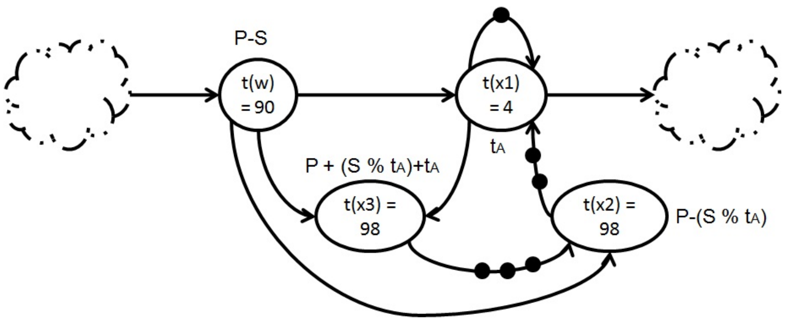

Figure 5.2: Proposed data-flow model

the temporal behavior defined by the LPR-model in comparison to the LR-model [1].

Iterations $\rightarrow$	**1**	**2**	**3**	**4**	**5**	**6 ...**
Actual finish time	94	98	192	196	200	294 ...
LR-modeled finish time	130	170	210	250	290	330 ...
LPR-modeled finish time	94	98	192	196	200	294 ...

Table 5.1: Comparison of obtained finish times using LR-model and our model

5.3 Construction of the model

Formally, a job executing on a TDM scheduled resource is described as a 3-tuple (P, S, t_x) where P is the replenishment period of the TDM arbitration, S is the size of the slice allocated to the job and t_x is the execution time of a single job iteration. The temporal behavior has a cyclic pattern of length q periods expressed as:

$$q = \frac{lcm(S, t_x)}{S} \tag{5.1}$$

We say that the sum of the slices, given by $q \cdot S$, is just sufficient to completely execute m job iterations, where $m = \frac{lcm(S,t_x)}{t_x}$. Each job can be modeled as an SRDF graph component $G(P, S, t_x) = (V, E, d, t)$. V is the set of actors of the model given by:

$$V = \{x_i | 1 \leq i \leq q + 1\} \cup \{w\}, \tag{5.2}$$

where each phase actor $x_i \in V$ describes a separate phase of the cyclic pattern. The latency actor w is used to represent the effect of the initial waiting time on the latency of a job iteration.

The edges are defined such that the phase actors form a cycle, and there is an edge from the latency actor w to each of the phase actors. There is also a self-edge from the actor

x_1 to itself.

$$E_A = \{(x_{i+1}, x_i) | 1 \leq i \leq q\} \cup \{(x_1, x_{q+1}), (x_1, x_1)\} \cup \{(w, x_i) | 1 \leq i \leq q+1\} \tag{5.3}$$

We now add initial tokens (delay) d on each edge, such that $d(x_{i+1}, x_i)$ for $1 \leq i \leq q$ represents the maximum number of job iterations that can execute in the i-th phase, given by:

$$d(x_j, x_i) = \begin{cases} \lfloor \frac{i \cdot S}{t_x} \rfloor - \lfloor \frac{(i-1) \cdot S}{t_x} \rfloor & \text{for } 1 \leq i \leq q \text{ and } j = i+1, \\ 1 & \text{for } i = j = 1. \end{cases} \tag{5.4}$$

The delay for all other edges in the LPR-model, i.e. the edges connecting the latency actor w to the phase actors $x_1 ... x_{q+1}$, is set to 0.

$$d(w, x_i) = 0 \qquad \text{for } 1 \leq i \leq q+1. \tag{5.5}$$

The firing time of the latency actor is given by $t(w) = P - S$. The firing time $t(x_i)$ of each phase actor $x_i \in V$ defines the start time of the phase slice based on the amount of execution time leftover from the previous phase slice after last iteration that finishes executing in that phase slice, expressed as:

$$t(x_i) = \begin{cases} t_x & \text{for } i = 1, \\ (i-1) \cdot P - [(i-1) \cdot S \ \% \ t_x] & \\ \qquad -((i-2) \cdot P - [(i-2) \cdot S \ \% \ t_x]) & \text{for } 2 \leq i \leq q \\ (q) \cdot P - [(q) \cdot S \ \% \ t_x] & \\ \qquad -((q-1) \cdot P - [(q-1) \cdot S \ \% \ t_x]) - t_x & \text{for } i = q+1. \end{cases} \tag{5.6}$$

We have defined the the set of actor V and the edges E connecting the actors in the LPR-model in Equations 5.2 and 5.3 respectively. Equation 5.4 defines the delay for each edge $e \in E$ in the LPR-model, while Equation 5.6 defines the firing time of each actor $v_i \in V$ for the LPR-model. Thus, we provide a formal definition for the construction of the LPR-model as (V, E, d, t) for the TDM arbitration setup for the job (P, S, t_x).

5.4 Analyzing the modeled bound for the worst-case temporal behavior

In this section we now analyze the LPR-model construction described in the previous section. We first establish the bound defined by the LPR-model for the worst-case temporal behavior of TDM arbitration. We then demonstrate that the LPR-model is conservative. We then compare the LPR-model with the state-of-the-art model [1, 2].

Initially we assume that the execution time of a single job iteration is smaller than the allocated slice. We later show how to extend the LPR-model to conservatively capture the worst-case temporal behavior of TDM arbitration of jobs with arbitrary execution times.

5.4.1 Establishing the LPR-modeled bound for the worst-case temporal behavior of TDM arbitration

We assume the self-timed behavior of the data-flow actors, i.e. an actor fires as soon as its enabling condition is satisfied. Based on the model construction in section 5.3, we can now derive a constraint on the finish time of the first iteration in each phase as:

$$\hat{f}(i) \geq \max_{1\leq j\leq q}\{\hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + (P-S) + (j-1)P - [(j-1)\cdot S \ \% \ t_x] + t_x\} \quad (5.7)$$

There are total of $\frac{q\cdot S}{t_x} = \frac{lcm(S,t_x)}{t_x} = m$ tokens in the cycle of phase actors. This gives us a constraint for the start of the next cycle given by:

$$\hat{f}(i) \geq \hat{f}(i-m) + q\cdot P, \quad (5.8)$$

where $m = \frac{lcm(S,t_x)}{t_x}$ and i is the first iteration in the new cycle.

The actor x_1 has a self loop with a single token, ensuring non-overlapped execution of job iterations. This bounds the finish time of a job iteration to finish at least t_x time after the finishing of the previous iteration, expressed as:

$$\hat{f}(i) \geq \hat{f}(i-1) + t_x \quad (5.9)$$

The above construction generates a *max*-expression to bound the firing of the i-th iteration of the job given by

$$\hat{f}(i) = \max\begin{cases}\max_{1\leq j\leq q}\{\hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + (P-S) + \\ \qquad (j-1)\cdot P - [(j-1)\cdot S \ \% \ t_x] + t_x\} \\ \hat{f}(i-m) + q\cdot P, \\ \hat{f}(i-1) + t_x, \end{cases} \quad (5.10)$$

where $q = \frac{lcm(S,t)}{S}$. Thus, we establish the LPR-modeled bound for the worst-case finish time for each iteration of a TDM scheduled job.

5.4.2 Conservativity of the model

We now prove that our model is conservative in defining the worst-case temporal behavior of TDM arbitration. The formal definition of conservativity as it is used in data-flow graphs [6] usually guarantees both the over-approximation of the behavior of a single job, as well as the compositionality of over-approximations when jobs are combined in a larger model. In section 3.4, we have defined the conservativity property of a data-flow model for TDM arbitration. Recall, that the model is conservative if:

$$\forall_{i \geq 0}\ a(i) \leq \hat{a}(i) \ \Rightarrow\ \forall_{i \geq 0}\ f(i) \leq \hat{f}(i), \tag{5.11}$$

where $a(i)$ and $f(i)$ are the actual start time and finish time, respectively, of the i-th job iteration, while $\hat{a}(i)$ and $\hat{f}(i)$ are the corresponding LPR-modeled start and finish time respectively.

Theorem 1. *The graph $G(P, S, t_x)$ as defined in the previous section, is a conservative model for a TDM-scheduled job with worst-case execution time t_x, that is assigned a slice S per period P.*

Proof. We have formalized the worst-case behavior of TDM arbitration in section 3.2 as:

$$f_n = \begin{cases} s_n + \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P & \text{if } n \cdot t_x \ \% \ S = 0 \\ s_n + \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P + (P - S) + (n \cdot t_x \ \% \ S) & \text{if } n \cdot t_x \ \% \ S \neq 0 \end{cases}, \tag{5.12}$$

where s_n and f_n are the start time and finish time respectively, of a consecutive sequence of multiple job iterations of length n. In order to prove conservativity of the LPR-model, we show that worst-case temporal behavior of TDM arbitration defined by the LPR-model is an upper bound to this formalization defined in section 3.2. We show that each iteration can be associated (represented in the model) with a token present on the cycle of phase actors in the proposed model. The execution behavior of a job iteration can be enumerated as follows:

Case I: If a job iteration arrives in isolation i.e. if the arrival of the i-th job iteration is after the finishing of its previous iteration ($a(i) > f(i-1)$), it can be considered as the first job iteration of a consecutive sequence (i.e. $n = 1$). We say that the start time of the sequence is given by, i.e. $s_n = a(i)$. The finish time f_n of the sequence is also the finish time of the i-th iteration $f(i)$. Since we know that $t_x \leq S$, we can define the finish time as $f(i) = a(i) + (P - S) + t_x$. We can associate this iteration to the first token of the first phase, i.e. putting $j = 1$ in Equation 5.7. This bounds the finish time of the iteration as $\hat{f}(i) \geq \hat{a}(i) + P - S + t_x$. As $a(i) \leq \hat{a}(i)$, we say that:

$$f(i) = a(i) + (P - S) + t_x \leq \hat{a}(i) + P - S + t_x = \hat{f}(i) \tag{5.13}$$

If the i-th iteration of the job belongs to a consecutive sequence of job iterations i.e. $[a(i) < f(i-1)]$, such that the $(i-k)$-th iteration is the first iteration of the consecutive sequence, we sub-divide its execution behavior into three categories based on the value of k.

Case II: The $i-k$-th iteration is the first iteration in the consecutive sequence [i.e. $a(i-k) > f(i-k-1)$] such that $[k < \frac{lcm(S,t_x)}{t_x} - 1]$. The start time of the consecutive sequence is the arrival time of the $(i-k)$-th iteration (i.e. $s_n = a(i-k)$) where the length of the sequence is $n = (k+1)$. Since $k < \frac{lcm(S,t_x)}{t_x} - 1$ we know that $(k+1) \,\%\, S \neq 0$. The finish time of the sequence is the finish time of the i-th iteration (i.e $f_n = f(i)$)) and is defined as:

$$f(i) = a(i-k) + \lfloor\frac{(k+1)\cdot t_x}{S}\rfloor \cdot P + (P-S) + ((k+1)\cdot t_x \,\%\, S) \tag{5.14}$$

If the i-th iteration is the first iteration of a j-th phase slice, we associate it with the leading token of that phase actor. The $(i-k)$-iteration is then associated with the first token of the cycle. We can express this association as $k = \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$, i.e. the leading token of the j-th phase actor is $\lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$ tokens after the first token in the cycle. From this association we can say that $\frac{k\cdot t_x}{S} \leq (j-1) < \frac{(k+1)\cdot t_x}{S}$. Equation 5.7 gives the bounds:

$$\hat{f}(i) = \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + (P-S) + (j-1)P - [(j-1)S \,\%\, t_x] + t_x \tag{5.15}$$

Using the relations between k, S, and t_x, we can derive that $\lfloor\frac{(k+1)\cdot t_x}{S}\rfloor \cdot P \leq (j-1)P$ and $(k+1)\cdot t_x \,\%\, S \leq [(j-1)S \,\%\, t_x] + t_x$. Due the association $k = \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$, we know that $a(i-k) \leq \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor)$. Adding all these inequalities we get the required bound expressed as:

$$\begin{aligned} f(i) &= a(i-k) + \lfloor\frac{(k+1)\cdot t_x}{S}\rfloor \cdot P + (P-S) + ((k+1)\cdot t_x \,\%\, S) \\ &\leq \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + (P-S) + (j-1)P - [(j-1)S \,\%\, t_x] + t_x \\ &\leq \hat{f}(i) \end{aligned} \tag{5.16}$$

Consider now that the i-th iteration is not the first iteration of the j-th phase slice. We define a u such that $(i-u)$-th iteration is the first iteration in this phase. Using the association that $k - u = \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$ we repeat the entire procedure described for Equation 5.16 to get:

$$f(i-u) \leq \hat{f}(i-u). \tag{5.17}$$

The self loop of actor x_1 gives the constraint $\hat{f}(i) \geq \hat{f}(i-1) + t_x$. If applied u times, this constraint gives us the bound $\hat{f}(i) \geq \hat{f}(i-u) + u\cdot t_x$. As the i-th iteration executes in the same slice as the $(i-u)$-th iteration, we can derive that $f(i) = f(i-u) + u\cdot t_x$. Using Equation 5.17 we can now say that:

$$f(i) = f(i-u) + u\cdot t_x \leq \hat{f}(i-u) + u\cdot t_x = \hat{f}(i). \tag{5.18}$$

Case III: The $(i-k)$-th iteration is the first iteration of the sequence such that $[k = \frac{lcm(S,t_x)}{t_x} - 1]$. The start time of the sequence is $s_n = a(i-k)$. Since the length of the sequence $n = (k+1) = \frac{lcm(S,t_x)}{t_x}$, we know that $(k+1)$ % $S = 0$. The finish time $f_n = f(i)$ of the sequence is then defined as:

$$f(i) = a(i-k) + \frac{(k+1)\cdot t_x}{S} \cdot P \tag{5.19}$$

We use the same approach that was used to derive Equation 5.18 for the u-th token in the j-th phase. The i-th iteration is now associated with the last token of the cycle, which is also the last token of the q-th phase slice (i.e. $j = q$). We can now derive that $(q-1)S \,\%\, t_x + (u+1)\cdot t_x = S$, where $(i-u)$-th token is associated with first iteration of the q-th phase slice. Using this in Equation 5.15 we model the finish time of the i-th iteration as:

$$\begin{aligned} \hat{f}(i) &= \hat{a}(i - \lfloor\frac{(q-1)\cdot S}{t_x}\rfloor) + (P-S) + (q-1)P + S \\ &= \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + q\cdot P \end{aligned} \tag{5.20}$$

As $q = \frac{lcm(S,t_x)}{S} = \frac{(k+1)\cdot t_x}{S}$ and $a(i-k) \leq \hat{a}(i - \lfloor\frac{(q-1)\cdot S}{t_x}\rfloor$, we get:

$$f(i) = a(i-k) + \frac{(k+1)\cdot t_x}{S}\cdot P \leq \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + q\cdot P = \hat{f}(i) \tag{5.21}$$

Case IV: The $i-k$-th iteration is the first iteration of the sequence such that $[k \geq \frac{lcm(S,t_x)}{t_x}]$. The execution behavior of the i-th iteration can be defined such that there will be $m = \frac{lcm(S,t_x)}{t_x}$ job iterations left to execute consecutively after the finishing of the $(i-m)$-th iteration. We then say that the start time of this consecutive sequence is $s_n = f(i-m)$ with length m. We now define the finish time $f_n = f(i)$ of the sequence as $f(i) = f(i-m) + \frac{m\cdot t_x}{S}\cdot P$ We associate the i-th iteration with the first iteration of a new cycle, bounded by the cyclic constraint defined in Equation 5.8. We can inductively show that $f(i-m) \leq \hat{f}(i-m)$. Since $q = \frac{lcm(S,t_x)}{S} = \frac{m\cdot t_x}{S}$ we get:

$$f(i) = f(i-m) + \frac{m\cdot t_x}{S}\cdot P \leq \hat{f}(i-m) + q\cdot P = \hat{f}(i) \tag{5.22}$$

□

Through the above enumeration, we have covered all possible cases of the worst-case temporal behavior of a job scheduled using TDM. We have shown in each case that the LPR-modeled finish time of job iterations are conservative with respect to the corresponding actual finish time.

5.4.3 Comparison with the LR-model:

Our model is truly tighter than the LR-model [1] such it is always more conservative than our approach. We show this by proving that the LR-model gives an upper-bound to the temporal behavior modeled by our model. The LR-model defines the worst-case temporal behavior of TDM arbitration as:

$$f'(i) = max(a'(i) + (P - S) + \frac{P \cdot t_x}{S}, f'(i-1) + \frac{P \cdot t_x}{S}) \tag{5.23}$$

where $f'(i)$ and $a'(i)$ define respectively the modeled finish time and modeled arrival time of the i-th iteration in the LR-model.

Theorem 2. *The LPR-model gives a tighter estimation on the worst-case finish time than the LR-model, i.e.*

$$\forall_{i \geq 0}\ \hat{a}(i) \leq a'(i) \ \Rightarrow\ \forall_{i \geq 0} \hat{f}(i) \leq f'(i).$$

Proof. We again enumerate all the possible cases for the worst-case temporal behavior modeled using our approach in comparison to the LR-model [1].

1. For isolated iterations, it is trivially proven that $\hat{a}(i) + (P - S) + t_x \leq a'(i) + (P - S) + \frac{P \cdot t_x}{S}$ given that $\hat{a}(i) = a'(i)$ and $P \leq S \Rightarrow t_x \leq \frac{P \cdot t_x}{S}$.

2. For consecutive iterations with length $k \geq \frac{lcm(S,t_x)}{t_x}$, the LR-model [1] can be shown to give the constraint $f'(i) = f'(i-m) + m \cdot \frac{P \cdot t_x}{S}$, where $m = \frac{lcm(S,t_x)}{t_x}$ which is equivalent to $\hat{f} = \hat{f}(i-m) + q \cdot P$.

3. For all consecutive iterations of length k where $1 \leq k < \frac{lcm(S,t_x)}{t_x}$, the bound is given by the LR-model [1] is shown as $f'(i) = f'(i-k) + k \cdot \frac{P \cdot t_x}{S} = a'(i-k) + (P - S) + (k+1) \cdot \frac{P \cdot t_x}{S}$. To show that Equation 5.7 gives us a tighter bound than the LR-model we try to prove by contradiction that.

$$(j-1)P - (j-1)S \ \% \ t_x + t_x > (k+1) \cdot \frac{P \cdot t_x}{S} \tag{5.24}$$

where $\lfloor \frac{j-1) \cdot S}{t_x} \rfloor = k$ (i.e. $\frac{k \cdot t_x}{S} \leq (j-1) < \frac{(k+1) \cdot t_x}{S}$). Expanding $(j-1)S \ \% \ t_x$ in Equation 5.24, we derive that $(j-1) > \frac{(k+1) \cdot t_x}{S}$ thereby giving us a contradiction. Therefore we show that $(j-1)P - (j-1)S \ \% \ t_x + t_x > (k+1) \cdot \frac{P \cdot t_x}{S}$.

These enumerations show that for all cases of worst-case temporal behavior of TDM, our model gives tighter estimates as compared to the LR-model. □

5.4.4 Comparison to Staschulat's LR model

Let us now compare the LPR-model with Staschulat's LR-model [2]. Staschulat's LR-model assumes that size of the slice allocated to a job is an integer multiple of the execution time of a single job iteration. The LPR-model does not make this assumption, and is a more generic approach to model the temporal behavior or jobs scheduled using TDM arbitration.

However, under the assumption made by Staschulat's LR-model, we can show that its model construction directly coincides with the LPR-model. Consider a TDM setup of period P and slice S for a job with execution time t_x such that $\frac{S}{t_x} = n$. Figure 5.3 shows Staschulat's LR-model for this setup.

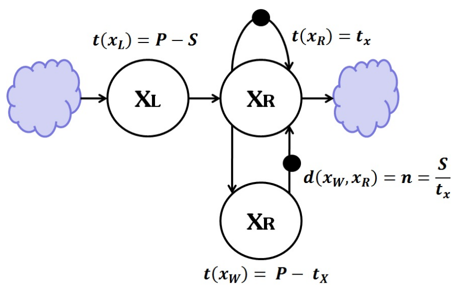

Figure 5.3: Staschulat's LR-model

In the proposed approach for the LPR model, we first identify the length of the cyclic pattern as $q = \frac{lcm(S,t_x}{S}$ periods. As S is the integer multiple of t_x we can say that the cyclic pattern is of the length $q = 1$ period. The number of phase actors for this model is $q + 1$ with firing times $t(x_1) = t_x$ and $t(x_2) = P - (S \% t_x) - t_x)$. As $S \% t_x = 0$, this firing time is reduced to $t(x_2) = P - t_x$. The firing time of initial waiting time actor w is unchanged ,i.e. $t(w) = P - S$ and the actors are connected as shown in Figure 5.4. The number of tokens on the edge (x_2, x_1) is given by $d(x_2, x_1) = \frac{S}{t_x}$.

The only difference between Staschulat's LR-model and the LPR-model is the additional

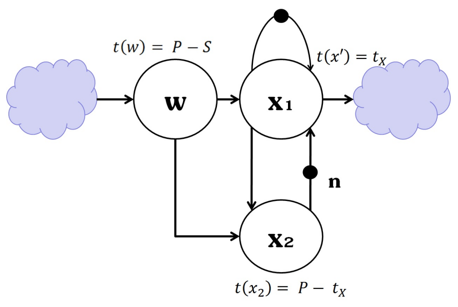

Figure 5.4: LPR-model under the assumptions in Staschulat's LR-model

edge (w, x_2) in the LCR-model. We observe that the firing constraint on the edge due to edge (w, x_2) is $a(i-n)+P-S$, while the firing constraint due to edge (x_1, x_2) is $f(i-n)$. The overall firing constraint on the actor x_2 is expressed as $\max(a(i-n)+P-S, f(i-n)$. We can show that the finish time $f(i-n)$ of the $(i-n)$-th iteration itself has the constraint $f(i-n) \geq a(i-n)+P-S$. Therefore, the firing constraint due to (w, x_2) is implicitly always satisfied by the firing constraint due to the edge (x_1, x_2). We can now safely remove the edge (w, x_2) from the model without affecting the modeled temporal behavior. The obtained LCR-model in now exactly the same as Staschulat's LR-model. We can conclude that the LCR-model is a generalized version of Staschulat's LR-model.

5.4.5 Modeling arbitrary execution times:

We have shown that we can conservatively model the worst-case temporal behavior of a TDM scheduled job with execution time t_x for a single iteration and allocated slice size S, such that $t_x \leq S$. We now remove this restriction to model arbitrary execution times allocated a slice such that $S \leq P$, where P is the replenishment period.

Theorem 3. *The worst-case effect of TDM arbitration on a job, for an arbitrary execution time t_x and a slice size S, can be modeled using a virtual slice $S' = y \cdot S$ and virtual period $P' = y \cdot P$, where $y \in \mathbb{N}$ such that $y \cdot S \geq t_x > (y-1) \cdot S$.*

Proof. We can show using induction that for all $1 \leq i \leq \frac{lcm(S,t_x)}{t_x}$, the worst-case temporal behavior of TDM arbitration given by Equation 5.12 for (P, S) is conservatively modeled for (P', S'). The entire proof is provided in Appendix B.3.

We observed that this approach to extend the use of our model has a negative impact on the accuracy of the model. In most cases, the proposed LPR-model would show improved accuracy in estimating the finish times of job iterations, in comparison to the LR-model. In some cases, however, it was observed that the model looses accuracy and the estimates on the finish time of certain job iterations may be worse than the LR-model. The complete proof of this anomaly has been provided in Appendix B.4. $\square$

We can conclude that the LPR-model is a definite improvement for modeling the worst-case temporal behavior of TDM arbitration under the assumption that the execution time of a single job iteration is less than or equal to the slice size. In the reverse case, however, our solution may not be optimal.

Multi-Rate and Cyclo-Static Data Flow:

We now illustrate that our model can also be used in multi-rate [12] and cyclo-static [15] environments, i.e. we can represent multi-rate data-flow based job actors using the

LPR-model. The arrival time is modeled as the time when all the firing conditions are met. The model does not restrict these conditions from requiring different number of tokens per incoming edge. The model can thus handle both multi-rate and cyclo-static consumption rates. Similarly, our model can also handle different production rates per outgoing edge. However, cyclo-static data flow actors are allowed to have different execution times per firing. As our model assumes a fixed execution time for all firings, it cannot handle varying execution times.

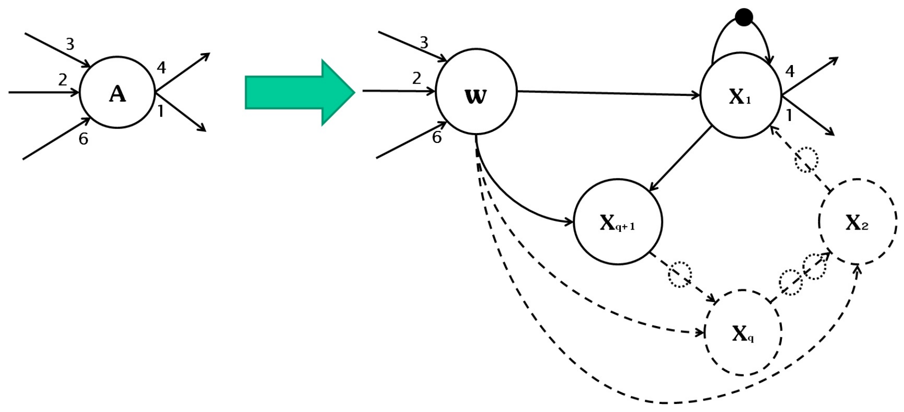

Figure 5.5: Representing a multi-rate job actor using the LPR-model

Figure 5.5 shows how a multi-rate actor can be represented using the LPR-model. The consumption rules defined on the incoming edges of job A are modeled as the consumtption rules for the actor w in the LPR-model. Similarly the production rules defined for the outgoing edges of A are modeled as the production rules for the outgoing edges for actor x_1 in the LPR-model.

5.5 Experimental Results

In this section, we quantify the difference between the LPR-model and the LR-model of [1], by comparing the over-estimation for a random set of arrivals of job-iterations. The over-estimation by a model is calculated as the percentage error in the model-defined response times from the actual response time. The response time of an iteration the difference between its arrival time and its finish time. We distinguish three categories of inter-arrival rates, as shown in Table 5.2, based on the TDM period and the execution time of a job. It is evident that our model has much tighter estimates of the worst-case temporal behavior of TDM arbitration.

Figure 5.6(a) illustrates the over-estimations made by the LPR-model and the LR-model with respect to the amount of service provided for a single random run. The LR-model displays an erratic and high over-estimation while the LPR-model shows a more stable

Inter-arrival Rates ↓	**LR-Model**	**LPR-Model**
Long ($rate \geq P$)	50% – 70%	< 1%
Medium ($P > rate > t_x$)	60% - 90%	20% – 30%
Small ($t_x \geq rate$)	18% – 25%	< 1%

Table 5.2: Comparison of overestimation for different arrival rate categories

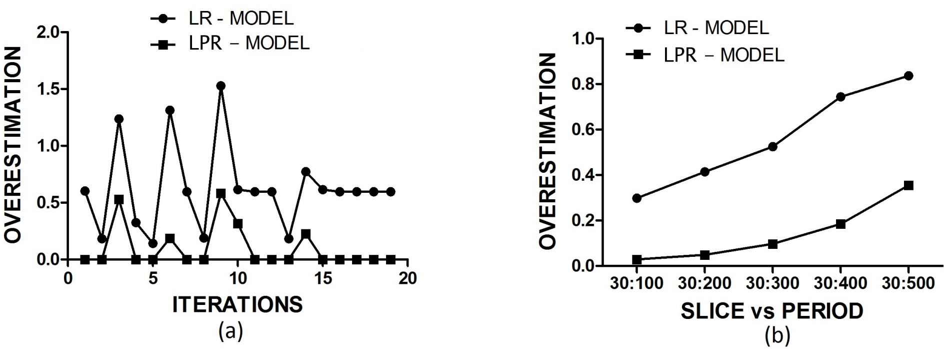

Figure 5.6: Comparison of over-estimations of the modeled response times

and tight overestimation.

Figure 5.6(b) illustrates the over-estimation of the different models for different slice to period ratio. We plot the average over-estimation made by each model in each case. Again, we observe that the over-estimation of the LPR-model is much tighter than the LR-model.

5.6 Summary

In this chapter we described a single-rate data-flow model for estimating the worst-case temporal behavior of TDM arbitration. The execution of continuous iterations of a job using TDM can be shown to have a cyclic pattern over a fixed number of replenishment periods. Our approach models this patterns using data-flow by capturing the execution behavior for each replenishment period separately. We initially assume that the execution time of a single job iteration is smaller that the size of a single slice. Thereby, we relax the restriction of Staschulat's LR-model [2] which requires that the size of the slice should be only an integer multiple of the execution time of a single job iteration. For this setup of TDM arbitration, we demonstrate that the model provides a conservative estimation that is tighter than the existing state-of-the-art models i.e. the LR-model [1].

We then show that we can extend this model to capture the effect of TDM arbitration for jobs with arbitrary execution times. We do this by modeling TDM arbitration using a virtual slice and period setup such that a single slice can accommodate the execution of a single job iteration. However, when we extend this model to larger execution times, we demonstrate that the modeled estimations are comparable to the LR-modeled estimations but not always better. We may conclude that in such cases the proposed approach is not optimal.

Chapter 6

Modeling TDM Arbitration: Latency-Cyclic-Rate Model

In the previous chapter, we presented a modeling approach that provides an improved estimation of the worst-case temporal behavior of TDM arbitration as compared to the LR-model [1]. However, the modeled bound is still an over-estimation to the actual worst-case execution of job iterations scheduled using TDM arbitration. Also, in the case when the execution time of a single job iteration is greater than the size of a slice, the model loses accuracy in its estimation. Thus, the requirement to design a generic approach to model TDM arbitration is not completely satisfied. To overcome the shortcomings of the previous model, we re-draw our approach to propose a more generic model, called the *Latency-Cyclic-Rate (LCR) model.*

In this chapter, we first give an overview of the new approach. We then illustrate with an example, the new approach to model the cyclic execution pattern for this example. We then formalize the construction of the LCR-model and analyze the modeled bound for the worst-case temporal behavior of TDM arbitration. We demonstrate that the approach presented in this chapter, is in fact an accurate estimation of the worst-case temporal behavior of TDM arbitration. We then discuss the features of the LCR-model with the state-of-the-art models and the model proposed in the previous chapter.

6.1 Overview

In this chapter, we present another approach to model TDM arbitration using single-rate data-flow. The approach is similar to the previous model such that we still try to capture the cyclic pattern in the execution behavior of job iterations. The difference is we model the execution behavior of individual iterations rather than capturing the execution of these iterations according to individual slices in the cyclic pattern.

6.2 Sketch of the proposed approach

Consider the second example as depicted in section 4.1.1. Figure 6.1 shows a time line of the simulated execution. The first iteration executes after initially waiting for the first slice and executes for the entire slice from time 90 to 100. At the end of the slice at time 100 the execution of this job iteration is pre-empted and has to wait for the next slice to resume execution. The next slice allocation is from time 190, at which the execution of the first iteration resumes and finishes execution at time 195. The second iteration starts immediately from time 195 until the end of the slice time 200. The second iteration must wait for the next slice allocation and starts executing at time 290 and finishes exactly at the end of the slice at time 300. This pattern repeats itself from the third iteration onwards.

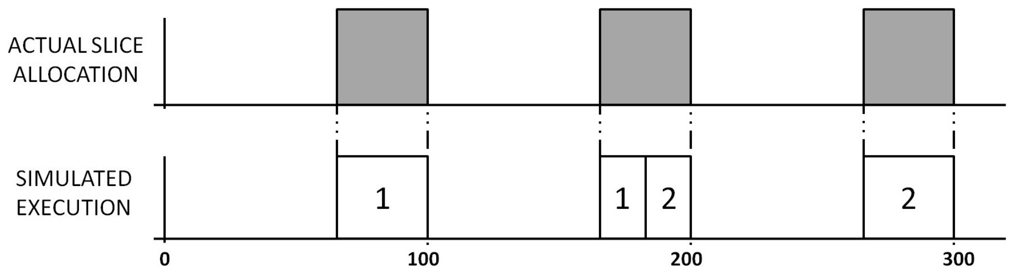

Figure 6.1: Modeled temporal behavior for the given TDM schedule

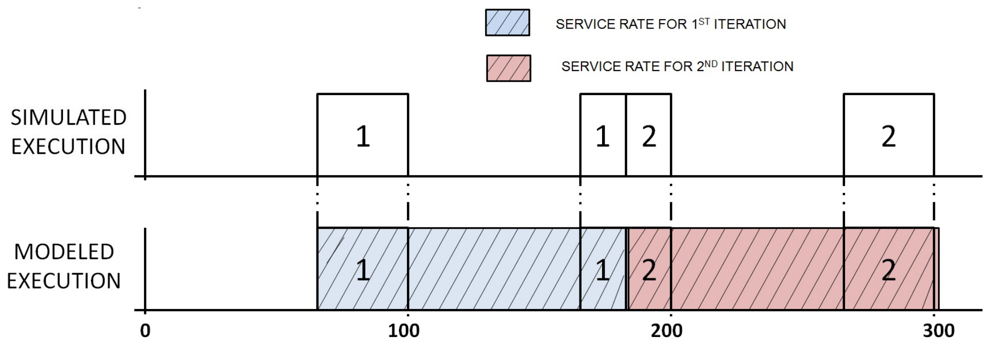
SERVICE RATE FOR 1ST ITERATION
SERVICE RATE FOR 2ND ITERATION
SIMULATED EXECUTION
1
1
2
2
MODELED EXECUTION
1
1
2
2
0
100
200
300

Figure 6.2: Modeled temporal behavior for the given TDM schedule

As explained in the previous approach, the arrival time and finish time of a job iteration are the only relevant aspects of the temporal behavior, irrespective of when the execution actually happens. Instead of modeling the actual execution of the iterations and the intermittent times separately, we club the entire time, from the start of the execution of an iteration to its finish, as the total service time for that iteration. This allows us to define the rate per iteration of the job execution for the entire cyclic pattern, as shown in Figure 6.2. The service rate of each iteration in the cyclic pattern is modeled as a separate node in the data-flow model as chained dependency on the service rate of the previous iteration, as shown in Figure 6.3. Table 6.1 shows the temporal behavior defined by the LCR-model in comparison to the LR-model proposed in [1].

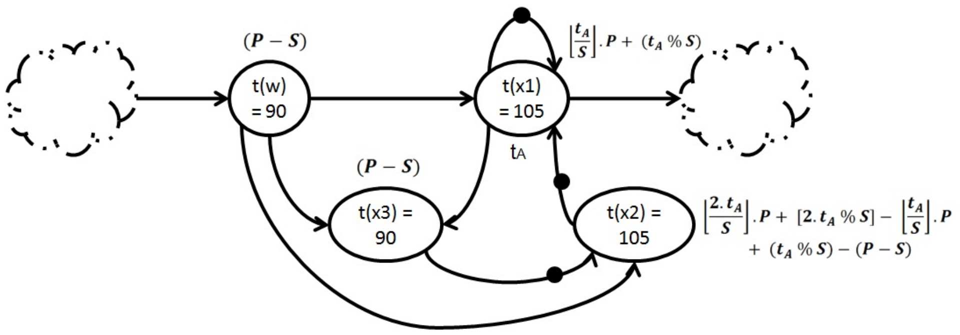
$(P-S)$
t(w) = 90
$\left\lfloor \frac{t_A}{S} \right\rfloor . P + (t_A \% S)$
t(x1) = 105
tA
$(P-S)$
t(x3) = 90
t(x2) = 105
$\left\lfloor \frac{2.t_A}{S} \right\rfloor . P + [2.t_A \% S] - \left\lfloor \frac{t_A}{S} \right\rfloor . P + (t_A \% S) - (P-S)$

Figure 6.3: Proposed data flow model

Iterations $\rightarrow$	**1**	**2**	**3**	**4**	**5**	**6 ...**
Actual Finish Time	195	300	495	600	795	900 ...
LR-Modeled Finish Time	240	390	540	690	842	990 ...
LCR-modeled Finish Time	195	300	495	600	795	900 ...

Table 6.1: Comparison of obtained finish times using LR-model and our model

6.3 Constructing the proposed data flow model

Formally, a job executing on a TDM scheduled resource is described as a 3-tuple $j_A = (P, S, t_x)$ where P is the replenishment period of the TDM arbitration, S is the size of the slice allocated to the job and t_x is the execution time of a single job iteration. The temporal behavior has a cyclic pattern of length q periods expressed as:

$$q = \frac{lcm(S, t_x)}{t_x} \tag{6.1}$$

We say that that the total execution time $q.t_x$, require exactly q slices to complete execution. Each job can be modeled as a SRDF graph component $G(P, S, t_x) = (V, E, d, t)$. V is the set of actors of the model given by:

$$V = \{x_i | 1 \leq i \leq q+1\} \cup \{w\}, \tag{6.2}$$

where each rate actor $x_i \in V$ describes the service rate for a specific iteration in the cclic pattern with the exception of the actor x_{q+1}. The rate actor x_{q+1} is an additional actor, placed to complete the cyclic behavior to be modeled. The latency actor w is used to represent the effect of the initial waiting time on the latency of a job iteration.

The edges are defined such that the phase actors form a cycle, and there is an edge from the latency actor w to each of the phase actors. There is also a self-edge from the actor x_1 to itself.

$$E_A = \{(x_{i+1}, x_i) | 1 \leq i \leq q\} \cup \{(x_1, x_{q+1})\} \cup \{(w, x_i) | 1 \leq i \leq q+1\} \tag{6.3}$$

We now add initial tokens (delay) d on each edge. Each token in the cycle of $x_i \in V$ rate actors corresponds to a single job iteration. Hence there will be a single token on each edge in the cycle. The delay can be expressed as:

$$d(u,v) = \begin{cases} 1 & \text{for } u = x_{i+1} \ \& \ v = x_i \text{ and } 1 \leq i \leq q, \\ 0 & \text{for } u = x_1 \ \& \ v = x_q \\ 0 & \text{for } u = w \ \& \ v = x_i \text{ and } 1 \leq i \leq q. \end{cases} \tag{6.4}$$

The firing time of the latency actor is given by $t(w) = P - S$ while the firing time $t(x_i)$ of each phase actor $x_i \in V$ defines the execution behavior of the associated job iteration, expressed as:

$$t(v) = \begin{cases} P - S & \text{for } v \in \{w, x_{q+1}\}, \\ \lfloor \frac{i \cdot t_x}{S} \rfloor \cdot P + [i \cdot t_x \ \% \ S] & \\ \qquad - \lfloor \frac{(i-1) \cdot t_x}{S} \rfloor \cdot P + [(i-1) \cdot t_x \ \% \ S] & \text{for } v = x \ \& \ 1 \leq i < q, \\ \lfloor \frac{i \cdot t_x}{S} \rfloor \cdot P + [i \cdot t_x \ \% \ S] & \\ \qquad - \lfloor \frac{(i-1) \cdot t_x}{S} \rfloor \cdot P + [(i-1) \cdot t_x \ \% \ S] - (P - S) & \text{for } v = x_q. \end{cases} \tag{6.5}$$

We have defined the the set of actor V and the edges E connecting the actors in the LCR-model in Equations 6.2 and 6.3 respectively. Equation 6.4 defines the delay for each edge $e \in E$ in the LPR-model, while Equation 6.5 defines the firing time of each actor $v_i \in V$ for the LPR-model. Thus, we provide a formal definition for the construction of the LCR-model as (V, E, d, t) for the TDM arbitration setup for the job (P, S, t_x).

6.4 Analyzing the worst-case bound of the constructed model

In this section we now analyze the LCR-model construction described in the previous section. We first establish the bound defined by the LPR-model for the worst-case temporal behavior of TDM arbitration. We then demonstrate that the LCR-model is conservative. We then compare the LCR-model with the state-of-the-art model [1, 2]. We do not make any assumptions on the execution time of a single job iteration or the size of the allocated slice. This makes the LCR-model a truly generic model.

6.4.1 Establishing the LCR-modeled bound for the worst-case temporal behavior for TDM arbitration

Similar to the previous approach, we again derive a *max*-equation for the worst-case temporal behavior of TDM arbitration represented using the LCR-model. There are $q+1$ paths from the actor w to the actor x_1. Additionally there is a cyclic path from the actor x_1 to itself via all the other rate-actors, i.e. the path $\{x_1, x_{q+1}, x_q, ..x_1\}$. Summing up the firing times of all actors in each path, we derive the max-equation as

$$\hat{f}(i) = \max \begin{cases} \max_{1 \le j < q}\{\hat{a}(i-j) + (P-S) + \lfloor \frac{(j+1)\cdot t_x}{S} \rfloor \cdot P + [(j+1)\cdot t_x \ \% \ S]\}, \\ \hat{a}(i-(q-1)) + \frac{q \cdot t_x}{S} \cdot P, \\ \hat{f}(i-q) + \frac{q \cdot t_x}{S} \cdot P. \end{cases} \tag{6.6}$$

where $q = \frac{lcm(S,t_x)}{t_x}$. The above equation defines the LCR-modeled bound for the worst-case temporal behavior of a job scheduled using TDM arbitration.

6.4.2 Conservativity of the model:

We now prove that our model is conservative in defining the worst-case temporal behavior of TDM arbitration. The formal definition of conservativity as given in section 3.4 is expressed as:

$$\forall_{i \ge 0} \ a(i) \le \hat{a}(i) \ \Rightarrow \ \forall_{i \ge 0} \ f(i) \le \hat{f}(i) \tag{6.7}$$

where $a(i)$ and $f(i)$ are the actual start time and finish time, respectively, of the i-th job iteration, while $\hat{a}(i)$ and $\hat{f}(i)$ are the corresponding LPR-modeled start and finish time respectively.

Theorem 4. *The graph $G(P, S, t_x)$ as defined for the LCR-model, is a conservative model for a TDM-scheduled job with worst-case execution time t_x, that is assigned a slice S per period P.*

Proof. We have formalized the worst-case behavior of TDM arbitration in section 3.2 as:

$$f_n = \begin{cases} s_n + \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P & \text{if } n \cdot t_x \ \% \ S = 0 \\ s_n + \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P + (P-S) + (n \cdot t_x \ \% \ S) & \text{if } n \cdot t_x \ \% \ S \ne 0 \end{cases}, \tag{6.8}$$

where s_n and f_n are the start time and finish time respectively, of a consecutive sequence of multiple job iterations of length n. In order to prove conservativity of the LCR-model, we show that worst-case temporal behavior of TDM arbitration defined by the LCR-model is an upper bound to this formalization defined in section 3.2. We show that each iteration can be associated (represented in the model) with a token present on the cycle of rate actors $\{x_1, \cdots, x_n + 1\}$ in the proposed model. The execution behavior of a job iteration can be enumerated as follows:

1. We can say that $n \cdot t_x \ \% \ S = 0$ only when $n.t_x$ is an integral multiple of $lcm(S, t_x)$ with the smallest value of this multiple is given by $n = \frac{lcm(S,t_x)}{t_x}$. We say that in a sequence of $n = \frac{lcm(S,t_x)}{t_x}$ consecutive job iterations the start of the sequence is given by $s_n = a(i - (n-1))$ and the finish time of the sequence is given by the finish time of the last iteration i.e. $f_n = f(i)$. We can now express the finish time of the sequence as:
$$f(i) = a(i - (n-1)) + \frac{n \cdot t_x}{S} \cdot P \tag{6.9}$$
In the max-plus bound of our model in Equation 5.10, we get
$$\hat{f}(i) = \hat{a}(i - (q-1)) + \frac{q \cdot t_x}{S} \cdot P \tag{6.10}$$
We know that $n = \frac{lcm(S,t_x)}{t_x} = q$ and that $a(i) \leq \hat{a}(i)$, we can say that:
$$f(i) = a(i-n) + \frac{n \cdot t_x}{S} \cdot P \leq \hat{a}(i-q) + \frac{q \cdot t_x}{S} \cdot P = \hat{f}(i) \tag{6.11}$$
All sequences of length $n > \frac{lcm(S,t_x)}{t_x}$ can be expressed as consecutive sequences of length $n' = \frac{lcm(S,t_x)}{t_x} = q$ from the finish of the $(i-n')$-th job iteration. Since $n' \cdot t_x \ \% \ S = 0$ we express this behavior as:
$$f(i) = f(i-n') + \frac{n' \cdot t_x}{S} \cdot P \tag{6.12}$$
This behavior can be realized in our model with the bound:
$$\hat{f}(i) = \hat{f}(i-q) + \frac{q \cdot t_x}{S} \cdot P \tag{6.13}$$

2. For a sequence of consecutive iterations whose length is given by $n < \frac{lcm(S,t_x)}{t_x}$ we know that $n.tA \ \% \ S \neq 0$, we can consider that $(i-k)$-th job iteration is first iteration in the consecutive sequence and the i-th iteration is the last iteration in the sequence (i.e. $s_n = a(i-k)$ and $f_n = f(i)$). The length of the sequence is given by $n = k+1$ where $k < \frac{lcm(S,t_x)}{t_x} - 1$. We can now express the worst-case temporal behavior of this sequence as:
$$f(i) = a(i-k) + (P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + (k+1) \cdot t_x \ \% \ S \tag{6.14}$$
This behavior is realized in our model with the bound:
$$\max_{1 \leq j < q} \{ \hat{a}(i-j) + (P-S) + \lfloor \frac{(j+1) \cdot t_x}{S} \rfloor \cdot P + [(j+1) \cdot t_x \ \% \ S], \tag{6.15}$$
where $q = \frac{lcm(S,t_x)}{t_x} - 1$ and $j = k$.

□

Corollary 1. *If the modeled arrival time of job iterations are same as the actual arrival times of the job iterations, then the graph $G(P, S, t_x)$ as defined for the LCR-model, is an exact representation a TDM-scheduled job with worst-case execution time t_x, that is assigned a slice S per period P.*

$$\forall_{i \geq 0}\ a(i) = \hat{a}(i) \ \Rightarrow\ \forall_{i \geq 0}\ f(i) = \hat{f}(i) \tag{6.16}$$

Proof. We observe from the above conservativity proof that, for each enumeration, the modeled bound exactly coincides with the formal definition of the worst-case temporal behavior of a job scheduled using TDM arbitration. We can, therefore, conclude that our model is an accurate representation of the worst-case temporal behavior of TDM arbitration of a job. □

6.4.3 Comparison with the LR-model

In chapter 4 we have already discussed that the LR-model is pessimistic in defining the worst-case temporal behavior of TDM arbitration. In other words the finish-times of job iterations estimated by the LR-model are an over approximation of their actual finish times. This also implies that the temporal estimation made using the LR-model will be less accurate than the LCR-model, as the LCR-model is in fact an accurate representation of the worsy-case temporal behavior of TDM arbitration.

6.4.4 Comparison with Staschulat's LR-model

Staschulat's LR-model assumes that size of the slice allocated for a job is an integer multiple of the execution time of a single job iteration. The LCR-model does not make this assumption, and is a generic approach to model the temporal behavior or jobs scheduled using TDM arbitration. Our model can accommodate arbitrary execution times and arbitrary slice sizes.

However, under the assumption made by Staschulat's LR-model, we can show that its model construction coincides with an optimized version of the proposed model. Consider a TDM setup of period P and slice S for a job with execution time t_x such that $\frac{S}{t_x} = n$. Figure 6.4 shows Staschulat's LR-model for this setup.

In our approach, we show that the cyclic pattern repeats after $q = \frac{lcm(S,t_x}{t_x} = \frac{S}{t_x} = n$ job iterations. Our model represents the execution each iteration using a separate actor.

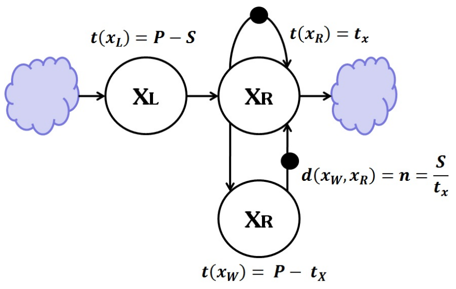

Figure 6.4: Staschulat's LR-model

The firing time for the actor of all but the last iteration, according to Equation 6.5 , is given by:

$$\begin{aligned} t(x_i) &= \lfloor \frac{i \cdot t_x}{S} \rfloor \cdot P + [i \cdot t_x \ \% \ S] \\ &\quad - \lfloor \frac{(i-1) \cdot t_x}{S} \rfloor \cdot P + [(i-1) \cdot t_x \ \% \ S], \end{aligned} \tag{6.17}$$

where $1 \leq i < n$. As we know that $n = \frac{S}{t_x}$, we can conclude that $i \cdot t_x < S$ for $1 \leq i < n$. Therefore we can reduce Equation 6.17 to $i.t_x - (i-1).t_x$ or just t_x. For the n-th iteration we define the firing time of its corresponding actor in the model is defined by

$$\begin{aligned} t(x_n) &= \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P + [n \cdot t_x \ \% \ S] \\ &\quad - \lfloor \frac{(n-1) \cdot t_x}{S} \rfloor \cdot P + [(n-1) \cdot t_x \ \% \ S] - (P - S). \end{aligned} \tag{6.18}$$

Again, as we know that $n = \frac{S}{t_x}$, this expression also gets reduced to t_x. We now have a model with n actors with firing time t_x for the n iterations in the cyclic pattern and two additional actors w and x_{n+1} with firing times $t(w) = t(x_{n+1}) = P - S$, as shown in Figure 6.5.

For each actor x_i where $1 \leq i \leq n$ there is an incoming edge (x_{i+1}, x_i) with delay $d(x_{i+1}, x_i) = 1$. There is an additional edge for each of these actors from the actor w with no delay. According to optimization techniques for data-flow described in [11], we can represent this sequence of n identical actors with a single actor x' having firing time t_x and a self-edge with a single delay, and an edge from actor x_{n+1} with delay $d(x_{n+1}, x') = n$. Also the firing time of the actor x_{n+1} is now the sum of all the actors in the cycle and deducting from this some the firing time of actor x'. The can be expressed

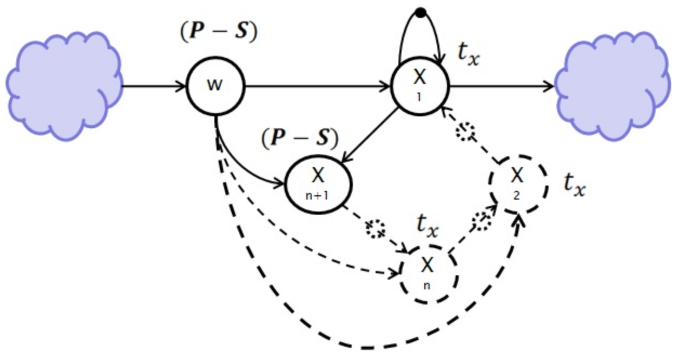

Figure 6.5: The LCR-model under the assumptions of Staschulat's LR-model

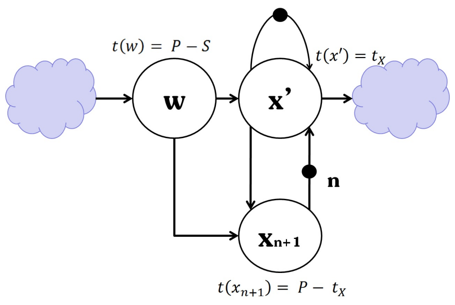

Figure 6.6: Optimization of the LCR-model under the assumptions of Staschulat's LR-model

as $\sum_{1\leq i\leq n+1} t(x_i) - t(x')$. As $\sum_{1\leq i\leq n+1} = P$ the firing time of actor x_{n+1} is expressed as $t(x_{n+1}) = P - t_x$. The optimized model is shown in Figure 6.6

As also explained in the LPR-model, the firing constraint due to the edge (w, x_{n+1}) is implicitly always satisfied by the firing constraint due to the edge (x', x_{n+1}), and hence can be removed from the model without affecting the modeled temporal behavior. We can observe that this model directly coincides with Staschulat's LR-model. We can conclude that the LCR-model is truly a generic model as it does not make any assumption on the characteristics of TDM arbitration. Also, under the assumptions considered by the other approaches, the LCR-model is a superposition of these models.

Multi-Rate and Cyclo-Static Data Flow:

We now show that our model can also be used in multi-rate [12] and cyclo-static [15] environments. The arrival time is modeled as the time when all the firing conditions are met. The model does not restrict these conditions from requiring different number of tokens per incoming edge. The model can thus handle both multi-rate and cyclo-static consumption rates. Similarly, our model can also handle different production rates per outgoing edge. However, cyclo-static data flow actors are allowed to have different execution times per firing. As our model assumes a fixed execution time for all firings, it cannot handle varying execution times.

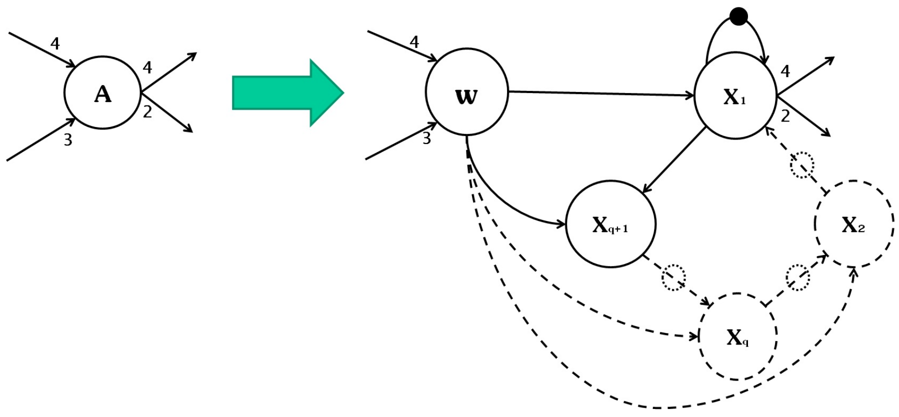

Figure 6.7: Representing a multi-rate job actor using the LCR-model

Figure 6.7 shows how a multi-rate actor can be represented using the LCR-model. The consumption rules defined on the incoming edges of job A are modeled as the consumtption rules for the actor w in the LPR-model. Similarly the production rules defined for the outgoing edges of A are modeled as the production rules for the outgoing edges for actor x_1 in the LPR-model.

6.5 Summary

In this chapter, we propose a new approach to model the cyclic patter of execution of a job under the effect of TDM arbitration. The proposed LCR-model overcomes the disadvantages in the LPR-model (proposed in chapter 5) such that it does not make any assumption on the execution time of a single job iteration. The LCR-models the execution rate of each individual iteration instead of the modeling the execution rate per phase slice of the cyclic pattern.

We demonstrated that this model provides an accurate estimation of the worst-case temporal behavior of an arbitrary TDM setup. We also conclude that the LCR-model is better than the existing state-of-the-art models and the proposed LPR-model. We also demonstrated that under the assumptions on TDM arbitration made by Staschulat's LR-model [2], the LCR-model obtained for the same TDM setup can be further optimized using data-flow optimization techniques [11] to obtaing a reduced model, that directly coincides with Staschulat's LR-model. The LCR-model is, therefore, a generic model that captures he worst-case temporal behavior of job iterations scheduled using TDM arbitration.

Chapter 7

Modeling TDM Arbitration: Multi-Rate Data-Flow Approach

Till now we have presented single-rate data-flow approaches to model the worst-case temporal behavior of TDM arbitration. In this chapter, we use a different flavor of data-flow, namely multi-rate data-flow, to capture the effect of TDM arbitration on a job. We first recapitulate the concept of multi-rate data-flow, and give a brief introduction to our approach. We then illustrate with an example, the use of multi-rate data-flow to capture the effect of arbitration of a job using TDM. Following this, weco formalize the construction of the proposed multi-rate data-flow model. In the analysis of this model, we argue that the MRDF-model is conservative. Due to lack of time, we are not able to present a formal proof of its conservativity and accuracy, which is left as future work. Next, we describe the conversion of this model into a single-rate data-flow model and address the effects of this expansion. We also address some approaches to optimize this expansion. We then compare of our multi-rate model with respect to the other models proposed in this thesis, and the state-of-the-art model existing in literature.

7.1 Overview

Let us briefly re-visit the concepts of multi-rate data-flow. Multi-Rate Data-Flow graphs (MRDF graphs) are a generalization on Single-Rate Data-Flow (SRDF) graphs such that, the number of tokens consumed and produced for all firing of a multi-rate actor is fixed to a non-negative integer. MRDF graphs are more expressive than SRDF graphs, as they can model complex behavior in a more condensed form. For instance, consider that we have two jobs A and B such that for every execution of the job A there are three executions of the job B. Figure 7.1(a) shows the use of single-rate data-flow to models this behavior, while Figure 7.1(b) models this behavior using multi-rate data-flow. The SRDF equivalent of MRDF graphs is an exponential expansion of the MRDF graph having a large number of redundant actors [11, 12].

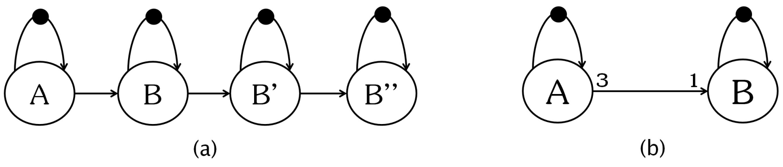

Figure 7.1: For every execution of job A there are two executions of job B.

In chapters 5 and 6, we presented two approaches that capture the cyclic pattern of execution of consecutive job iterations and obtain a bound on this behavior using single-rate data-flow. In this chapter, we present an approach which captures the effect of TDM arbitration on individual iterations of a job.We can make use of the expressiveness of multi-rate data-flow to capture the actual consumption of the slice per replenishment period, by the iterations of a job as and when they arrive. Instead of modeling a job iteration as a single token, we split it into multiple tokens according to the total execution time of a single job iteration. Each token represents to a fixed amount of execution time of an iteration and is modeled such that it consumes the equivalent amount of time from the alloted slice. The availability of slice time for execution is replenished after every period. On the output side, all tokens of a single iteration are consumed in a single firing to produce an output token.

7.2 Sketch of the proposed approach

Consider a simple TDM scheduler with a replenishment period $P = 10$ that schedules a job with execution time $t_x = 5$ in a slice $S = 3$ per replenishment period. Like in the illustrations for the other models, we assume that each job iteration can execute as soon as the previous iteration finishes, i.e. all iterations are assumed to be ready to execute at the start of the system. The slice allocated per period, is located just before the start of the next period, as shown in Figure 7.2.

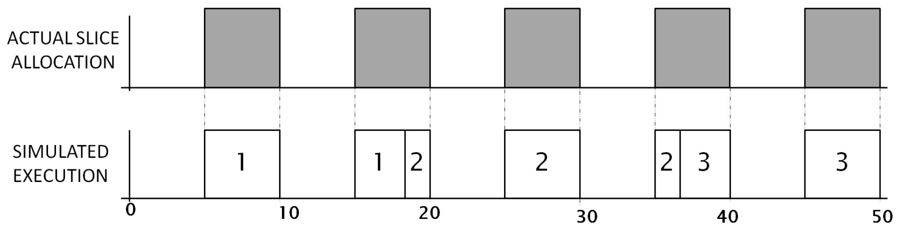

Figure 7.2: Actual temporal behavior for the given TDM schedule

The above scenario can be modeled by our MRDF-model as shown in Figure 7.3. Each input token arriving on the incoming edge of the actor w models the arrival of a job iteration. Let us consider, for now, only a single iteration represented as a single token on the incoming edge of the actor w. The firing condition for w requires a single input token on its incoming edge to fire. Upon firing, actor w will produce 5 tokens (since the execution time of a single iteration is also 5) according to the production rule defined on edge (w, x_1). According to the consumption rules defined on edges (w, x_1) and (x_2, x_1), each firing of actor x_1 consumes a single token from both these edges. Also for each firing of x_1 a single token is produced on edges (x_1, w') and (x_1, x_2), as defined by the production rules of these edges respectively.

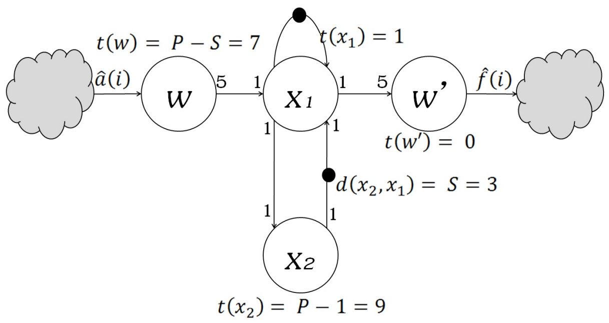

Figure 7.3: Multi-rate data-flow model for a job scheduled using TDM

Since the job iteration is already ready at the start of the system, actor w fires immediately and produces five tokens on edge (w, x_1) at time 7. The first three tokens will be consumed one after the other (due to the self edge on x_1 with a single delay) at time 7,8, and 9. Similarly, upon the three consecutive firings of x_1, output tokens will be produced on the outgoing edges (x_1, w') and (x_1, x_2) at time 8,9, and 10. At time 10, there are no more tokens present on the edge (x_2, x_1), even though there are still two tokens from the first iteration to be consumed on the edge (w, x_1). The first firing of the actor x_2 will commence at time 8, when the first token is produced on the edge (x_1, x_2), and produce a token at time 17. Similarly the second and third firing of x_2 will produce tokens on the edge (x_1, x_2) at times 18, and 19. The tokens produced on the edge (x_1, x_2) at times 17, and 18 will be consumed by x_1 as soon as they are produced and correspondingly produce tokens on edges (x_1, w') and (x_1, x_2) at time 18 and 19. At time 19, a total on five tokens are present on the edge (x_1, x_2) (i.e. tokens produced at times 8,9,10,18, and 19). All five tokens are consumed in a single firing of the actor w', with a firing time $t(w') = 0$, an produce one output token at the end of the firing. The purpose of modeling the actor w' is only to combine the individual tokens, produced as the result of execution of a single job iteration, into a single output token that corresponds to the finish of a single iteration. To summarize the modeled execution behavior, the first iteration executes for the entire first slice, from time 7 to 10, and then resumes execution in the second slice at time 17 and completes execution at time 19. We can observe that this execution behavior directly coincides with the actual execution of the first iteration according to Figure 7.2. Table 7.1 shows the actual finish times and the modeled finish times of each iteration for the current example.

Iterations →	**1**	**2**	**3**	**4**	**5...**
Actual Finish Time	19	38	50	69	88 ...
LR-modeled Finish Time	23.66	40.33	56.99	73.66	90.33 ...
MRDF-modeled Finish Time	19	38	50	69	88 ...

Table 7.1: Comparison of obtained finish times using LR-model and our model

7.3 Construction of the proposed model

The MRDF model can be represented as a 5-tuple (V, E, d, f, t). The set of actors V constitutes of 4 actors w, x_1, x_2, w'. The edges for the model are given by $E = (w, x_1), (x_1, x_2), (x_2, x_1), (x_1, w')$. The delay for the edge (x_2, x_1) is given by $d(x_2, x_1) = S$ where S is the size of the slice allocated for the job. The delay for all other edges in the model in 0. The firing constraints for the edges (w, x_1) and (x_1, w') is expressed as $f(w, x_1) = (t_x, 1)$ and $f(x_1, w') = (1, t_x)$, where t_x is the execution time for a single job iteration. The firing constraint for other edges is set to $(1, 1)$. Figure 7.4 shows the stated construction.

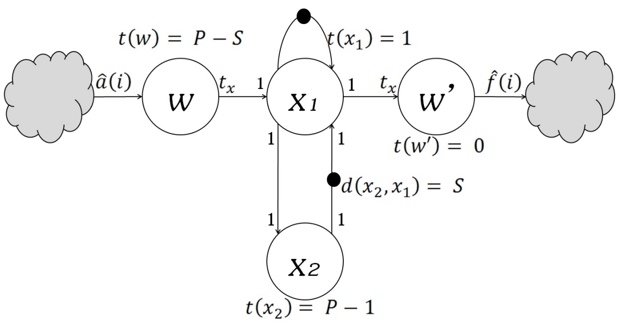

Figure 7.4: Multi-rate data-flow model construction for a job scheduled using TDM

7.4 Discussion

In this section, we now present a discussion on various issues of the proposed multi-rate model. We first analyze the model construction and argue that it is conervative. We then show that the given model can be converted into a single-rate data flow model and discuss the effects of this conversion on the size of the obtained single-rate graph and its analysis. We then present a few optimization techniques to address the issues of this multi-rate to single-rate conversion of our model. Lastly, we discuss the model presented in this chapter to the other models presented in chapters 5 and 6, and the state-of-the-art models present in literature.

7.4.1 Analysis of the correctness the constructed model:

We argue that the constructed model has a direct co-relation the consumption of slice by individual job iterations. The entire proof for the correctness of the model is not presented in this thesis due to lack of time and is left for future work. We consider, however, that the behavior of the model is intuitive enough to argue that it accurately models the worst-case temporal behavior of TDM arbitration.

7.4.2 Converting the MRDF-model to a SRDF-model:

We have explained in the section 2.3 that MRDF graphs can be converted to SRDF graphs using conversion algorithms [11, 12]. Using this conversion technique, we can expand our MRDF model for the example shown in Figure 7.3 into a SRDF model, shown in Figure 7.5.

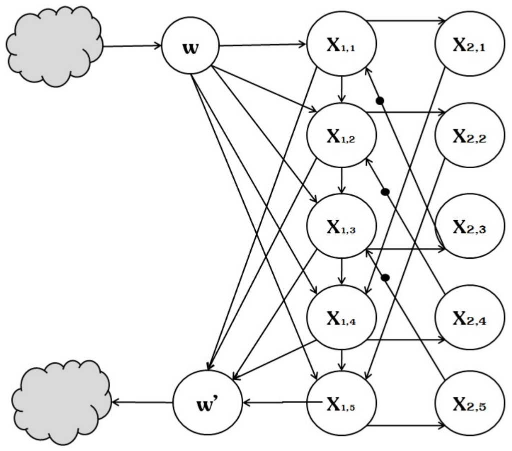

Figure 7.5: SRDF model equivalent of the MRDF model

The repetition vector for the MRDF-actors w, x_1, x_2, w' can be stated as

$$\vec{q} = \begin{pmatrix} 1 \\ t_x \\ t_x \\ 1 \end{pmatrix} \tag{7.1}$$

Each actor is replicated $q(v_i)|v_i \in V$ times according to the repetition vector $\vec{q}$. The edges and delay are distributed such that the firing of the actors in the SRDF model, is identical to the firing of the corresponding actors in the MRDF-model.

7.4.3 Optimizing the obtained SRDF model

In section 7.4.2, we observe that converting the proposed MRDF model into an equivalent SRDF model results in a huge expansion of the graph. Also the example used for illustrating the model is rather simple. The actual execution times of a job iteration and the slice size are much larger for the jobs of actual applications. Expansion of the MRDF-models for these jobs will result in a huge state-explosion problem. We now suggest some basic optimization techniques to address the problems of expanding the proposed MRDF-model.

Eliminating unwanted edges from the SRDF expansion:

The graph obtained by expanding the MRDF-model is quite complex. Let us now try to optimize this graph. If there are multiple paths between two actors, we only consider the longest path (in terms of firing time). It must be noted that, for our model, we perform optimizations on paths that have zero delay on them (i.e we do not consider optimizing that have non-zero delays). We suspect that further optimizations of our model is possible, but this has not been addressed due to lack of time. Observe in Figure 7.5, there are edges from the actor w to each of the x_1 actors. The x_1 actors themselves are connected in a chain. If we consider all the paths from actor w to actor $x_{1,5}$, excluding the ones that have non-zero delays on them, we observe that the longest path is $\{w, x_{1,1}, x_{1,2}, x_{1,3}, x_{1,4}, x_{1,5}\}$. We can, therefore remove all the edges in other paths which are not present in the longest path. The same trick can be applied for all the paths from the actor $x_{1,1}$ to the actor w'. Figure 7.6 shows the optimized single-rate expansion for the multi-rate model described in Figure 7.3.

We also observe that the actor w' in the SRDF expansion serves no purpose as it simply transfer tokens from its incoming edge to its outgoing edge without consuming any additional time. We can, therefore, remove this actor from the SRDF expansion without affecting the behavior of the model.

Optimizing the initial MRDF model:

The number of nodes in the SRDF expansion depends upon the number of tokens produced by actor w per firing in the MRDF model. In other words, the number of nodes in the SRDF expansion depends upon the number of segments in which we divide the

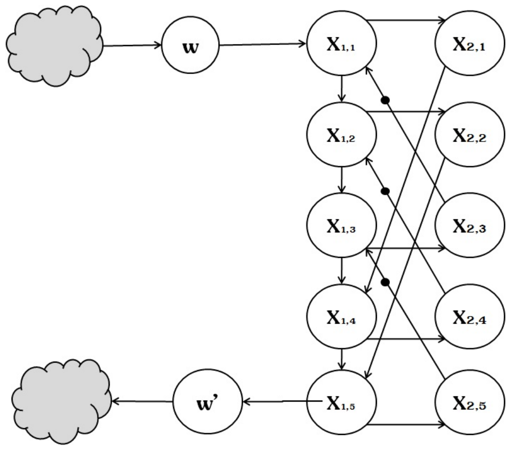

Figure 7.6: Optimized SRDF model equivalent of the MRDF model

the total execution time for a single iteration. By reducing the number of tokens produced per firing of the actor w in the MRDF model, we can reduce the size of the SRDF expnsion.

Currently, we divide the execution time into the finest resolution possible, i.e. the each token represents one unit of the execution time of a single job iteration. If each token represents a larger segment, say z time units, of the total execution time of a job iteration, the total number of tokens per firing is reduced to $\frac{t_x}{z}$. As each segment will take more time to execute and also consume a larger portion of the slice, we change the firing to of actor x_1 such that $t(x_1) = z$ and the delay on the edge (x_2, x_1) such that $d(x_2, x_1) = \frac{S}{z}$. It should be noted that the delay on an edge is a non-negative integer, hence the size of the segment (i.e. the value of z) is such that $S \ \% \ z = 0$. The maximum size of z, using this approach is expressed as $z = \gcd(S, t_x)$ where t_x is the total execution time of a single job iteration and S is the slice size allocated to the job. Figure 7.7 shows the optimized MRDF model. Expanding the optimized MRDF-model will result in fewer nodes in the expanded single-rate model equivalent of the optimized multi-rate model.

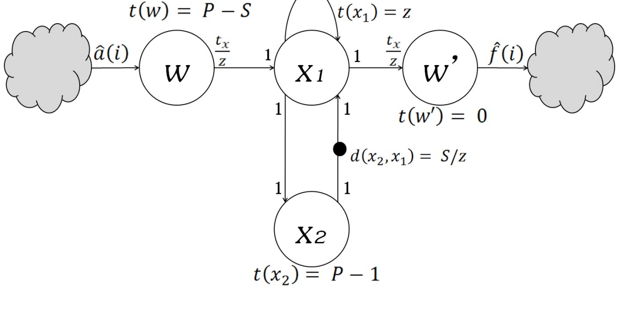

Figure 7.7: Optimized multi-rate data-flow model construction for a job scheduled using TDM

7.4.4 Comparison with other models

The MRDF-model is an intuitive approach to model the execution behavior of individual job iterations, under the effect of TDM arbitration. Meanwhile, the previous models presented in chapters 5 and 6, try to model the cyclic pattern of behavior consecutive sequences of job iterations.

The proposed MRDF model is, intuitively, an accurate model representation of the worst-case temporal behavior of TDM scheduled jobs as compared to the LR-model [1], which is an over-estimation of the temporal behavior of TDM arbitration. Under the assumptions made in [2], we claim that the proposed MRDF model can express the same behavior as Staschulat's LR-model [2]. If we assume that the slice size S is an non-negative integer multiple of the execution time t_x of a single job iteration, we can say that $z = \gcd(S, t_x) = t_x$. Using the optimization presented in section 7.4.3 we get the firing time of actor x_1 as $t(x_1) = z = t_x$. The production rule for the edge (w, x_1) (and consumption rule for the edge $(x_1, w'))$ is defined as $\frac{t_x}{z} = \frac{t_x}{t_x} = 1$. Lastly, the delay on the edge (x_2, x_1) is defined as $d(x_2, x_1) = \frac{S}{z} = \frac{S}{t_x}$. The optimized model now obtained directly coincides with Staschulat's LR-model. Thus, we can conclude that the proposed MRDF model is a multi-rate data-flow extension of Staschulat's LR-model such that it can model arbitrary execution times for arbitrary slice allocation.

7.5 Summary

In this chapter, we presented a different approach to model the worst-case temporal behavior of TDM scheduled jobs, i.e. using multi-rate data-flow. Instead of modeling a job iteration as a single token, we split it into multiple tokens according to the total execution time of a single job iteration. Each token represents to a fixed amount of execution time of an iteration and is modeled such that it consumes the equivalent amount of time from the alloted slice. The availability of slice time for execution is replenished after every period. On the output side, all tokens of a single iteration are consumed in a single firing to produce an output token.

We argued about the correctness and the accuracy of the model through an illustrative example. We addressed issues, such as expanding the multi-rate model into an equivalent single-rate data-flow model and also on various optimizations possible on the models. We also compared the multi-rate model to the other models proposed in this thesis and the existing state-of-the-art models. We argue that the proposed MRDF model is more accurate the the LR-model [1]. We also claim that Staschulat's LR-model [2] can be perceived as a special instance of the proposed MRDF model.

Chapter 8

Discussion and Future Scope

In this chapter, we address some observations made during the course of this project and certain key areas that should be explored further as future work. We first describe the complexity of the generated graphs using the models proposed in this thesis in comparison with the existing state-of-the-art models, namely the LR-model [1] and Staschulat's LR-model [2]. We then address the issue of using the proposed models for TDM arbitration with static-ordering. Lastly we address the issue extending the use of our model to a broader class of budget schedulers.

8.1 Complexity of the Generated Graphs

We now present the complexity of the proposed models in comparison with the existing state-of-the-art models. in capturing the worst-case behavior of jobs scheduled using TDM arbitration. We address the complexity of the models with respect to two factors, namely the size of the generated graph and the techniques used to analyze the generated graphs.

8.1.1 Size Complexity

The worst-case execution behavior of jobs under TDM arbitration is observed as a cyclic pattern that repeats after a fixed number of consecutive job iterations. The LR-model defines a linear bound on the this cyclic pattern of TDM arbitration by generalizing the execution rate of consecutive job iterations. It is further observed that, the bound defined by the LR-model is the tightest possible linear bound on this worst-case temporal behavior.

The models proposed in this thesis define a periodic non-linear bound, enabling more accurate estimation of the worst-case behavior of the jobs scheduled using TDM. The accuracy of these models comes at the cost of using more actors to capture the worst-case behavior of TDM arbitration. For instance, the LPR-model proposed in chapter 5 defines the cyclic pattern in terms of phased-slices such that the number of actors used for describing the total phase-slices is given by $q = \frac{lcm(S,t_x)}{S} + 1$, where S is size of the slice allocated to the job with execution time t_x for a single job iteration. The there is an additional actor for modeling the initial waiting time. The total number of actors used in the LPR-model for a job is given as $\frac{lcm(S,t_x)}{S} + 2$. We can say the the size complexity of the LPR-model is linear in the execution time t_x of a single job iteration.

The LCR-model defines a non-linear bound for the worst-case temporal behavior of jobs under TDM arbitration, based on the rate of execution of each individual iteration in the cyclic pattern. The number of actors for defining the cyclic pattern is one more than the number of iterations in the cyclic pattern expressed as $q = \frac{lcm(S,t_x)}{t_x} + 1$ and the total number of actors in the model, including the actor for the initial waiting time, is given as $\frac{lcm(S,t_x)}{t_x} + 2$. We can say the the size complexity of the LCR-model is linear in slice size S allocated for a job.

The number of actors used in both the proposed SRDF models, i.e. the LPR-model and the LCR-model, depends on the size of the slice allocated for the job and the execution time for a single job iteration. In the case the execution time of a job iteration is more than the size of slice allocated per replenishment period, it is observed that the LCR-model will require lesser number of actors than the LPR-model, while if the execution time of a single iteration is less than the slice size, the LPR-model will have lesser number of actors for that job.

In the case of the multi-rate data-flow model proposed in chapter 7, the model in its multi-rate form will always have a four actors. The SRDF expansion equivalent of the MRDF-model will have a single actor for modeling the initial waiting time, multiple actors for modeling the actual execution of the jobs, and a collector actor w'. We observe that the SRDF equivalent of the MRDF-model is an exponential expansion of the original model. The single-rate expansion of the MRDF-model depends on the production rule of the actor representing the initial waiting time, (see figure 7.4. In section 7.4.3, we show that the collector actor can be eliminated from in the expanded SRDF equivalent of the original model without affecting the behavior of the model. In section 7.4.3, we addressed the issue of optimizing the MRDF-model such that we can reduce the size of the equivalent SRDF expansion. We proposed that the number of tokens produce by the actor w (the actor that model the initial waiting time) is set to $\frac{t_x}{\gcd(S,t_x)}$ where t_x is the execution time of a single iteration of the modeled job and S is the size of the slice allocated per replenishment period for this job. The number of actors then required to model the execution of the job iterations is expressed as $2 \times (\frac{t_x}{\gcd(S,t_A)})$ and the total number of tokens in the expanded SRDF equivalent of the MRDF-model is given by $1 + 2 \times (\frac{t_x}{\gcd(S,t_A)})$.

8.1.2 Analysis Complexity

In sections 2.5.1 and 2.5.2, we described two techniques, namely simulation based analysis and static analysis, for analyzing the temporal behavior of data-flow modeled embedded streaming applications. The time taken by both techniques directly depends upon the number of actors and the number of paths across these actors. Each of the models proposed in this thesis have more number of actors and paths with respect to the existing state-of-the-art models [1], [2]. Intuitively, it is expected that the temporal analysis of an applications graph, obtained by representing each job in the application one of the proposed models instead of the existing state-of-the-art models, would take longer. The empirical quantification of this time, however, is not presented as part this project due to lack of time. This quantification is left for future work.

8.2 Applicability of the proposed models for other budget schedulers

It is shown in [1] that TDM arbitration belongs to the class of budget schedulers, called latency-rate servers, such that its behavior can be captured using data-flow based models called Latency-Rate models [10]. This latency-rate model has been used extensively in literature for modeling the temporal behavior of different latency-rate servers, such as [21, 2, 22].

This project focuses on the data-flow based temporal analysis of TDM arbitration to establish hard real-time guarantees for independent application running on a shared resource MPSoC. We have shown that the modeling techniques presented in this thesis are a more accurate representation of the worst-case temporal behavior of TDM arbitration as compared to the existing state-of-the-art- techniques.

We suspect that our techniques can also be extended to capture the worst-case temporal behavior for a broader class of latency-rate servers. For instance the data-flow model for the high priority job in *Priority based Budget Scheduling* (PBS) [21] is shown in Figure 8.1(a). The advantage in PBS is that the high priority job has a much smaller initial waiting time and then has a generalized rate of execution given by $\frac{t_x \cdot P}{B}$ where t_x is the execution time for a single job iteration which has an allocated budget (or slice) B per period P. The modeled behavior is claimed to be a tight conservative (over-) estimate of the worst-case temporal behavior of jobs scheduled using PBS. An accurate estimation can be obtained using, for instance, the LCR-model (Figure 8.1(b)) or the MRDF model (Figure 8.1(c)) proposed in this thesis by only changing the firing time of the actor w to $t(w) = B$, and replacing the slice allocation S with the budget allocation B in the rest of the model.

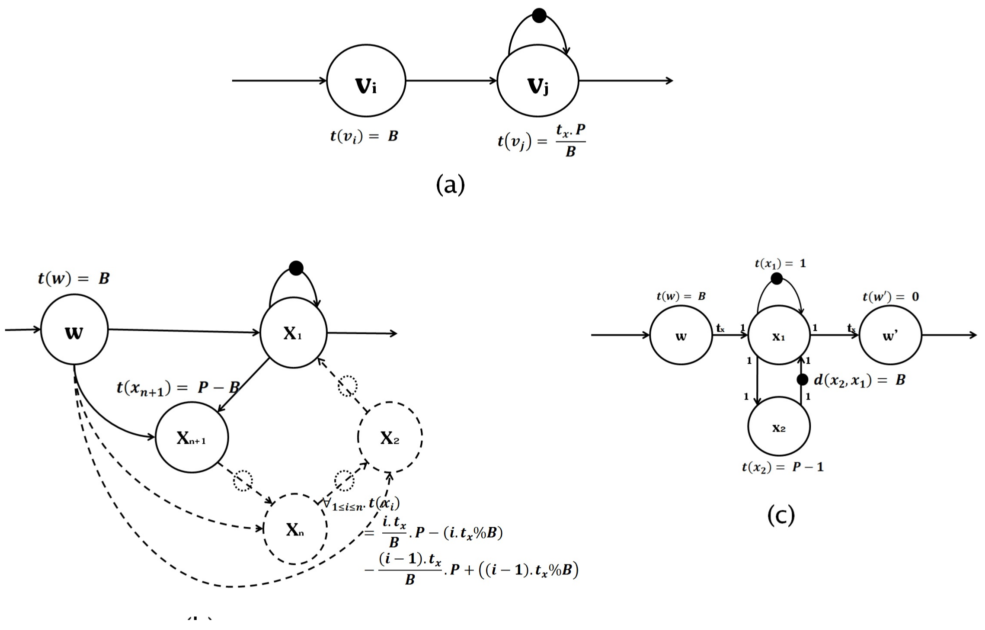

Figure 8.1: Data-flow based modeling of Priority based Budget Scheduling

Intuitively, we observe that the pessimism present in the LR-model is due to the generalization of the the rate of execution across all job iterations to establish a linear worst-case bound on the temporal behavior of resource shared jobs. Meanwhile, the models proposed in this thesis do not model this generalization such that we define a more accurate non-linear bound that coincides with cyclic pattern of the actual worst-case temporal behavior of these jobs. Further study may lead to more conclusive results and we strongly recommend that this extending the use of the proposed models for other budget schedulers should be investigated.

8.3 Using the models for TDM with Static-Ordering

In section 3.5, we described the concept of TDM arbitration with static-ordering in which a group of mutually exclusive jobs of an application, that are mapped onto the same resource, are grouped together in staic-order and given a single slice per replenishment period. *Moreira et al* [8] shows that the response times of jobs scheduled via TDM with static-ordering is much better than if they had an independent slice allocation. In *Moreira et al* [8], it is also shown how TDM with static-ordering can be modeled using the LR-model. For instance, Figure 8.2 shows the slice allocation for a group of two static-ordered jobs and its corresponding LR-model.

We suggest to model this group of static-ordered jobs as a single job that has a cyclo-

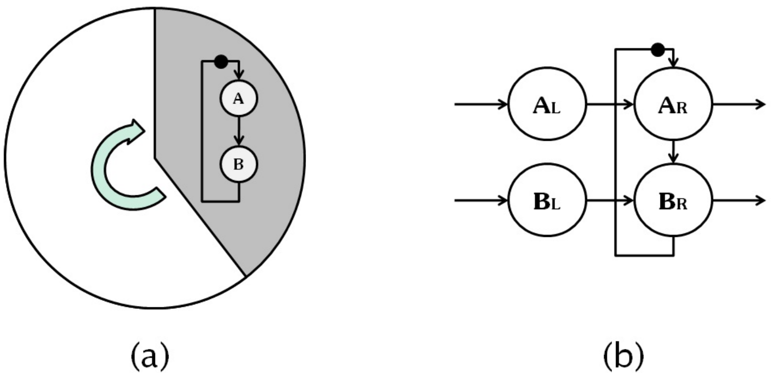

Figure 8.2: Modeling TDM with static-order using the LR-model

static regime of execution time. In other words we represent this group of jobs as a single job having a fixed order of varying execution times. We can then redefine the worst-case behavior of this job under the effect of TDM arbitration as:

$$f_n = \begin{cases} s_n + \lfloor \frac{\sum_{1\le j\le n} t_x^j}{S} \rfloor \cdot P & \text{if } \sum_{1\le j\le n} t_x^j \ \% \ S = 0 \\ s_n + \lfloor \frac{\sum_{1\le j\le n} t_x^j}{S} \rfloor \cdot P + (P-S) + (\sum_{1\le j\le n} t_x^j \ \% \ S) & \text{if } \sum_{1\le j\le n} t_x^j \ \% \ S \neq 0 \end{cases} . \tag{8.1}$$

8.3.1 Extending the MRDF-model for TDM with static-order

We suggest that the MRDF-model proposed in chapter 7 can be extended to a CSDF-model, to capture the worst-case temporal behavior of jobs under the effect of TDM with static ordering. This is shown in Figure 8.3. Further verification and validation on the correctness of the obtained CSDF-model is left for future work.

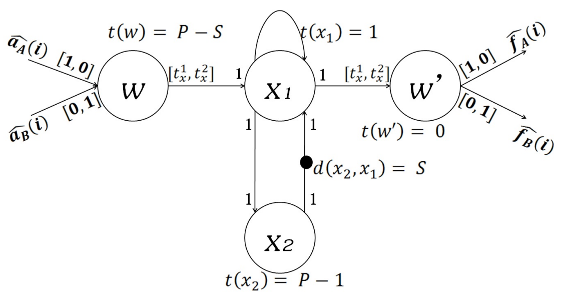

Figure 8.3: Modeling TDM with static-ordering using the proposed MRDF (CSDF) -model

8.3.2 Extending the LCR-model for TDM with static-ordering

We suspect that the LSPRI-model proposed in chapter 6 can also be extended by making the following changes in the model construction.

The firing time of the latency actor is given by $t(w) = P - S$ while the firing time $t(x_i)$ of each phase actor $x_i \in V$ defines the execution behavior of the associated job iteration, expressed as:

$$t(v) = \begin{cases} P - S & \text{for } v \in w, x_{q+1}, \\ \lfloor \frac{\sum_{1 \le j \le i} t_x^j}{S} \rfloor \cdot P + [\sum_{1 \le j \le i} t_x^j \ \% \ S] & \\ \qquad - \lfloor \frac{\sum_{1 \le j \le i-1} t_x^j}{S} \rfloor \cdot P + [\sum_{1 \le j \le i-1} t_x^j \ \% \ S] & \text{for } v = x \ \& \ 1 \le i < q, \\ \lfloor \frac{\sum_{1 \le j \le n} t_x^j}{S} \rfloor \cdot P + [i \cdot t_A \ \% \ S] & \\ \qquad - \lfloor \frac{\sum_{1 \le j \le n-1} t_x^j}{S} \rfloor \cdot P + [\sum_{1 \le j \le n-1} t_x^j \ \% \ S] - (P - S) & \text{for } v = x_q. \end{cases}, \tag{8.2}$$

where $n = \frac{lcm(S, \sum_{1 \le j \le n} t_x^j)}{\sum_{1 \le j \le n} t_x^j}$. We would like to inform here that the proposed construction is only suspected to be correct. We recommend a detailed future study to establish adequate claims on this approach.

Chapter 9

Conclusions

An increasing number of embedded radio applications for MPSoC hardware is modeled using data-flow to establish hard real-time guarantees such as minimum throughput maximum latency. The influence of resource sharing amongst multiple applications by using budget based schedulers such as TDM is usually bounded. This thesis proposes various modeling techniques that make use of data-flow graphs to capture the worst-case temporal behavior of jobs scheduled using TDM arbitration. The existing modeling techniques [6, 1], are pessimistic in capturing this worst-case temporal behavior thus providing weaker guarantees. Stronger guarantees can be established via the state-of-the-art modeling approaches by over-allocating resources to the jobs of an application. This results in under-utilization of system resources, and may cause unnecessary rejection of applications that could in fact be accommodated within the given MPSoC framework. Models such as in [2], enable more accurate estimation of the worst-case temporal behavior of TDM arbitration. The model, however, is only applicable to exceptional cases of TDM arbitration, as the model makes the assumes that the size of the slice allocated for a job is an integer multiple of the execution time of a single job iteration.

The main focus of this thesis was to use data-flow to design a conservative model that accurately capture the worst-case temporal behavior of jobs scheduled using TDM arbitration. The project goal is stated as:

"To design a data-flow based modeling techniques to capture the worst-case behavior of jobs scheduled using TDM arbitration, such that:

- it provides more accurate estimation of the worst-case behavior of a TDM scheduled job as compared to the existing state-of-the-art models.
- it is conservative i.e. it will define an upper-bound to the worst-case temporal behavior of jobs scheduled using TDM arbitration.

- it is generic i.e. it should not make unnecessary assumptions on the characteristics of TDM arbitration, such as slice sizes".

In this thesis, we presented various data-flow based models, namely the LPR-model, the LCR-model, and the MRDF-model, that provide more accurate estimation of the worst-case temporal behavior of TDM arbitration in comparison to the existing state-of-the-art [6, 8, 1, 21]. Unlike [2], we do not make assumptions, or pose restrictions, on the characteristics of TDM arbitration such as size of the slice allocated for a job. This makes our proposed models more generic and applicable for arbitrary TDM setup. We enlist the characteristics of each of the proposed models as:

1. LPR-model: The LCR-model is a single-rate data-flow model for estimating the worst-case temporal behavior of TDM arbitration. The execution of continuous iterations of a job using TDM can be shown to have a cyclic pattern over a fixed number of replenishment periods. Our approach models this patterns using data-flow by capturing the execution behavior for each replenishment period separately. We initially assume that the execution time of a single job iteration is smaller that the size of a single slice. Thereby, we relax the restriction of Staschulat's LR-model [2] which requires that the size of the slice should be only an integer multiple of the execution time of a single job iteration. For this setup of TDM arbitration, we demonstrate that the model provides a conservative estimation that is tighter than the existing state-of-the-art models i.e. the LR-model [1].

   We then show that we can extend this model to capture the effect of TDM arbitration for jobs with arbitrary execution times. We do this by modeling TDM arbitration using a virtual slice and period setup such that a single slice can accommodate the execution of a single job iteration. However, when we extend this model to larger execution times, we demonstrate that the modeled estimations are comparable to the LR-modeled estimations but not always better. We may conclude that in such cases the proposed approach is not optimal.

2. LCR-model: The LCR-model is a new approach to model the cyclic patter of execution of a job under the effect of TDM arbitration. The proposed LCR-model overcomes the disadvantages in the LPR-model such that it does not make any assumption on the execution time of a single job iteration. The LCR-models the execution rate of each individual iteration instead of the modeling the execution rate per phase slice of the cyclic pattern.

   We demonstrated that this model provides an accurate estimation of the worst-case temporal behavior of an arbitrary TDM setup. We also conclude that the LCR-model is better than the existing state-of-the-art models and the proposed LPR-model. We also demonstrated that under the assumptions on TDM arbitration made by Staschulat's LR-model [2], the LCR-model obtained for the same TDM setup can be further optimized using data-flow optimization techniques [11] to obtaing a reduced model, that directly coincides with Staschulat's LR-model. The LCR-model is, therefore, a generic model that captures he worst-case temporal behavior of job iterations scheduled using TDM arbitration.

3. MRDF-model: The MRDF-model uses multi-rate data-flow to model the worst-case temporal behavior of TDM scheduled jobs. Instead of modeling a job iteration as a single token, we split it into multiple tokens according to the total execution time of a single job iteration. Each token represents to a fixed amount of execution time of an iteration and is modeled such that it consumes the equivalent amount of time from the alloted slice. The availability of slice time for execution is replenished after every period. On the output side, all tokens of a single iteration are consumed in a single firing to produce an output token.

   We argued the correctness and the accuracy of the model through an illustrative example. We addressed issues, such as expanding the multi-rate model into an equivalent single-rate data-flow model and also on various optimizations possible on the models. We also compared the multi-rate model to the other models proposed in this thesis and the existing state-of-the-art models. We argue that the proposed MRDF model is more accurate the the LR-model [1]. We also claim that Staschulat's LR-model [2] can be perceived as a special instance of the proposed MRDF model.

To show that the models are conservative we first formalized the worst-case temporal behavior of TDM arbitration. Although this formalization is rather intuitive and straight forward, such formalization of the worst-case behavior of TDM arbitration is not presented in literature to the best of our knowledge. We must then demonstrate that a model defines an upper bound to this worst-case temporal behavior and is, thus, conservative. The existing state-of-the-art models do not demonstrate conservativity via this approach, but rather they sketch the worst-case temporal behavior of TDM arbitration and argue that these models are conservative. This thesis provides a formal proof of their conservative properties in Appendix A. Similarly we also present the proof for conservativity of the models proposed in this thesis in Appendix B and also demonstrate that the proposed models are more accurate than the existing state-of-the-art.

We addressed various issues regarding the the proposed model. We discussed that the improved accuracy in the estimation of the worst-case temporal behavior of TDM arbitration via our models comes at the cost of increased complexity of the data-flow model both in terms of size of the generated data-flow model and the time required to analyze these models. Due to lack of time, we do not quantify this increase in the complexity, and leave it for future work. In case of the MRDF-model, we also presented the some optimization techniques that reduce the complexity of the model and its SRDF expansion. We also addressed the issue of extending the proposed modeling techniques for a broader category of budget schedulers and illustrated the use of our models for the Priority based Budget Schedulers [21]. Lastly we also address the issue of extending the proposed models to capture the effect of TDM arbitration with static-ordering.

References

[1] M. H. Wiggers, M. J. G. Bekooij, and G. J. M. Smit, "Modelling run-time arbitration by latency-rate servers in dataflow graphs," in *Proceedingsof the 10th international workshop on Software & compilers for embedded systems*, SCOPES '07, pp. 11–22, 2007.

[2] J. Staschulat and M. Bekooij, "Dataflow models for shared memory access latency analysis," in *Proceedings of the seventh ACM international conference on Embedded software*, EMSOFT '09, pp. 275–284, 2009.

[3] K. van Berkel, A. Burchard, D. van Kampen, P. Kourzanov, A. Piipponen, R. Raiskila, S. Slotte, M. van Splunter, and T. Zetterman, "A Multi-Radio SDR Technology Demonstrator," in *SRD Forum Technical Conference*, 2009.

[4] A. Ahtinen, K. van Berkel, D. van Kampen, A. Piipponen, and T. Zetterman, "Multi-radio Scheduling and Resource Sharing on a Software Defined Radio Computing Platform," in *SRD Forum Technical Conference*, 2008.

[5] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, "Vector processing as an enabler for software-defined radio in handheld devices," *EURASIP J. Appl. Signal Process.*, vol. 2005, pp. 2613–2625, January 2005.

[6] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, B. Mesman, J. D. Mol, S. Stuijk, V. Gheorghita, and J. van Meerbergen, "Dataflow analysis for real-time embedded multiprocessor system design," *Dynamic and Robust Streaming in and between Connected Consumer Electronic Devices*, pp. 81–108, 2005.

[7] O. Moreira, J.-D. Mol, M. Bekooij, and J. v. Meerbergen, "Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix," in *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pp. 332–341, 2005.

[8] O. Moreira, F. Valente, and M. Bekooij, "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Proceedings of the 7th ACM*

& IEEE international conference on Embedded software, EMSOFT ’07, pp. 57–66, 2007.

[9] O. Moreira, T. Basten, M. Geilen, and E. Stuijk, “Buffer sizing for rate-optimal single-rate dataflow scheduling revisited,” *IEEE Transactions on Computers*, 2010.

[10] D. Stiliadis and A. Varma, “Latency-rate servers: a general model for analysis of traffic scheduling algorithms,” *IEEE/ACM Trans. Netw.*, vol. 6, pp. 611–624, October 1998.

[11] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization.* New York, NY, USA: Marcel Dekker, Inc., 1st ed., 2000.

[12] E. Lee and D. Messerschmitt, “Synchronous data flow,” *Proceedings of the IEEE*, vol. 75, pp. 1235 – 1245, sept. 1987.

[13] P. Butkovic, *Max-Linear Systems: Theory and Algorithms.* Springer Monographs in Mathematics, Springer, 2010.

[14] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat, *Synchronization and Linearity.* New York, NY, USA: John Wiley & Sons, 1992.

[15] G. Bilsen et al, “Cyclo-static dataflow,” *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, 1996.

[16] P. Wauters, M. Engels, R. Lauwereins, and J. Peperstraete, “Cyclo-dynamic dataflow,” in *Parallel and Distributed Processing, 1996. PDP ’96. Proceedings of the Fourth Euromicro Workshop on*, pp. 319 –326, jan 1996.

[17] T. Parks, J. Pino, and E. Lee, “A comparison of synchronous and cycle-static dataflow,” in *Signals, Systems and Computers, 1995. 1995 Conference Record of the Twenty-Ninth Asilomar Conference on*, vol. 1, pp. 204 –210 vol.1, oct-1 nov 1995.

[18] M. Geilen, “Reduction techniques for synchronous dataflow graphs,” in *Proceedings of the 46th Annual Design Automation Conference*, DAC ’09, pp. 911–916, 2009.

[19] S. Stuijk, M. Geilen, and T. Basten, “Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs,” in *Proceedings of the 43rd annual Design Automation Conference*, DAC ’06, pp. 899–904, 2006.

[20] M. H. Wiggers, M. J. G. Bekooij, M. C. W. Geilen, and T. Basten, “Simultaneous budget and buffer size computation for throughput-constrained task graphs,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE ’10, pp. 1669–1672, 2010.

[21] M. Steine, M. Bekooij, and M. Wiggers, “A priority-based budget scheduler with conservative dataflow model,” in *Proceedings of the 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, DSD ’09, pp. 37–44, 2009.

[22] M. H. Wiggers, M. J. Bekooij, and G. J. Smit, “Monotonicity and run-time scheduling,” in *Proceedings of the seventh ACM international conference on Embedded software*, EMSOFT ’09, pp. 177–186, 2009.

[23] S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal, “Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs,” in *Proceedings of the 44th annual Design Automation Conference*, DAC ’07, pp. 777–782, 2007.

[24] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij, “Throughput analysis of synchronous data flow graphs,” in *Proceedings of the Sixth International Conference on Application of Concurrency to System Design*, pp. 25–36, 2006.

[25] O. M. Moreira and M. J. G. Bekooij, “Self-timed scheduling analysis for real-time applications,” *EURASIP Journal on Advances in Signal Processing*, 2007.

[26] R. Reiter, “Scheduling parallel computations,” *J. ACM*, vol. 15, pp. 590–599, October 1968.

[27] A. Dasdan, “Experimental analysis of the fastest optimum cycle ratio and mean algorithms,” *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, pp. 385–418, October 2004.

Appendix A

Conservativity of the state-of-the-art models

A.1 Basic Definitions

Let us first familiarize ourselves with some basic terms and definitions that we will use to formulate our proofs. We assume that each iteration is invoked by arrival of a new input data container, and finishes on producing an output data container. These basic definitions allow us to formalize the modeled behavior of worst-case TDM arbitration.

- P replenishment period of the TDM scheduler
- S time slice alloted to a job
- t_A execution time of each iteration of an arbitrary job A
- $a(i)$ the actual arrival time of the i-th job iteration marked by arrival of the i-th input container
- $f(i)$ the actual finish time of the i-th job iteration marked by production of the i-th output container
- $\hat{a}(i)$ the modeled arrival time of the token representing i-th input container
- $\hat{f}(i)$ the modeled finish time of token representing the i-th output container

Let us also re-state the worst-case temporal behavior of TDM arbitration formalized in section 3.2. We formalize the temporal behavior of TDM arbitration by expressing the finish time of an iteration in relation to the the finish times or start times of the previous iterations. A sequence of consecutive job iterations is where every iteration, except the first iteration in the sequence, arrives before its previous iteration finishes executing. The execution of each iteration will have to wait until all previous iterations have finished executing.

Consider that for a sequence of n consecutive job iterations, s_n is the arrival time of the first iteration in the sequence and f_n is the finish time of the last iteration in the sequence. If each job iteration takes t_A time to execute, $n \cdot t_A$ is the total execution time required. This amount of time can be expressed in terms of the slice size as $\lfloor \frac{n \cdot t_A}{S} \rfloor \cdot S + (n \cdot t_A \ \% \ S)$ i.e. we will require $\lfloor \frac{n \cdot t_A}{S} \rfloor$ complete slices and and additional $(n \cdot t_A \ \% \ S)$ time for the remaining execution. We require $\lfloor \frac{n \cdot t_A}{S} \rfloor$ periods for each slice that is consumed completely, and if $n \cdot t_A \ \% \ S \neq 0$ then an additional waiting time of $P - S$ is required before the job is allotted a slice to complete the remaining $(n \cdot t_A \ \% \ S)$ of the execution time. We define the execution of a sequence of n consecutive iterations as.

$$f_n = \begin{cases} s_n + \lfloor \frac{n \cdot t_A}{S} \rfloor \cdot P & \text{if } n \cdot t_A \ \% \ S = 0 \\ s_n + \lfloor \frac{n \cdot t_A}{S} \rfloor \cdot P + (P - S) + (n \cdot t_A \ \% \ S) & \text{if } n \cdot t_A \ \% \ S \neq 0 \end{cases} \tag{A.1}$$

Definition 2. *A data-flow model is* conservative *if, whenever the modeled arrival times $\hat{a}$ are an upper bound to the actual arrival times a, the modeled finish times $\hat{f}$ are an upper bound to the actual finish times f. Formally, this means that:*

$$\forall_{i \geq 0} \ a(i) \leq \hat{a}(i) \ \Rightarrow \ \forall_{i \geq 0} \ f(i) \leq \hat{f}(i) \tag{A.2}$$

We will now show that the each of the state-of-the-art models [6, 1, 2]. We show that the finish times modeled by these models of each job iteration is an upper bound to the actual finish times of job iterations as per Equation A.1.

A.2 Conservativity of the Singe-actor response model

Bekooij et al [6] propose a single-actor response model for modeling the worst case temporal behavior of TDM arbitration as shown in Figure A.1. The model bound for this worst-case temporal behavior is expressed as:

$$\begin{aligned} \hat{f}(i) &= \max(\hat{a}(i), \hat{f}(i-1)) + t(v_x) \\ &= \max(\hat{a}(i), \hat{f}(i-1)) + (P - S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S] \end{aligned} \tag{A.3}$$

To show that the single-actor response model defines an upper bound to the worst-case temporal behavior of TDM arbitration we enumerate the following cases:

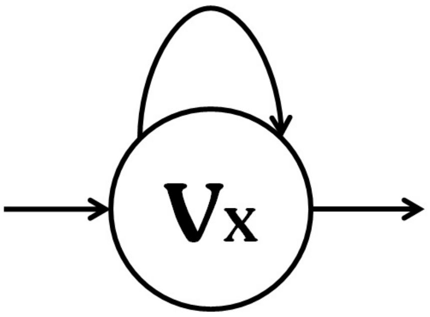

Figure A.1: Representing a job using the single-actor response-model

1. Consider a sequence of consecutive job iteration such that the $(i-k)$-th iteration is the first iteration of the sequence, i.e. $s_n = a(i-k)$. The length of the sequence is $n = k+1$. Consider, for now, that the length $(k+1) < \frac{lcm(S,t_x)}{t_x}$ such that $(k+1) \cdot t_x \ \% \ S \neq 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = a(i-k) + (P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + (k+1) \cdot t_x \ \% \ S \tag{A.4}$$

The single-actor response model gives the constraint for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1) + (P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S]$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) = \hat{a}(i-k) + (k+1) \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S] \tag{A.5}$$

To show that $f(i) \leq \hat{f}(i)$ we show that the predicate

$$(P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + [(k+1) \cdot t_x \ \% \ S] \leq (k+1) \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S], \tag{A.6}$$

is satisfied. To prove this, we split the above inequality into three parts.

- Trivially we can show that $(P-S) \leq (k+1) \cdot (P-S)$.
- For $(k+1) \leq \frac{lcm(S,t_x)}{t_x}$, we know that $\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor = (k+1) \cdot \lfloor \frac{t_x}{S} \rfloor$. Multiplying this inequality by P we get $\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P = (k+1) \cdot \lfloor \frac{t_x}{S} \rfloor \cdot P$

- Similarly, the inequality $(k+1) \cdot [t_x \ \% \ S] \leq (t_x \ \% \ S) \cdot (k+1)$ can be reduced to $(k+1) \cdot \lfloor \frac{t_x}{S} \rfloor \leq \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor$. For $(k+1) \leq \frac{lcm(S,t_x)}{t_x}$, this is trivially satisfied.

2. Consider a sequence of consecutive job iteration such that the $(i-k)$-th iteration is the first iteration of the sequence, i.e. $s_n = a(i-k)$. The length of the sequence is $n = k+1$. Now consider that the length $(k+1) = \frac{lcm(S,t_x)}{t_x}$ such that $(k+1) \cdot t_x \ \% \ S = 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = a(i-k) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \tag{A.7}$$

The single-actor response model gives the constraint for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1) + (P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S]$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) = \hat{a}(i-k) + (k+1) \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S] \tag{A.8}$$

To show that $f(i) \leq \hat{f}(i)$ we show that the predicate

$$\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \leq (k+1) \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S], \tag{A.9}$$

is satisfied. We know that $a(i-k) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P < a(i-k) + (P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + (k+1) \cdot t_x \ \% \ S$. As we have already shown that $(P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + [(k+1) \cdot t_x \ \% \ S] \leq (k+1) \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S]$, it is trivial to show that $\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \leq (k+1) \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S]$.

3. Consider that the $(i-k)$-th iteration is the first iteration of the consecutive sequence such that the length of the sequence $n = k+1 > \frac{lcm(S,t_x)}{t_x}$. In such cases we can consider the sequence to start from an iteration such that $s_n = f(i-n')$, where $n' = \frac{lcm(S,t_x)}{t_x}$. The new length of the sequence is n' such that $n' \cdot t_x \ \% \ S = 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = f(i-n') + \lfloor \frac{n' \cdot t_x}{S} \rfloor \cdot P \tag{A.10}$$

The single-actor response model gives the constraint for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1) + (P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + [t_x \ \% \ S]$. Applying this constraint n' time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) = \hat{f}(i-n') + n' \cdot [(P-S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S] \tag{A.11}$$

We have already shown that $\lfloor \frac{n' \cdot t_x}{S} \rfloor \cdot P \leq n' \cdot [(P - S) + \lfloor \frac{t_x}{S} \rfloor \cdot P + t_x \ \% \ S]$ and, inductively, we say that $f(i - n) \leq \hat{f}(i - n')$. Adding these two inequalities we can conclude that $f(i) \leq \hat{f}(i)$.

The above enumeration covers all cases of the worst-case temporal behavior of TDM arbitration such that we conclude for all cases that $f(i) \leq \hat{f}(i)$. Hence the single-actor response model is conservative.

A.3 Conservativity of the LR-model

Wiggers et al [1] propose a Latency-Rate model for modeling the worst case temporal behavior of TDM arbitration as shown in Figure A.2. The model bound for this worst-case temporal behavior is expressed as:

$$\begin{aligned} \hat{f}(i) &= \max(\hat{a}(i) + t(x_L) + t(x_R), \hat{f}(i-1) + t(x_R)) \\ &= \max(\hat{a}(i) + (P - S) + \frac{P \cdot t_x}{S}, \hat{f}(i-1) + \frac{P \cdot t_x}{S}) \end{aligned} \tag{A.12}$$

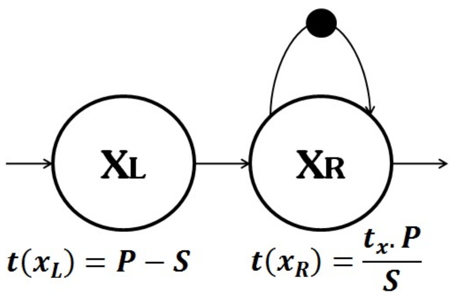

Figure A.2: Representing a job using the LR-model

To show that the LR-model defines an upper bound to the worst-case temporal behavior of TDM arbitration we enumerate the following cases:

1. Consider a sequence of consecutive job iteration such that the $(i - k)$-th iteration is the first iteration of the sequence, i.e. $s_n = a(i - k)$. The length of the sequence is $n = k + 1$. Consider, for now, that the length $(k + 1) < \frac{lcm(S, t_x)}{t_x}$ such that $(k + 1) \cdot t_x \ \% \ S \neq 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = a(i - k) + (P - S) + \lfloor \frac{(k + 1) \cdot t_x}{S} \rfloor \cdot P + (k + 1) \cdot t_x \ \% \ S \tag{A.13}$$

The LR-model defines the constraints for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1) + \frac{t_x \cdot P}{S}$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) \geq \hat{a}(i-k) + (P-S) + (k+1) \cdot \frac{t_x \cdot P}{S} \tag{A.14}$$

To show that $f(i) \leq \hat{f}(i)$ we show that the predicate

$$(P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + [(k+1) \cdot t_x \ \% \ S] \leq (P-S) + (k+1) \cdot \frac{t_x \cdot P}{S}, \tag{A.15}$$

is satisfied. To prove this, we reduce this predicate as:

$$\begin{aligned} \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + [(k+1) \cdot t_x \ \% \ S] &\leq (k+1) \cdot \frac{t_x \cdot P}{S} \\ \equiv [(k+1) \cdot t_x \ \% \ S] &\leq (k+1) \cdot \frac{t_x \cdot P}{S} - \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P && \text{(A.16)} \\ & && \text{(A.17)} \end{aligned}$$

Multiplying both sides by S we get:

$$\begin{aligned} S \cdot [(k+1) \cdot t_x \ \% \ S] &\leq (k+1) \cdot_x \cdot P - \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot S \cdot P \\ \equiv S \cdot [(k+1) \cdot t_x \ \% \ S] &\leq ((k+1) \cdot_x \cdot - \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot S) \cdot P \\ \equiv S \cdot [(k+1) \cdot t_x \ \% \ S] &\leq [(k+1) \cdot t_x \ \% \ S] \cdot P && \text{(A.18)} \\ & && \text{(A.19)} \end{aligned}$$

As $S \leq P$ the above inequality is satisfied.

2. Consider a sequence of consecutive job iteration such that the $(i-k)$-th iteration is the first iteration of the sequence, i.e. $s_n = a(i-k)$. The length of the sequence is $n = k+1$. Now consider that the length $(k+1) = \frac{lcm(S,t_x)}{t_x}$ such that $(k+1) \cdot t_x \ \% \ S = 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = a(i-k) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \tag{A.20}$$

The LR-model gives the constraint for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1)\frac{P \cdot t_x}{S}$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) = \hat{a}(i-k) + (P-S) + (k+1) \cdot \frac{P \cdot t_x}{S} \tag{A.21}$$

To show that $f(i) \leq \hat{f}(i)$ we show that the predicate

$$\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \leq (P-S) + (k+1) \cdot \frac{P \cdot t_x}{S}, \tag{A.22}$$

is satisfied. We know that $a(i-k) + \lfloor \frac{(k+1)\cdot t_x}{S} \rfloor \cdot P < a(i-k) + (P-S) + \lfloor \frac{(k+1)\cdot t_x}{S} \rfloor \cdot P + (k+1) \cdot t_x \ \% \ S$. As we have already shown that $(P-S) + \lfloor \frac{(k+1)\cdot t_x}{S} \rfloor \cdot P + [(k+1) \cdot t_x \ \% \ S] \leq (P-S) + (k+1) \cdot \frac{P \cdot t_x}{S}$, it is trivial to show that $\lfloor \frac{(k+1)\cdot t_x}{S} \rfloor \cdot P \leq (P-S) + (k+1) \cdot \frac{P \cdot t_x}{S}$.

3. Consider that the $(i-k)$-th iteration is the first iteration of the consecutive sequence such that the length of the sequence $n = k+1 > \frac{lcm(S,t_x)}{t_x}$. In such cases we can consider the sequence to start from an iteration such that $s_n = f(i-n')$, where $n' = \frac{lcm(S,t_x)}{t_x}$. The new length of the sequence is n' such that $n' \cdot t_x \ \% \ S = 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = f(i-n') + \lfloor \frac{n' \cdot t_x}{S} \rfloor \cdot P \tag{A.23}$$

The LR-model gives the constraint for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1)\frac{P \cdot t_x}{S}$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) = \hat{f}(i-n') + n' \cdot \frac{P \cdot t_x}{S} \tag{A.24}$$

As we know that, inductively, $f(i-n') \leq \hat{f}(i-n')$ - we can show that

$$\begin{aligned} f(i-n') + n' \cdot \frac{P \cdot t_x}{S} &\leq \hat{f}(i-n') + n' \cdot \frac{P \cdot t_x}{S} \\ &\equiv f(i) \leq \hat{f}(i). \end{aligned} \tag{A.25}$$

The above enumeration covers all cases of the worst-case temporal behavior of TDM arbitration such that we conclude for all cases that $f(i) \leq \hat{f}(i)$. Hence the LR-model is conservative.

A.4 Conservativity of Staschulat's LR-model

Staschulat et al [2] propose a model for memory arbiters based on the latency-rate model with some additional considerations. Staschulat's LR-model considers that the size of the slice assigned for a job is an integer multiple of the execution time of a single job iteration, i.e. $S \;\%\; t_x = 0$ and $\frac{S}{t_x} = m$. Figure A.3 The bound defined by Staschulat's LR-model for the worst-case temporal behavior of TDM arbitration is expressed as:

$$\hat{f}(i) = \max \begin{cases} \hat{a}(i) + (P - S) + t_x \\ \hat{f}(i-1) + t_x \\ \hat{f}(i-n) + P \end{cases} \tag{A.26}$$

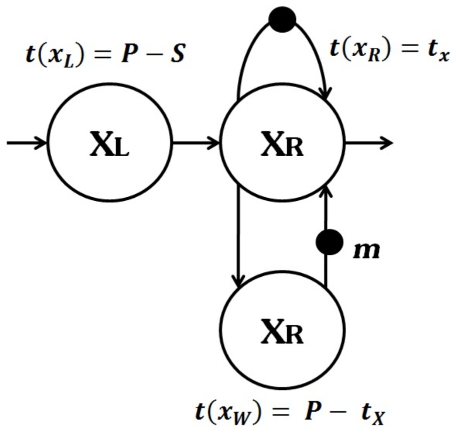

Figure A.3: Representing a job using Staschulat's LR-model

To show that Staschulat's LR-model defines an upper bound to the worst-case temporal behavior of TDM arbitration we enumerate the following cases:

1. Consider a sequence of consecutive job iteration such that the $(i-k)$-th iteration is the first iteration of the sequence, i.e. $s_n = a(i-k)$. The length of the sequence is $n = k+1$. Consider, for now, that the length $(k+1) < \frac{lcm(S,t_x)}{t_x}$ such that $(k+1) \cdot t_x \;\%\; S \neq 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = a(i-k) + (P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + (k+1) \cdot t_x \;\%\; S \tag{A.27}$$

Staschulat's LR-model defines the constraints for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1) + t_x$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) \geq \hat{a}(i-k) + (P-S) + (k+1) \cdot t_x \tag{A.28}$$

To show that $f(i) \leq \hat{f}(i)$ we show that the predicate

$$(P-S) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P + [(k+1) \cdot t_x \ \% \ S] \leq (P-S) + (k+1) \cdot t_x \tag{A.29}$$

is satisfied. Let us look at only the lefi-hand side of this inequality. The consideration for Staschulat's LR-model is that $S \ \% \ t_x = 0$. Hence, we say that $lcm(S, t_x) = S$. As $k+1 < \frac{lcm(S,t_x)}{t_x}$, we say that $k < \frac{S}{t_x}$, i.e. $\frac{(k+1) \cdot t_x}{S} < 1$. We can now show that $\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor = 0$ and $(k+1) \cdot t_x \ \% \ S = (k+1) \cdot t_x$. The left-hand side is now reduced to $(P-S) + (k+1) \cdot t_x$, which is the same as the right hand side. Hence the above predicate is satisfied.

2. Consider a sequence of consecutive job iteration such that the $(i-k)$-th iteration is the first iteration of the sequence, i.e. $s_n = a(i-k)$. The length of the sequence is $n = k+1$. Now consider that the length $(k+1) = \frac{lcm(S,t_x)}{t_x}$ such that $(k+1) \cdot t_x \ \% \ S = 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = a(i-k) + \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \tag{A.30}$$

Staschulat's LR-model defines the constraints for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i-1) + t_x$. Applying this constraint $(k+1)$ time we derive the modeled finish time of the i-th job iteration as:

$$\hat{f}(i) \geq \hat{a}(i-k) + (P-S) + (k+1) \cdot t_x \tag{A.31}$$

To show that $f(i) \leq \hat{f}(i)$ we show that the predicate

$$\lfloor \frac{(k+1) \cdot t_x}{S} \rfloor \cdot P \leq (P-S) + (k+1) \cdot t_x, \tag{A.32}$$

is satisfied. Let us first consider the right-hand side of the above inequality. We know that $k + 1 = \frac{lcm(S,t_x)}{t_x} = \frac{S}{t+x}$. Therefore we can say that $(k + 1) \cdot t_x = S$ or $P - S + (k + 1) \cdot t_x = P$. Similarly, the left-hand side of the inequality can be expressed as $\frac{lcm(S,t_x) \cdot t_x}{t_x \cdot S} \cdot P = P$. As both the left-had side and right-hand side of the inequality are same, we can conclude that the above predicate is satisfied.

3. Consider that the $(i-k)$-th iteration is the first iteration of the consecutive sequence such that the length of the sequence $n = k + 1 > \frac{lcm(S,t_x)}{t_x}$. In such cases we can consider the sequence to start from an iteration such that $s_n = f(i - n')$, where $n' = \frac{lcm(S,t_x)}{t_x} = \frac{S}{t_x} = m$. The new length of the sequence is n' such that $n' \cdot t_x \ \% \ S = 0$. The finish time of the sequence f_n is the same as the finish time of the i-th job iteration and expressed as:

$$f(i) = f(i - n') + \lfloor \frac{n' \cdot t_x}{S} \rfloor \cdot P$$

   For $n' = \frac{lcm(S,t_x)}{t_x} = \frac{S}{t_x}$ we can say that $\lfloor \frac{n' \cdot t_x}{S} \rfloor \cdot P = P$. The above expression now reduces to:

$$f(i) = f(i - n') + P \tag{A.33}$$

   Staschulat's LR-model defines the constraints for consecutive job iterations expressed as $\hat{f}(i) \geq \hat{f}(i - m) + P$ where $m = \frac{S}{t_x} = n'$. As we know that , inductively, $f(i-n') \leq \hat{f}(i-m)$, where $n' = m$ - we can show that $f(i-n')+P \leq \hat{f}(i-m)+P$, i.e. $f(i) \leq \hat{f}(i)$.

The above enumeration covers all cases of the worst-case temporal behavior of TDM arbitration such that we conclude for all cases that $f(i) \leq \hat{f}(i)$. Hence Staschulat's LR-model is conservative.

Appendix B

Analysis of the LPR-model

This chapter provides a detailed proof about the various issues of the LPR-model. We first provide a proof of the conservativity of the model. We present the proof that we can cosnservatively model arbitratry execution times and slice sizes using the LPR-model. The behavior of the LPR-model in the case when the execution time of a single job iteration is greater than the slice size is not necessarily better than the LR-model. We provide a detailed proof of this anomalous behavior.

B.1 Basic Definitions

We will adhere to the basic definitions stated in Appendix A.1.

B.2 Conservativity of the LPR-model

In this section we show that the LPR-model is conservative in describing the worst-case temporal behavior of a job scheduled using TDM arbitration. The LPR-modeled bound for the worst-case temporal behavior of TDM arbitration is expressed as:

$$\hat{f}(i) = \max \begin{cases} \max_{1 \leq j \leq q}\{\hat{a}(i - \lfloor \frac{(j-1)\cdot S}{t_x} \rfloor) + (P - S) + \\ \qquad\qquad (j-1)\cdot P - [(j-1)\cdot S \ \% \ t_x] + t_x\} \\ \hat{f}(i-m) + q \cdot P, \\ \hat{f}(i-1) + t_x, \end{cases} \tag{B.1}$$

We enumerate all the cases of the worst-case temporal behavior of TDM arbitration (refer equation A.1) and show that the model defined

1. For execution of a task iteration in isolation $[a(i) > f(i-1)]$, since $a(i) \leq \hat{a}(i)$, we can bound the execution behavior such that
$$a(i) + t_x + (P - S) \leq \hat{a}(i) + t_x + (P - S) \tag{B.2}$$

2. Equation 5.19 defines the execution behavior of consecutive sequences of task iterations where $k \geq \frac{lcm(S,t_x)}{t_x}$. We can replace $n = \frac{lcm(S,t_x)}{t_x}$ in the expression $\frac{n \cdot t_x}{S} \cdot P$ to reduce it to $q \cdot P$ where $q = \frac{lcm(S,t_x)}{S}$. We can therefore bound Equation 5.19 such that:
$$f(n-1) + \frac{n \cdot t_x}{S} \cdot P \leq \hat{f}(n-1) + q \cdot P \tag{B.3}$$

3. By, construction, the total number of tokens within the cycle of the $q+1$ rate nodes is equal to $\frac{q \cdot S}{t_x}$ i.e. $\frac{lcm(S,t_x)}{t_x}$ tokens. Therefore for all k such that $1 \leq k < \frac{lcm(S,t_x)}{t_x}$, there is an associated token in the dataflow. To show that our model bounds execution behavior of consecutive sequences of task iterations given by Equation 5.14 where $k < \frac{lcm(S,t_x)}{t_x}$. we consider two situations:

   - Firstly, we consider the situation where the token associated with i-th iteration of the task is the leading token on the edge $E(R_{j+1}, R_j)$ i.e. it is the $\lfloor \frac{(j-1) \cdot S}{t_x} \rfloor$-th token in the entire cycle from node R_1. By construction, we can associate the arrival of $(i-k)$-th iteration of task execution with $(i - \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor)$-th token [i.e. $a(i-k) \leq \hat{a}(i - \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor)$]
$$k = \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor \tag{B.4}$$
$$\equiv k \leq \frac{(j-1) \cdot S}{t_x} < k+1 \tag{B.5}$$
$$\equiv \frac{k \cdot t_x}{S} \leq (j-1) < \frac{(k+1) \cdot t_x}{S} \tag{B.6}$$

     For $k < \frac{lcm(S,t_x)}{t_x} \leq S \cdot t_x$ and $t_x \leq S$, we can state that $kt < S$, and conclude that
$$\lfloor \frac{k \cdot t_x}{S} \rfloor < \frac{k \cdot t_x}{S} \leq (j-1) \tag{B.7}$$
     Also, since $\frac{k \cdot t_x}{S} \leq (j-1) < \frac{(k+1) \cdot t_x}{S}$, we can say that:
$$\lfloor \frac{k \cdot t_x}{S} \rfloor - \lfloor \frac{(k+1) \cdot t_x}{S} \rfloor = 1 \tag{B.8}$$

From Equations B.6 and B.8 we can deduce the following expressions:

$$\lfloor\frac{(k+1)\cdot t_x}{S}\rfloor = j-1 \tag{B.9}$$

Multiplying both sides by P, we get

$$\equiv \lfloor\frac{(k+1)\cdot t_x}{S}\rfloor\cdot P = (j-1)\cdot P \tag{B.10}$$

Similarly, we can also multiply both sides of Equation B.9 by S to get:

$$\lfloor\frac{(k+1)\cdot t_x}{S}\rfloor\cdot S = (j-1)\cdot S$$

$$\equiv (k+1)\cdot t_x - \lfloor\frac{(k+1)\cdot t_x}{S}\rfloor\cdot S = (k+1)\cdot t_x - (j-1)\cdot S$$

$$\equiv [(k+1)\cdot t_x \ \%\ S] = (k+1)\cdot t_x + (j-1)\cdot S$$

Subtracting both sides from t_x we get:

$$\equiv t_x - [(k+1)\cdot t_x \ \%\ S] = t_x - (k+1)\cdot t_x + (j-1)\cdot S \tag{B.11}$$

$$\equiv t_x - [(k+1)\cdot t_x \ \%\ S] = (j-1)\cdot S - k\cdot t_x \tag{B.12}$$

On a parallel line, using Equation B.5 we can that

$$(j-1)\cdot S - k\cdot t_x = (j-1)\cdot S - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor\cdot t_x \tag{B.13}$$

$$\equiv (j-1)\cdot S - k\cdot t_x = [(j-1)\cdot S \ \%\ t_x] \tag{B.14}$$

We can conclude from Equations B.12 and B.14 that

$$t_x - [(k+1)\cdot t_x \ \%\ S] = [(j-1)\cdot S \ \%\ t_x] \tag{B.15}$$

Therefore,

$$t_x - [(j-1)\cdot S \ \%\ t_x] = [(k+1)\cdot t_x \ \%\ S] \tag{B.16}$$

By adding Equations B.10 and B.16 we see that $(j-1)\cdot P - [(j-1)\cdot S \ \%\ t_x] + t_x = (k+1)\cdot P + [(k+1)\cdot t_x \ \%\ S]$. Since $a(i-k) \leq \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor)$ we finaly deduce that

$$a(i-k) + \lfloor\frac{(k+1)\cdot t_x}{S}\rfloor\cdot P + ((k\cdot t_x + 1)\ \%\ S) + (P-S) \leq$$
$$\hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + (P-S) + (j-1)P - [(j-1)S \ \%\ t_x] + t_x$$

- We now consider the case when the token associated with the i-th iteration is not the leading token. This situation in the dataflow model can be expressed mathematically as

$$\lfloor \frac{(j) \cdot S}{t_x} \rfloor) > k > \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor) \tag{B.17}$$

Let u be such that $u = k - \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor)$, that is the token associated with the i-th iteration is the u-th token on the edge $E(R_{j+1}, R_j)$. Due the dataflow constraint that $f(i) \geq f(i-1) + t_x$ posed by the self loop at node $R-1$, the constraint on the finishing time of the $(i-u)$-th iteration can be expressed as:

$$\hat{f}(i) = \hat{f}(i-u) + u \cdot t_x \tag{B.18}$$

Also the finishing time of the $(i-u)$-th token is the leading token on the same edge, its finishing time is given by

$$\hat{f}(i-u) = \hat{a}(i - \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor) + t_x + (P-S) + (j-1)P - [(j-1)S \% t_x] \tag{B.19}$$

Using Equation B.19 in Equation B.18, we get

$$\hat{f}(i) = \hat{a}(i - \lfloor \frac{(j-1) \cdot S}{t_x} \rfloor) + (u+1)t_x + (P-S) + (j-1)P - [(j-1)S \% t_x] \tag{B.20}$$

There are $(k-u+1)$ task iteration from start of iteration $(i-k)$ to end of iteration $(i-u)$ such that its execution behavior is expressed as:

$$f(i-u) = a(i-k) + \lfloor \frac{(k-u+1) \cdot t_x}{S} \rfloor \cdot P + [(k-u+1) \cdot t_x \% S] + (P-S) \tag{B.21}$$

We can prove according to the previous case that $f(i-u) \leq \hat{f}(i-u)$ we can say that

$$f(i-u) + u \cdot t_x \leq \hat{f}(i-u) + u \cdot t_x \tag{B.22}$$

By expanding $f(i-u)$ and $\hat{f}(i-u)$ in the inequality we get

$$a(i-k) + \lfloor\frac{(k-u+1)\cdot t_x}{S}\rfloor \cdot P + [(k-u+1)\cdot t_x \ \% \ S] + (P-S)$$
$$+u\cdot t_x \leq \hat{a}(i - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor) + (u+1)t_x + (P-S) + (j-1)P$$
$$-[(j-1)S \ \% \ t_x] \quad \text{(B.23)}$$

Substituting the right hand side according to Equation B.20:

$$a(i-k) + \lfloor\frac{(k-u+1)\cdot t_x}{S}\rfloor \cdot P + [(k-u+1)\cdot t_x \ \% \ S]$$
$$+(P-S) + u\cdot t_x \leq \hat{f}(i) \quad \text{(B.24)}$$

We also know that there are u task iterations left after the finishing of the $(i-u)$-th iteration, such that the finishing time of the −-th iteration is given as:

$$f(i) = f(i-u) + \lfloor\frac{u\cdot t_x}{S}\rfloor \cdot P + [u\cdot t_x \ \% \ S] + P - S \quad \text{(B.25)}$$

The difference $\frac{(k+1)\cdot t_x}{S} - \frac{(k-u+1)\cdot t_x}{S}$ is $\frac{u\cdot t_x}{S}$. Since $\lfloor\frac{j\cdot S}{t_x}\rfloor > k > \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$ and $u = k - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$, we can say that $u < \lfloor\frac{j\cdot S}{t_x}\rfloor - \lfloor\frac{(j-1)\cdot S}{t_x}\rfloor$ i.e. $u < \frac{S}{t_x}$. Therefore:

$$\frac{u\cdot t_x}{S} < 1 \quad \text{(B.26)}$$

Using Equation B.26 we can derive that $\lfloor\frac{u\cdot t_x}{S}\rfloor \cdot P = 0$ and that $u\cdot t_x \ \% \ S = u\cdot t_x$. We can now re-write Equation B.25 as:

$$\begin{aligned} f(i) &= a(i-k) + \lfloor\frac{(k-u+1)\cdot t_x}{S}\rfloor \cdot P + [(k-u+1)\cdot t_x \ \% \ S] \\ &\quad +\lfloor\frac{u\cdot t_x}{S}\rfloor \cdot P + [u\cdot t_x \ \% \ S] + P - S \end{aligned} \quad \text{(B.27)}$$

Again, please note that we need to consider the worst case waiting time of $(P-S)$ only once. Using Equation B.27 in Equation B.24 we conclude that

$$f(i) \leq \hat{f}(i) \quad \text{(B.28)}$$

B.3 The LPR-model extension for arbitrary execution times and slice sizes

The LPR-model initially assumes that the execution time of a single job iteration is less that the slice size allocated to that job. We now show that we can extend the LPR-model to capture the behavior of larger execution times as well. Consider a TDM setup for a job with execution time t_x and allocated a slice of size S per period P such that $t_x > S$. We can express the execution time as $t_x = k \cdot S$, where $k \in \mathbb{R}$ and $k \geq 1$. To capture the worst-case temporal behavior of this setup using the LPR-model, we replace the period and slice values in our model by a virtual slice and virtual period value. Let P' and S' represent the virtual period and virtual slice such that $P' = \lceil k \rceil \cdot P$ and $S' = \lceil k \rceil \cdot S$. We claim that by replacing P with P' and S with S' in the LPR-model we can conservatively model the job with execution time $t_x > S$. To show this we simple need to show that the worst-case temporal behavior defined using equation A.1 using the values P' and S' is an upper bound to the behavior defined using the values P and S. Let us enumerate the cases as follows:

1. For a continuous sequence of length n job iterations, where $n.t_x \ \% \ S = 0$ we define the worst case behavior for the TDM period-slice pair (P, S) as

$$f_n = s_n + \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P \tag{B.29}$$

Similarly for the TDM period-slice pair (P', S') we define the behavior as:

$$\begin{aligned} f_n &= s_n + \lfloor \frac{n \cdot t_x}{S'} \rfloor \cdot P' \\ &= s_n + \lfloor \frac{n \cdot t_x}{\lceil k \rceil \cdot S} \rfloor \cdot \lceil k \rceil \cdot P \end{aligned} \tag{B.30}$$

As we can show that $\lfloor \frac{n \cdot t_x}{\lceil k \rceil \cdot S} \rfloor \cdot \lceil k \rceil \geq \lfloor \frac{n \cdot t_x}{S} \rfloor$, we say that

$$s_n + \lfloor \frac{n \cdot t_x}{\lceil k \rceil \cdot S} \rfloor \cdot \lceil k \rceil \cdot P \geq s_n + \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P \tag{B.31}$$

2. Similarly, for a continuous sequence of length n job iterations, where $n.t_x \ \% \ S \neq 0$ we need to satisfy the predicate:

$$\lfloor \frac{n \cdot t_x}{S'} \rfloor \cdot P' + (P' - S') + (n \cdot t_x \ \% \ S') \geq \lfloor \frac{n \cdot t_x}{S} \rfloor \cdot P + (P - S) + (n \cdot t_x \ \% \ S) \tag{B.32}$$

We can split this predicate such that

- We have already shown that $\lfloor \frac{n \cdot t_x}{\lceil k \rceil \cdot S} \rfloor \cdot \lceil k \rceil \geq \lfloor \frac{n \cdot t_x}{S} \rfloor$
- Next, we can trivially show that $(P' - S') = \lceil k \rceil (P - S) \geq (P - S)$ where $k \geq 1$.
- Lastly we need to show that $n \cdot t_x \ \% \ S' \geq n \cdot t_x \ \% \ S$. Expanding the inequality we get:
$$n \cdot t_x - \lfloor \frac{n \cdot t_x}{S'} \rfloor \cdot S' \geq n \cdot t_x - \lfloor \frac{n \cdot t_x}{S'} \rfloor \cdot S' \qquad \text{(B.33)}$$
Reducing this equation we get $\lfloor \frac{n \cdot t_x}{\lceil k \rceil \cdot S} \rfloor \cdot \lceil k \rceil \leq \lfloor \frac{n \cdot t_x}{S} \rfloor$. We can show that this always holds.

By combining these three cases we can conclude that the above predicate is satisfied.

The above enumeration shows that for all cases the TDM behavior modeled using the virtual period-slice pair (P', S') is an upper bound to the TDM behavior modeled using the actual period-slice pair $(P - S)$. As the LPR-model is conservative for the (P', S') pair, it is implicitly also conservative for the (P, S) pair of the TDM setup.

B.4 The LPR-model anomaly

In Appendix B.3, we demonstrated how we can model execution times that are larger than the size of the allocated slice, using the LPR-model. However, we observe that this extension to the LPR-model causes deterioration of the accuracy of the model. The LPR-model may provide a more pessimistic estimate than the LR-model. We will now just show the counter-condition in which we observe this anomaly.

Consider a continuous sequence of job iteration of the length expressed as $\lfloor \frac{j \cdot S'}{t_x} \rfloor + 1$. If $\hat{f}(i)$ defines the finish time of the i-th iteration using the LPR-model and $f'(i)$ defines the finish time of the same iteration using the LR-model, if the LPR-model is tighter than the LR-model then $\hat{f}(i) \leq f'(i)$. To prove this, we need to satisfy the predicate:

$$j \cdot P' - [j \cdot S' \ \% \ t_x] + P' - S' + t_x \leq (\lfloor \frac{j \cdot S'}{t_x} \rfloor + 1) \cdot \frac{t_x \cdot P}{S} \qquad \text{(B.34)}$$

Solving the above predicate, we reduce it to:

$$(j + 1) \cdot \frac{\lceil k \rceil}{k} \leq (\lfloor \frac{j \cdot \lceil k \rceil}{k \cdot S} \rfloor + 1), \qquad \text{(B.35)}$$

where $k = \frac{S}{t_x}$. For $k > 1$, this predicate is not satisfied. Hence we can conclude that the extension proposed for the LPR-model to represent the worst-case behavior of jobs whose execution time is grater than the size of the allocated slice, is not better than the LR-model.