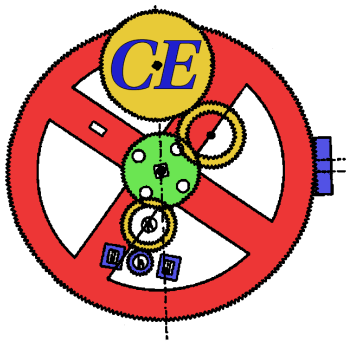




MSc THESIS

Improved static data-flow model for TDM scheduler

Kaushal Rajeev Butala

Abstract

A streaming application like software defined radio (SDR) executed on a heterogeneous multi-processor system on chip (MPSoC) consists of various transceiver jobs that have to be scheduled on different processors concurrently. The hard real time performance requirements of these jobs can be guaranteed on the multi-processor system on chip (MPSoC) using a Time Division Multiplexing (TDM) scheduler for each processor. The TDM scheduler allocates a fixed amount of time slot for each job and removes the timing inter-dependence

CE-MS-2012-05

between the concurrently executing jobs.

In order to guarantee that a job scheduled using a TDM scheduler would meet its deadlines, temporal analysis can be performed by modelling an application as a data-flow graph, called as an application graph. Scheduling framework maps nodes of the data-flow graph, called actors, on to the processors. It generates a schedule or an order in which actors of an application can execute on a processor. This order is called a static order. Data-flow model for a TDM scheduler is a timed data-flow graph which uses the TDM scheduler settings to predict worst case finish times of a job scheduled on a TDM scheduler. Using this mapping information and a data-flow model, an analysis graph is obtained from the application graph by replacing each application actor by a data-flow model and is analysed for its temporal behaviour. Various such data-flow models, e.g., the latency rate model, are proposed in the literature, but they over-estimate the worst case finish time of an application. This over-estimation causes over-allocation of resources on the processor.

In this thesis, we propose a data-flow model based on multi-rate data flow (MRDF) graph, called as a multi-rate model, that is conservative and more accurate than the existing models. We provide a detailed analysis of the multi-rate model and prove that the model is conservative. We also show that the multi-rate model provides worst case finish times that are more accurate than existing models like the latency rate model.

We implemented the multi-rate model, simulation of the TDM behaviour and techniques to reduce the utilizations of the processors on an existing data-flow analysis tool developed at ST Ericsson. We used our implementation to study the impact of improved accuracy in the estimation of the finish times of the application tasks due to the multi-rate model. The reductions in the processor utilization obtained using the multi-rate model were found to be better than the latency rate model and in some cases, better than the simulation technique. The experimental results show that the improved modelling accuracy helped us to achieve up to 40% reduction in the utilization of a processor, over the latency rate model based method for a WLAN application. Thus, the multi-rate model is a more accurate model for the analysis of the TDM scheduler arbitration as compared to the state-of-the-art.

Improved static data-flow model for TDM scheduler

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

EMBEDDED SYSTEMS

by

Kaushal Rajeev Butala
born in Khed, India

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Improved static data-flow model for TDM scheduler

by Kaushal Rajeev Butala

Abstract

A streaming application like software defined radio (SDR) executed on a heterogeneous multi-processor system on chip (MPSoC) consists of various transceiver jobs that have to be scheduled on different processors concurrently. The hard real time performance requirements of these jobs can be guaranteed on the multi-processor system on chip (MPSoC) using a Time Division Multiplexing (TDM) scheduler for each processor. The TDM scheduler allocates a fixed amount of time slot for each job and removes the timing inter-dependence between the concurrently executing jobs.

In order to guarantee that a job scheduled using a TDM scheduler would meet its deadlines, temporal analysis can be performed by modelling an application as a data-flow graph, called as an application graph. Scheduling framework maps nodes of the data-flow graph, called actors, on to the processors. It generates a schedule or an order in which actors of an application can execute on a processor. This order is called a static order. Data-flow model for a TDM scheduler is a timed data-flow graph which uses the TDM scheduler settings to predict worst case finish times of a job scheduled on a TDM scheduler. Using this mapping information and a data-flow model, an analysis graph is obtained from the application graph by replacing each application actor by a data-flow model and is analysed for its temporal behaviour. Various such data-flow models, e.g., the latency rate model, are proposed in the literature, but they over-estimate the worst case finish time of an application. This over-estimation causes over-allocation of resources on the processor.

In this thesis, we propose a data-flow model based on multi-rate data flow (MRDF) graph, called as a multi-rate model, that is conservative and more accurate than the existing models. We provide a detailed analysis of the multi-rate model and prove that the model is conservative. We also show that the multi-rate model provides worst case finish times that are more accurate than existing models like the latency rate model.

We implemented the multi-rate model, simulation of the TDM behaviour and techniques to reduce the utilizations of the processors on an existing data-flow analysis tool developed at ST Ericsson. We used our implementation to study the impact of improved accuracy in the estimation of the finish times of the application tasks due to the multi-rate model. The reductions in the processor utilization obtained using the multi-rate model were found to be better than the latency rate model and in some cases, better than the simulation technique. The experimental results show that the improved modelling accuracy helped us to achieve up to 40% reduction in the utilization of a processor, over the latency rate model based method for a WLAN application. Thus, the multi-rate model is a more accurate model for the analysis of the TDM scheduler arbitration as compared to the state-of-the-art.

Laboratory : Computer Engineering
Codenummer : CE-MS-2012-05

Committee Members :

Advisor: Dr. Orlando Moreira, ST-Ericsson

Advisor: Dr. Anca Molnos, CE, TU Delft

Chairperson: Dr. Sorin Cotofana, CE, TU Delft

Member: Dr. Arjan Van Genderen, CE, TU Delft

Member: Dr. Kees Goossens, Electronic systems, TU Eindhoven

In loving memory of my Grandfather

Contents

List of Figures	x
List of Tables	xi
Acknowledgements	xiii

1 Introduction	1
1.1 Application domain	1
1.1.1 Real time applications	1
1.1.2 Streaming Applications	2
1.1.3 Software Defined Radio (SDR)	3
1.2 Platform	4
1.3 Design flow	5
1.4 Problem Description	6
1.5 Contributions	7
1.6 Outline of the thesis	8
2 Data-flow concepts	9
2.1 Data-flow Graphs	9
2.2 Type of data-flow graphs	10
2.2.1 Multi rate data-flow graphs	11
2.2.2 Single rate data-flow graphs	13
2.2.3 Cyclo static data-flow graphs	14
2.2.4 Conversion to single rate data-flow graphs	15
2.3 Properties of data-flow graphs	18
2.3.1 FIFO property and monotonicity	18
2.3.2 Maximum Cycle Mean (MCM) of a SRDF	18
2.3.3 Self-time schedule (STS) and Static periodic schedule (SPS)	19
2.4 Unrolling of SRDF graphs	20
2.5 Summary	20
3 TDM scheduler and data-flow based models: related work	21
3.1 Description of TDM scheduler	21
3.2 Definitions of terms	23
3.3 Formalization of the temporal behaviour of a TDM scheduler	23
3.4 Periodic behaviour of TDM scheduler	24
3.5 Data-flow models for TDM scheduler	26
3.5.1 Existing models for modelling TDM arbitration	27
3.6 Problem description	30
3.6.1 Conservativity of data-flow models	30

3.6.2	Effect of pessimism	30
3.6.3	TDM scheduler and static ordering	31
3.6.4	Restrictive applicability of Staschulat's LR model	33
3.7	Summary	34
4	Multi rate model for TDM scheduler	35
4.1	definitions and conventions	35
4.2	Multi-rate model	36
4.2.1	Static ordering with multi-rate model	38
4.3	Composition of the the multi-rate model	39
4.4	Formal expressions for the single rate equivalent of the multi-rate model .	41
4.5	Formal expressions for the unrolled SRDF equivalent graph of the multi-rate model	43
4.6	Formal expressions for CSDF static order model	44
4.7	Summary	46
5	Temporal properties for multi-rate model	47
5.1	Definitions and conventions	47
5.2	Conservativity of the multi-rate model	47
5.2.1	Outline of the proof	48
5.2.2	Redundant edges in the graph G_S	51
5.2.3	Impulse response	52
5.2.4	Burst response	60
5.2.5	Conservativity for the burst response	80
5.3	Improvements over the LR model	83
5.3.1	Comparison between the impulse responses	83
5.3.2	Comparison between the burst responses	84
5.4	Summary	85
6	Implementation	87
6.1	Data-flow analysis and scheduling tool - Heracles	87
6.1.1	Inputs to Heracles	87
6.1.2	Scheduler	88
6.1.3	Analyser	91
6.1.4	Simulator	92
6.2	Features added to Heracles	93
6.2.1	Analysis graph builder for multi-rate model	93
6.2.2	Time wheel simulation	99
6.2.3	Optimizing schedules by rescheduling	100
6.2.4	Optimizing the schedules without rescheduling	102
6.3	Summary	102
7	Experimental results	105
7.1	Single actor response	105
7.1.1	Setup	106
7.1.2	Impulse response results	106

7.1.3	Burst Response	107
7.2	Application graphs used for evaluation	109
7.3	Effect of reduced pessimism in multi-rate model	111
7.4	Time taken by modelling, conversion and analysis	116
7.5	Binary slicer results	119
7.6	Randomized slicer results	122
7.7	Summary	123
8	Future work and conclusions	129
8.1	Discussion	129
8.1.1	Complexity of analysis graphs	129
8.2	Conclusions	130
8.3	Future work	131
8.3.1	Formal proofs for CSDF static order model	131
8.3.2	Applicability of the multi-rate model to other budget schedulers	132
8.3.3	Reducing the analysis and conversion time	132
	Bibliography	135

List of Figures

1.1	Broad-level architecture of a typical transceiver system (adapted from [20])	3
1.2	Typical MPSoC architecture relevant to field of study	4
1.3	Top level design flow	5
2.1	Directed Graph	9
2.2	Directed Multi-graph	9
2.3	Example of multi rate data-flow graph	11
2.4	Example of single rate data-flow graph	13
2.5	Example of a cyclo static data-flow graph	14
2.6	Example of MRDF to SRDF conversion	16
2.7	Example of CSDF to SRDF conversion	17
3.1	Example of TDM time wheel	22
3.2	Example of multiple executions of a task scheduled on a TDM scheduler .	22
3.3	Example for periodic behaviour of TDM scheduler	25
3.4	Example for Data-flow modelling	27
3.5	single actor response model	28
3.6	latency rate model	28
3.7	Staschulat's LR model	29
3.8	Clustering of statically ordered actors	32
3.9	Static ordering using LR model	33
4.1	Initial multi rate model	36
4.2	server wait emulator	36
4.3	modified multi-rate model	37
4.4	static ordering with multi rate model	38
4.5	static ordering with modified multi rate model	39
4.6	single rate form of a multi rate model	42
4.7	Example for the unrolled form of a SRDF graph	44
5.1	Proof overview	49
5.2	Delayed input token is the beginning of new burst	50
5.3	redundant (s, x) edges	51
5.4	redundant (x, c) edges	52
5.5	Impulse response example for $\lfloor \frac{S}{\tau} \rfloor > 0$	53
5.6	DAG for Figure 5.5 showing longest forward path	53
5.7	Impulse response example for $\lfloor \frac{S}{\tau} \rfloor = 0$	54
5.8	DAG for Figure 5.7 showing longest forward path	54
5.9	Example for impulse response proof with $S \geq \tau$	56
5.10	Possible arrival times for $S \geq \tau$	58
5.11	Possible arrival times for $S < \tau$	58
5.12	structure of SRDF equivalent graph for case $S = k\tau$	61
5.13	structure for the case when $\gcd(S, \tau) \neq 1$	65

5.14	structure of SRDF equivalent graph when $S \neq k\tau$ and $S\% \tau = 1$	67
6.1	Simplified flowchart for the software tool	89
6.2	application graph to analysis graph conversion	90
6.3	input graph	92
6.4	simulated graph	92
6.5	Simplified flowchart for the initiation phase	94
6.6	Simplified flowchart for the execution phase	96
6.7	Equivalent model for GCD based reduction	98
6.8	Position of Time wheel simulation in the Heracles	99
6.9	Position of slicers in Heracles	101
7.1	Setup for single actor response	105
7.2	Arrival time vs Finish time plot for experiment 3	110
7.3	Arrival time vs Finish time plot for experiment 4	110
7.4	TDS-CDMA application graph	112
7.5	Flowchart used for obtaining pessimism results	114
7.6	Effect of pessimism in the models on the finish times for TDSCDMA . . .	115
7.7	Effect of pessimism in the models on the finish times for WLAN	115
7.8	Modelling, conversion and analysis time for TDS-CDMA	118
7.9	Modelling, conversion and analysis time for WLAN	118
7.10	Reduced utilizations for ARM	120
7.11	Reduced utilizations for EVP	120
7.12	Reduced utilizations for SWC	120
7.13	Total utilization	120
7.14	Delayed firing of the 'Detect' actor in time wheel simulation	121
7.15	WLAN application graph	124
7.16	Gantt chart for TDS-CDMA with time wheel simulation	125
7.17	Gantt chart for TDS-CDMA with multi-rate model based analysis	126
7.18	Gantt chart for TDS-CDMA with LR model based analysis	127

List of Tables

7.1	Impulse response experiment 1: $S < \tau$	107
7.2	Impulse response experiment 2: $S \geq \tau$	107
7.3	Burst response experiment 3: $S < \tau$	108
7.4	Burst response experiment 4: $S \geq \tau$	109
7.5	Start times and finish times for two iterations of time wheel simulation for TDS-CDMA	111
7.6	Start times and finish times for two iterations of multi-rate analysis graph for TDS-CDMA	113
7.7	Start times and finish times for 2 iterations of LR analysis graph for TDS- CDMA	113
7.8	Number of actors and edges handled by the analyser for TDS-CDMA . . .	116
7.9	Time taken by Heracles for analysing TDS-CDMA application graph . . .	118
7.10	Time taken by Heracles for analysing WLAN application graph	119
7.11	WLAN binary slicer results	119
7.12	Time taken by binary slicer to compute reduced utilizations for WLAN application	122
7.13	Randomized slicer results for WLAN using LR model	122
7.14	Randomized slicer results for WLAN using multi-rate model	122
7.15	Randomized slicer results for WLAN using time wheel simulation	123

Acknowledgements

The work presented in this thesis was carried out at the ST Ericsson B.V. in Eindhoven in the Netherlands as a part of Computer Engineering group of the Delft University of Technology, Delft, the Netherlands.

I am thankful to Dr. Sorin Cotofana for being my supervising at the university. I am also thankful to Dr. Anca Molnos for supervising my work on day-to-day basis and reviewing my work as well as this thesis. I enjoyed working at ST Ericsson under the guidance of Dr. Orlando Moreira. He helped me immensely to understand the difficult concepts related to data-flow graphs. I am amazed by his expertise in this field of study and the way new ideas come to him. This work would not have been possible without him. I am also thankful to the colleagues and the staff of ST Ericsson for their encouragement and support.

I am extremely thankful to my parents and my brother who always stood by me and were also a source of inspiration and support.

Kaushal Rajeev Butala
Delft, The Netherlands
September 7, 2012

Introduction

The field of computing is no longer limited to servers, desktops and laptops. It has expanded to many other unobtrusive computing devices that we may not even notice very easily. These devices are in our cars, mobile phones, tablets, TV sets, music-players, DVD players, washing machines, gaming consoles etc. Some of these devices play critical role in applications like flight control, railways signalling, nuclear power plant control, and many other industrial processes. Such systems, equipped with computing devices are commonly known as embedded systems. Formally, the term "embedded systems" has been defined by *Marwedel P.* in [17] as *Embedded systems are information processing systems embedded into enclosing products*. In this chapter, we visit the application domain for our work and develop the understanding of the problem that is being addressed in this thesis. We first introduce the concept of software defined radio (SDR) and the need to use a multiprocessor environment for SDR. We then describe a design flow to map an SDR application on the multiprocessor platform. Later, we discuss the problem that we are going to address in this thesis.

1.1 Application domain

In this section, we introduce the domain where the work of this thesis is being applied.

1.1.1 Real time applications

Some embedded applications are time-critical. For example, the Anti-lock Braking System (ABS) used in cars, which is responsible for accurate braking even in slippery surface conditions, has to function correctly and also apply brakes at accurate times. If braking does not happen in time, there can be a mishap. Thus, criticality is not only associated with the correctness of the result produced but also with the time at which the result is produced [23]. This leads us to another classification of systems (not limited to embedded systems), based on their real time behaviour. Such systems are called real-time systems and are classified as:

- *hard real-time* if missing a deadline has a catastrophic consequence,
- *firm real-time* if missing a deadline is not acceptable but does not have catastrophic consequence,
- *soft real-time* if missing a deadline is not desired, but it is tolerated and the consequences are not catastrophic.

The ABS example seen earlier is an example application of a hard real-time system. A cellular mobile handset is an example application of a firm real-time system due to

regulatory restrictions concerning efficient bandwidth or network usage. For example, when a wireless communication is taking place with a handset acting as a receiver, if an acknowledgement for an input packet is not received by the transmitter in the stipulated time, then the transmitter retransmits the same packet. If a receiver frequently delays the acknowledgement, then there will be frequent retransmissions in the network. This shall reduce the network bandwidth. An example of a soft real-time system is a video/DVD player where missing a few frames is tolerated.

1.1.2 Streaming Applications

There is a class of embedded systems which are required to support streaming applications. *Streaming applications* [26] are defined as those which operate over (potentially infinitely long) large data streams. In order to handle the possibly infinitely long streams of data in a processing system with finite memory, the processing system has to work in a non-terminating way, discarding the already processed stream. Wi-Fi baseband processing (the term baseband processing is described later), audio and video transmission, playback, are some examples of streaming applications.

Baseband processing is the term used to describe the signal processing functionality that converts raw data bits into transmission-ready symbols before these are converted into analog signal for transmission over a communication link (equivalently to convert samples received from the analog to digital converter into raw data bits during the reception of a signal). Modulation, encoding etc. are the activities involved in digital baseband processing. Various communication standards like Wi-Fi, GSM, CDMA, Blue-tooth etc. may use different techniques for each of these activities and hence they may have different signal processing requirements.

If a streaming application demands real time guarantees, then the processing system must be able to provide them. All the communication standards mentioned above would have to meet certain timing requirements in order to achieve functional correctness and comply to communication and regulatory standards. Baseband processing of wireless communication standards is performed by firm real-time streaming applications. Real-time streaming applications generally are said to have two types of timing requirements namely, the throughput requirements and the latency requirements. Throughput requirements are associated with the rate at which an repetitive application produces results and latency requirements are associated with restrictions on the time interval between the arrival time of the input and the time at which the output becomes available. Thus, real time streaming applications would have to meet the criteria of minimum throughput and maximum latency requirements. For example, Wi-Fi application has both latency and throughput requirements. The latency requirements are due to the acknowledgements that have to be sent after the Short Inter-Frame Space (SIFS) interval. FM radio requires only throughput guarantees, since transmission happens only in one direction and acknowledgements are not sent back to the transmitter. For some applications there can be multiple latency requirements. The present day mobile phones not only support multiple mobile communication standards but also have other capabilities like wireless Internet, GPS navigation, music and video playback, video capturing, gaming, etc. Thus,

the mobile phones, tablets, etc. are required to support streaming applications. The support for multiple communication standards, wireless Internet, GPS navigation requires a mobile phone or a tablet to support multiple wireless communications standards. In the next section, we introduce a special type of system capable of handling multiple streaming applications called as software defined radio.

1.1.3 Software Defined Radio (SDR)

Today, mobile phones use dedicated hardware for each type of communication. As the number of such communications to be supported increases, the cost of such a solution also increases. Reconfigurability of such dedicated hardware towards the new standard is limited. Hence a search for a more flexible, configurable solution led the research community to the concept of Software Defined Radio (SDR).

Software Defined Radio (SDR) is a term coined by *J. Mitola* in [18]. Software defined radio as the name suggests, is a technique in which certain signal processing tasks, more specifically baseband processing tasks, related to transmission and reception are performed on a programmable processor with the help of software instead of dedicated hardware. The reason why SDR is an attractive proposition is due to the fact that, broadly, various communication standards have a common level architecture. Such a generalized architecture is shown in Figure 1.1.

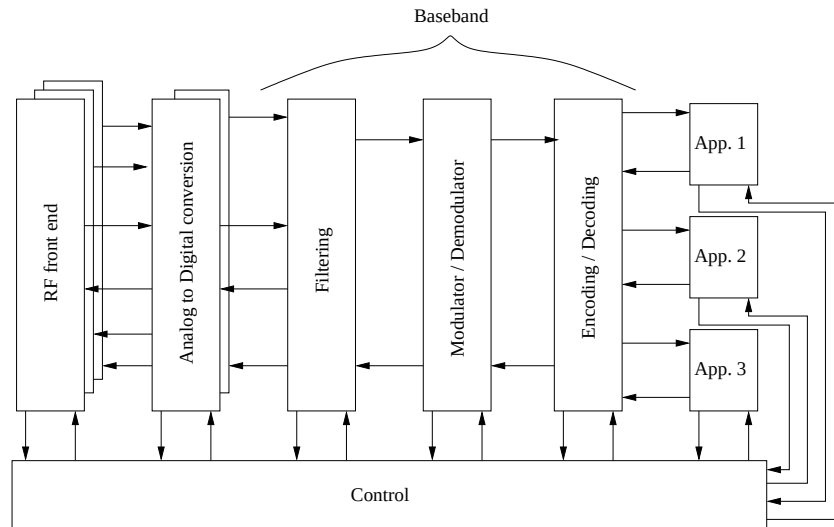


Figure 1.1: Broad-level architecture of a typical transceiver system (adapted from [20])

Referring to Figure 1.1, the baseband section is typically implemented digitally. This section includes modulation (demodulation), encoding (decoding) and filtering stages. These stages are, in general, different for different protocols and may require dedicated hardware for each protocol. Thus, if baseband processing can be implemented on a programmable hardware, then supporting multiple protocols on a single hardware is possible. Such a system can not only support multiple standards at a time but also provides a capability to support newer standards in the future, provided the RF section can

support it. With a single device like mobile phone supporting multiple communication standards like IEEE 802.11 a/b/g/n for WLAN; UMTS, TDS-CDMA, HSDPA, LTE for 3G/4G; GSM, GPRS, EDGE for 2G; Bluetooth for WPAN and GPS for navigation, SDR is a very interesting prospect.

1.2 Platform

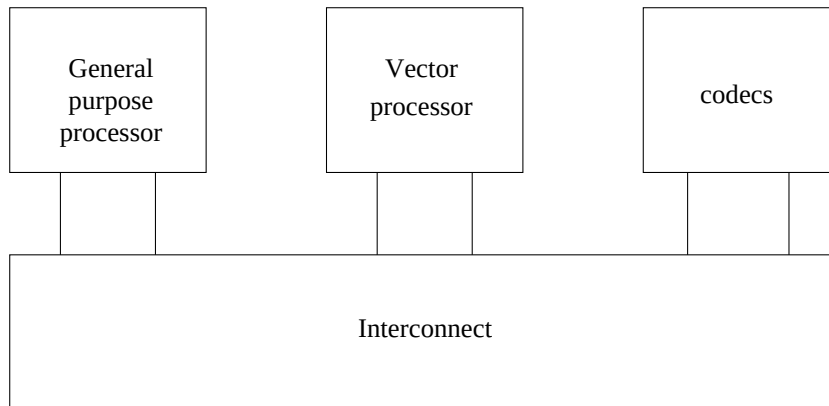


Figure 1.2: Typical MPSoC architecture relevant to field of study

What kind of programmable hardware do we really need? This is a very important question and has been addressed to a great detail in [28]. There are different hardware requirements for each stage. The control section can be implemented on a general purpose processor like ARM. The modulator/demodulator stage (also called as the inner receiver) has large variety of algorithms with lot of matrix operations and thus programmable hardware like a vector processor is more appropriate. The encoder/decoder stage (also called as outer receiver) deals mostly with bit manipulation and here dedicated accelerators can be a good choice. Thus, a heterogeneous multiprocessing platform on a chip (MPSoC) is a good fit for SDR. There have been few architectures proposed for SDR, notably amongst them are SODA by University of Michigan [16] (ARM Ard-beg was its early commercial form but now it forms a basis for a spin-out out of ARM called Cognovo [30]), MuSIC by Infineon [21] etc. Since all of the protocols listed above are streaming real-time applications with different baseband processing and timing requirements, the SDR system has to provide real time guarantees for each protocol, independent of other communication taking place simultaneously. For example, consider a case where a mobile phone user is using internet to play music onto his Blue-tooth headset from his (or her) phone, while the phone is communicating with the base-stations periodically so that it remains connected to the network. This is a fairly common use case. Thus, being able to provide real time guarantees is an important requirement of a SDR system and is also a design challenge. The SDR system should be capable to handling dynamic resource requests like in the case where another type of communication is turned on while the current communication is still going on.

The architecture that we consider for our work is a heterogeneous MPSoC with one

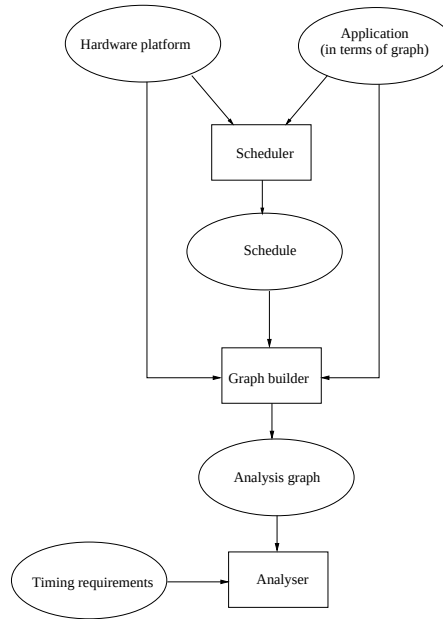


Figure 1.3: Top level design flow

or more general purpose processors like ARM, one or more vector processors like EVP, and dedicated software codecs (for example Viterbi or Turbo coder and decoder) or accelerators for application specific usage. The inter-connect should be the one which gives predictable timing behaviour, something similar to a bus interconnect. The example for such an architecture is shown in Figure 1.2. The application domain for our work is providing a mechanism to guarantee if the real time streaming applications, more specifically SDR, meet their respective timing requirements when they are being processed on a Multiprocessor system on chip (MPSoC), similar to the one described above.

1.3 Design flow

In SDR, we have seen that meeting the timing requirements for each radio application, has to be independent of the other applications running at that time. Each streaming application is divided into a number of entities called tasks which perform certain computations and have a worst case execution time. In order to achieve this goal, the tasks of different streaming applications have to be scheduled in such a way that they do not affect timing performance of each other. There is a class of schedulers called *budget schedulers* [25] that provide fixed budget to each task in a fixed time interval. Time division multiplexed (TDM) scheduling is one such scheduler technique that provides fixed budget to each task thus providing temporal independence to the tasks. The most useful property of TDM scheduler is that it switches the tasks at known times. Thus, it is possible to calculate an upper bound for the finish time of each task independently of the other tasks. This also makes analysis of temporal behaviour possible. It is a simple scheduler with low run time overhead. Hence we base our study on TDM scheduler. We

shall cover the TDM scheduler in detail in Chapter 3.

A streaming application can be represented by using a data-flow graph [5]. The vertices of such a graph called actors represent the tasks and the edges represent First-In-First-Out (FIFO) communication channels. The graph thus formed is called an application graph. With some restrictions, data-flow graphs are analysable, possess well defined formal properties and can also be used for code synthesis. We will see the important types of data-flow graphs and their properties in chapter 2. Data-flow graphs have been used in number of research problems like buffer sizing, temporal analysis, effect of scheduler arbitration etc. A software tool called Heracles is being developed to support the design effort of mapping the tasks in a SDR onto the MPSoC such that the time guarantees are met. The tool has a design flow as soon in Figure 1.3. It accepts the application and the hardware platform. The hardware platform also has information about the scheduler used on the each processor. It generates a schedule based on these inputs.

Data-flow graphs are used extensively for modelling the effect of the TDM scheduler arbitration on the temporal behaviour of real time tasks. Such data flow models have been proposed in [29] [5] [24]. These data-flow models are in-fact data-flow graphs that are constructed with help of TDM scheduler parameters like TDM cycle-time (also called period), the slice duration allotted to a real time task and the worst-case execution time (WCET) of the task being modelled. A data-flow graph created from the data-flow models described above is denoted as an analysis graph. The parameters of an analysis graph are based on the temporal attributes of the scheduler and the tasks. If temporal analysis of such a graph is carried out, then we are actually analysing the application for the given resource mapping and the given slice time allocation for each real time task. If the analysis is favourable then we can say that the given mapping of resources and slice time allocations are sufficient to meet the timing requirements of the application. The graph builder in Figure 1.3 uses the schedule and other inputs to generate an analysis graph based on some data-flow model of TDM scheduler, as described above. This graph is analysed for temporal validity against the temporal requirements. The design flow in Figure 1.3 finally generates a list of valid schedules after iteratively (iterations not shown in the Figure 1.3) going through all the stages for a number of times.

1.4 Problem Description

With multiple radios operating simultaneously, each having its own real-time requirements and sharing processor times on a MPSoC with the help of a TDM scheduler, we need an accurate temporal analysis technique to ensure that all the active applications meet their timing requirements while avoiding over-utilization of resources and unnecessary rejections of applications that might start later.

We have discussed in the previous section, how to develop an analysis graph from a given application graph using the data-flow models for TDM scheduler. A model is said to be pessimistic if it predicts a finish time that is greater than the actual finish time of a task due it being scheduled on a processor. Models used for modelling TDM scheduler arbitration like the single actor response model [5] and the latency rate model [29] are pessimistic and hence the slice times required to get a favourable analysis result would be

over-estimated. This over-estimation leads to wastage of resources. The over-estimation is reflected in the analysis graph, by virtue of its construction. The latency rate model proposed by *Staschulat et al* in [24] is accurate but cannot be used in all conditions like when the allocated slice time is smaller than the WCET of a task (modelled as an actor). The Latency Cyclic Rate (LCR) model [13] is also accurate but does not support static ordering (static ordering is an optimization that we shall see in Section 3.6.3), which also leads to over-estimation of resources. We shall see these models and their drawbacks in greater detail in Chapter 3. Thus, there is a need to have a model that is accurate, conservative, and generic and supports static ordering to model the worst case temporal response of a task scheduled on a resource managed by TDM scheduler. The multi-rate model proposed in [13] is a promising candidate for developing an accurate model, but no theoretical analysis is done so far for the multi rate model and nothing could be said about its accuracy as well as conservativity. A detailed problem definition is covered in Chapter 3 in Section 3.6.

1.5 Contributions

The contributions of our work are discussed below:

- We propose certain modifications to the multi-rate model [13] to support static ordering of actors. We propose a new way of representing static order of actors.
- We provide theoretical proofs that the multi rate model is accurate and conservative. In doing so, we also prove a number of interesting properties about the model such as Maximum Cycle Mean(MCM), cyclicity and the number of iterations of the model in the transition phase of the periodic execution.
- Since multi rate graphs have to be converted into single rate for analysis, the analysis time increases many-fold. We investigate and propose a method of simplifying the multi-rate model under certain conditions, in order to reduce the analysis time. We prove that the reduction technique is exact and does not impact the accuracy or conservativity of the model.
- We prove that the multi-rate model is less pessimistic than the latency rate model.
- We present techniques to simplify the model even further by compromising on the amount of pessimism but which will reduce the analysis time.
- We implement the analysis graph builder which uses the multi-rate model and reduction techniques mentioned above.
- We implement techniques for obtaining optimized schedules in terms of lesser resource utilization. These techniques are called slicing techniques.
- We present experimental results with WLAN and TDS-CDMA applications running on an MPSoC hardware showing the effect of less pessimism of modelled start and finish times. We also present results showing reduced utilization of the processors due to the improved model.

1.6 Outline of the thesis

The thesis is organized as follows. In Chapter 2, we explain the basic concepts of data-flow and describe various properties of data-flow graphs. In Chapter 3, we describe resource management using TDM arbitration. We capture the behaviour of the TDM scheduler and formalize it. We also describe the existing models that capture the TDM scheduler behaviour. We explain the problems with the existing models which is also a reason why we need a better model. We present the improvements made to the multi-rate model and formalize the modified model in Chapter 4. We analyse the behaviour of TDM scheduler under various input conditions in Chapter 5. We also prove certain properties of the multi-rate model and then use these properties to prove that the model is conservative. In Chapter 6, we present the implementation details of the existing software tool and the features added to it during the course of this thesis. In Chapters 7 and 8, we present the results, conclusions, insights and future work.

Data-flow concepts

In this chapter, we describe the data-flow formalism. We then discuss the reason for which we are interested in the data-flow graphs. We present an overview of their properties and their analysis techniques which we will be using in further chapters. We will be referring to the sections in this chapter frequently. Data-flow graphs are covered in a greater detail by *Sriram et al* in [22].

2.1 Data-flow Graphs

Data-flow is a concept of expressing an application as a set of tasks and their precedence constraints determined by the data dependencies between these tasks. The data-flow graph is a formal technique to represent such an application using a directed graph structure.

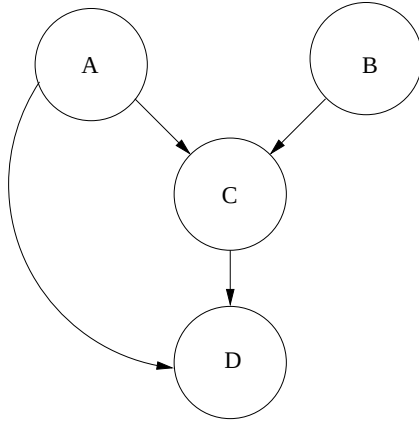


Figure 2.1: Directed Graph

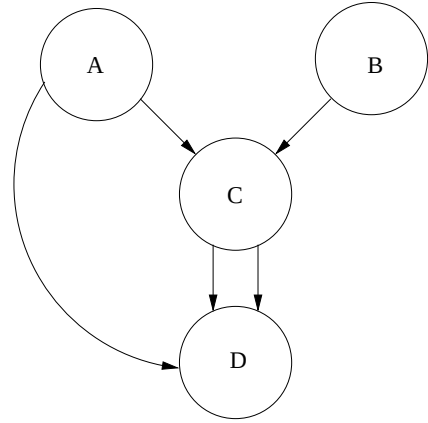


Figure 2.2: Directed Multi-graph

A *directed graph* is formally defined as an ordered pair (V, E) , where V is a set of vertices and E is the set of edges. An edge E is a set of ordered pair of vertices (v_1, v_2) , with $v_1, v_2 \in V$. Any edge $e \in E$ is said to be directed from v_1 to v_2 , if $e = (v_1, v_2)$. Node v_1 is called a source of edge e denoted by $src(e)$ while node v_2 is called a sink of edge e denoted by $snk(e)$. Figure 2.1 shows an example for directed graph. A *directed multi-graph* (see Figure 2.2) is superset of directed graphs which allows a source vertex and a sink vertex to have more than one edges between them.

A *path* in a graph is a sequence of edges $(e_1, e_2, e_3, \dots, e_n)$ such that $snk(e_i) = src(e_{i+1})$, $0 \leq i \leq n-1$, where n is non-zero and finite. A simple path is the one in which all actors are visited once. A *cycle* is a simple path with the repetition of the first node such that last node is same as the first node.

A data-flow graph is a directed multi-graph, in which vertices represent computa-

tional tasks and edges represent FIFO (First-in first-out) queues that pass the data of one computational task to another computational task. Thus, edges represent the data dependencies between the vertices. The vertices of a data-flow graph are commonly called actors. Actors consume data from the input edges and produce data on the output edges. Data is assumed to have communicated in terms of atomic containers called *tokens*. Also, execution of a computational task is called as *firing* of an actor. Thus, during every firing, an actor consumes tokens from its input edges and produces tokens onto its output edges after completion. The number of tokens consumed and produced per edge are respectively called consumption and production rates (the word rate is used to imply the amount for tokens produced/consumed per firing) denoted usually by $cons(e)$ and $prod(e)$ respectively. Some edges may have an initial distribution of tokens. These are called *delay* tokens.

Data-flow graph is a model of computation which can successfully capture the properties of streaming applications [3]. The tasks of streaming applications have to be performed iteratively. These tasks also have data dependencies between them. Hence, these tasks consume and produce fixed amount data in every execution. This behaviour can be captured by tokens. The edges capture the notion of communication channels and the actors capture the notion of tasks.

The data-flow execution model is based on the idea that an actor can *fire* only when sufficient number of tokens are available on each of its input edges. Hence, it is also called a data-driven approach to computation. A condition under which an actor can fire (a task can execute) is called a *firing rule*. Actors with incoming edges can fire only at certain favourable instances of time when *firing rule* is satisfied. The firing of an actor depends upon the availability of data. This property suits the data-driven streaming applications like SDR. There might be cases where data is available to multiple tasks at the same time enabling them to execute simultaneously. Hence, data-flow supports the notion of concurrency of computations.

In our application domain, data-flow graphs are used to represent SDR streaming applications, which typically handle the infinitely long data sequences. These applications have to work in a periodic fashion to be able to work using finite memory. Hence, we need to obtain periodic schedules for these graphs. In order to achieve this, the graph should return to a state where its original token distribution is reached again. The graphs which can be scheduled in finite memory are called correctly constructed graphs. Also, deadlocked graphs do not allow all actors in the graph to be iterated infinitely. Hence, we use graphs which are correctly constructed and deadlock free [22]. We also define the concept of a timed graph for each type of data-flow graph for the purpose of scheduling and timing analysis of graphs.

2.2 Type of data-flow graphs

We briefly discuss various types of data-flow graphs in this section. We cover the data-flow graph types related to our work in more detail than the other types of data-flow graphs.

2.2.1 Multi rate data-flow graphs

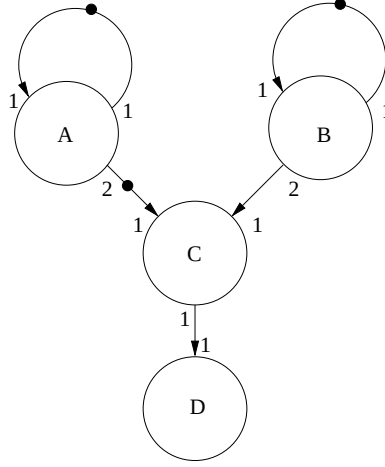


Figure 2.3: Example of multi rate data-flow graph

Multi Rate Data-Flow (MRDF) graph, also known as synchronous data-flow (SDF) graph [11] [22] is a directed graph, in which the number of tokens produced and consumed per edge is known and it is a fixed integer. Each edge may have initial distribution of tokens which is referred as delay of the edge.

An example of a MRDF graph is shown in Figure 2.3. The black dots indicate the initial tokens or delays. Each edge has a number written next to the point where it connects to the source actor and to the sink actor. These numbers are production and consumption rates respectively of that edge.

MRDF graphs have a fixed firing rule for each actor. These are

- An actor consumes a fixed number of tokens along its input edges and produces fixed number of tokens on its output edges. The number of tokens consumed and produce on every edge is known *a priori*, that is, at compile time and it is same for every firing of the actor.
- An actor can fire only when the requisite number of tokens are available along its input edges.

A timed MRDF graph is defined by the tuple $(V, E, d, t, prod, cons)$. The set of timed MRDF graph actors are denoted by V and have a valuation $t : V \rightarrow \mathbb{N}$; where, $t(i)$ is the execution time of actor i . The set of timed MRDF graph edges have a valuation $d : E \rightarrow \mathbb{N}$; where, $d(i, j)$ is the initial number of tokens (delays) of arc (i, j) . Edges also have following valuations: $prod : E \rightarrow \mathbb{N}$ and $cons : E \rightarrow \mathbb{N}$. Valuation $prod(e)$ gives the fixed number of tokens produced by the source of edge e and valuation $cons(e)$ gives the fixed number of tokens consumed by the sink of edge e . A MRDF graph can be denoted by a *topology matrix* [12]. A topology matrix is denoted by Γ . In a topology matrix, the number of rows is equal to $|E|$ and number of columns is $|V|$. A number in the i^{th} row and the j^{th} column of a topology matrix is

- positive, if tokens are produced by node j along edge i
- negative, if tokens are consumed by node j along edge i
- equal to difference between the number of tokens produced and consumed, if tokens are produced as well as consumed by node j along edge i .
- zero, there is no edge between i and j .

If a MRDF graph is correctly constructed, then we should be able to schedule it on a finite memory resource. If a correctly constructed MRDF graph returns to its original state (i.e. initial distribution of tokens) after r_i firings of an actor i , $\forall i \in V$, then we can number of firings for each actor as a column vector $\vec{r}$. When an actor a fires r_a (r_a is the column in the $\vec{r}$ related to repetition of a) times, it produces a total of $r_a.prod(a, j)$ tokens along its outgoing edges towards actors j . Similarly, it consumes a total of $r_a.prod(i, a)$ tokens along its incoming edges coming from actor i . The tokens total tokens consumed and produced in r_a firings should be equal to keep the number of tokens bounded. Hence, for all edges $e(i, j)$ we can write

$$r_i.prod(i, j) = r_j.cons(i, j) \quad (2.1)$$

Equation 2.1 can be written in matrix form as

$$\Gamma.\vec{r} = \vec{0} \quad \vec{r} \neq \vec{0}. \quad (2.2)$$

Equation 2.2 is called the balance equation of the MRDF graph. It has one trivial solution $\vec{r} = \vec{0}$. Apart from this trivial solution, if there is no solution to Equation 2.2, then the MRDF graph denoted by Γ is inconsistent and it cannot be scheduled in finite memory.

2.2.1.1 Repetitions vector for MRDF graphs

If V is a set of actors, then the repetitions vector is defined as a vector of length $|V|$, in which each entry corresponds to a minimum number of firings of an actor in the graph such that, when each actor fires for the given number of times, the edges of the graph will have the same token distribution as the initial token distribution. The smallest integer vector which satisfies the balance equation 2.2 is the repetitions vector. An *iteration of a MRDF graph* is considered to be complete when each actor in the MRDF graph fires as many times as the number corresponding to it in the repetitions vector. Thus, the solution of the balance equation gives us how many firings per actor are executed in an iteration of a MRDF graph.

2.2.2 Single rate data-flow graphs

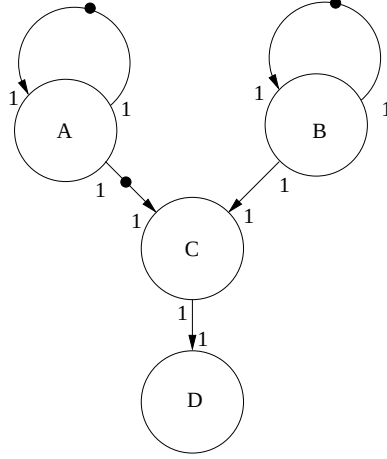


Figure 2.4: Example of single rate data-flow graph

A Single Rate Data-Flow (SRDF) graph, also known as a Homogeneous data-flow (HSDF) graph is a special case of MRDF (SDF) graph, where the number of tokens produced and consumed per edge is unity. Because of this, SRDF graphs have lesser expressive power as compared to the MRDF graphs. This can be seen from the repetitions vector entry for a multi-rate actor. For each actor in the multi-rate graph, we need

A timed SRDF graph is defined by the tuple (V, E, d, t) . The set of Timed SRDF graph actors is denoted by V and have a valuation $t : V \rightarrow \mathbb{N}$; where, $t(i)$ is the execution time of actor i . Timed SRDF graph edges have a valuation $d : E \rightarrow \mathbb{N}$; where, $d(i, j)$ is the initial number of tokens (delays) of arc (i, j) .

An example of a SRDF graph is shown in Figure 2.4. The black dots indicate the initial tokens or delays.

Since, SRDF is a special case of a MRDF graph, all the expressions that have been provided for MRDF graphs in Section 2.2.1 are applicable to SRDF graphs as well with $prod(i, j) = cons(i, j) = 1$. Apart from the trivial solution $\vec{r} = \vec{0}$, the only other possible solution is $\vec{r} = [1 \ 1 \ 1 \dots |V| \text{ times}]^T$. If such non-trivial solution exists, then the SRDF graph is consistent and can be scheduled in finite memory.

An *iteration* of a SRDF graph is considered to be complete when all the actors fire once.

2.2.3 Cyclo static data-flow graphs

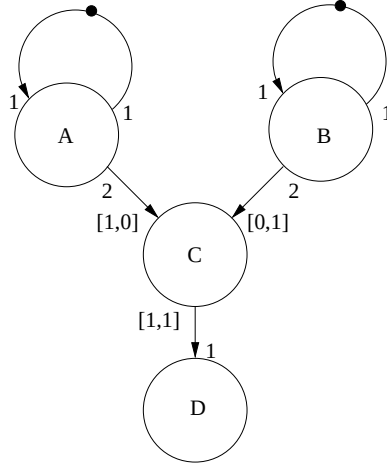


Figure 2.5: Example of a cyclo static data-flow graph

Cyclo-static data-flow (CSDF) [6] is a data-flow graph in which there is at least one actor that has a pre-defined sequence of production and consumption rates along its incoming and outgoing edges for its consecutive firings. This sequence repeats periodically. For example, the actor C in the Figure 2.5 has a sequence of 2 phases. It acts like a selector switch, selecting outputs from actors A and B alternately, during each phase. We cannot model such functionality using a MRDF graph. Thus, CSDF has more expressive power than MRDF graphs. In fact, every MRDF graph can also be expressed as a CSDF graph with production and consumption sequences having same number repeated r_i times for an actor i . Thus, CSDF can be seen as a generalized form of MRDF graph.

A timed CSDF graph is defined by the tuple $(V, E, d, t, prod, cons)$. The set of Timed CSDF graph actors is denoted by V and have a valuation $t : V \rightarrow \mathbb{N}$; where, $t(i)$ is the execution time of actor i . Timed CSDF graph edges have a valuation $d : E \rightarrow \mathbb{N}$; where, $d(i, j)$ is the initial number of tokens (delays) of arc (i, j) . Edges also have following valuations: $prod : E \rightarrow \mathbb{N}$ and $cons : E \rightarrow \mathbb{N}$. $prod(e)$ gives the sequence of tokens production rates of edge e and $cons(e)$ gives the sequence of consumption rates of edge e .

A CSDF graph can also be denoted by a *topology matrix* [6]. A topology matrix is denoted by Γ . In a topology matrix, the number of rows is equal to $|E|$ and number of columns is $|V|$. A number in the i^{th} row and the j^{th} column of a topology matrix is

- positive sum of all production rates in the production sequence, if tokens are produced by node j along edge i
- negative sum of all consumption rates in the consumption sequence, if tokens are consumed by node j along edge i
- equal to difference between the sum of all production rates and sum of all consumption rates, if tokens are produced as well as consumed by node j along edge i .

- zero, if there is no edge between i and j

For CSDF, the following condition also holds true.

$$\Gamma.\vec{r} = \vec{0} \quad \vec{r} \neq \vec{0}. \quad (2.3)$$

Here $\vec{r}$ is a vector obtained as a smallest integral solution of equation 2.3. It has one trivial solution $\vec{r} = \vec{0}$. If apart from this trivial solution, if there is no solution to Equation 2.3, then the CSDF graph is inconsistent and it cannot be scheduled in finite memory.

2.2.3.1 Repetitions vector for CSDF

If $\vec{r}$ is the smallest positive integral solution of equation 2.3, then repetitions vector for CSDF can be written as

$$\vec{q} = P\vec{r}. \quad (2.4)$$

where,

$\vec{q}, \vec{r}$ are vectors of length $|V|$.

P is the diagonal matrix of the number of phases of each actor.

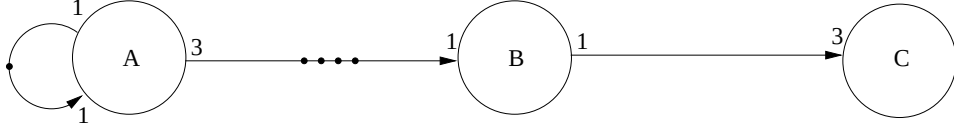
An *iteration of a CSDF graph* is considered to be complete when each actor in the CSDF graph fires as many times as the number corresponding to it in the repetitions vector.

2.2.4 Conversion to single rate data-flow graphs

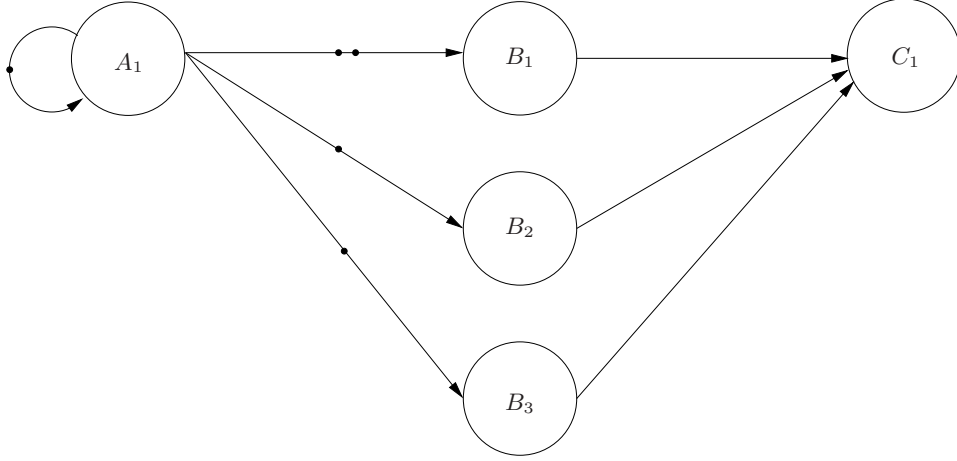
Conversion from MRDF to SRDF or from CSDF to SRDF becomes necessary for purpose of analysis, since there are some analysis techniques and properties that can be used for single rate graphs only. For example, the Maximum Cycle Mean (MCM) (defined in Section 2.3.2) is only defined for a SRDF graph. Also, it is easy to calculate the Static Periodic Schedule (SPS) (defined in Section 2.3.3.3) bounds for a SRDF graph.

2.2.4.1 Conversion of MRDF graphs to Single rate data-flow graphs

A well-constructed MRDF graph can be converted to SRDF graph. Each actor a in a MRDF graph has r_a copies in the converted SRDF graph, where r_a is the number of firings of the actor a as given by the repetitions vector. If p_a is the production rate of an edge (a, b) in a MRDF graph and c_b is the consumption rate of that edge, then there are p_a edges outgoing from each one of the r_a copies of actor a in the SRDF graph and there are c_b edges incoming towards each of the r_b copies of actor b in the SRDF graph. The edges in the SRDF graph are connected in such a way that the firing of every actor in the SRDF graph remains identical to the firing of the actors in the MRDF graph. The k^{th} firing of i^{th} copy of an actor a in a SRDF graph with a total of r_a copies corresponds to $(r_a.k + i)^{th}$ firing of that actor in the MRDF graph. If there are d delay tokens along an edge (a, b) in the MRDF graph, then these are distributed along the r_b incoming edges in the following way. There will one delay token along j^{th} incoming edge based on the rule $j = \{i \% r_b | 0 \leq i < d\}$. Let us see an example to understand this transformation.



(a) Example of MRDF graph



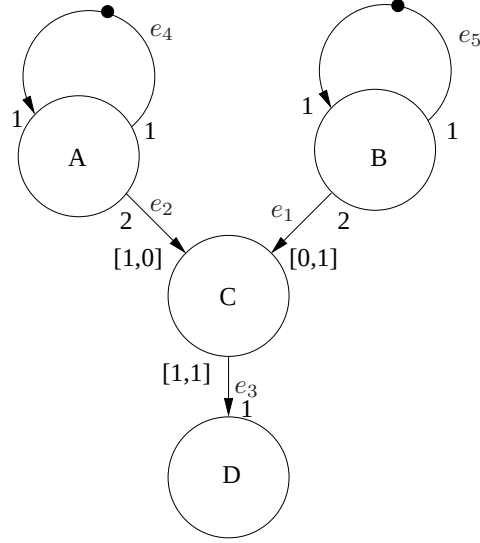
(b) Equivalent SRDF graph

Figure 2.6: Example of MRDF to SRDF conversion

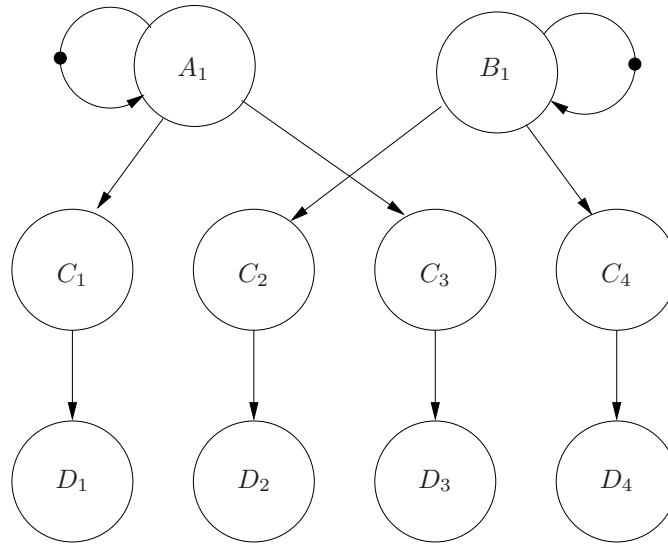
As per Figure 2.6, The production rate of edges (A, B) and (B, C) are 1 and 3 respectively while consumption rates are 3 and 1 respectively. Thus, the repetition vector is $[1 \ 3 \ 1]^T$. We get one copy of actors A, C and 3 copies of actor B . The production rate of edge (A, B) in the MRDF graph is 3. Hence, there are 3 edges outgoing from A . Similarly, the consumption rate of edge (A, B) is 1. Hence, each of the C actors in the SRDF graph has an incoming edge. The remaining edges in the SRDF graph can also be explained in the similar way.

2.2.4.2 Conversion of CSDF graphs to Single rate data-flow graphs

A well-constructed CSDF graph can be converted to a SRDF graph. Each actor a in a CSDF graph has q_a copies in the converted SRDF where q_a is the number of firings of actor a given by the repetitions vector $\vec{q}$. The edges in the SRDF graph are connected in such a way that the firing of every actor remains identical to the firing of the actors in the CSDF graph. Let us see an example.



(a) Sample CSDF Graph



(b) Equivalent SRDF graph

Figure 2.7: Example of CSDF to SRDF conversion

As per Figure 2.7, the matrix Γ is given by

$$\Gamma = \begin{matrix} & \begin{matrix} A & B & C & D \end{matrix} \\ \begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{matrix} & \begin{pmatrix} 0 & 2 & -1 & 0 \\ 2 & 0 & -1 & 0 \\ 0 & 0 & 2 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{matrix}$$

From equation 2.3, we can calculate $\vec{r}$. The value of $\vec{r} = [1 \ 1 \ 2 \ 4]^T$. From equation 2.4, we can calculate the value of $\vec{q}$. Therefore,

$$\vec{q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 4 \\ 4 \end{bmatrix}.$$

Thus, there are 4 copies of actor C and D each and one copy of actors A and B each. Thus the equivalent SRDF graph can be drawn as in Figure 2.7(b).

2.3 Properties of data-flow graphs

In this section, we will see the properties of various data-flow graphs.

2.3.1 FIFO property and monotonicity

FIFO (First in, first out) property suggests that two consecutive executions of an actor will always produce output tokens in the order of their firing. Any data-flow graph can conserve FIFO property if it has fixed execution time for all executions. If, for an actor i with start time $s(i, k)$ and finish time $f(i, k)$ for its k^{th} firing, and start time $s(i, k-1)$ and finish time $f(i, k-1)$ for its $(k-1)^{th}$ firing, the FIFO property can be stated formally as if $s(i, k) \geq s(i, k-1)$, then $f(i, k) \geq f(i, k-1)$. If the actor i has constant execution time τ , then $s(i, k) + \tau = f(i, k)$ and $s(i, k-1) + \tau = f(i, k-1)$. Adding τ to both sides of the equation $s(i, k) \geq s(i, k-1)$, we get $(s(i, k) + \tau) \geq (s(i, k-1) + \tau)$ or $f(i, k) \geq f(i, k-1)$.

Monotonicity property states that early finish (or decrease in execution time) of any actor in a data-flow graph cannot cause the other firings in the graph to occur later in time (i.e. increase in their start time).

2.3.2 Maximum Cycle Mean (MCM) of a SRDF

The Maximum cycle mean (MCM) of a SRDF graph is defined as the maximum ratio of the sum of execution time in a cycle to the number of delay tokens along the cycle, over all cycles. If $C(G)$ is the set of all simple cycles in a graph, $V(c)$ is a set of all vertices along a cycle c ($\in C(G)$), $E(c)$ is a set of all edges along a cycle c ($\in C(G)$), t_i is the execution time of actor i and d_e is the number of delay tokens along edge e . Then, MCM is defined as

$$\mu(G) = \max_{c \in C(G)} \frac{\sum_{i \in V(c)} t_i}{\sum_{e \in E(c)} d_e} \quad (2.5)$$

The MCM is defined only for a SRDF graph. It is equal to reciprocal of the throughput of a strongly connected SRDF graph [20]. Any graph can be considered as a connection of strongly connected sub-graphs. If MCM of a graph is infinity, then there exists a cycle in the graph which does not have any delay tokens. Such a graph is said to be deadlocked. Any well-constructed CSDF or MRDF graph can be converted into a SRDF graph. Thus, any graph that can be converted to a SRDF graph can be checked for

deadlock freedom by computing its MCM. Hence, MCM is an important property for temporal analysis.

2.3.3 Self-time schedule (STS) and Static periodic schedule (SPS)

In this section, we define the two type of schedules and present the relation between them. We also define the concept of cyclicity of a graph.

2.3.3.1 Self-time schedule (STS)

In this section, we discuss the self-time schedules and static periodic schedules for SRDF graphs, knowing that the MRDF graphs and cyclo-static graphs can be converted to single rate for the purpose of analysis.

A self-time schedule STS for a data-flow graph is defined as a schedule in which actors fire whenever requisite number of tokens are available on all their input edges. A STS where every actor in the data-flow graph takes its worst worst-case time to execute, then it is called worst-case self-time schedule (WCSTS). For any graph, WCSTS is unique.

2.3.3.2 Cyclicity and Iteration count in transition phase (K) for a SRDF graph

The WCSTS of a strongly connected SRDF graph G enters periodic execution after a transition phase of K iterations. The K is a constant for a given SRDF graph and is called as iteration count in transition phase. The periodic execution has a period of $N(G) \cdot \mu(G)$, where $N(G)$ is the cyclicity of the graph and $\mu(G)$ is the maximum cycle mean (MCM). The term cyclicity ($N(G)$) is defined as the greatest common divisor of the sums of delays along all the critical cycles of the graph. Critical cycles are those cycles which have cycle length to delay ratio equal to the MCM. If $s(i, k)$ is the start time of actor i , then the property mentioned above can be written as [20]

$$s(i, k + N(G)) = s(i, k) + N(G) \cdot \mu(G), \quad \forall k \geq K. \quad (2.6)$$

This equation is true for all actors in a strongly connected graph and under a STS schedule.

2.3.3.3 Static periodic schedule (SPS)

A static periodic schedule of a SRDF graph is defined by the relation [20]

$$s(i, k) = s(i, 0) + T \cdot k \quad \forall i \in V, \quad \forall k \quad (2.7)$$

Here,

T is the period of the SPS,

$s(i, k)$ is the start time of the k^{th} iteration,

$s(i, 0)$ is the start time of the first iteration.

If period of a SPS T is equal to $\mu(G)$, then it is called rate-optimal static periodic schedule. A related theorem presented in [20] states that SPS exists for a SRDF graph provided the period of the SPS is not less than the MCM of the graph. Thus, MCM bounds the rate of SPS.

2.3.3.4 Relation between STS and SPS

As per a theorem in [20], Any SPS schedule start time is an upper bound on the start time of any STS schedule start time for the same firing and same actor. Thus, SPS bounds WCSTS. This theorem enables us to compute an upper bound on the finish times obtained from any self-time execution of the graph using a SPS.

Hence, MCM becomes a very important tool for temporal analysis of the streaming applications which are expressed as data-flow graphs.

2.4 Unrolling of SRDF graphs

In this section, we define the unrolling of a SRDF graph. We have seen earlier in Section 2.2.2 that one iteration of an SRDF graph is complete when all the actors fire once. Unrolling is an operation which represents multiple iterations of a SRDF graph as a single graph. This helps us to visualise how multiple graph iterations interact with each other. We have also seen in Section 2.3.3.2 that a SRDF graph enters a periodic regime after certain number of iterations. We use the unrolling technique to analyse the this initial transition behaviour in the next chapters.

If a SRDF graph is unrolled by a factor of u , ($u \geq 1$), then each actor will have u copies. Then the k^{th} firing ($k \in [0, \infty)$) of the i^{th} copy ($i \in [0, u)$) of an actor a is in the unrolled graph is equivalent to the $(k.u + i)^{th}$ iteration of that actor in SRDF graph.

2.5 Summary

In this chapter, we have presented various type of data-flow graphs that will be used later in this thesis. We have also seen the conditions and ways to convert the MRDF and CSDF graphs into SRDF graphs. We have covered the theory related to these data-flow graphs, their properties and schedules. We would need these definitions in the further chapters. To summarize, we have presented the theoretical framework related to data-flow graphs which is important for the proofs that we provide in the later chapters.

TDM scheduler and data-flow based models: related work

3

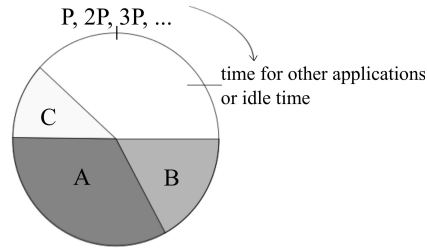
In case of SDR, meeting the timing requirements for each radio application has to be independent of the other applications running at that time. In order to achieve this goal, the tasks (of all applications running at any given moment) have to be scheduled in such a way that they do not affect the timing performance of each other. One way to achieve the desired timing independence is to use *budget schedulers*. A *Budget scheduler* [25] provides a fixed budget for each task in a fixed time interval. Time Division Multiplexed (TDM) is a basic type of budget scheduler, which provides certain budget to each application task by dividing a chosen time interval into multiple slots and allocating each slot to a single task.

In this chapter, we describe the basics of the TDM scheduler. We present the analytical expressions for the temporal behaviour of the TDM scheduler in various cases. We then present various data-flow based models that model the temporal behaviour of TDM scheduler and then finally come up with the requirements for a better model to represent the temporal behaviour of the TDM scheduler.

It is important to note that tasks may have variable execution times, depending upon the state of control flow instructions at that point of time. However, there is always an upper bound to the execution time of real time tasks. We make this assumption for real time tasks in our context. It is called *worst case execution time* (WCET). So, whenever we refer to execution time of a task, we are in-fact referring to its WCET. We thus, define task as a set of instructions grouped together, executing concurrently, to achieve certain functionality which may take variable amount of time to execute but will definitely have an upper bound, every time it executes. Thus, task execution time is the time spent between its finish and its dispatch. Thus, task execution time is variable. Actor execution time is constant and is equal to the WCET of task being modelled by the actor.

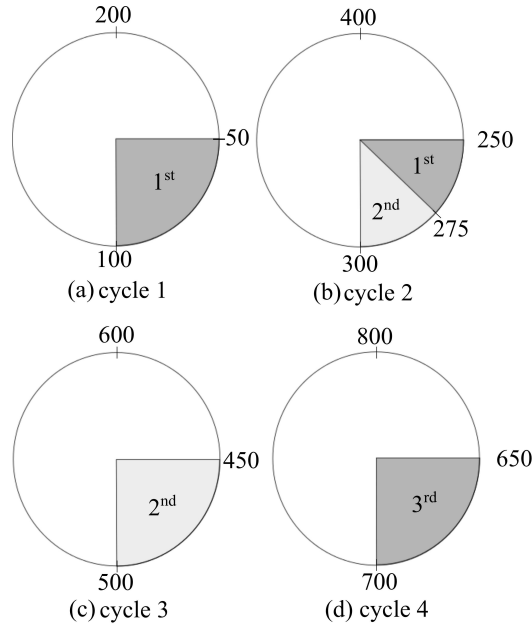
3.1 Description of TDM scheduler

Time Division Multiplexed (TDM) scheduler is a type of a pre-emptive budget scheduler. Each task is allotted a fixed time-budget called *slice* S on a computing resource from within a total available time-budget called *period* P . The slice sizes can be different for different tasks but the total period is constant for a resource. With TDM scheduler, a task can execute only within its slice. If the execution of the task is not finished until the end of the current slice, the task is pre-empted and can only continue during the next slice in the subsequent period. Thus, a task may require multiple slices to finish a single execution. The TDM scheduler is cyclic in nature and the scheduling process is repeated every period. Hence, the period is also referred as cycle time. Figure 3.1 shows a case where tasks A, B and C are mapped on the same resource. In this example, Task



A, B and C are three tasks scheduled on a TDM scheduler with period P . The shaded regions show the slices allotted to each of them

Figure 3.1: Example of TDM time wheel



S

Figure 3.2: Example of multiple executions of a task scheduled on a TDM scheduler

A has the biggest allocated slice and C has the lowest. The total period is constant and is represented by a circle to suggest a periodic behaviour. Task A, B and C are scheduled in the same sequence in every period.

In real-time streaming applications, tasks are invoked repetitively. Hence we assume that the tasks are iterative in nature. When a task finishes execution, it can begin next execution provided the data for the next execution is available and also if the slice time that was allocated to it has not expired. Figure 3.2 illustrates this with an example.

Let us assume that any task A with an execution time of 75 time-units is scheduled using a TDM scheduler with a period of 200 time-units on a processor. The slice time allocated to this task is 50 time-units. In 3.2(a), we can see that the slice is not sufficient for the task to finish its execution within its slice and hence it may get pre-empted. In the next period, the task finishes its execution at 275 time units (3.2(b)). Let us now

assume that the data dependencies for this task are met already so that it is ready to be scheduled again. Since, the slice is still available, the task again starts execution until the end of the slice at 300 time-units. The task is pre-empted and continues execution in the third period (3.2(c)), where the task finishes its second execution at time 500 units. This cycle continues if the data dependencies for the next execution of the task are met before reaching the finish time of the previous execution. This condition is formalized in the next section.

3.2 Definitions of terms

We use the following notations in this chapter.

$a \% b = a - (\lfloor \frac{a}{b} \rfloor)b$	definition of modulo operator
τ	the worst case execution time of the application actor
P	the period of the TDM scheduler in time units
$t(i)$	execution time of an actor i
S	the slice of the TDM scheduler in time units such that $S \leq P$
$s(i, k)$	the start time of $(k + 1)^{th}$ iteration of the an actor i
$f(i, k)$	the finish time of $(k + 1)^{th}$ iteration of an actor i
$a(k)$	arrival time of the $(k + 1)^{th}$ token at the input
$T(k)$	the arrival time of $(k + 1)^{th}$ execution of a task on a TDM scheduler
$F(k)$	the finish time of $(k + 1)^{th}$ execution of a task on a TDM scheduler
t_x	parametric constant for the multi-rate model, equal to 1 unless specified otherwise
G	the multi rate model
G_S	the single rate form of multi rate model
$G_{SU}(n)$	the n times unrolled single rate form of multi rate model
G_{SO}	the model for static ordering of actors

3.3 Formalization of the temporal behaviour of a TDM scheduler

A task can start executing as soon as the data dependencies for its execution are satisfied. The arrival time of the task is defined as time when all its data dependencies have been satisfied. Let T be the arrival time for a single execution of a task and F be the finish time for a single execution of a task. Let τ be its WCET. The beginning of its slice and arrival time of the task are two independent events. The task can be activated exactly at the beginning of the slice, at any moment within the slice or outside the slice. Let us assume that the slice is of S units and the TDM scheduler is of period P . We also define a parameter ϕ which determines the start time of the slice in the time wheel, relative to a arbitrarily chosen point marked as $\phi_0 = 0$. Hence, if the slice of time S units starts at any time ϕ , then duration of the slice is from ϕ to $(S + \phi)$. If the task is activated exactly at the start of the slice, then it will take $\lfloor \tau/S \rfloor$ complete slices and will take $\tau \% S$ time in the next slice, after a waiting of $(P - S)$ units, to complete one execution. $\lfloor \tau/S \rfloor$ complete slices also means $\lfloor \tau/S \rfloor$ complete waiting times of $(P - S)$. Thus, finish

time in this case is given by

$$\begin{aligned}
F &= T + \left\lfloor \frac{\tau}{S} \right\rfloor S + \left\lfloor \frac{\tau}{S} \right\rfloor (P - S) + (\tau \% S) \\
&= T + \left\lfloor \frac{\tau}{S} \right\rfloor P + (\tau \% S) + \\
&= T + \left\lfloor \frac{\tau}{S} \right\rfloor P + (\tau - \left\lfloor \frac{\tau}{S} \right\rfloor S) \\
&= T + \tau + \left(\left\lfloor \frac{\tau}{S} \right\rfloor - 1 \right) (P - S)
\end{aligned} \tag{3.1}$$

If the task is activated outside the slice, it has to wait till the beginning of the next slice. The waiting time in this case is given by $(\phi - T) \% P$. We take a modulus in order to avoid negative values of $(\phi - T)$. It will be added to finish time obtained in 3.1.

$$F = T + \tau + \left(\left\lfloor \frac{\tau}{S} \right\rfloor - 1 \right) (P - S) + (\phi - T) \% P. \tag{3.2}$$

If the task is activated within a slice, then the finish time is given either by 3.1 or by 3.3, depending upon the amount of time it executed in the first slice.

$$F = T + \tau + \left(\left\lfloor \frac{\tau}{S} \right\rfloor \right) (P - S). \tag{3.3}$$

To Summarize, the finish time for a single execution of a task scheduled using a TDM scheduler on a resource will have the finish time given by [14]

$$F = \begin{cases} T + \tau + \left(\left\lfloor \frac{\tau}{S} \right\rfloor \right) (P - S); & \text{for } (\phi + S - T) \% P < \tau - \left(\left\lfloor \frac{\tau}{S} \right\rfloor - 1 \right) S \\ T + \tau + \left(\left\lfloor \frac{\tau}{S} \right\rfloor - 1 \right) (P - S) + (\phi - T) \% P; & \text{for } (\phi - T) \% P \leq (P - S) \\ T + \tau + \left(\left\lfloor \frac{\tau}{S} \right\rfloor - 1 \right) (P - S); & \text{otherwise} \end{cases} \tag{3.4}$$

3.4 Periodic behaviour of TDM scheduler

A task can start executing only if its data dependencies are satisfied and its previous execution has finished (i.e. the resource is free in the slice allocated to the task). If we assume that the data dependencies of the task for next execution are always satisfied before the finish time of the previous execution and there are multiple such executions, then we can say that the finish time of all executions, except the first, will only depend upon the previous finish time. This condition is called a burst and the behaviour can be modelled as follows [13].

With the aforementioned assumption of multiple executions, consider a sequence of n consecutive executions of a task on a resource with a TDM scheduler. When one

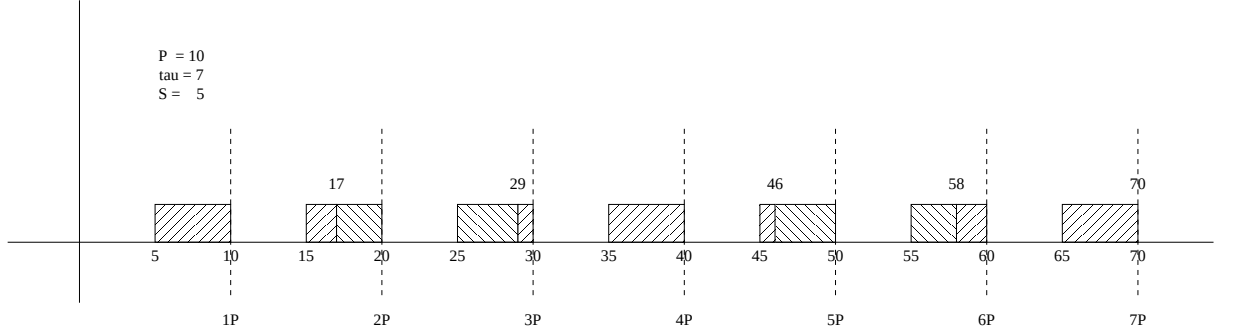


Figure 3.3: Example for periodic behaviour of TDM scheduler

execution finishes, the next one begins, and the slice allocated to this task is never wasted. If the slice is not wasted at all, the time taken for n executions is given by $n\tau$. Time $n\tau$ is chopped into slices and the number of slices is given by $\lfloor \frac{n\tau}{S} \rfloor$. Thus n executions will take $\lfloor \frac{n\tau}{S} \rfloor \cdot S$ full slices and $n\tau \% S$ time for the reminiscent execution, after a delay of $(P - S)$. If the end of slice matches the finish of the n^{th} execution, then $n\tau \% S = 0$. Thus, if f is the finish time for burst of n executions and s is the start time, then

$$F(n) = \begin{cases} T(0) + (\lfloor \frac{n\tau}{S} \rfloor)P + (P - S) + (n\tau \% S); & \text{for } n\tau \% S \neq 0 \\ T(0) + (\lfloor \frac{n\tau}{S} \rfloor)P; & \text{for } n\tau \% S = 0 \end{cases} \quad (3.5)$$

A burst is defined as a sequence of arrivals of a task meeting the criterion, $T(k) \leq F(k - 1)$, where $T(k)$ is the arrival of the $(k + 1)^{th}$ execution of the task. and $F(k - 1)$ is the finish time of the k^{th} execution of the task. We present the temporal behaviour of a system with TDM scheduler for a burst of large number of executions of a task with execution time τ . We consider first q executions of the task finishing in exactly r slices of slice time S . If there are q executions of a task in r slices of slice time S , then the time $q\tau$ is the total execution time and the total time spent in r slices is rS . The finish time of some q^{th} execution will match with finish time of some r^{th} slice. When they match, the time spent in q executions will be equal to r slices, since the task can only execute within the slice in a TDM schedule. Thus we can write,

$$\tau q = rS \quad (3.6)$$

We present an example in figure 3.3 to explain this idea. Here the period P is 10 units and slice S is 5 units. The task has an execution time τ of 7 units. We can see that there are 5 executions of the task marked by finish times $\{17, 29, 46, 58, 70\}$ in 7 slices, with the end of 7^{th} slice matching with the finish time of 5^{th} execution.

In fact, the time when the q^{th} execution matches with the finish time of the r^{th} slice will be nothing but the least common multiple of S and τ , denoted by $lcm(S, \tau)$. To find

the expressions for q and r , we write,

$$\tau\left(\frac{lcm(S, \tau)}{\tau}\right) = S\left(\frac{lcm(S, \tau)}{S}\right) \quad (3.7)$$

Comparing (3.6) and (3.7), we can say that

$$q = \frac{lcm(S, \tau)}{\tau} \text{ and } r = \frac{lcm(S, \tau)}{S} \quad (3.8)$$

From basic mathematics, we know that for two positive integers S, τ ,

$$lcm(S, \tau).gcd(S, \tau) = S\tau \quad (3.9)$$

Using (3.8) and (3.9), we can write another set of expressions for q and r .

$$q = \frac{S}{gcd(S, \tau)} \text{ and } r = \frac{\tau}{gcd(S, \tau)} \quad (3.10)$$

Furthermore, the start of r TDM periods also marks the beginning of a periodic behaviour. Since, we have one slice per TDM period, the r^{th} slice will be present in r^{th} period. For every start of the task execution after time rP and before $2rP$ the pattern of first q starts will repeat, but with an offset of rP . Thus,

$$T(0 + q) = T(0) + rP,$$

$$T(1 + q) = T(1) + rP$$

.

.

.

$$T(q - 1 + q) = T(q - 1) + rP.$$

In general, we can write,

$$T(k + q) = T(k) + rP; \quad k \geq q \quad (3.11)$$

This equation models the periodic behaviour of the TDM scheduler.

3.5 Data-flow models for TDM scheduler

In a data flow based approach of analysis of real time systems, each real time task is represented by an actor in the data flow graph. Every actor is modelled to have an execution time that is equal to its worst case execution time of the task being modelled by the actor. The edges between these actors represent the precedence constraints of these tasks. The input and output token rates represent the data that is shared between these tasks. The graph, thus generated, is called an application graph. The streaming application which is represented by an application graph has some latency and throughput requirements. When an actor of an application graph is scheduled on a processor running a TDM scheduler, the start time and finish time of the task is dependent upon the TDM scheduler. Data-flow modelling is a technique that is used extensively for tem-

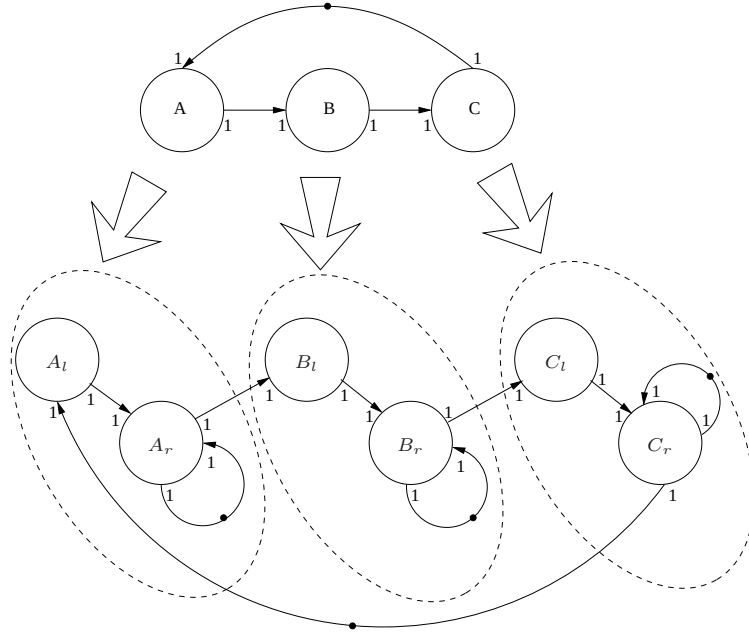


Figure 3.4: Example for Data-flow modelling

poral analysis of streaming applications. It can also be used to study the effect of TDM scheduler on the start and finish times of the tasks and thus on the overall application. The main idea is to use a model, which is also a data-flow component, that captures the impact on the start and finish times of a task because of it being scheduled on a processor using a TDM scheduler. If we have such a model, then each actor in the original application graph can be replaced by its equivalent model to create an analysis graph for the complete application graph and then its temporal behaviour can be analysed. The example of analysis graph is shown in Figure 3.4. In this example, the application graph of 3 actors A, B and C is converted into an analysis graph using Latency Rate model (discussed in section 3.5.1.2) for each of them.

3.5.1 Existing models for modelling TDM arbitration

We present the various models for the TDM arbitration in this section.

3.5.1.1 Single Actor Response Model

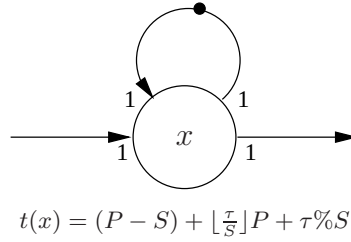


Figure 3.5: single actor response model

As discussed in [5], the single actor response model (Figure 3.5) converts an actor in the original application graph with worst case execution time τ into an actor whose execution time given by 3.12. Execution time of the model-actor represents the mapping of the original actor on a resource running a TDM scheduler with period P units. The slice allocated to this actor is of S units.

$$t(x) = (P - S) + \lfloor \frac{\tau}{S} \rfloor P + \tau \% S \quad (3.12)$$

Here, $(P - S)$ models the worst case waiting time and assumes that the actor is ready to execute just when the slice ends. The remaining terms in 3.12 model the number of slices required to complete the execution. If $a(k)$ is the arrival time of actor of the $(k + 1)^{th}$ input token and $f(x, k)$, $f(x, k - 1)$ are the finish times of actor x during $(k + 1)^{th}$, k^{th} executions respectively, then the max equation [4] for this model can be written as

$$f(x, k) = \max(a(k), f(x, k - 1)) + (P - S) + \lfloor \frac{\tau}{S} \rfloor P + \tau \% S \quad (3.13)$$

3.5.1.2 Latency Rate (LR) model

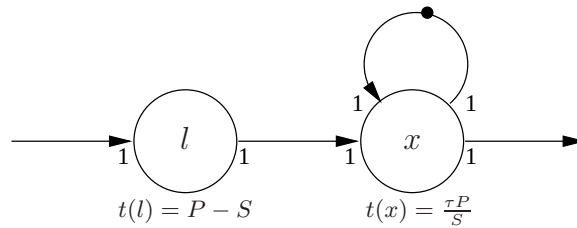


Figure 3.6: latency rate model

In LR model [29], the waiting time is modelled by a separate actor called *latency actor* having execution time $P - S$. Another actor called *rate actor* models the number of

slices required to complete the execution, once it has started execution.

$$\begin{aligned} t(l) &= (P - S) \\ t(x) &= \frac{\tau P}{S} \end{aligned} \quad (3.14)$$

If $a(k)$ is the arrival time of $(k + 1)^{th}$ token and $f(x, k), f(x, k - 1)$ are the finish times of actor x during $(k + 1)^{th}, k^{th}$ executions respectively, then the max equation for this model can be written as

$$f(x, k) = \max(a(k) + (P - S), f(x, k - 1)) + \frac{\tau P}{S} \quad (3.15)$$

The LR model is conservative [13] (explained later in Section 3.6.1) and it can be combined with static ordering of actors [20]. (explained later in Section 3.6.3).

3.5.1.3 Staschulat's LR model

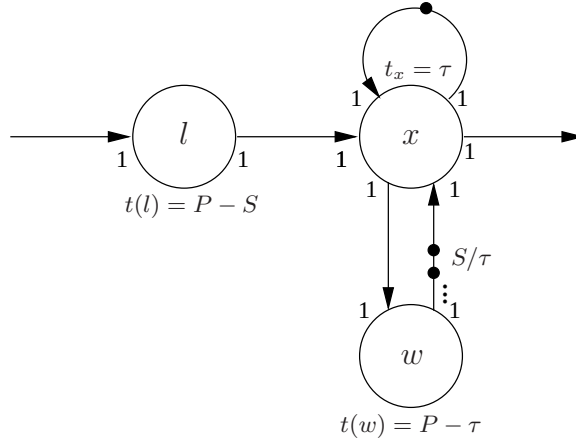


Figure 3.7: Staschulat's LR model

Staschulat's LR model [24] was presented in the context of arbitration of shared memory but it can be adapted in case of analysis model for TDM scheduler as well. The l actor models the initial latency of $P - S$. The actor x models the occupation of the resource for τ and w models the waiting time. The model is accurate and conservative [24] [13]. If $a(k)$ is the arrival time of $(k + 1)^{th}$ token and $f(x, k), f(x, k - 1)$ are the finish times of actor x during $k^{th}, k - 1^{th}$ execution respectively, then the max equation for this model can be written as

$$f(x, k) = \max \begin{cases} a(k) + (P - S) + \tau \\ f(x, k - 1) + \tau \\ f(x, k - \frac{S}{\tau}) + P \end{cases}$$

3.5.1.4 Latency cyclic rate (LCR) model

This model was proposed in [13]. The LCR model presents models the TDM scheduler behaviour in a slightly different way. We have seen earlier the periodic behaviour of the TDM scheduler for a large number of executions of a task. In this case, the periodic behaviour is captured by the equation 3.11. LCR models the behaviour for one complete cycle of q executions with the help of $q + 1$ actors. These actors have different execution times, and each actor models the finish time during each of the q^{th} execution. Since the number of actors are not fixed and are dependent upon value of q , we shall see an example for LCR model.

LCR model can be considered as a more general form of the Staschulat's LR model. Under the constraints of $S/\tau = n, n \in \mathbb{N}$, the LCR model looks similar to Staschulat's LR model.

3.6 Problem description

We have seen the theory for the TDM scheduler and various models for modelling the behaviour of the TDM scheduler for timing analysis. The prime requirements for such a model is that it should be conservative towards the actual start times and the actual finish times (explained later in Section 3.6.1), it should be less pessimistic towards (explained later in Section 3.6.2) and it should support static ordering (explained later in Section 3.6.3). We first define conservativity and pessimism in our context, and then explain static ordering. While discussing each requirement, we revisit the models to evaluate them against the requirement. Then, we come up with the problem statement for our work.

3.6.1 Conservativity of data-flow models

A data-flow model is nothing but a way to predict the start times and finish times of actor of an application graph when they are scheduled on a resource which is being managed by a TDM scheduler. It is important that the model predicts timings which are worse or equal to the actual worst case, we would not be able to provide a guarantee that the application meets the real time requirements. The conservativity is a property that covers such a behaviour [13]. It is defined as follows:

Definition: *A data-flow model is considered conservative if, for all actors, the modelled start times are greater than or equal to the actual start times and the modelled finish times are greater than or equal to the actual finish times.*

The single actor response model is conservative [5], [13]. Also, the LR and LCR model is conservative [13]. The Staschulat's LR model is accurate and conservative [24] [13].

3.6.2 Effect of pessimism

A model is said to be pessimistic if it predicts a finish time that greater than the actual finish time that could be achieved due to impact of TDM scheduler. When we represent all the actors in a application graph with their pessimistic models in order to analyse the temporal behaviour, the pessimism in each actor's model adds up giving us a high

overestimation of the timing behaviour of the entire application. In order to meet this overestimation, we need to provide more resources, which can be costly. Hence, accurate estimation is necessary. We will show this with an example.

Consider the example in Figure 3.4. Let us assume that all the three actors have been mapped on three different processors and each of them has a worst case execution time of 2 units, a slice time of 10 units has be allocated to all of them and period for all the three schedulers is 20 units. Let us also assume that interprocessor communication time is 1 unit. Now, the actual start times and finish times of the actors are unknown. Hence, the actual waiting time cannot be known. Hence, $(P - S)$ is considered as an upper bound on the waiting time. This value will be 10 units for all three models. Let us assume that the actor A fires just at the start of the slice such that actual waiting time is zero. Then, actor A shall finish at time = 2 units and B at 5 units and C and 8 units. But, as per the model, the finish time of A would be 14 units, B would be 28 units and C would be 42 units. So we can see that the pessimism in finish time for A is 12 units but for C it builds up to 34 units.

The single actor response model is highly pessimistic [19]. The high level of pessimism in the single actor response model is due to the fact that consecutive executions of the same actor have been modelled with a waiting time of $(P - S)$ which is not true. In cases when an actor is ready to fire before the end of the previous execution (i.e. bursty behaviour), the start time of next execution is immediately after the finish of the current execution. The waiting time in these cases can be zero. For streaming applications, bursty behaviour is the most important use case.

LR model [29] tries to improve the extent of pessimism by modelling the waiting time with latency actor. Rate actor models the number of slices required to complete the execution, once it has started execution. However, the LR model is also pessimistic, though less than the single actor response model. From Equation 3.14, we can see that the term $\frac{\tau P}{S}$ is a reason for pessimism, apart from the upper bound on waiting time $(P - S)$. For a TDM scheduler, $P \geq S$. Hence, the term $\frac{P}{S} \geq 1$ meaning τ will always be multiplied by a term which is greater than one, resulting into a value higher than τ as a execution time of actor x [13]. Hence, the predicted finish times by the LR model are always pessimistic with respect to the actual finish times.

LCR model and Staschulat's LR model are both less pessimistic than LR and single actor model.

3.6.3 TDM scheduler and static ordering

Static ordering is defined as an order of precedence in execution given to the actors mapped to same resource [19]. Usually each actor is allocated a dedicated slice on a resource, irrespective of whether all the tasks mapped on that resource belong to the same application or to different applications. If the actors mapped onto a resource belong to a same application, then we can say that these actors are mutually exclusive since only one of such actors can execute at a time. In this case, we can actually have a single slice serving all of them instead of separate slices for each one of them and statically ordering these actors. This optimization is suggested in [19] by *Moreira et al* and can lead to better utilization of the resources. Since static order is an order of precedence, we

can actually think of it as a schedule for actors belonging to the same application. We also expect that the TDM models should provide a way to represent the static ordering of actors. Static ordering of actors can be looked upon a static schedule for actors of an application graph that are mapped on the same processor.

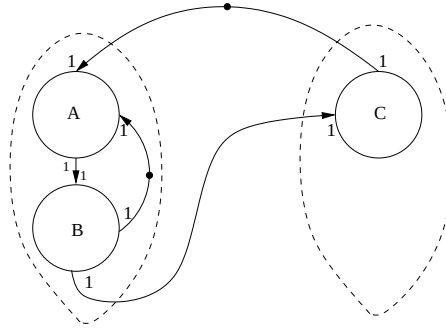


Figure 3.8: Clustering of statically ordered actors

Static ordering of actors is an important modelling criteria for scheduling actors on the same processor. Now, consider the same example of Figure 3.4. Let us now assume that actors A and B have been mapped onto same processor, while C is mapped on a different processor. If actors A and B are allotted different slices, then the analysis graph will look exactly like the Figure 3.4, since we do not have any information about relative positioning of two slices meant for actor A and B. In this case, the predicted finish times of A and B will be similar to those given in Section 3.6.2. However, if actor A and B are allotted the same slice, then actor B can start executing as soon as actor A finishes, without any waiting time of $P - S$ for actor B. Actors A and B execute one after the other, in a mutual exclusive way and hence these can be modelled as shown in Figure 3.8. The dotted lines show imaginary processor boundaries. The important thing to note in this figure is the extra back edge that connects from actor B to A, with a unit delay. This guarantees the mutual exclusivity of their execution.

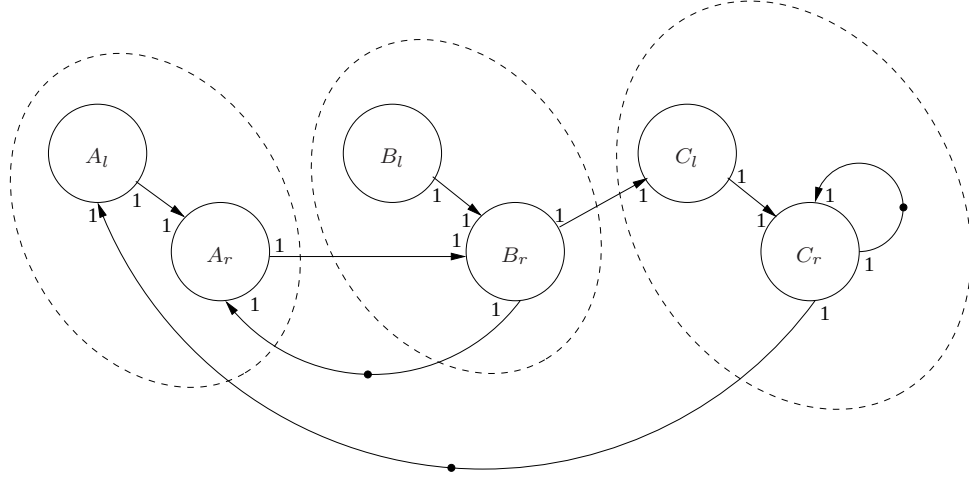


Figure 3.9: Static ordering using LR model

The LR model supports static ordering. Let us use the example mentioned above to show how LR model supports static ordering and how can static ordering of actors help reducing the pessimism in finish times. The analysis model based on LR model after static ordering actor A and B is shown in Figure 3.9. The latency actor B_l is not used. Instead, A_r is connected to B_r and an extra back edge is added from the B_r to A_r actor. This optimization helps in modelling the back to back execution of A and B without any waiting time for B. If we use the same example from section Section 3.6.2, the actual finish time would be 2 units for actor A, 4 units for actor B and 7 units for actor C while the predicted finish times would be 14 units for A, 18 units for B and 32 units for C. Thus, there is an appreciable reduction in the predicted finish times due to static ordering.

It is still unknown if LCR can support static ordering of actors.

3.6.4 Restrictive applicability of Staschulat's LR model

Staschulat's LR model cannot be used in certain cases. These cases are given below:

- The ratio S/τ has to be a integer, since the number of delay tokens can only be represented by integers. This means there are only few choices for either choosing S given a τ values.
- The model cannot be used to model cases where a smaller slice is allotted to a task with a large execution time τ i.e. $S < \tau$.

In practical applications, we may have cases where the worst case execution times of actors are order of magnitude higher than the slice time allotted to them. In such cases, this model cannot be used for modelling. Hence we have to use other models which are more pessimistic than this model in cases where $S \leq \tau$, which would again lead to over-estimation.

We find that either the current models are pessimistic, restrictive or do not support static ordering. We will see the effect of each of these in the following section. Having seen the problems with the existing models, the requirements for a new model can be listed down as per below:

- The application should not miss any timing deadlines. Hence, the model should be conservative.
- The model should accurately model the TDM behaviour or at least should be as accurate as the previous model, so that over-estimation of resources is as less as possible.
- The model should support all possible cases of relative values of actor execution times and slice times.
- The model should support static ordering of actors which are mapped on same processor and belong to the same application graph, so that we get lesser over-estimation of resources.

3.7 Summary

This chapter covers the basic theory about the TDM scheduler. It also presents the current data-flow models that have been proposed for modelling TDM arbitration. It also tries to capture various problems with the existing models and their side effects on the resource estimation. Finally, the chapter tries to present the need for a new and a better model for TDM arbitration and also lists down the requirements for the same.

Multi rate model for TDM scheduler

4

In the previous chapter, we have seen several models used for TDM arbitration along with their drawbacks. The LCR model presented earlier in section 3.5.1.4 models the consecutive periodic iterations of an actor using a SRDF graph. In this chapter, we propose a different approach for modelling TDM arbitration. This approach is based on modelling each execution of an application actor on a TDM-scheduled resource using the expressiveness of the MRDF graph. Depending on the execution time of an application actor, each execution of that actor is modelled using several tokens in a MRDF graph. Each token represents a fixed portion of the allotted slice. This model, since it is based on a MRDF graph, is called multi-rate model. Its first version was presented in [13].

In this chapter, we quickly review the original multi-rate model and then propose changes to the original multi-rate model in order to improve it for the case of static ordering, as explained later. We formalize the multi-rate model, its single rate equivalent form and its unrolled single rate equivalent form. These expressions are necessary since we use SRDF equivalent forms of the model in the later chapter to prove certain properties of the multi-rate model. We also formalize the static ordering of the actors.

4.1 definitions and conventions

We repeat the definitions from the previous chapter, for the sake of convenience and better readability.

$a \% b = a - (\lfloor \frac{a}{b} \rfloor)b$	definition of modulo operator
τ	the worst case execution time of the application actor
P	the period of the TDM scheduler in time units
$t(i)$	execution time of an actor i
S	the slice of the TDM scheduler in time units such that $S \leq P$
$s(i, k)$	the start time of $(k + 1)^{th}$ iteration of the an actor i
$f(i, k)$	the finish time of $(k + 1)^{th}$ iteration of an actor i
$a(k)$	arrival time of the $(k + 1)^{th}$ token at the input
$T(k)$	the arrival time of $(k + 1)^{th}$ execution of a task on a TDM scheduler
$F(k)$	the finish time of $(k + 1)^{th}$ execution of a task on a TDM scheduler
t_x	parametric constant for the multi-rate model, equal to 1 unless specified otherwise
G	the multi rate model
G_S	the single rate form of multi rate model
$G_{SU}(n)$	the n times unrolled single rate form of multi rate model
G_{SO}	the model for static ordering of actors

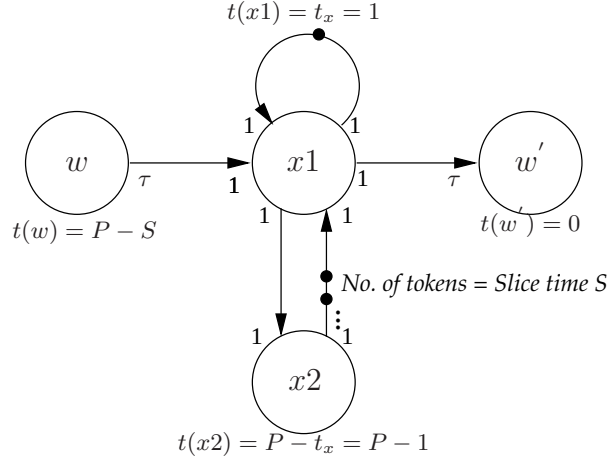


Figure 4.1: Initial multi rate model

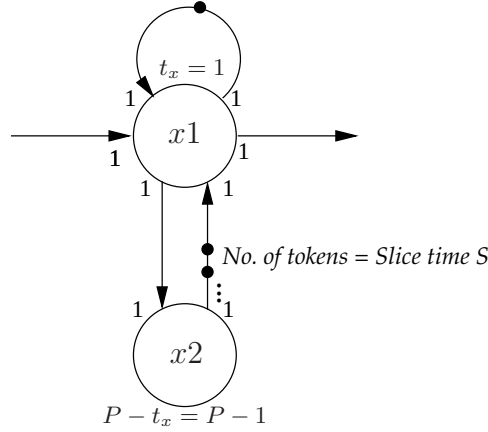


Figure 4.2: server wait emulator

4.2 Multi-rate model

The multi-rate model for TDM scheduling is as shown in Figure 4.1. Each actor in the application graph, which is mapped on a processor with a TDM scheduler, can be modelled using the multi-rate model. We are presenting it in this chapter so that it is easy to understand the modifications that we have suggested. The latency actor w models the worst case waiting time for every incoming token. It's execution time is $t(w) = P - S$ similar to the models discussed in the previous chapter. One firing of the w actor leads to the firing of τ tokens on the $(w, x1)$ edge. Actor $x1$, having a self-edge with a unit delay token and execution time t_x , acts as a server that provides a service in t_x increments or quanta of time. Parameter t_x is used a constant throughout this chapter and can be assigned different values. We use the value of t_x as 1 since it provides maximum granularity. The $(x2, x1)$ edge has S delay tokens and hence the

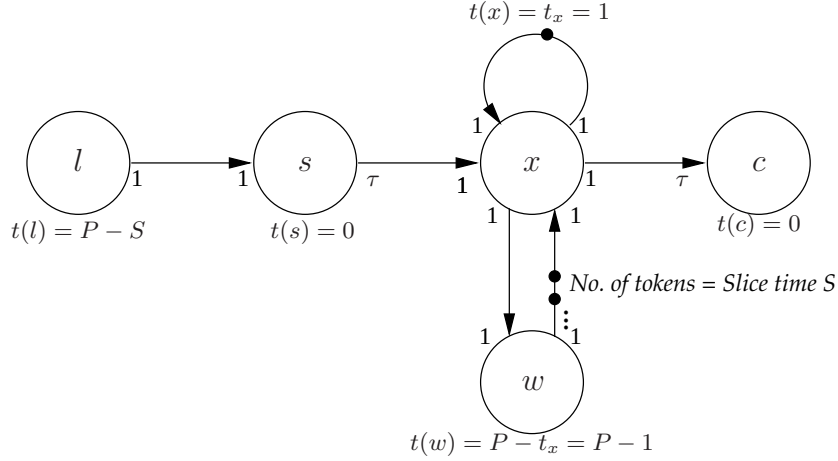


Figure 4.3: modified multi-rate model

service duration is S units of time. Every firing of the actor x_1 also fires the x_2 actor making a provision for its firing P units of time in the future, due to the generation of a token on the (x_2, x_1) edge, after $(P - t_x)$ units of time. Due to S initial delay tokens, there will be S firings of actor x_1 , and then the next set of S tokens shall be available only after P units of time. Actor x_1 cannot fire, till the time token is available on the (x_2, x_1) edge. Thus, with $t_x = 1$, the arrangement in Figure 4.2 models the service time of S units in total time of P units. Consumption of τ tokens lead to one firing of w' actor. Out of τ tokens fired by the w actor, S are processed by the x_1, x_2 pair in one cycle and those left, if any, are processed after P units of time. This cycle continues until all the τ tokens have been processed by the x actor. Once all the τ tokens are collected at the input of the w' actor. Actor w' accumulates τ tokens together before firing once. The end of a firing of w' indicates one execution of the application actor. We have renamed the actor names from the original work, based on their role in the model. Since the actor w models worst case latency, we rename it as *latency node* and represent it by letter l . Actor w' collects the τ tokens and hence we rename it as *collector node* and represent it by letter c . Similarly, Actor x , originally called x_1 , is renamed as *executing node*. Actor w , originally called x_2 , is renamed as *wait node*. We modify the model in Figure 4.1 by adding an extra actor. The modified multi-rate model is shown in Figure 4.3. The extra actor is called the *splitter node*, and it represented by s . It fires τ tokens per input token. Thus, the actor w in the Figure 4.1 has been replaced by two actors l and s in the Figure 4.3.

We have seen an example earlier in Section 3.5.1.2 in chapter 3, which explains the behaviour of LR model. We will use the same example to explain the multi-rate model. Let us consider the case when a burst of tokens arrives at the input of the actor, i.e., the next token arrives at the input of the actor before the finish of its previous firing. Formally, a bursty input is defined as $f(c, k - 1) + P - S \leq a(k)$.

Let $\tau = 7$ units, $S = 5$ units and $P = 10$ units. Let us also assume that the first token in the burst appears at time $t = 0$. Then actor l will fire at time $t = 0$ units

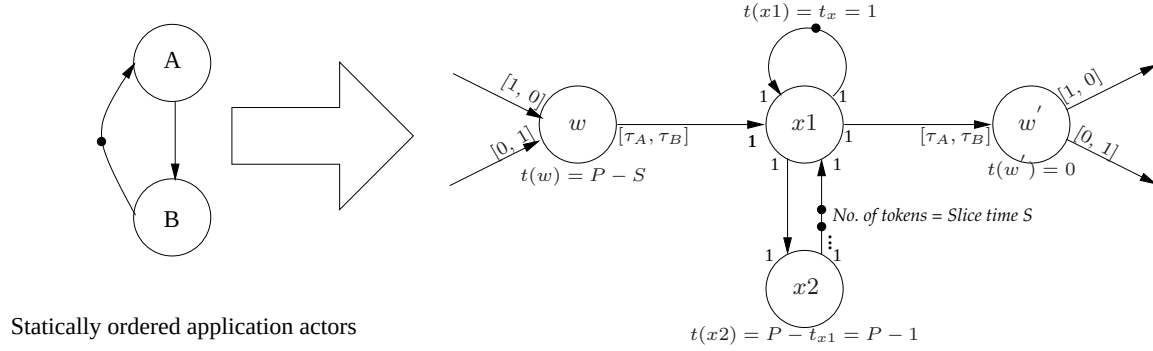


Figure 4.4: static ordering with multi rate model

and the finish the firing at $t = 10 - 5 = 5$ units. Actor s will fire at $t = 5$ producing τ tokens on its output edge. Actor x will fire for $S = 5$ times at $t = 6, 7, 8, 9, 10$ and then will be stalled. Actor w will fire at $t = 6, 7, 8, 9, 10$ producing output tokens at $t = 15, 16, 17, 18, 19$. Hence, actor x will again fire at $t = 15, 16$ producing output tokens at $t = 16, 17$. At $t = 17$, τ tokens are available at the input of actor c and actor c will fire at $t = 17$ producing the first output of the model at $t = 17$. If we continue this analysis further, we can see that the finish times are given by the set $\{17, 29, 46, 58, 70\}$. Thus, the actual finish times of the TDM scheduler match the finish times predicted by the model.

4.2.1 Static ordering with multi-rate model

The original work has proposed the following changes to their multi-rate model [13] to support static ordering of application actors. Actor w and w' are considered cyclo-static actors. The consumption rate of the incoming edges of actor w and the production rate of the outgoing edges of w' are adjusted to support the firing of actors in static order as shown in Figure 4.4. However, this arrangement has a problem. In this arrangement, a token from an actor in a static order to the next (immediate) actor in same static order, would also have to face the waiting time of $(P - S)$ i.e. actors that are mapped on the same processor are being modelled to face the initial waiting time of $P - S$ in spite of sharing the same slice. The concept of static order was designed with the basic idea that actors that are mapped on the same processor and share the same slice need not be modelled with the waiting time of $P - S$ [19]. Hence, the arrangement in the original proposal contradicts the requirement of having the static order of the actors mapped on the same processor. Also, If each application actor in the static order has multiple incoming edges or outgoing edges from/to other actors which are mapped onto other processors, we may get a number of incoming edges on the actor w and outgoing edges on the actor w' . It is difficult to distinguish edges belonging to a specific actor. Also, each incoming edge will have a cyclo-static consumption rate and each outgoing edge will have a cyclo-static production rate. To avoid these problems, we try to separate the cyclo-static behaviour from the incoming and the outgoing edges. Hence, we introduce an actor s , which takes care of the edges within the model, due to the static ordering.

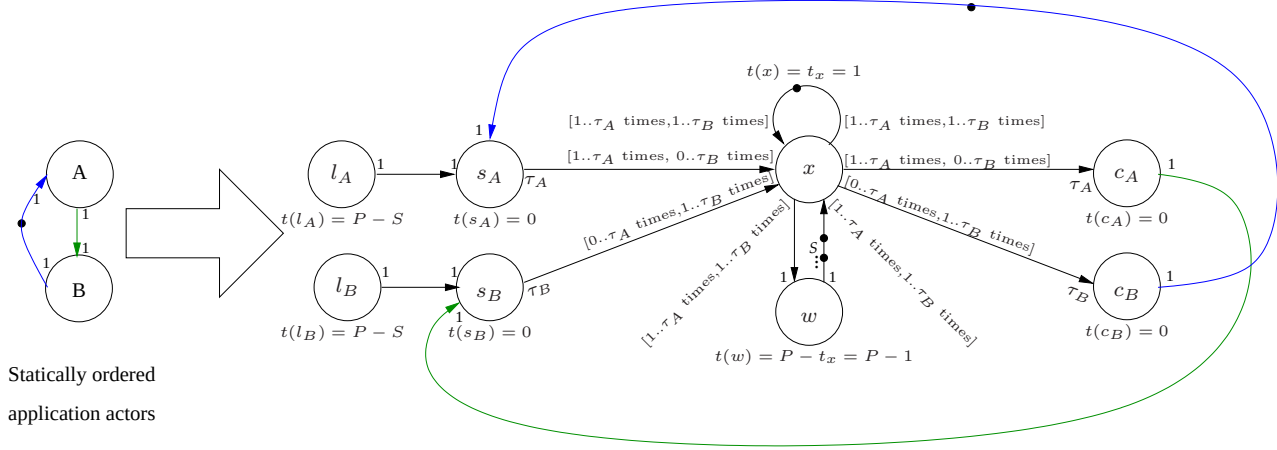


Figure 4.5: static ordering with modified multi rate model

Instead of having cyclo-static behaviour on actors like w and w' which act like an interface to the actors mapped on other processors, we propose it on actor x , which is internal to the model. The first τ_1 phases of x correspond to the first actor in static order, the next τ_2 phases correspond to second actor in static order and so on. due to this modification, incoming tokens from actors mapped on other processors towards the actors in the static order have to face the initial waiting time of $P - S$, but tokens coming from actors mapped on the same processor will not have to face this waiting time. Please refer to the Figure 4.5 for an example of two actors in a static order.

4.3 Composition of the the multi-rate model

A multi-rate model of an actor is a multi-rate graph $G(P, S, \tau) \equiv (V, E, d, t, prod, cons)$. V is the set of nodes called actors and it is denoted by

$$V = \{l\} \cup \{s\} \cup \{x\} \cup \{w\} \cup \{c\} \quad (4.1)$$

Each edge is a tuple $(source, destination)$ denoting the source and the destination actor of that edge. E is the set of edges and it is defined by

$$E = \{(l, s)\} \cup \{(s, x)\} \cup \{(w, x)\} \cup \{(x, c)\} \cup \{(x, x)\} \cup \{(x, w)\} \quad (4.2)$$

Execution time of the node in the timed multi-rate graph are given by

$$t(v) = \begin{cases} P - S & \text{for } v = l \\ t_x & \text{for } v = x \\ P - t_x & \text{for } v = w \\ 0 & \text{for } v = c \\ 0 & \text{for } v = s \end{cases} \quad (4.3)$$

where, $v \in V$

The number of initial tokens is denoted by $d(i, j)$ and defined as

$$d(i, j) = \begin{cases} 1 & \text{for } src = x \text{ and } dest = x \\ S & \text{for } src = w \text{ and } dest = x \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

where, $i, j \in V$.

The number of tokens produced along an edge is given by function $prod$ and it is defined as

$$prod(e) = \begin{cases} \tau & \text{for } e = (s, x) \\ 1 & \text{otherwise} \end{cases} \quad (4.5)$$

where, $e \in E$.

Also, the number of tokens consumed along an edge is given by function $cons$ and it is defined as

$$cons(e) = \begin{cases} \tau & \text{for } e = (x, c) \\ 1 & \text{otherwise} \end{cases} \quad (4.6)$$

where, $e \in E$

In order to analyse the multi-rate model for its conservativity and studying its properties, we will use the concepts that we have discussed in Chapter 2. Some of the properties can be only be studied for SRDF graphs (like MCM) while some others make analysis easy when they are applied to single rate data flow. Hence, we derive a formal way of converting a multi-rate model into its single rate equivalent form, based on the theory discussed in Section 2.2.4.1. Also, for certain proofs, we also need to unroll the single rate equivalent graph of the multi-rate model by some factor. Hence, we also derive formal expressions for the unrolled form of the single rate equivalent graph of the multi-rate model.

4.4 Formal expressions for the single rate equivalent of the multi-rate model

Multi-rate model is a MRDF graph with the repetitions vector given by

$$\vec{r} = \begin{bmatrix} r_l \\ r_s \\ r_x \\ r_w \\ r_c \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \tau \\ \tau \\ 1 \end{bmatrix}$$

Since, the multi-rate model has a valid repetitions vector, it can be converted into an equivalent SRDF graph. The equivalence is based on the fact that the k^{th} firing of i^{th} copy of an actor a in a SRDF equivalent graph with a total of r_a copies corresponds to $(r_a \cdot k + i)^{th}$ firing of that actor in the MRDF graph, where r_a is the entry corresponding the MRDF actor a in the repetitions vector. The single rate equivalent form of multi-rate model will have τ copies of x and w actors and a single copy of the other actors. The single rate equivalent form of a sample multi-rate model is shown in Figure 4.6. We use the example from Figure 4.6.

The timed single rate equivalent graph can be defined by the relation $G_S(P, S, \tau) = (V_s, E_s, d_s, t)$. Here, the set of actors V_s is defined as

$$V_s = \{x_i | 0 \leq i < \tau\} \cup \{w_i | 0 \leq i < \tau\} \cup \{l\} \cup \{s\} \cup \{c\} \quad (4.7)$$

The set of edges E_s is defined as

$$E_s = \{(s, x_i), (x_i, w_i), (w_{\tau+(i-S)\% \tau}, x_i), (x_i, c) | 0 \leq i < \tau\} \\ \cup \{(x_i, x_{i+1}) | 0 \leq i < \tau - 1\} \cup \{(x_{\tau-1}, x_0)\} \cup \{(l, s)\} \quad (4.8)$$

During conversion to SRDF, the S/t_x delay tokens on the (w, x) edge get distributed in a particular pattern, as described in Chapter 2. We can express the distribution of delay tokens along the various (w, x) edges as follows:

$$d(w_j, x_{(j+S)\% \tau}) = \begin{cases} \lfloor S/\tau \rfloor & \text{if } j < \tau - S\% \tau \text{ and } S \neq k\tau \\ (\lfloor S/\tau \rfloor) + 1 & \text{if } j \geq \tau - S\% \tau \text{ and } S \neq k\tau \\ \lfloor S/\tau \rfloor & \text{if } 0 \leq j \leq \tau \text{ and } S = k\tau \end{cases} \quad (4.9)$$

where,
 $0 \leq j < \tau$,
 $k \in \mathbb{N}$.

The equation (4.9) describes the edges between copies of w and x actor and it also describes how the delay tokens are distributed along these edges. The equation

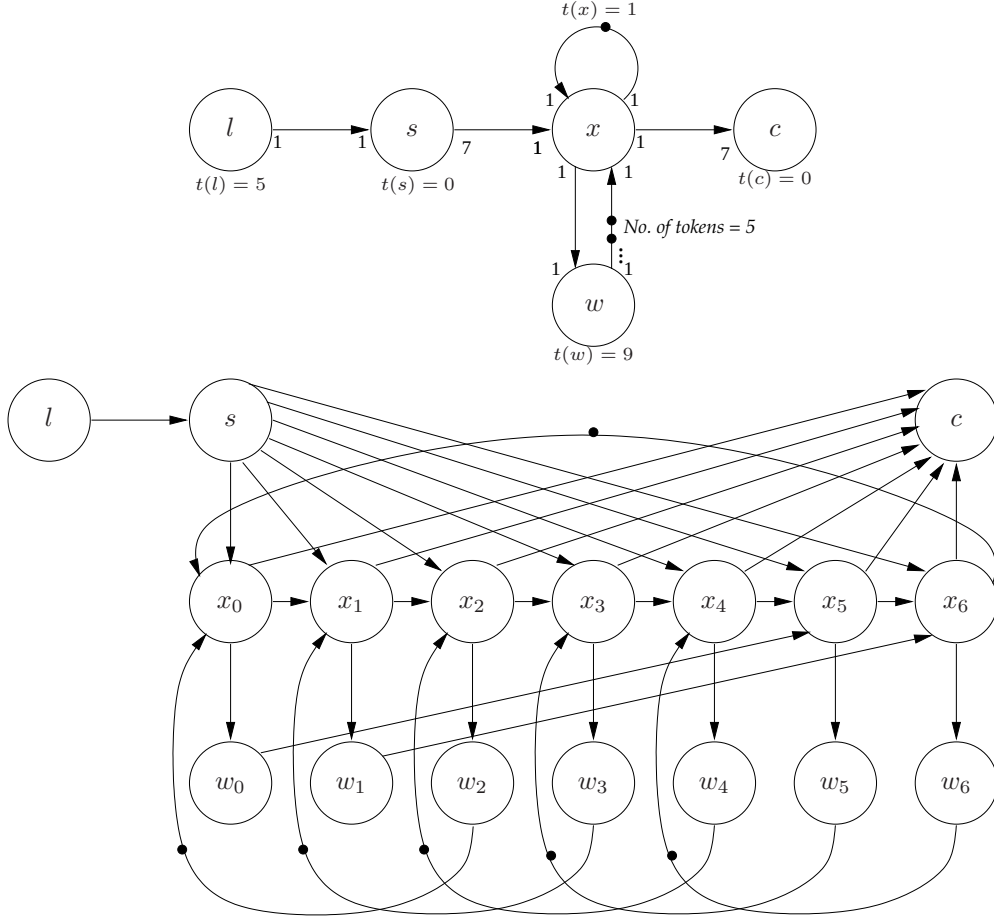


Figure 4.6: single rate form of a multi rate model

(4.9) can be written in a form where the indices of w actors are derived from those of x actors. Such an equation is given by equation (4.10).

$$d(w_{(i-S)\% \tau}, x_i) = \begin{cases} (\lfloor S/\tau \rfloor) + 1 & \text{if } i < S\% \tau S \neq k\tau \\ (\lfloor S/\tau \rfloor) & \text{if } i \geq S\% \tau S \neq k\tau \\ \lfloor S/\tau \rfloor & \text{if } 0 \leq i \leq \tau \text{ and } S = k\tau \end{cases} \quad (4.10)$$

where,
 $0 \leq i < \tau$,
 $k \in \mathbb{N}$.

Using equation (4.10), the delay along all the edges is defined as

$$d_s(src, dest) = \begin{cases} 1 & \text{for } src = x_{\tau-1} \text{ and } dest = x_0 \\ d(w_{(i-S)\% \tau}, x_i) & \text{for } src = w_{(i-S)\% \tau} \text{ and } dest = x_i \\ 0 & \text{otherwise} \end{cases} \quad (4.11)$$

where,
 $(src, dest) \in E_s$,
 $0 \leq i < \tau$,

4.5 Formal expressions for the unrolled SRDF equivalent graph of the multi-rate model

We need the unrolled SRDF equivalent graph in order to study the periodicity of the behaviour in a bursty mode. As per Section 2.4, If we unroll the timed single-rate graph by a factor q , we will get $q\tau$ copies of the w and x actors and a q number of l, s and c actors. The equivalence is based on the fact that the k^{th} firing of i^{th} copy of an actor a in an unrolled graph corresponds to $(q.k + i)^{th}$ firing of that actor in the SRDF graph. The timed unrolled graph can be denoted by a relation $G_{SU}(P, S, t) \equiv (V_{su}, E_{su}, d_{su}, q\tau)$. Here, the set of actors V_{su} is represented by

$$V_{su} = \{x_i | 0 \leq i < q\tau\} \cup \{w_i | 0 \leq i < q\tau\} \cup \{l_i | 0 \leq i < q\} \\ \cup \{s_i | 0 \leq i < q\} \cup \{c_i | 0 \leq i < q\} \quad (4.12)$$

The set of edges E_{su} is defined by

$$E_{su} = \{(x_i, w_i), (w_{((i-S)\% (q\tau))}, x_i) | 0 \leq i < q\tau\} \\ \cup \{(x_i, x_{(i+1)}) | 0 \leq i < (q\tau - 1)\} \cup \{(x_{(q\tau-1)}, x_0)\} \cup \{(l_i, s_i) | 0 \leq i < q\} \\ \cup \{(s_i, x_{i\tau}) | 0 \leq i < q\} \cup \{(x_{(i\tau-1)}, c_{i-1}) | 0 < i \leq q\} \quad (4.13)$$

The delays along the various (w, x) edges are given as

$$d_{su}(w_j, x_{(j+S)\% (q\tau)}) = \begin{cases} \lfloor S/(q\tau) \rfloor & \text{if } j < ((q\tau) - S\%(q\tau)) \text{ and } S \neq k\tau \\ (\lfloor S/(q\tau) \rfloor) + 1 & \text{if } j \geq ((q\tau) - S\%(q\tau)) \text{ and } S \neq k\tau \\ \lfloor S/(q\tau) \rfloor & \text{if } 0 \leq j \leq ((q\tau) - S\%(q\tau)) \text{ and } S = k\tau \end{cases} \quad (4.14)$$

where,
 $0 \leq j < (q\tau)$,
 $k \in \mathbb{N}$.

Using equation (4.14), the delay along for each edge is defined as

$$d_{su}(src, dest) = \begin{cases} 1 & \text{for } src = x_{q\tau-1} \text{ and } dest = x_0 \\ d_u(w_j, x_{(j+S)\% (q\tau)}) & \text{for } src = w_j \text{ and } dest = x_{(j+S)\% (q\tau)} \\ 0 & \text{otherwise} \end{cases} \quad (4.15)$$

where,
 $(src, dest) \in E_{su}$,

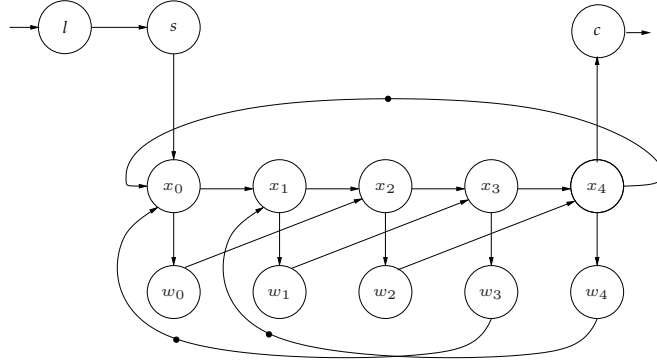
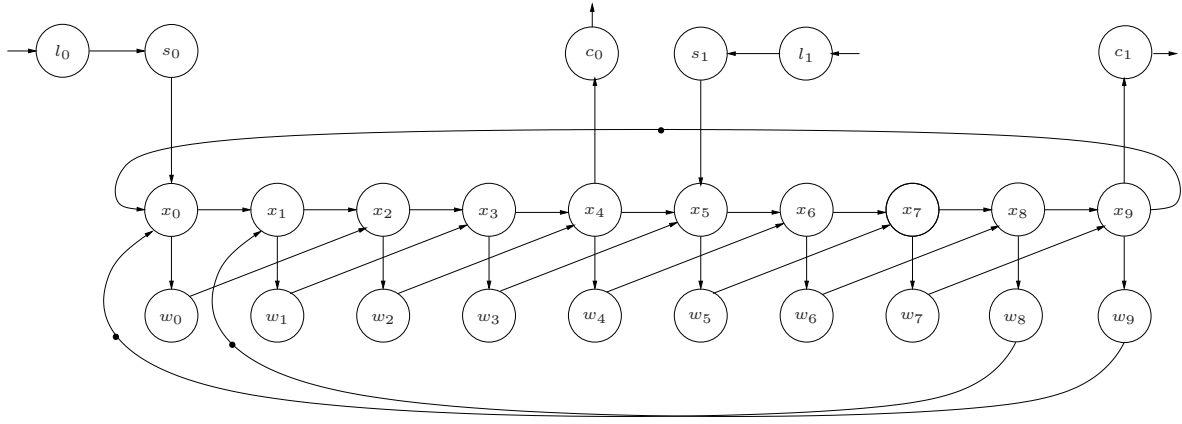
(a) SRDF equivalent graph for multi-rate model with $S = 2, \tau = 5$ (b) unrolled form of the SRDF graph in (a) with a unrolled factor of $q = 2$

Figure 4.7: Example for the unrolled form of a SRDF graph

$$0 \leq j < q\tau,$$

An example of conversion of SRDF graph into its a graph which is unrolled by a factor of q is shown in Figure 4.7. In this example, $S = 2$, $\tau = 5$ and unroll factor $q = 2$. There are $q \cdot \tau = 10$ copies of x and w actors.

4.6 Formal expressions for CSDF static order model

We have seen an example of static ordering in Figure 4.5. In this section we provide the formal expressions for modelling TDM arbitration of statically ordered actors. If there are h actors in a static order, then a model for TDM arbitration of a static order of h actors is modelled by a CSDF graph $G_{SO}(P, S, T) \equiv (V_{so}, E_{so}, d_{so}, t_{so}, prod_{so}, cons_{so})$.

V_{so} is the set of nodes called actors represented by

$$V_{so} = \{l_i | 1 \leq i \leq h\} \cup \{s_i | 1 \leq i \leq h\} \cup \{x\} \cup \{w\} \cup \{c_i | 1 \leq i \leq h\} \quad (4.16)$$

Due to the static ordering of h actors, there will be h copies of l, s and c actors. These are connected by edges in E_{so} . Each edge is a tuple (*source, destination*) indicating a source and a destination actor. E_{so} is the set of edges called arcs and is defined by

$$E_{so} = \{(l_i, s_i), (s_i, x), (x, c_i) | 1 \leq i \leq h\} \cup \{(c_i, s_{i+1}) | 1 \leq i < h\} \cup \{(c_h, s_1)\} \cup \{(w, x)\} \cup \{(x, x)\} \cup \{(x, w)\} \quad (4.17)$$

If τ_i is the execution time of an actor i in the static order and $T = \sum_{i=1}^h \tau_i$, then we define sequence (ordered set) Ω_T as a sequence of T ones in the following way:

$$\Omega_T = \langle n_i | n_i = 1, \forall i \in [1, T] \rangle \quad (4.18)$$

Similarly, we define sequence Ω_{τ_i} as a sequence of τ_i ones and Z_{τ_i} as the sequence of τ_i zeros. Formally, Ω_{τ_i} and Z_{τ_i} can be written as,

$$\Omega_{\tau_i} = \langle n_j | n_j = 1, \forall j \in [1, \tau_i] \rangle \text{ where, } 1 \leq i \leq h \quad (4.19)$$

$$Z_{\tau_i} = \langle n_j | n_j = 0, \forall j \in [1, \tau_i] \rangle \text{ where, } 1 \leq i \leq h \quad (4.20)$$

Using (4.18), we can write an expression for the execution time in the timed CSDF graph in the following way:

$$t(v) = \begin{cases} P - S & \text{for } v = l_i; 1 \leq i \leq h \\ \Omega_T & \text{for } v = x \\ P - 1 & \text{for } v = w \\ 0 & \text{for } v = c_i; 1 \leq i \leq h \\ 0 & \text{for } v = s_i; 1 \leq i \leq h \end{cases} \quad (4.21)$$

where, $v \in V_{so}$.

The initial tokens are denoted by d_{so} and are defined by

$$d_{so}(src, dest) = \begin{cases} 1 & \text{for } src = x, dest = x \\ S & \text{for } src = w, dest = x \\ 1 & \text{for } src = c_h, dest = s_1 \\ 0 & \text{otherwise} \end{cases} \quad (4.22)$$

where, $(src, dest) \in E_{so}$.

The number of tokens produced along an edge $e \in E_{so}$ is given by $prod_{so}$ and is defined by

$$prod_{so}(e) = \begin{cases} \tau_i & \text{for } e = (s_i, x) | 1 \leq i \leq h \\ \Omega_{\tau_1} \cup Z_{\tau_2} \cup Z_{\tau_3} \dots \cup Z_{\tau_h} & \text{for } e = (x, c_1) \\ Z_{\tau_1} \cup \Omega_{\tau_2} \cup Z_{\tau_3} \dots \cup Z_{\tau_h} & \text{for } e = (x, c_2) \\ Z_{\tau_1} \cup Z_{\tau_2} \cup \Omega_{\tau_3} \dots \cup Z_{\tau_h} & \text{for } e = (x, c_3) \\ \vdots & \\ \vdots & \\ \vdots & \\ Z_{\tau_1} \cup Z_{\tau_2} \cup Z_{\tau_3} \dots \cup \Omega_{\tau_h} & \text{for } e = (x, c_h) \\ \Omega_T & \text{for } e = (x, x) \\ \Omega_T & \text{for } e = (x, w) \\ 1 & \text{otherwise} \end{cases} \quad (4.23)$$

where, $e \in E_{so}$.

Also, the number of tokens consumed along an edge $e \in E_{so}$ is given by $cons_{so}$ and is defined by

$$cons_{so}(e) = \begin{cases} \tau_i & \text{for } e = (x, c_i) | 1 \leq i \leq h \\ \Omega_{\tau_1} \cup Z_{\tau_2} \cup Z_{\tau_3} \dots \cup Z_{\tau_h} & \text{for } e = (s_1, x) \\ Z_{\tau_1} \cup \Omega_{\tau_2} \cup Z_{\tau_3} \dots \cup Z_{\tau_h} & \text{for } e = (s_2, x) \\ Z_{\tau_1} \cup Z_{\tau_2} \cup \Omega_{\tau_3} \dots \cup Z_{\tau_h} & \text{for } e = (s_3, x) \\ \vdots & \\ \vdots & \\ \vdots & \\ Z_{\tau_1} \cup Z_{\tau_2} \cup Z_{\tau_3} \dots \cup \Omega_{\tau_h} & \text{for } e = (s_h, x) \\ \Omega_T & \text{for } e = (x, x) \\ \Omega_T & \text{for } e = (w, x) \\ 1 & \text{otherwise} \end{cases} \quad (4.24)$$

where, $e \in E_{so}$.

4.7 Summary

We have covered the preliminaries of the multi-rate model in this chapter. We have seen the formal expressions of the multi-rate model and its various forms. We will be using these forms in the next chapter for proving certain properties of the multi-rate model. We have also seen how multi-rate model can be modified to handle static ordering of the actors. We also formalized the modifications that we have suggested in the model for static ordering of the actors.

Temporal properties for multi-rate model

5

This chapter presents the proofs, properties and the modes related to the multi-rate model. In this chapter, we present the hypothesis for the correctness of the multi-rate model, based on the data-flow graph theory presented in Chapter 2, TDM scheduler theory presented in Chapter 3 and formal expressions presented in Chapter 4. We then prove that the multi-rate model is conservative towards the actual start and finish times of the actual TDM scheduler.

5.1 Definitions and conventions

We repeat the definitions from the previous chapter, for the sake of convenience and better readability.

$a \% b = a - (\lfloor \frac{a}{b} \rfloor)b$	definition of modulo operator
τ	the worst case execution time of the application actor
P	the period of the TDM scheduler in time units
$t(i)$	execution time of actor i
S	the slice of the TDM scheduler in time units such that $S \leq P$
$s(i, k)$	the start time of $(k + 1)^{th}$ iteration of the an actor i
$f(i, k)$	the finish time of $(k + 1)^{th}$ iteration of an actor i
$a(k)$	arrival time of the $(k + 1)^{th}$ token at the input
$T(k)$	the arrival time of $(k + 1)^{th}$ execution of a task on a TDM scheduler
$F(k)$	the finish time of $(k + 1)^{th}$ execution of a task on a TDM scheduler
t_x	parametric constant for the multi-rate model, equal to 1 unless specified otherwise
G	the multi rate model
G_S	the single rate form of multi rate model
$G_{SU}(n)$	the n times unrolled single rate form of multi rate model
G_{SO}	the model for static ordering of actors
$\lambda_F(a, b)$	longest forward path between actors a and b

In the multi-rate model, the l actor has no self-edge and it is the actor at the input side. This actor can fire as soon as a token arrives at its input. Hence, arrival time $a(k)$ of the k^{th} token at the input of the multi-rate model coincides with the start time of the latency actor $s(l, k)$. $a(k)$ and $s(l, k)$ are interchangeably used in this chapter. Using the same argument, the finish time of actor c marks the departure of a token.

5.2 Conservativity of the multi-rate model

Theorem 1: The multi-rate model is a conservative representation of the TDM scheduler i.e. $f(c, k) \geq F(k)$.

5.2.1 Outline of the proof

The multi-rate model of an actor is a MRDF graph. We have seen in Section 2.2.1 in Chapter 2 that a MRDF graph is said to be consistent or correctly constructed if it returns to its original token distribution after a finite number of firings of each actor. The number of firings of an actor in a MRDF graph after which, the token distribution returns to the initial state is equal to an entry corresponding to that actor in the repetitions vector for the graph. Thus, for a correctly constructed MRDF graph, the repetitions vector exists. The repetitions vector for the multi-rate model is given by Equation 4.7. If a MRDF graph is correctly constructed, it can be converted into an equivalent SRDF graph. We convert the multi-rate model into its equivalent SRDF graph G_S for the purpose of analysis. The reasons for this conversion are explained in Chapter 4 Section 4.3.

The amount of time taken by the multi-rate model to process an input token is called its response time. When a token appears at the input of the model, the response time of the model depends upon the current state of the delay tokens inside the model.

The model will have the lowest response time when the arrival time of the input token is such that all the delay tokens are in their original position so that they do not stall the input token at the input of any actor of the model. When the delay tokens do not arrive at the original position at the stipulated time, the input token gets stalled at the input of some actor and the response time of the model is higher. The condition when the input sees the lowest response time is called the impulse response. The condition that the impulse response is guaranteed to an input token when its arrival is $(P - S)$ units of time after the finish time of the previous token i.e. $a(k) \geq f(c, k - 1)$.

Input is said to be bursty if the rate of arrival of input tokens is higher than the time taken for the tokens to return to their original positions. If the start of a burst is at the time when the model is in steady state, with all the tokens already in their original positions, then the first token of the burst sees the lowest response time from the model. All the subsequent tokens will see a higher delay in the model as compared to the first. This is called the burst response. The condition for a guaranteed burst response is when the arrival of a new input token is $(P - S)$ units of time before the finish time of the previous token i.e. $a(k) + (P - S) < f(c, k - 1)$.

The Figure 5.1 gives an outline of how the proof is organised. The multi-rate model should be a correct and conservative representation of the TDM scheduler in all cases - cases when the input is bursty and when the input is an impulse.

5.2.1.1 Outline of the proof for bursty input

We have also seen in Section 2.3.3.2 in chapter 2 that a strongly connected, correctly constructed SRDF graph with fixed execution time for its actors enters a periodic regime after the initial K firings. The period is given by $N(G_S) \cdot \mu(G_S)$ units of time. The SRDF equivalent of MRDF graph, G_S would also enter a periodic regime after K_{G_S} number of firings. These K_{G_S} firings represent K_{G_S} different states of execution of the graph, which

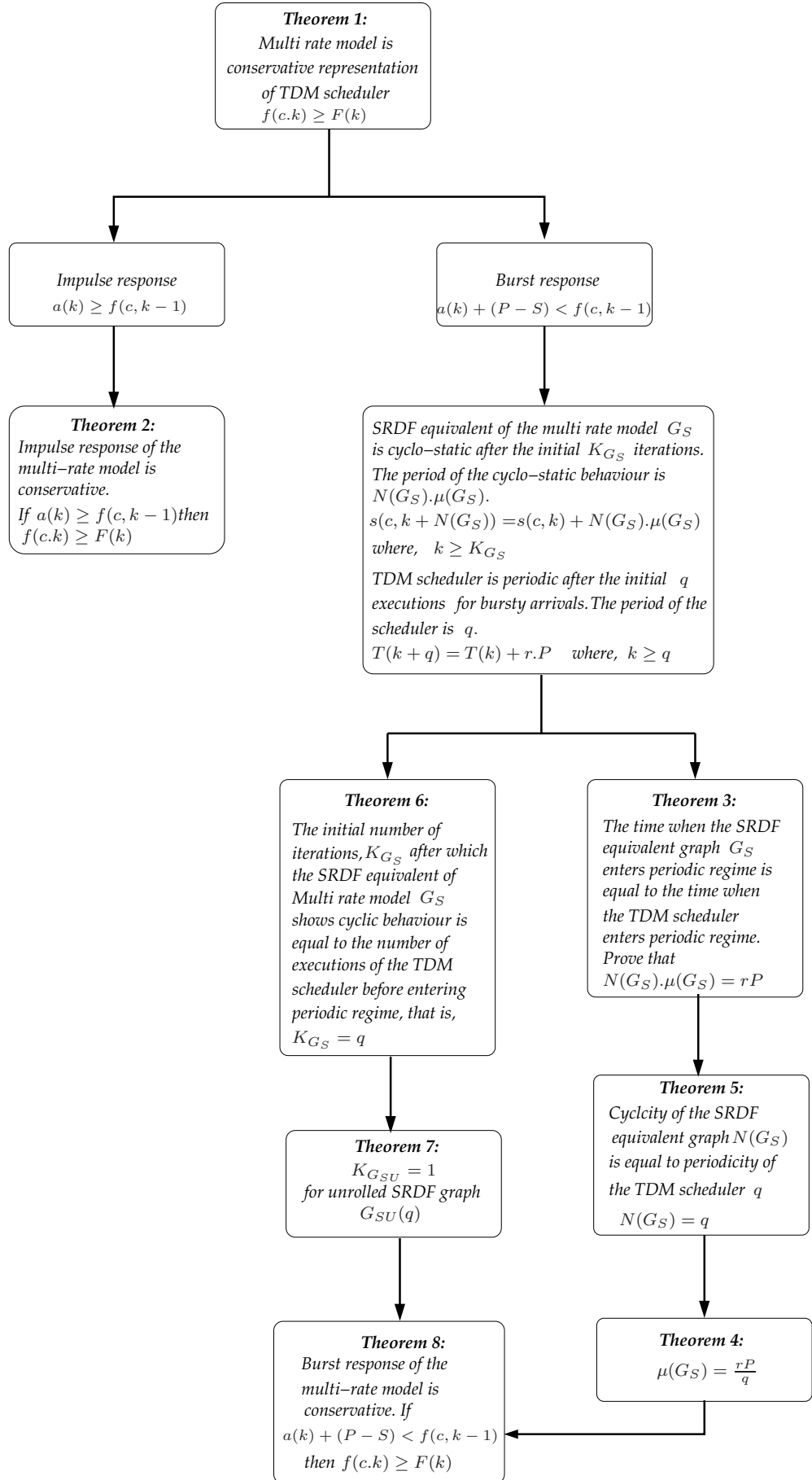


Figure 5.1: Proof overview

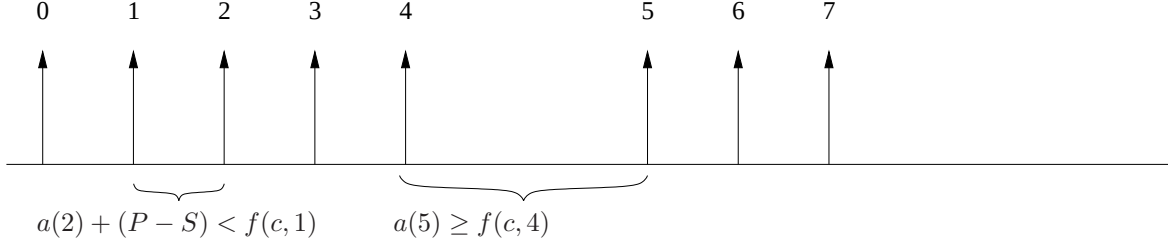


Figure 5.2: Delayed input token is the beginning of new burst

occur periodically. Such behaviour is also called cyclo-static behaviour. The cyclo-static behaviour can be formalized as

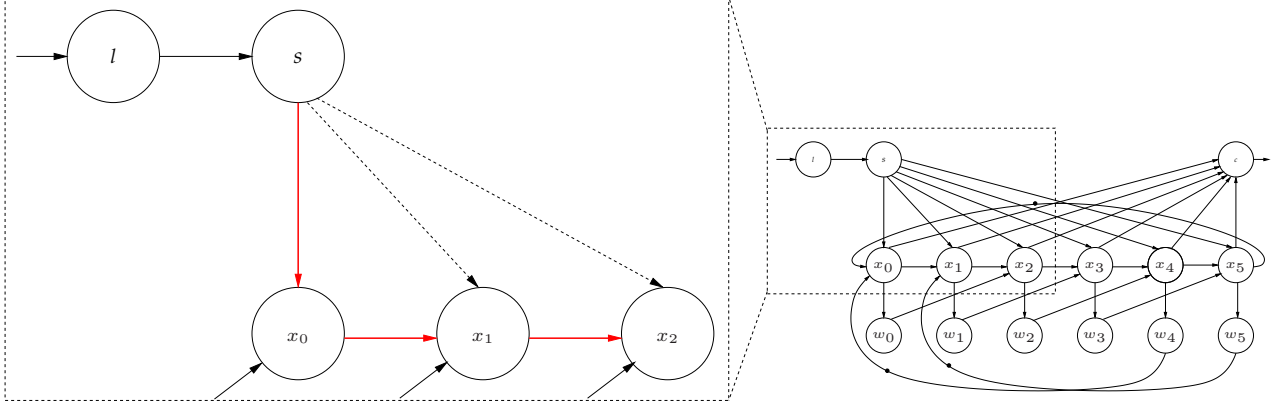
$$f(c, k + N(G_S)) = f(c, k) + N(G_S) \cdot \mu(G_S); \quad \text{for } k \geq K_{G_S} \quad (5.1)$$

We have seen in Section 3.4 in Chapter 3 that the TDM scheduler has a periodic behaviour. If we could prove that the period of the periodic regime of SRDF equivalent of multi-rate model and that of the TDM scheduler are same, then we can say that the long term periodic behaviour of the TDM scheduler is correctly represented by the model.

We have seen earlier in Section 3.4 in Chapter 3 that the periodic behaviour of the TDM scheduler is seen after q executions. On the other hand, the graph G_S would enter a periodic regime after K_{G_S} iterations. If we could prove that the K_{G_S} is equal to q of the TDM scheduler, and that the first K_{G_S} iterations of the G_S are conservative towards the finish times of the first q executions of the TDM scheduler, then we could say that the model is a conservative representation of TDM scheduler.

Thus, if we could prove that the model is conservative towards the finish time of the first q executions, then due to the periodic nature of the model and the TDM scheduler, both having the same period, we could say that the model is conservative towards finish times of all executions. Hence, we find the value of K_{G_S} of the graph G_S . For confirming the value of $K_{G_S} = q$, we unroll the graph G_S by a factor of q to obtain a graph $G_{SU}(q)$. We then prove that $K_{G_{SU}(q)} = 1$ for $G_{SU}(q)$. This is analogous to proving that $K_{G_S} = q$ for G_S . Having proved $K_{G_S} = q$, we provide the equations for the finish times of the first $K_{G_{SU}(q)}$ iterations of the graph based on the unrolled graph $G_{SU}(q)$. With the help of these equations, we show that the finish times provided by these equations are conservative.

To show that the period of G_S and that of the TDM scheduler is the same, we first need to find the period of the graph G_S . In order to find the period of G_S , we need to find the cyclicity $N(G_S)$ of the graph G_S and show that $N(G_S)$ is equal to q of the TDM scheduler. Then, we need to show that the MCM $\mu(G_S) = \frac{r \cdot P}{q}$ of the graph G_S . Once we have both $N(G_S)$ and $\mu(G_S)$, we have to show that the period of the graph $N(G_S) \cdot \mu_{G_S}$ is equal to period $\frac{r \cdot P}{q}$ of the TDM scheduler.

Figure 5.3: redundant (s, x) edges

5.2.1.2 Outline of the proof for impulse response

If the multi-rate model is in steady state and an input appears at the input of the model, then it faces minimal delay inside the model.

If the model is processing a burst (input tokens arriving such that $a(k) + (P - S) < f(c, k - 1)$) and one of the input tokens gets delayed such that its arrival after the finish time of the previous input token $a(k) \geq f(c, k - 1)$, then all the delay tokens would have returned to their original position. The delayed token, thus faces a minimal delay inside the model which is equal to lowest response time of the multi-rate model, like the first token of the burst and marks the beginning of the new burst, starting with this token (please refer to the Figure 5.2).

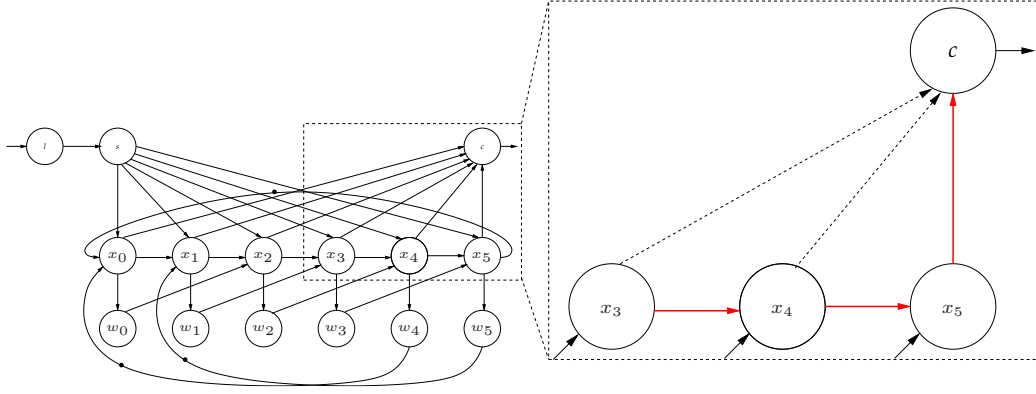
Both the cases mentioned above see the minimal delay inside the model. We have to find an expression for impulse response and prove that the impulse response is conservative.

5.2.2 Redundant edges in the graph G_S

Before starting with the actual proofs, we present the simplifications that we have used in our analysis. In the graph G_S , certain dependencies can never be critical (longer in terms of path lengths) and hence these can be ignored for temporal analysis. The edges which are showed as dotted in 5.3 and 5.4 can never be critical as compared to the bold red edges. We call these dotted edges as redundant edges. It is explained in detail below.

Consider the single rate equivalent of the multi rate model, G_S . Referring to figures 5.3, 5.4, the edges (s, x_i) , (x_j, c) where $i \neq 0, j \neq (\tau - 1)$ can be considered redundant since there exists a path from actor s to any actor x_i and from any actor x_j to actor c which is longer than the direct path (s, x_i) or (x_j, c) . The longer path is shown in red colour. The longer path between actors s and x_i can be described as $\max(0, t(x).i) = i$ and the path longer between actors x_j and c can be described as $\max(0, \tau - 1 - j) = \tau - 1 - j$.

Thus, we can remove all edges (s, x_i) where, $i \in [1, (\tau - 1)]$ and (x_j, c) where, $j \in$

Figure 5.4: redundant (x, c) edges

$(0, (\tau - 2)]$ for the purpose of our analysis. We use this fact in the analysis and proofs hence forth and ignore the edges (s, x_i) and (x_j, c) where $i \neq 0, j \neq (\tau - 1)$.

5.2.3 Impulse response

In this section, we find the finish time of the multi-rate model when a single token or the first token of a burst appears at its input. We have seen earlier that the impulse response of the model is the lowest possible response time. We first try to find the response time in this case and then prove that the finish time obtained is conservative.

5.2.3.1 Impulse response equation

In the multi-rate model, the l actor has no self-edge and it is the actor at the input side. This actor can fire as soon as a token arrives at its input. Hence, arrival time $a(k)$ of the k^{th} token at the input of the multi-rate model is nothing but the start time of the latency actor $s(l, k)$. Using the same argument, the finish time of actor c marks the departure of a token.

Consider the single rate equivalent of the multi rate model G_S . Assume that the model is in steady state, that is, all its delay tokens are in their initial position before an input token appears. We also assume that the redundant edges are not considered for analysis as per the argument presented in Section 5.2.2. We have seen in Equation 4.11 that the delays along (w, x) edges are either $\lfloor \frac{S}{\tau} \rfloor$ or $1 + \lfloor \frac{S}{\tau} \rfloor$. Also, there is a token on the edge $(x_{\tau-1}, x_0)$. When a token arrives at the input of the graph G_S , it will either pass through without getting stalled if all the tokens are in their original position or it will stall due to absence of a token on some edge. The first token will always pass through when the graph is in steady state. The departure time of a token will, however, depend upon the longest forward path from the actor x_0 to actor $x_{\tau-1}$.

If $\lfloor \frac{S}{\tau} \rfloor > 0$ or $S \geq \tau$ then there is at least one token along all (w, x) edges (refer to Figure 5.5). Since all delay tokens are in their original position, the edges with delays can be ignored. Hence, we can represent the graph G_S in form of a directed acyclic graph (DAG) after removing the edges with delays as shown in Figure 5.6. As per the DAG

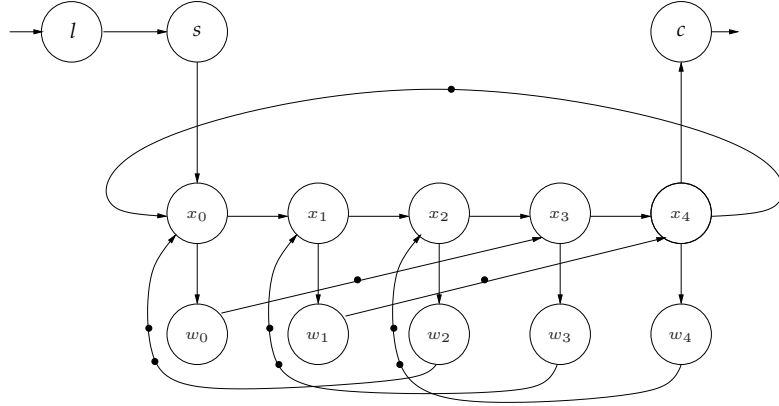
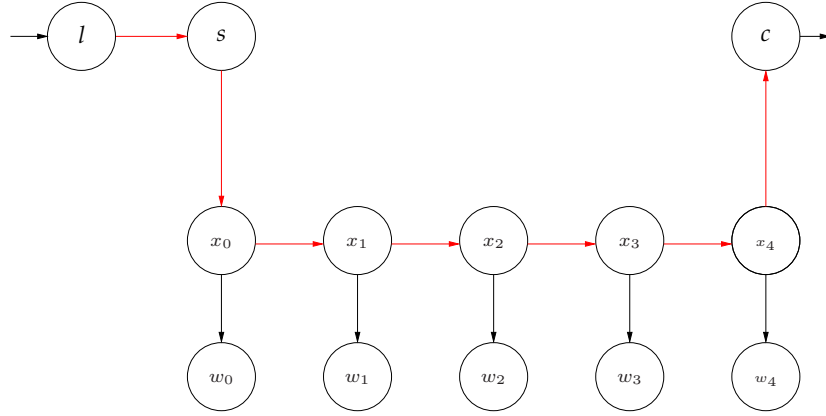
Figure 5.5: Impulse response example for $\lfloor \frac{S}{\tau} \rfloor > 0$ 

Figure 5.6: DAG for Figure 5.5 showing longest forward path

in Figure 5.6, the longest forward path from the actor x_0 to actor $x_{\tau-1}$ in the DAG is equal to τ .

Similarly, if $\lfloor \frac{S}{\tau} \rfloor = 0$ or $S < \tau$ (the case shown in Figure 5.7), there are two paths from any actor x_j to the actor $x_{(j+S)\% \tau}$ - one with path length $(S\% \tau) \cdot t(x)$ and the other with path length P . Since $\max((S\% \tau) \cdot t(x), P) = \max(S, P) = P$, the longest path from an actor x_j to actor $x_{(j+S)\% \tau}$ is through actor w_j . The Figure 5.8 shows the two paths. The longest path from the actor x_0 to the actor $x_{\tau-1}$ will be through alternate $x - w$ pairs. When we traverse along the longest path thus obtained, we take every $(S\% \tau)^{th}$ $x - w$ actor pair, until we reach an x actor which is one of the last S x actors. Also, there are in all τ number of x actors. Hence, one factor in the total path length from x_0 to $x_{\tau-1}$ is $\lfloor \frac{\tau}{S\% \tau} \rfloor \cdot P$. For every $(S\% \tau)^{th}$ $x - w$ actors traversed in this way, we miss

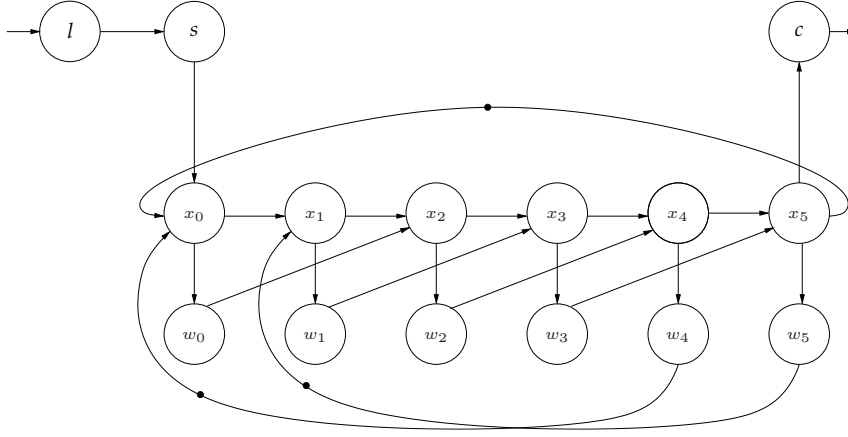
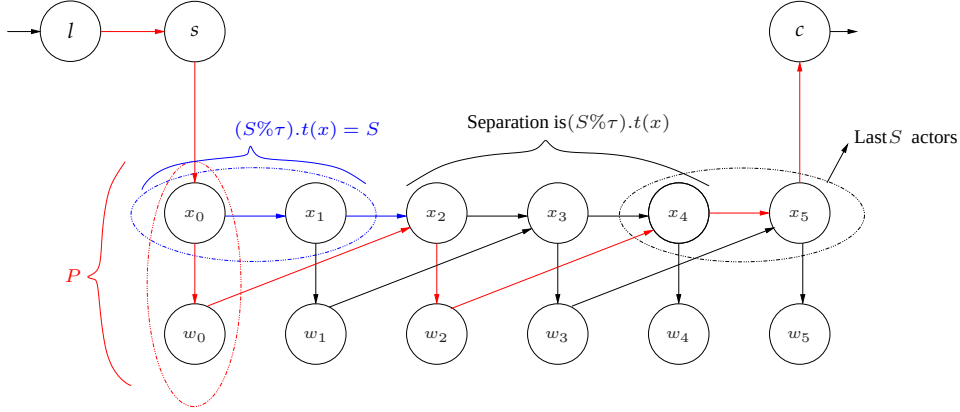
Figure 5.7: Impulse response example for $\lfloor \frac{S}{\tau} \rfloor = 0$ 

Figure 5.8: DAG for Figure 5.7 showing longest forward path

$(S\% \tau)$ x actors in between (shown as separation in Figure 5.8).

After reaching one of the last S x actors, the path traversal should only be along the x actors until the $x_{\tau-1}$ actor is reached. Thus, the path length along last $(S\% \tau)$ x actors is given by the difference between the total number of x actors and the number of x actors missed while traversing the path along the $x-w$ pairs multiplied by the execution time of actor x i.e. $(\tau - \lfloor \frac{\tau}{S\% \tau} \rfloor \cdot S) \cdot t(x)$. Hence total path length for the longest path in the DAG becomes $\lfloor \frac{\tau}{S\% \tau} \rfloor \cdot P + (\tau - (\lfloor \frac{\tau}{S\% \tau} \rfloor \cdot S)) \cdot t(x)$. To summarize, if $t(x) = t_x = 1$, then the longest forward path λ_F in the DAG from the actor l to actor c is given by,

$$\lambda_F(s, c) = \begin{cases} P - S + \tau & \text{if } S \geq \tau \\ P - S + \lfloor \frac{\tau}{S\% \tau} \rfloor \cdot P + (\tau - (\lfloor \frac{\tau}{S\% \tau} \rfloor \cdot S)) & \text{if } S < \tau \end{cases} \quad (5.2)$$

The finish time for the first token is nothing but its arrival time added to the $\lambda_F(s, c)$ given in Equation 5.2. Thus, the impulse response of the model is given by

$$f(c, k) = s(l, k) + \lambda_F(s, c) \quad (5.3)$$

where, $k = 0$ for a single token or $k \in \mathbb{N}$ for a delayed input during a burst.

5.2.3.2 Proof for the impulse response

We have to prove that the Equation 5.3 provides a conservative estimate of the finish time during an impulse.

Theorem 2: The impulse response of the SRDF equivalent graph of the multi-rate model is conservative i.e. $f(c, k) \geq F(k)$ if $a(k) \geq f(c, k - 1)$.

We first have to prove that if the condition for the impulse response is true, then the input token faces minimum amount of delay. Then, for a given arrival time $a(k)$ for the $(k + 1)^{th}$ token, the finish time predicted by the multi-rate model is greater than or equal to the actual finish time of the $(k + 1)^{th}$ execution of the TDM scheduler. The finish time of the multi-rate model is the finish time of actor c . Hence we have to prove that $f(c, k) \geq F(k)$ when the input is the impulse i.e. $(a(k) \geq f(c, k - 1))$.

Proof: We divide the proof into two cases, depending upon the relation between S and τ . In both the cases, we show that the Equation 5.3 is conservative.

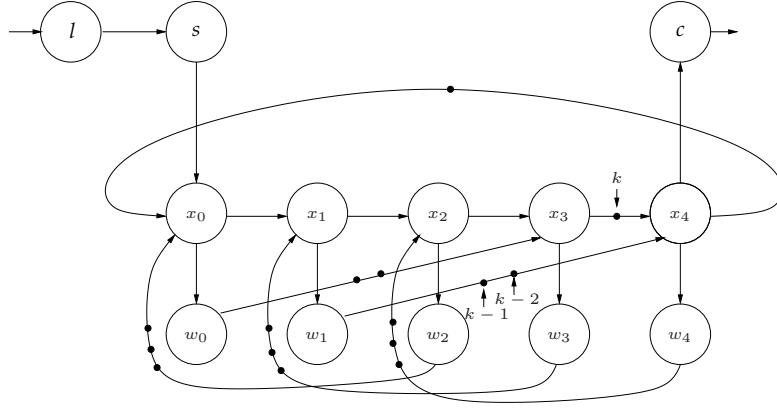
Case 1: $S \geq \tau$

In Section 5.2.1.2, we have seen that the tokens along the (w, x) edges should be in their original position so that the current iterations face minimal delay. There are at least $\lfloor \frac{S}{\tau} \rfloor$ tokens along the (w, x) edges. Thus we have to prove that, when $S \geq \tau$ and $a(k) > f(c, k - 1)$, these $\lfloor \frac{S}{\tau} \rfloor$ tokens due to the previous $\lfloor \frac{S}{\tau} \rfloor$ iterations are in their initial position for the next $\lfloor \frac{S}{\tau} \rfloor$ iterations. Let us assume that we have a steady state.

Now, consider the example in the Figure 5.9. For the k^{th} token along the (x_3, x_4) edge due to the k^{th} arrival, the $(k - 2)^{th}$ token should be available. Hence, we consider that the $k - \lfloor \frac{S}{\tau} \rfloor$ be the first token to arrive. Let the arrival of the $(k - \lfloor \frac{S}{\tau} \rfloor)^{th}$ be the first token to arrive. This token shall pass through without facing any stall. Then, we can write

$$f(c, k - \frac{S}{\tau}) = a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + \tau \quad (5.4)$$

Now going back to the example in Figure 5.9, the time of arrival of this token should be such that it consumes the next token along the (w, x) edges before replenishment. Hence, in general, all input tokens, starting with $(k - \lfloor \frac{S}{\tau} \rfloor + 1)^{th}$ token up to $(k - 1)$ token, should consume tokens before their replenishment. If this happens, the finish time for

Figure 5.9: Example for impulse response proof with $S \geq \tau$

the next input token can be written with the help of Equation 5.4 as

$$\begin{aligned}
 f(c, k - \lfloor \frac{S}{\tau} \rfloor + 1) &= \max(a(k - \lfloor \frac{S}{\tau} \rfloor + 1) + P - S, f(c, k - \lfloor \frac{S}{\tau} \rfloor)) + \tau \\
 \Leftrightarrow f(c, k - \lfloor \frac{S}{\tau} \rfloor + 1) &= \max(a(k - \lfloor \frac{S}{\tau} \rfloor + 1) + P - S, a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + \tau) + \tau \\
 \Leftrightarrow f(c, k - \lfloor \frac{S}{\tau} \rfloor + 1) &= a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + 2\tau
 \end{aligned} \tag{5.5}$$

In general, for the next y input tokens (where, $0 < y < \lfloor \frac{S}{\tau} \rfloor$), we can write the expression for finish time as

$$f(c, k - \lfloor \frac{S}{\tau} \rfloor + y) = a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + (y + 1)\tau \quad \text{where, } 0 < y < \lfloor \frac{S}{\tau} \rfloor \tag{5.6}$$

Now, even if all the previous finishes have consumed the tokens before replenishment, we have to show that the replenishment has happened when the k^{th} token arrives along the edge (x_3, x_4) in our example, when $a(k) \geq f(c, k - 1)$ is satisfied. Now, with the help of Equation 5.6 and $y = \lfloor \frac{S}{\tau} \rfloor - 1$, we can write

$$\begin{aligned}
 a(k) &\geq f(c, k - 1) \\
 \Leftrightarrow a(k) &\geq a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + \lfloor \frac{S}{\tau} \rfloor \cdot \tau
 \end{aligned} \tag{5.7}$$

For the k^{th} token, we have to show that

$$f(w_{(j-S)\% \tau}, k - \lfloor \frac{S}{\tau} \rfloor) \leq f(x_{j-1}, k) \quad \text{where, } S \% \tau \leq j < \tau \tag{5.8}$$

The finish times in Equation 5.8 can be written in terms of the arrival times as given in Equations 5.9 and 5.10.

$$f(w_{(j-S)\% \tau}, k - \lfloor \frac{S}{\tau} \rfloor) = a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + (j - S) \% \tau + P \tag{5.9}$$

$$f(x_{j-1}, k) = a(k) + P - S + j \quad (5.10)$$

Substituting the values of Equations 5.9 and 5.9 in Equation 5.8, we get,

$$a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + (j - S)\% \tau + P \leq a(k) + P - S + j \quad (5.11)$$

Using Equation 5.7, we can write 5.11 as

$$\begin{aligned} a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + (j - S)\% \tau + P &\leq a(k - \lfloor \frac{S}{\tau} \rfloor) + P - S + \lfloor \frac{S}{\tau} \rfloor \cdot \tau + P - S + j \\ \Leftrightarrow (j - S)\% \tau + P &\leq \lfloor \frac{S}{\tau} \rfloor \cdot \tau + P - S + j \quad \text{where, } S\% \tau \leq j < \tau \\ \Leftrightarrow (j - S)\% \tau + S - \lfloor \frac{S}{\tau} \rfloor \cdot \tau &\leq j \\ \Leftrightarrow (j - S)\% \tau + S\% \tau &\leq j \\ \Leftrightarrow j &\leq j \end{aligned} \quad (5.12)$$

Thus, we prove that, tokens along (w,x) edges will be available just at the right time so that the k^{th} token does not stall.

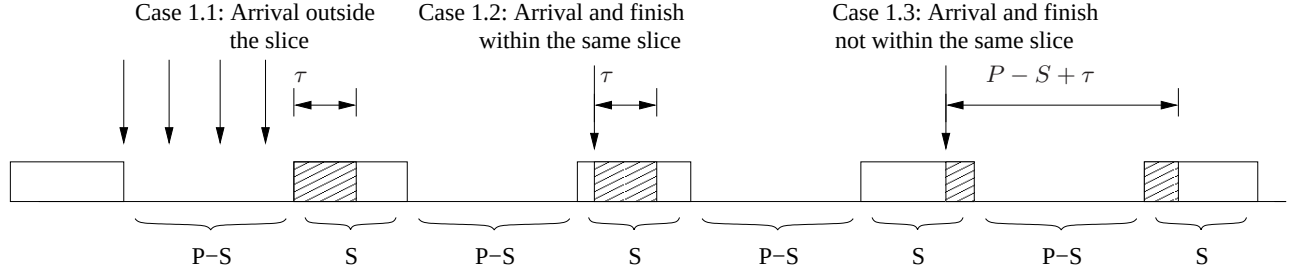
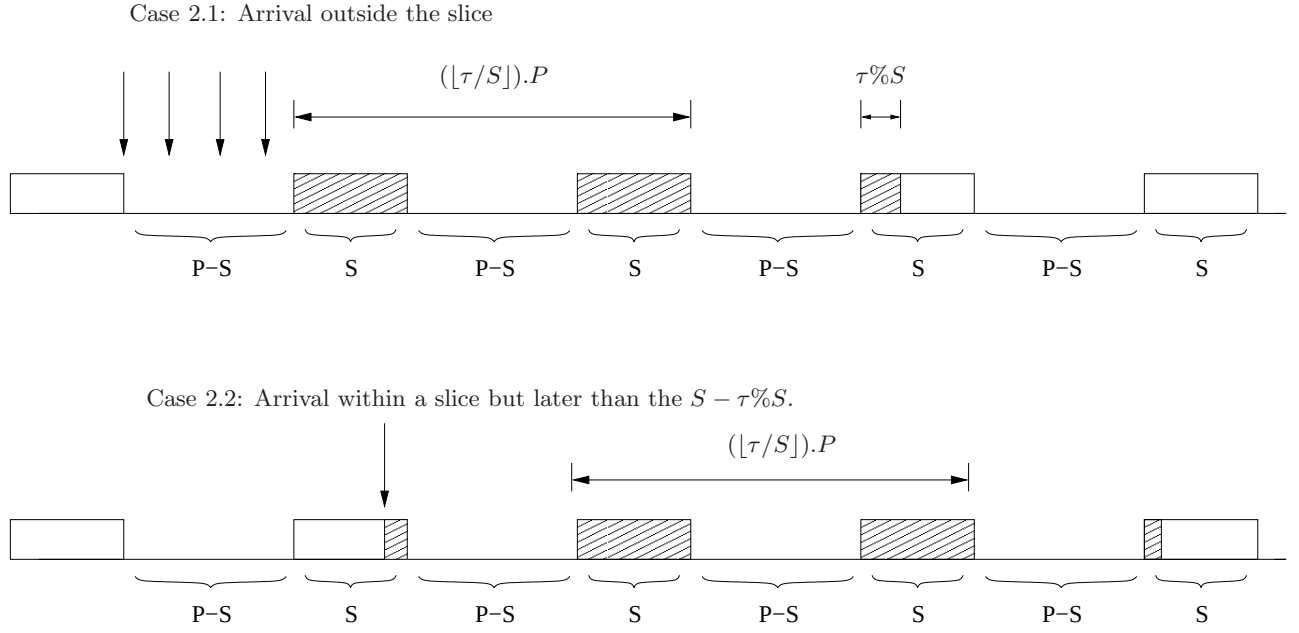
Similarly, we can prove that, for all incoming tokens after the k^{th} that

$$\begin{aligned} f(w_{(j-S)\% \tau}, k - \lfloor \frac{S}{\tau} \rfloor + y) &\leq f(x_{j-1}, k + y) \quad \text{where, } S\% \tau \leq j < \tau, 0 < y < \lfloor \frac{S}{\tau} \rfloor \\ \Leftrightarrow a(k - \lfloor \frac{S}{\tau} \rfloor) + (y + 1) \cdot \tau + P - S + (j - S)\% \tau + P &\leq a(k) + (y + 1) \cdot \tau + \lfloor \frac{S}{\tau} \rfloor \cdot \tau + P - S + j \end{aligned} \quad (5.13)$$

The rest of the proof remains exactly the same. Hence, we have proved that the tokens will be in their initial position, if the condition of the impulse response holds and $S \geq \tau$.

Please refer to Figure 5.10. In this figure, case 1.1 shows (with vertical arrows pointing downwards) the possibilities of arrival of an actor outside the slice. If this happens, then the time at which the TDM scheduler finishes processing is given by (arrival time + waiting time + execution time). The waiting time depends upon the time of arrival. In the worst case, the arrival may happen right at the finish of the slice. If the arrival happens at any time before, it shall get serviced immediately. Hence, the waiting time cannot be greater than $(P - S)$. As a result, the worst case finish time in the TDM scheduler is $a(k) + P - S + \tau$. The estimated finish time by the multi-rate model is $s(l, k) + P - S + \tau$. (referring to 5.3). But, as per explanation provided in beginning of the Section 5.2.3.1, $s(l, k) = a(k)$. Hence, the finish time predicted by the model is equal or greater.

In Figure 5.10, case 1.2 shows the possibility of the arrival and finish within a slice. In this case, the finish time is equal to (arrival time + execution time) since no waiting time is required. Thus, the finish time is $a(k) + \tau$. As per the Equation 5.3, the finish time is $s(l, k) + P - S + \tau = a(k) + P - S + \tau$ which is greater than the finish time predicted by the TDM scheduler.

Figure 5.10: Possible arrival times for $S \geq \tau$ Figure 5.11: Possible arrival times for $S < \tau$

In Figure 5.10, case 1.3 shows the possibility of the arrival and finish within a slice. In this case, the finish time is equal to (arrival time + execution time + full waiting time). Thus, the finish time is $a(k) + \tau$. As per the Equation 5.3), the finish time is $s(l, k) + P - S + \tau = a(k) + P - S + \tau$ which is equal to the finish time predicted by the TDM scheduler.

Case 2: $S < \tau$

In Section 5.2.1.2, we have seen that the tokens along the (w, x) edges should be in their original position so that the current iterations face minimal delay. There is at least one token along some (w, x) edges. Thus, we have to prove that, when $S \geq \tau$ and $a(k) > f(c, k-1)$, the single token due to the previous iteration is in its initial position for the next iteration. Let us assume that we have a steady state.

In Section 5.2.1.2, we have seen that the tokens along the (w, x) edges should be in their original position so that the current iteration faces minimal delay. Thus we have to prove that, when $S < \tau$ and $a(k) > f(c, k-1)$, the tokens are in their initial position. In terms of the SRDF graph, since there can be only one token along the (w, x) edges when $S < \tau$, it means that for any actor x_j , the condition in Equation 5.14 should hold. There are in all S such actors. Thus, we get $0 < j \leq S$.

$$f(w_{j-S\% \tau}, k-1) \leq f(x_{j-1}, k) \quad (5.14)$$

$$\Leftrightarrow a(k-1) + (P-S) + \lambda_F(x_0, x_{\tau+j}) \leq a(k) + P - S + j \quad (5.15)$$

But, $a(k) \geq a(k-1) + (P-S) + \lambda_F(x_0, x_{\tau-1})$ due to the relation with the previous finish time. Hence, substituting this condition in Equation 5.15, we get,

$$\begin{aligned} a(k-1) + (P-S) + \lambda_F(x_0, x_{\tau+j}) &\leq a(k-1) + (P-S) + \lambda_F(x_0, x_{\tau-1}) + P - S + j \\ \Leftrightarrow \lambda_F(x_0, x_{\tau+j}) &\leq \lambda_F(x_0, x_{\tau-1}) + P - S + j \quad \text{where, } 0 < j \leq S \end{aligned} \quad (5.16)$$

We use the same reasoning provided in Section 5.2.3.1 for the calculation of $\lambda_F(s, c)$ to find the values of $\lambda_F(x_0, x_{\tau+j})$ and $\lambda_F(x_0, x_{\tau-1})$. Substituting these values we get,

$$\begin{aligned} \lambda_F(x_0, x_{\tau+j}) &\leq \lambda_F(x_0, x_{\tau-1}) + P - S + j \quad \text{where, } 0 < j \leq S \\ \Leftrightarrow \lfloor (\tau+j)/S \rfloor P + (\tau+j) - \lfloor (\tau+j)/S \rfloor S &\leq P - S + j + \lfloor \tau/S \rfloor P + \tau - \lfloor \tau/S \rfloor S \\ \Leftrightarrow \lfloor (\tau+j)/S \rfloor (P-S) &\leq (1 + \lfloor (\tau)/S \rfloor)(P-S) \end{aligned} \quad (5.17)$$

Using the maximum value of $j = S$, we get

$$\begin{aligned} \lfloor (\tau+j)/S \rfloor (P-S) &\leq (1 + \lfloor (\tau)/S \rfloor)(P-S) \\ \Leftrightarrow \lfloor 1 + (\tau/S) \rfloor (P-S) &\leq (1 + \lfloor (\tau)/S \rfloor)(P-S) \end{aligned} \quad (5.18)$$

Using the maximum value of $\tau/S = r$ where $r \in \mathbb{N}$, we get the right hand side equal to the left hand side in Equation 5.18. Using the minimum value of $\tau/S = 1$, we also get right hand side equal to left hand side. Hence, the condition in Equation 5.18 holds. Hence, we have proved that the tokens will be in their initial position, if the condition of the impulse response holds and $S < \tau$.

When $S < \tau$, the execution of the actor with execution time τ gets divided into a number of slices. The number of full slices for the execution of the actor will be given by $\lfloor \tau/S \rfloor$. The remaining part $\tau \% S$ will be executed in the next consecutive slice, after a delay of $(P-S)$. Thus total time between the beginning and end of execution of an actor on a TDM scheduler is thus given by $\lfloor \tau/S \rfloor \cdot S + (\lfloor \tau/S \rfloor - 1)(P-S) + \tau \% S$. The waiting time depends upon the time of arrival. In the worst case, the arrival may happen right at the finish of the slice. If the arrival happens at any time before, it shall get serviced immediately. Hence, the waiting time cannot be greater than $(P-S)$.

Now, in case 2.1 in Figure 5.11, the arrival time of the actor is between the two slices. In this case, the time at which the TDM scheduler finishes processing is given by measured from the arrival is given by arrival time + waiting time + $\lfloor \tau/S \rfloor \cdot S + (\lfloor \tau/S \rfloor - 1)(P-S) + \tau \% S$. As a result, the worst case finish time in the TDM scheduler is given

by

$$\begin{aligned}
& a(k) + P - S + \lfloor \tau/S \rfloor \cdot S + (\lfloor \tau/S \rfloor - 1)(P - S) + \tau \% S \\
& = a(k) + \lfloor \tau/S \rfloor \cdot S + (\lfloor \tau/S \rfloor)(P - S) + \tau \% S \\
& = a(k) + \lfloor \tau/S \rfloor \cdot P + \tau \% S
\end{aligned} \tag{5.19}$$

The finish time predicted by the model is given by

$$\begin{aligned}
& s(l, k) + (P - S) + \lfloor \tau/S \% \tau \rfloor \cdot P + (\tau - (\lfloor \tau/S \% \tau \rfloor \cdot S)) \\
& a(k) + (P - S) + \lfloor \tau/S \rfloor \cdot P + (\tau - \lfloor \tau/S \rfloor \cdot S) \\
& a(k) + (P - S) + \lfloor \tau/S \rfloor \cdot P + \tau \% S
\end{aligned} \tag{5.20}$$

Thus, the finish time predicted by the model in Equation 5.20 is greater than the time required by the TDM scheduler as per Equation 5.19.

In case 2.2 in Figure 5.11, the arrival happens inside the slice but later than the last $\tau \% S$ time duration of the slice. As a result an additional slice is required to finish the left over part of τ . Hence, the additional time of $(P - S)$ is added to the Equation 5.19. Hence, the time at which the TDM scheduler finishes processing is given by

$$a(k) + \lfloor \tau/S \rfloor \cdot P + \tau \% S + P - S \tag{5.21}$$

Thus, from Equations 5.20, 5.21, we can say the finish time predicted by the model is same as that required by the TDM scheduler.

Hence, from case 1 and case 2, we see that the finish time predicted by the multi-rate model for same arrival times is either equal or greater than the actual time required by the TDM scheduler.

Hence, the Theorem 2 is proved.

Now consider the case of a delayed input token in a burst. For this token, the theorem 2 holds. Hence, the response time predicted by the multi-rate model is conservative. Now, if the subsequent tokens follow the burst condition, then they are conservative as proven later in Section 5.2.5.

5.2.4 Burst response

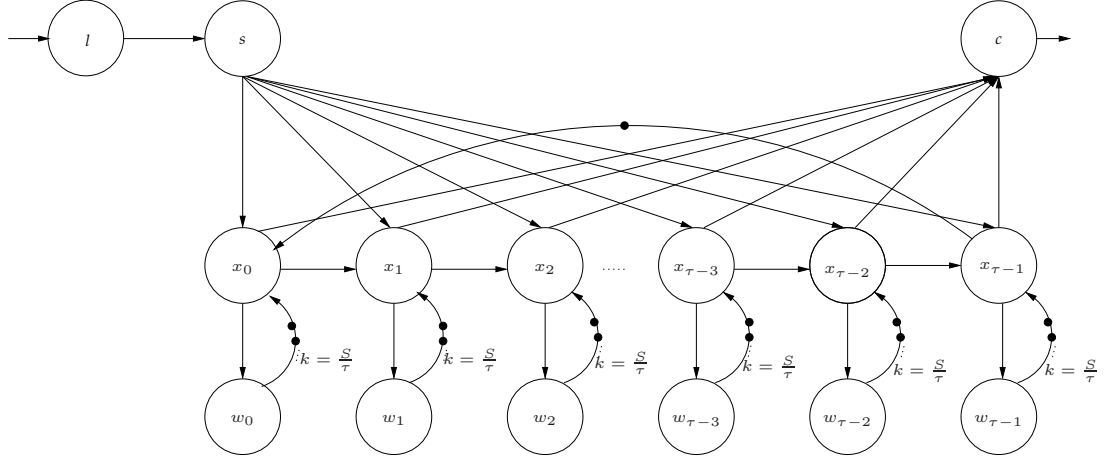
In this section, we first find the MCM, cyclicity and K_{G_S} for the SRDF graph G_S . Then we write the equations for the burst response and prove that they provide a conservative estimate for the finish time.

5.2.4.1 Periodicity of single rate equivalent graph G_S

Theorem 3: The periodicity of the single rate equivalent graph of the multi-rate model G_S is same as that of the TDM scheduler i.e. $r.P = \frac{\tau P}{\gcd(S, \tau)}$.

Proof:

In this section, we show that the periodicity of the single rate equivalent graph of the multi-rate model G_S is same as that of the TDM scheduler. We have seen in Section 3.4

Figure 5.12: structure of SRDF equivalent graph for case $S = k\tau$

that the periodicity of the TDM scheduler is rP . Hence we have to prove that the periodicity of the single rate equivalent graph of the multi-rate model G_S is $rP = \frac{\tau P}{\gcd(S, \tau)}$ (from equation 3.10). Hence we have to prove that the product of MCM ($\mu(G_S)$) and cyclicity ($N(G_S)$) is $\frac{\tau P}{\gcd(S, \tau)}$. First we provide the proof for MCM in the Section 5.2.4.2 and then for cyclicity in the Section 5.2.4.3.

5.2.4.2 MCM of the single rate equivalent graph G_S of multi-rate model

In this section, we present the proof for MCM of the single rate equivalent graph G_S of the multi-rate model.

Theorem 4: MCM of the single rate equivalent of multi-rate model G_S is $\frac{rP}{q} = \frac{\tau P}{S}$.

Proof:

First we have to prove that a cycle with cycle ratio $\frac{\tau P}{S}$ exists and then we have to prove that this ratio is equal to MCM. For the sake of simplicity, we divide the proof into three cases, depending upon relation between the slice time S and the worst case execution time of the modelled actor τ .

Case (1): when $S = k\tau$ or S is a multiple of τ . From definition of G_S (Equation 4.9), we know that there is an edge between an actor w_j and actor $x_{(j+S)\% \tau}$. The term $x_{(j+S)\% \tau}$ can also be written as

$$\begin{aligned} (j + S)\% \tau &= (j + k\tau)\% \tau \\ &= (j\% \tau + k\tau\% \tau)\% \tau \end{aligned} \quad (5.22)$$

Since $0 \leq j < \tau$, we get $j \% \tau = j$. Also, $k\tau \% \tau = 0$. Hence, equation 5.22 simplifies to

$$\begin{aligned}
 (j + S) \% \tau &= (j + k\tau) \% \tau \\
 &= (j \% \tau + k\tau \% \tau) \% \tau \\
 &= (j + 0) \% \tau \\
 &= j
 \end{aligned} \tag{5.23}$$

Thus, if $S = k\tau$, from equations 4.9 and 5.23, we can see that each actor w_j has an edge with an actor x_j . Thus, we have τ cycles, each of length $t_w + t_x = (P - t_x) + t_x = P$, and each having k delay tokens along edge (w_j, x_j) . Also, we have a cycle including all actors of length τ and having unity delay tokens. This structure is shown in figure 5.12. We have cycle ratios as $\frac{\tau}{1}$ and $\frac{P}{k} = \frac{P}{(\frac{S}{\tau})} = \frac{\tau P}{S}$. This proves that the cycle with cycle ratio $\frac{\tau P}{S}$ exists, when $S = k\tau$. First, we prove that the cyclicity of G_S is equal to q and then we prove that MCM of the G_S is equal to $\frac{\tau}{S}$.

MCM is given by,

$$\mu(G_S) = \max(\frac{\tau}{1}, \frac{\tau P}{S}, \frac{\tau P}{S}, \frac{\tau P}{S} \dots (\tau \text{ such cycles})) = \max(\tau, \frac{\tau P}{S})$$

For a TDM scheduler, $P \geq S$ or $\frac{P}{S} \geq 1$. Therefore, for a non-negative τ , $\frac{\tau P}{S} \geq \tau$.

Therefore, $\mu(G_S) = \max(\tau, \frac{\tau P}{S}) = \frac{\tau P}{S}$.

Conclusion: If S is a multiple of τ , the MCM of the graph G_S is $\frac{\tau P}{S}$.

Case (2): When $S \neq k\tau$ or S is not a multiple of τ .

We first prove in (2.1) that a cycle of cycle ratio $\frac{\tau P}{S}$ exists and then in (2.2), we show that this ratio is the MCM of the graph G_S .

part (2.1): We first prove that a cycle of cycle ratio $\frac{\tau P}{S}$ exists in G_S when $S \neq k\tau$.

In order to prove that MCM of the graph G_S is $\frac{\tau P}{S}$, we have to show that there exists a cycle of length τP and while traversing it, S tokens are encountered. There are in all τ copies of both x and w actors. Intuitively, if we want a cycle with τP length, we would have to traverse all w actors and all x actors exactly once before we reach the source. Then, length will be $(t_w + t_x)\tau = (P - t_x + t_x)\tau = \tau P$. First we show that there exists a path which goes through all w and x actors without reaching any w or x actor twice. We also show a special case when such a path does not exist.

We know from Equation 4.9 that there exists an edge $(w_j, x_{j+S \% \tau})$. Also, there exists an edge (x_i, w_i) in G_S . Let us assume that we begin traversing from an actor x_j with a decision that if we reach an actor x_i we would take the edge (x_i, w_i) and if we reach an actor w_k , we take the edge $(w_k, x_{(k+S) \% \tau})$. Thus the path becomes $x_j \rightarrow w_j \rightarrow x_{(j+S) \% \tau} \rightarrow w_{(j+S) \% \tau} \rightarrow x_{((j+S) \% \tau + S) \% \tau} \rightarrow w_{((j+S) \% \tau + S) \% \tau} \dots$ and so on.

Now, $((j + S)\% \tau + S)\% \tau$ can be simplified as

$$\begin{aligned}
 ((j + S)\% \tau + S)\% \tau &= (((j + S)\% \tau)\% \tau + S\% \tau)\% \tau \\
 &= ((j + S)\% \tau + S\% \tau)\% \tau \\
 &= (j + S + S)\% \tau \\
 &= (j + 2S)\% \tau
 \end{aligned} \tag{5.24}$$

We now use the principle of mathematical induction for prove that the path is extending till all the τ nodes. In the principle of mathematical induction, we prove that the condition is true for $n = 1$. Then we assume that it is true for any n and then we prove that it is true for $n + 1$. Thus, the condition in Equation 5.24 is true for $n = 1$. We now assume that the condition in Equation 5.24 is true for any n . Hence, we get

$$((j + nS)\% \tau + S)\% \tau = (j + (n + 1)S)\% \tau \tag{5.25}$$

Now for $n + 1$, we now prove that $((j + (n + 1)S)\% \tau + S)\% \tau = (j + (n + 2)S)\% \tau$

$$\begin{aligned}
 ((j + (n + 1)S)\% \tau + S)\% \tau &= (((j + (nS + S))\% \tau) + S)\% \tau \\
 &= (((j + nS) + S)\% \tau + S)\% \tau \\
 &= (((j + nS) + S)\% \tau)\% \tau + S\% \tau)\% \tau \\
 &= ((j + nS + S)\% \tau + S\% \tau)\% \tau \\
 &= ((j + nS + S + S)\% \tau)\% \tau \\
 &= ((j + (n + 2)S)\% \tau)\% \tau
 \end{aligned} \tag{5.26}$$

Thus, from (5.24), (5.25), (5.26) and principle of mathematical induction, for τ nodes the path will be

$$\begin{aligned}
 x_j \rightarrow w_j \rightarrow x_{(j+S)\% \tau} \rightarrow w_{(j+S)\% \tau} \rightarrow x_{(j+2S)\% \tau} \rightarrow w_{(j+2S)\% \tau} \\
 \rightarrow x_{(j+3S)\% \tau} \rightarrow w_{(j+3S)\% \tau} \dots \rightarrow x_{(j+\tau S)\% \tau}
 \end{aligned} \tag{5.27}$$

But,

$$\begin{aligned}
 (j + \tau S)\% \tau &= (j\% \tau + \tau S\% \tau)\% \tau \\
 &= (j + 0)\% \tau \\
 &= j
 \end{aligned} \tag{5.28}$$

This shows that, after traversing τ number of x actors and w actors, we reach the actor we had begun with. However, such a path will exist only if there is no number $g \in (1, \tau)$ in an expression $(j + gS)\% \tau$ such that $g.S$ is divisible by τ . If such a number exists, then we would reach the original node x_j without traversing all the x and w actors. Thus, we have an additional constraint that if no number between 1 and τ is a common divisor of both S and τ , then we are bound to traverse all w and x actors. The existence of a non-unity number g is a special case which we treat separately. This special arises when $\gcd(S, \tau) \neq 1$.

Since we have found a failure condition for our longest path traversal analysis when $\gcd(S, \tau)$ is non unity, we now consider two cases.

Case (2.1.a): $\gcd(S, \tau) = 1$

From Equation 5.28, for $\gcd(S, \tau) = 1$, we know that we reach the starting node after traversing all τ number of x and w nodes. Now we have to prove that no node is reached more than once. We will prove this by contradiction.

Let us assume that we reach a node twice during the traversal. Hence, the first time we reach it, we have $(j + uS) \% \tau$ and next time we have $(j + vS) \% \tau$, where $u \neq v$. Since we reach the same node, we can write $(j + uS) \% \tau = (j + vS) \% \tau$. Simplifying this further, we get

$$\begin{aligned} (j + uS) \% \tau - (j + vS) \% \tau &= 0 \\ (j + uS) - (\lfloor \frac{j + uS}{\tau} \rfloor) \tau - (j + vS) + (\lfloor \frac{j + vS}{\tau} \rfloor) \tau &= 0 \\ (u - v)S + \tau(\lfloor \frac{j + vS}{\tau} \rfloor - \lfloor \frac{j + uS}{\tau} \rfloor) &= 0 \end{aligned} \quad (5.29)$$

Since, τ and S are positive numbers, in the expression above, both $(u - v) = 0$ and $((\lfloor \frac{j + vS}{\tau} \rfloor) - (\lfloor \frac{j + uS}{\tau} \rfloor)) = 0$.

This means that $u = v$.

But we had assumed that $u \neq v$. Hence, our assumption was wrong. Thus, by contradiction, we prove that we traverse all w and x actors without visiting any actor twice and reach the original actor x_j . Since, we traverse all w and all x actors, we have the length as τP and we in this process we collect all S delays on all the (w, x) edges. Thus we prove in the cases where the $\gcd(S, \tau) = 1$ we have a cycle of length τP with S delays and thus cycle ratio is $\frac{\tau P}{S}$.

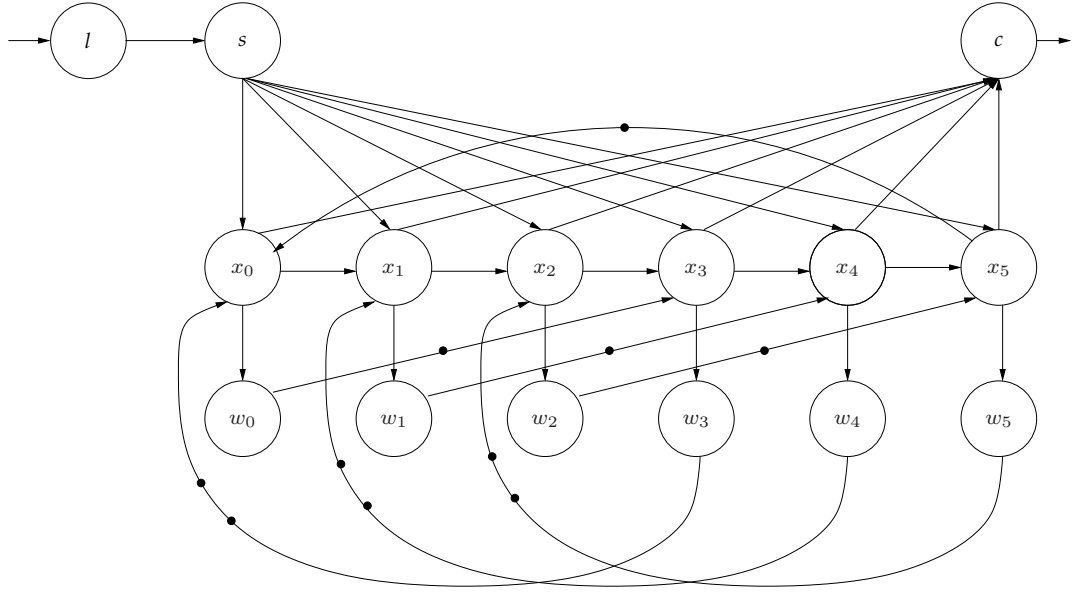
Case (2.1.b): $\gcd(S, \tau) \neq 1$ Now we consider the special case when there exists a $\gcd(S, \tau) = g$. In this case, we would reach the original node after traversing g number of x and w actors. Hence, we would have traversed a path of length $P(\frac{\tau}{g})$ and we would have collected $\frac{S}{g}$ delay tokens. There would be g such independent cycles. The cycle ratio is $P \cdot \frac{\tau/g}{S/g} = \frac{\tau P}{S}$. Example of such a structure is shown in figure 5.13 where $S = 9$ and $\tau = 6$ which gives $g = 3$.

Thus, we have proved that, in both the cases (2.1.a) and (2.1.b), there exists a cycle with the cycle ratio of $\frac{\tau P}{S}$.

part (2.2): Now we prove that the $\frac{\tau P}{S}$ is the MCM of the graph G_S in cases where $S \neq k\tau$.

The first step in proving that the MCM of G_S is to find an expression which gives the cycle ratios of all the remaining cycles in the graph G_S . Then, we prove that for all possible combinations, this expression has a value which is always less than $\frac{\tau P}{S}$.

The graph G_S has a cycle that covers all x actors and gathers one delay token, giving us a cycle ratio of $\tau/1 = \tau$. We have also shown that a cycle with path length τP exists in the graph G_S under the condition $S \neq k\tau$.

Figure 5.13: structure for the case when $\gcd(S, \tau) \neq 1$

However, there could be multiple other cycles in the graph G_S apart from the one with lengths τ and τP . If we traverse through all w actors, we would have also traversed through all x actors, since every w actor has an incoming edge from a unique x actor. Also, if we do not traverse through any w actor, we have to traverse through all x actors. We already know about these cycles. Hence, to find the remaining cycles, we have to traverse through at least one w actor and the most $(\tau - 1)$ w actors.

Starting from an actor x_j , we will have the following cycles through the edge $(x_{\tau-1}, x_0)$.

- through one w actor: $x_j \rightarrow w_j \rightarrow x_{(j+S)\% \tau} \rightarrow x_{1+(j+S)\% \tau} \rightarrow x_{2+(j+S)\% \tau} \dots \rightarrow x_{\tau-1} \rightarrow x_0 \rightarrow \dots x_{j-1} \rightarrow x_j$
- through two w actors: $x_j \rightarrow w_j \rightarrow x_{(j+S)\% \tau} \rightarrow w_{(j+S)\% \tau} \rightarrow x_{(j+2S)\% \tau} \rightarrow x_{1+(j+2S)\% \tau} \dots \rightarrow x_{\tau-1} \rightarrow x_0 \rightarrow \dots x_{j-1} \rightarrow x_j$
- through three w actors:...
- through four w actors:...
- .
- through $(\lfloor \frac{\tau}{(S\% \tau)} \rfloor)$ w actors: $x_j \rightarrow w_j \rightarrow x_{(j+S)\% \tau} \rightarrow w_{(j+S)\% \tau} \rightarrow x_{(j+2S)\% \tau} \rightarrow w_{(j+2S)\% \tau} \dots \rightarrow x_{\tau-1} \rightarrow x_0 \rightarrow \dots x_{j-1} \rightarrow x_j$

The longest such cycle will be the one through $\lfloor \frac{\tau}{(S\% \tau)} \rfloor$ w actors. This is because, when we take $x_j \rightarrow w_j \rightarrow x_{(j+S)\% \tau}$ path, we skip $S\% \tau$ number of x actors (and thus w actors). There are in total τ number of x actors. Hence the number of x actors (and thus w actors) encountered along the path is given by $(\lfloor \frac{\tau}{(S\% \tau)} \rfloor)$. Any $x_j \rightarrow w_j$ path contributes to a path length of $t(x) + t(w) = P$ units. If n such pairs are traversed, then

the total path length of all cycles through edge $(x_{\tau-1}, x_0)$ will be given by,

$$\text{path length} = nP + (\tau - n(S\% \tau)) \quad \text{where, } 1 \leq n \leq (\lfloor \frac{\tau}{(S\% \tau)} \rfloor) \quad (5.30)$$

From Equation 4.9, we know that each $(w_j, x_{(j+S)\% \tau})$ edge contributes to $(\lfloor \frac{S}{\tau} \rfloor)$ number of delay tokens. The unit delay along $(x_{\tau-1}, x_0)$ will make the total number of delay tokens along cycles through edge $(x_{\tau-1}, x_0)$ as,

$$\text{number of delay tokens} = n(\lfloor \frac{S}{\tau} \rfloor) + 1 \quad \text{where, } 1 \leq n \leq (\lfloor \tau / (S\% \tau) \rfloor) \quad (5.31)$$

The Equations 5.30 and 5.31 provide a generic expression for path length and number of delay tokens along all cycles through edge $(x_{\tau-1}, x_0)$.

The last $S\% \tau$ w actors have an edge to a x actor whose index is lower than that of the concerned w actor. Henceforth, we call such edges as back-edges. It is possible to have cycles due to these $S\% \tau$ back-edges. But even these cycles cannot be bigger than the longest cycle shown above. Consider that we traverse along $x_j \rightarrow w_j \rightarrow x_{(j+S)\% \tau} \rightarrow w_{(j+S)\% \tau} \rightarrow x_{(j+2S)\% \tau}$, starting an actor x_j until we reach one of the last $S\% \tau$ w actors. Consider that we take a back-edge and continue to traverse along alternate x and w actors until we reach an actor x with an index less than j such that any further traversing would not allow us to get back to x_j to complete the cycle. The index of such x actor will be $(j + (S\% \tau)S)\% \tau$. The delay tokens and the path length in this case exactly matches the one given by the equations 5.30 and 5.31.

To summarize, the cycle ratios of all possible cycles other than $\frac{\tau P}{S}$ and τ is given by

$$C_r = \frac{nP + (\tau - n(S\% \tau))}{n(\lfloor \frac{S}{\tau} \rfloor) + 1} \quad \text{where, } 1 \leq n \leq (\lfloor \frac{\tau}{(S\% \tau)} \rfloor) \quad (5.32)$$

Having found an expression for all the other cycles in the graph, we now prove that the Equation 5.32 has a value less than or equal to $\frac{\tau P}{S}$ in all cases. We divide this proof into three cases. These three cases are based on the maximum and minimum possible values of the term $S\% \tau$ respectively. The minimum value of $S\% \tau = 1$. This case is covered in case (2.2.a). The maximum value of $S\% \tau$ depends upon relation between τ and P . Hence, we need two cases - (2.2.b) and (2.2.c) to cover the maximum value of $S\% \tau$.

Case (2.2.a): When $S\% \tau = 1$:

Substituting $S\% \tau = 1$ in Equation 5.32, we get

$$C_r = \frac{nP + (\tau - n)}{n(\lfloor \frac{S}{\tau} \rfloor) + 1} \quad \text{where, } 1 \leq n \leq \tau \quad (5.33)$$

The maximum value of C_r should be less than the term $\frac{\tau P}{S}$. We have to check various conditions in order to show that $C_r \leq \frac{\tau P}{S}$. C_r will be maximum either when numerator is maximum or when the denominator is minimum.

1. The numerator of the expression 5.33 will be maximum, when $n = \tau$.

$$C_r = \frac{\tau P}{\tau(\lfloor \frac{S}{\tau} \rfloor) + 1} \quad (5.34)$$

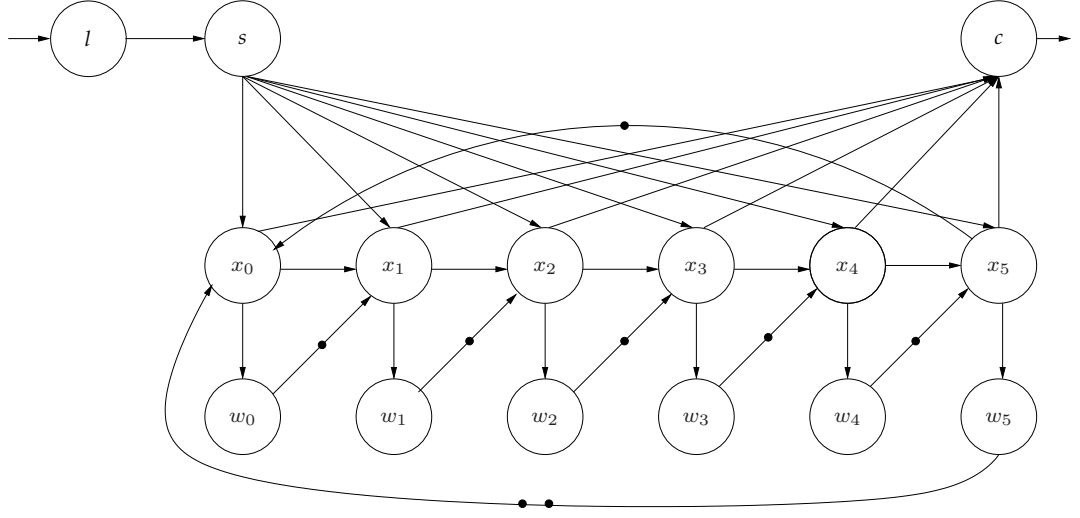


Figure 5.14: structure of SRDF equivalent graph when $S \neq k\tau$ and $S \% \tau = 1$

In Equation 5.34, maximum value of $\lfloor \frac{S}{\tau} \rfloor$ is $\frac{S}{\tau}$ and minimum value is 0. Also, if $\lfloor \frac{S}{\tau} \rfloor = 0$ and $S \% \tau = 1$, then it means that $S = 1$.

Using the minimum value of $\lfloor \frac{S}{\tau} \rfloor = 0$, we have to prove $C_r \leq \frac{\tau P}{S}$

$$\Leftrightarrow \frac{\tau P}{1} \leq \frac{\tau P}{S} \text{ with } S = 1 \quad (5.35)$$

Thus, the two terms are equal.

Using maximum value of $\lfloor \frac{S}{\tau} \rfloor = \frac{S}{\tau}$, We have to prove that $C_r \leq \frac{\tau P}{S}$

$$\begin{aligned} \Leftrightarrow \frac{\tau P}{\tau(\frac{S}{\tau}) + 1} &\leq \frac{\tau P}{S} \\ \Leftrightarrow (S + 1) &\geq S \text{ which is true for any positive } S. \end{aligned}$$

2. The denominator of the Equation 5.33 will be minimum, when $n = 1$.

$$C_r = \frac{P + \tau - 1}{(\lfloor \frac{S}{\tau} \rfloor) + 1} \quad (5.36)$$

In Equation 5.36, maximum value of $\lfloor \frac{S}{\tau} \rfloor$ is $\frac{S}{\tau}$ and minimum value is 0. Also, if $\lfloor \frac{S}{\tau} \rfloor = 0$ and $S \% \tau = 1$, then it means that $S = 1$.

Using the minimum value of $\lfloor \frac{S}{\tau} \rfloor = 0$, we get $C_r = \frac{(P+\tau-1)}{1}$. We have to

prove that $C_r \leq \frac{\tau P}{S}$.

$$\begin{aligned} &\Leftrightarrow \frac{\tau + P - 1}{1} \leq \frac{\tau P}{S} \text{ with } S = 1 \\ &\Leftrightarrow 1 \leq P \end{aligned}$$

For the TDM scheduler, $S \leq P$, with $S = 1$ in this case. Hence $1 \leq P$ is true.

Using maximum value of $\lfloor \frac{S}{\tau} \rfloor = \frac{S}{\tau}$, we get $C_r = \frac{P+\tau-1}{(\frac{S}{\tau})+1}$.

We have to prove that $\frac{P+\tau-1}{(\frac{S}{\tau})+1} \leq \frac{\tau P}{S}$

$$\begin{aligned} &\Leftrightarrow SP + S\tau - S \leq SP + \tau P \\ &\Leftrightarrow (\tau - 1)S \leq \tau P \end{aligned}$$

But we know that, for a TDM scheduler $S \leq P$ and for any positive number τ , $(\tau - 1) < \tau$. Hence, definitely, $(\tau - 1)S \leq \tau P$.

Thus, for case (2.2.a), the expression 5.32 is less than or equal to $\frac{\tau P}{S}$.

Case (2.2.b): When $S \% \tau = P$ (if $\tau > P$):

The term $S \% \tau$ can be equal to P if and only if τ is greater than P . Substituting $S \% \tau = P$ in Equation 5.32, we get

$$\begin{aligned} C_r &= \frac{nP + (\tau - nP)}{n(\lfloor \frac{S}{\tau} \rfloor) + 1} \quad \text{where, } 1 \leq n \leq \lfloor \frac{\tau}{P} \rfloor \\ &= \frac{\tau}{n\lfloor \frac{S}{\tau} \rfloor + 1} \end{aligned} \tag{5.37}$$

The maximum value of C_r in Equation 5.37 should be less than the term $\frac{\tau P}{S}$. C_r will be maximum either when numerator is maximum or when the denominator is minimum.

1. The denominator will be minimum, when $n = 1$. In Equation 5.37, maximum value of $\lfloor \frac{S}{\tau} \rfloor$ is $\frac{S}{\tau}$ and minimum value is 0.

Using the minimum value of $\lfloor \frac{S}{\tau} \rfloor = 0$, $C_r = \tau$. We have to prove that

$$\begin{aligned} C_r &\leq \frac{\tau P}{S} \\ &\Leftrightarrow \tau \leq \frac{\tau P}{S} \\ &\Leftrightarrow S \leq P \end{aligned}$$

We have assumed in the beginning that $S \leq P$. Hence, $\tau \leq \frac{\tau P}{S}$.

Using the maximum value of $\lfloor \frac{S}{\tau} \rfloor = \frac{S}{\tau}$, $C_r = \frac{\tau}{\frac{S}{\tau}+1}$. We have to prove that $C_r \leq \frac{\tau P}{S}$.

$$\begin{aligned} &\Leftrightarrow \frac{\tau}{\frac{S}{\tau}+1} \leq \frac{\tau P}{S} \\ &\Leftrightarrow \frac{\tau^2}{S+\tau} \leq \frac{\tau P}{S} \\ &\Leftrightarrow \frac{\tau}{S+\tau} \leq \frac{P}{S} \end{aligned}$$

For positive numbers S and τ , $\frac{\tau}{S+\tau} < 1$. Also, $P \geq S$ or $\frac{P}{S} \geq 1$. Hence $\frac{\tau}{S+\tau} \leq \frac{P}{S}$ is true.

There is only one case for $S \% \tau = P$ when $\tau > P$, since the numerator in Equation 5.37 is constant. Thus, for case (2.2.b), the Equation 5.32 is less than or equal to $\frac{\tau P}{S}$.

Case (2.2.c): $S \% \tau = \tau - 1$ (if $\tau \leq P$):

The term $S \% \tau$ can at most be equal to $\tau - 1$ if P is greater than or equal to τ . Substituting $S \% \tau = \tau - 1$ in Equation 5.32, we get

$$\begin{aligned} C_r &= \frac{nP + (\tau - n(\tau - 1))}{n(\lfloor \frac{S}{\tau} \rfloor) + 1} \quad \text{where, } 1 \leq n \leq \lfloor \frac{\tau}{\tau - 1} \rfloor \\ &= \frac{P + 1}{\lfloor \frac{S}{\tau} \rfloor + 1} \end{aligned} \tag{5.38}$$

The maximum value of C_r in Equation 5.38 should be less than the term $\frac{\tau P}{S}$. C_r will be maximum either when numerator is maximum or when the denominator is minimum.

1. The denominator will be minimum, when $\lfloor \frac{S}{\tau} \rfloor = 0$. Substituting this value in Equation 5.38, we get $C_r = P + 1$. We have to prove that $C_r \leq \frac{\tau P}{S}$. We have assumed that $S \leq P$ and in the current context, $S \% \tau = \tau - 1$, while $\tau \leq P$. Then we can say that $S = \tau - 1$.

$$\begin{aligned} &\Leftrightarrow P + 1 \leq \frac{\tau P}{S} \\ &\Leftrightarrow P + 1 \leq \frac{\tau P}{\tau - 1} \\ &\Leftrightarrow \tau \leq P + 1 \end{aligned}$$

We have assumed in the beginning that $\tau \leq P$. Hence, τ is definitely less than $P + 1$. Hence, it is proved that $P + 1 \leq \frac{\tau P}{S}$ in the current case.

There is only one case for $S \% \tau = \tau - 1$ with $\tau \leq P$ since the numerator in Equation 5.38 is constant. Thus, for case (2.2.c), the Equation (5.32) is less than or equal to $\frac{\tau P}{S}$.

In this way, we prove that the Equation 5.32 has a value less than or equal to

$\frac{\tau P}{S}$ under all conditions.

Now, MCM of the graph G_S when $S \neq k\tau$ and $GCD(S, \tau) = 1$ is given by

$$\mu(G_S) = \max(\tau, \frac{\tau P}{S}, \frac{nP + (\tau - n(S \% \tau))}{n(\lfloor \frac{S}{\tau} \rfloor + 1)}) \quad \text{where, } 1 \leq n \leq (\lfloor \frac{\tau}{(S \% \tau)} \rfloor)$$

Referring to cases (2.2.a), (2.2.b) and (2.2.c), we can write that $\frac{\tau P}{S} \geq \frac{nP + (\tau - n(S \% \tau))}{n(\lfloor S/\tau \rfloor + 1)}$ and $P \geq S$ implies that $\frac{\tau P}{S} \geq \tau$.

Therefore,

$$\mu(G_S) = \max(\tau, \frac{\tau P}{S}, \frac{nP + (\tau - n(S \% \tau))}{n(\lfloor \frac{S}{\tau} \rfloor + 1)}) = \frac{\tau P}{S} \quad (5.39)$$

where, $1 \leq n \leq (\lfloor \frac{\tau}{(S \% \tau)} \rfloor)$

Now we show that the $\frac{r.P}{q} = \frac{\tau P}{S}$.

As per the definition of r and q in Equation 3.10, we get

$$\begin{aligned} \frac{r.P}{q} &= \frac{\tau.P}{gcd(S, \tau)} / \frac{S}{gcd(S, \tau)} \\ &= \frac{\tau P}{S} \end{aligned}$$

Conclusion: If S is not a multiple of τ , the MCM of the graph G_S is $\frac{\tau P}{S}$.

Thus, we have proven that a cycle with cycle ratio $\frac{\tau P}{S}$ exists in the SRDF equivalent of multi-rate model G_S and it is equal to MCM of the graph G_S . Also the MCM of the SRDF equivalent graph G_S is equal to $\frac{r.P}{q}$.

Thus, Theorem 4 is proved.

5.2.4.3 Cyclicity of the single rate equivalent graph G_S of multi-rate model

In this section, we present the proof for the cyclicity of the single rate equivalent graph G_S of the multi-rate model. The cyclicity of a graph is defined as highest common factor of the number of delay tokens along all critical cycles in the graph. We prove that the cyclicity of the graph G_S is equal to q .

Theorem 5: The cyclicity $N(G_S)$ of the single rate equivalent of multi-rate model G_S is equal to $q = \frac{S}{gcd(S, \tau)}$.

Proof:

We break the proof in cases.

Case (1): $S = k\tau, k \in \mathbb{N}$ meaning that S is a multiple of τ :

When $S = k\tau$ (Figure 5.12), we have τ independent cycles of length P , each having a delay of $\frac{S}{\tau}$ in G_S . The only other cycle in the graph G_S is the one with length τ and a unity delay token.

For a TDM scheduler, $S \geq P$. We have assumed that $S = k\tau$. Due to this assumption, the relation between τ and P is $\tau < S \leq P$. Hence, the cycles with path length P are critical and delay along these cycles is equal to $k = S/\tau$. There are τ such cycles. Using the definition of cyclicity, we can write,

$$N(G_S) = \gcd\left(\frac{S}{\tau}, \frac{S}{\tau}, \frac{S}{\tau}, \dots, \tau \text{ times}\right) = \frac{S}{\tau}. \quad (5.40)$$

Since S is a multiple of τ , the greatest common divisor of S and τ is given by

$$\gcd(S, \tau) = \tau \quad (5.41)$$

From, equation 5.40 and 5.41, we can write,

$$N(G_S) = \frac{S}{\gcd(S, \tau)}. \quad (5.42)$$

Conclusion: $N(G_S) = \frac{S}{\gcd(S, \tau)} = q$, for the case where S is a multiple of τ .

Case (2): S is a not multiple of τ and $\gcd(S, \tau) = 1$:

For the cases where $\gcd(S, \tau) = 1$, we have shown earlier that there are cycles in G_S having lengths τP , τ and $nP + (\tau - n(S \% \tau))$, where $1 \leq n \leq \lfloor \frac{\tau}{(S \% \tau)} \rfloor$. The expression $nP + (\tau - n(S \% \tau))$ represents length of the remaining cycles in the graph G_S , of which the longest cycle is the one when value of n is $\lfloor \frac{\tau}{(S \% \tau)} \rfloor$.

The maximum value of n will be τ , when $S \% \tau = 1$ i.e. $S < \tau$. Hence, the maximum value of the expression $nP + (\tau - n(S \% \tau))$ is τP . Thus, we have proved that the longest cycle in the graph G_S , under any circumstances, will be of length τP .

We have seen earlier, that the number of delays along the cycle with length τP is S . Thus, if S is a not multiple of τ and $\gcd(S, \tau) = 1$, the cyclicity of the SRDF equivalent of the multi-rate model G_S is S .

$$\begin{aligned} N(G_S) &= \frac{S}{1} \\ &= \frac{S}{\gcd(S, \tau)} \end{aligned} \quad (5.43)$$

Conclusion: $N(G_S) = \frac{S}{\gcd(S, \tau)} = q$, for the case where S is not a multiple of τ and $\gcd(S, \tau) = 1$.

Case (3): S is a not multiple of τ and $\gcd(S, \tau) > 1$:

Let us assume that $\gcd(S, \tau) = g$, where $g > 1$. We have seen earlier that if

the greatest common divisor of (S, τ) is non-unity, there are g cycles of length $\frac{\tau P}{g}$ (Figure 5.13). The number of delay tokens along all these cycles is $\frac{S}{g}$.

$$\begin{aligned} N(G_S) &= gcd(\frac{S}{g}, \frac{S}{g}, \frac{S}{g} \dots g \text{ times}) \\ &= \frac{S}{g} \\ &= \frac{S}{gcd(S, \tau)} \end{aligned} \tag{5.44}$$

Conclusion: $N(G_S) = \frac{S}{gcd(S, \tau)} = q$, for the case where S is not a multiple of τ and $gcd(S, \tau) > 1$.

With cases (1),(2) and (3), we cover all possible cases between S and τ and in all the three cases, we have proved that the cyclicity of the graph G_S is equal to $\frac{S}{gcd(S, \tau)} = q$.

Hence, Theorem 5 is proved.

In Section 5.2.4.3 and Section 5.2.4.2, we have provided the proofs for MCM and cyclicity of the single rate equivalent graph G_S of the multi-rate model. MCM of G_S is $\mu(G_S) = \frac{\tau P}{S}$ and cyclicity of G_S is $N(G_S) = \frac{S}{gcd(S, \tau)}$. The periodicity of the graph G_S is given by

$$\begin{aligned} N(G_S) \cdot \mu(G_S) &= \frac{\tau P}{S} \cdot \frac{S}{gcd(S, \tau)} \\ &= \frac{\tau P}{gcd(S, \tau)} \end{aligned} \tag{5.45}$$

The periodicity of the TDM scheduler is given by Equation 3.11 and it is $r.P$ where r is defined by Equation 3.10 as $\frac{\tau}{gcd(S, \tau)}$. Hence we can write,

$$r.P = \frac{\tau P}{gcd(S, \tau)} \tag{5.46}$$

Conclusion: From Equations 3.11, 5.45 and 5.46, we have proved that the periodicity of the single rate equivalent graph of multi-rate model G_S and the TDM scheduler is equal.

Thus, Theorem 3 is proved.

5.2.4.4 Iterations of the graph G_S in transition phase

In this section, we show that the number of iterations in the transition period K_{G_S} of the single rate equivalent graph of the multi-rate model G_S is same as the number of executions of the TDM scheduler before the initiation of its periodic regime.

Theorem 6: The initial number of iterations in the transition period K_{G_S} of the graph G_S after which it enters into a periodic regime is equal to the number of executions of the TDM scheduler before the initiation of its periodic regime i.e. $S = q$.

Proof:

We have seen in Section 3.4 that the periodic regime of the TDM scheduler begins after q executions. Hence, we have to prove that the number of iterations in the transition period K_{G_S} of the single rate equivalent graph of the multi-rate model G_S is equal to q . In order to prove this theorem, we first unroll the SRDF graph G_S by a factor of q . Let the q -unrolled graph be represented by $G_{SU}(q)$. If we prove that the number of iterations in the transition period $K_{G_{SU}(q)}$ of graph $G_{SU}(q)$ is 1, then this is analogous to proving that the value of K_{G_S} for G_S is q . The expressions for the unrolled graph $G_{SU}(q)$ are provided in Section 4.5.

5.2.4.5 Delay distribution along (w, x) edges in $G_{SU}(q)$

The delay distribution d_u along (w, x) edges in a q -unrolled graph G_{SU} is given by the Equation 4.14 as

$$d_{su}(w_j, x_{(j+S)\%(q\tau)}) = \begin{cases} \lfloor S/(q\tau) \rfloor & \text{if } j < ((q\tau) - S\%(q\tau)) \text{ and } s \neq k\tau, k \in \mathbb{N} \\ (\lfloor S/(q\tau) \rfloor) + 1 & \text{if } j \geq ((q\tau) - S\%(q\tau)) \text{ and } s \neq k\tau, k \in \mathbb{N} \\ \lfloor S/(q\tau) \rfloor & \text{if } 0 \leq j \leq ((q\tau) - S\%(q\tau)) \text{ and } s = k\tau, k \in \mathbb{N} \end{cases}$$

In the current context, $q = \frac{S}{GCD(S, \tau)}$. The greatest common divisor of two numbers S and τ can, at the most, be equal to the smaller of the two. This means that the term $\lfloor S/(q\tau) \rfloor$ will be either equal to 1 or 0 as shown in Equation 5.47.

$$\lfloor S/(q\tau) \rfloor = \lfloor S/(\frac{\tau S}{GCD(S, \tau)}) \rfloor = \lfloor GCD(S, \tau)/\tau \rfloor = \begin{cases} 0 & \text{if } S \text{ is not a multiple of } \tau \\ 1 & \text{if } S \text{ is a multiple of } \tau \end{cases} \quad (5.47)$$

From Equation 4.14 and Equation 5.47, we can write expression for delay distribution d_u in the current context as,

$$d_{su}(w_j, x_{(j+S)\%(q\tau)}) = \begin{cases} 0 & \text{if } j < ((q\tau) - S\%(q\tau)) \text{ and } s \neq k\tau, k \in \mathbb{N} \\ 1 & \text{if } j \geq ((q\tau) - S\%(q\tau)) \text{ and } s \neq k\tau, k \in \mathbb{N} \\ 1 & \text{if } 0 \leq j \leq ((q\tau) - S\%(q\tau)) \text{ and } s = k\tau, k \in \mathbb{N} \end{cases} \quad (5.48)$$

5.2.4.6 Implication of $K = 1$ for $G_{SU}(q)$

In the previous section, we have found the delay distribution for (w, x) edges for the q -unrolled graph $G_{SU}(q)$. In this section, we use this delay distribution to create a proof statement for proving that $K = 1$ for the graph $G_{SU}(q)$.

For any edge (i, j) with delay $d_{su}(i, j)$ in the graph $G_{SU}(q)$, we can write the relation between the start times of various executions using self timed schedule (STS) (from Section 2.3.3.1) bounds as

$$s(j, k + 1) \geq s(i, k) + t(i) \quad \text{if } d_{su}(i, j) = 1 \quad (5.49)$$

$$s(j, k) \geq s(i, k) + t(i) \quad \text{if } d_{su}(i, j) = 0 \quad (5.50)$$

Using the definition of static periodic schedule (SPS), we can write the relation between the 0^{th} and the k^{th} firing of the graph $G_{SU}(q)$ as

$$s(j, k) = s(j, 0) + \mu_u k \quad (5.51)$$

Also, we can write similar relation between the 0^{th} and the $(k+1)^{th}$ firing of the graph $G_{SU}(q)$ as

$$s(j, k+1) = s(j, 0) + \mu_u(k+1) = s(j, 0) + \mu_u k + \mu_u \quad (5.52)$$

Here, μ_u is the MCM of the unrolled graph.

Using Equations 5.49, 5.51 and 5.52, we can write

$$\begin{aligned} s(j, 0) + \mu_u k + \mu_u &\geq s(i, k) + t(i) \\ s(j, k) + \mu_u &\geq s(i, k) + t(i) \\ s(j, k) &\geq s(i, k) + t(i) - \mu_u \end{aligned} \quad (5.53)$$

Using Equations 5.49, 5.50 and 5.52, we can write

$$s(j, k) \geq s(i, k) + t(i) \quad (5.54)$$

If the condition given by Equation 5.53 holds for all edges with delays on them even for $k = 0$, then we can say that the graph enters periodic execution right from the beginning. Graphically, in a linear programming space, it means that the STS constraints are tighter in all dimensions than the SPS constraints. Thus we have to show that the condition given by Equation 5.53 holds for $k = 0$. For $k = 0$, we get

$$\begin{aligned} s(j, 0) &\geq s(i, 0) + t(i) - \mu_u \\ s(j, 0) - s(i, 0) &\geq t(i) - \mu_u \\ s(i, 0) - s(j, 0) + t(i) &\leq \mu_u \end{aligned} \quad (5.55)$$

We have to prove that the condition in Equation 5.55 holds true. The $(s(i, 0) - s(j, 0) + t(i))$ term in Equation 5.55 can be considered as the path length from node j to node i including the execution time of i . In other words, the Equation 5.55 can be interpreted as if there exists no path from node j to any node i which is greater than μ_u , where (i, j) are the edges with unit delays. Hence, in the unrolled graph $G_{SU}(q)$, if we prove that there exists no path from node j to any node i which is greater than μ_u for edges (i, j) with unit delays, then we can say that the $K = 1$. Hence, it is sufficient to prove that the longest path in the graph $G_{SU}(q)$ is has length less than or equal to μ_u . It has been in the following section.

5.2.4.7 Expression for MCM of unrolled graph $G_{SU}(q)$

In order to prove the hypothesis mentioned above, we will need an expression for MCM of the unrolled graph $G_{SU}(q)$. Let this value be denoted by $\mu(G_{SU}(q))$. If we replace $q\tau$ in the expressions for V_{su}, E_{su}, d_{su} in the section 4.5 with τ' and compare them with the

expressions of V_s, E_s and d_s in the section 4.4, then the unrolled graph can be considered as a SRDF equivalent graph of multi-rate model of an application actor with execution time $q\tau$. Hence, the MCM of the graph $G_{SU}(q)$ will be $\tau'P/S$. Thus,

$$\begin{aligned}\mu(G_{SU}(q)) &= \tau'P/S \\ &= q\tau P/S \\ &= \frac{S}{\gcd(S, \tau)} \cdot \frac{\tau P}{S} \\ &= \frac{\tau P}{\gcd(S, \tau)}\end{aligned}\tag{5.56}$$

5.2.4.8 Proof for $K_{G_{SU}}(q) = 1$

We have found an analytical expression for the MCM of the unrolled graph $G_{SU}(q)$. Now, in this section we provide the proof for our hypothesis.

Theorem 7: The value of initial number of iterations in the transition stage for a q-unrolled SRDF equivalent graph of the multi-rate model is 1 i.e. $K_{G_{SU}}(q) = 1$.

Proof: As discussed in the Section 5.2.4.6, we have to prove that the longest path between the actors having an edge with a unit delay token between them in the q -unrolled single rate equivalent graph of the multi-rate model is less than or equal to the MCM of the unrolled graph $\mu(G_{SU}(q))$. We consider all cases for the proof, based upon the relation between S and τ .

Case (1): $S = k\tau, k \in \mathbb{N}$ i.e. S is a multiple of τ :

In this case, there is a unit delay token along all (w_j, x_j) edges and along $(x_{q\tau-1}, x_0)$ edge. The longest path from x_0 to $x_{q\tau-1}$ is just the path through all $q\tau$ actors. The longest path length from x_0 to $x_{q\tau-1}$ is just $q\tau$. The longest path length from actor (x_i, w_i) is $t_x + t_w = P$. This means we have to prove that $P \leq \mu(G_{SU}(q))$ and $q\tau \leq \mu(G_{SU}(q))$.

In this case, we have assumed that $S = k\tau$. Hence, $\gcd(S, \tau) = \tau$.

$$\mu(G_{SU}(q)) = \frac{\tau P}{\tau} = P\tag{5.57}$$

Thus, from 5.57, $P \leq \mu(G_{SU}(q))$ is proved. Furthermore,

$$q\tau = \frac{S\tau}{\gcd(S, \tau)} = \frac{S\tau}{\tau} = S\tag{5.58}$$

For a TDM scheduler, $S \leq P$. Hence, from 5.58, $q\tau \leq \mu(G_{SU}(q))$. is proved.

Conclusion: We have proved for the case where $S = k\tau$, longest path between actors having an edge between them with a unit delay token along it is less than

$\mu(G_{SU}(q))$.

Case (2): $S \neq k\tau, k \in \mathbb{N}$ i.e. S is not a multiple of τ :

In an unrolled graph, the last S actors of type w have an outgoing edge with unit delay to one of the first S actors of type x . Also there is an edge from $x_{q\tau-1}$ to x_0 with unit delay. Thus, in all, there are $S + 1$ edges with unit delays. Thus, we have to show that the longest path from x_0 to $x_{q\tau-1}$ and the longest paths from x_j to $w_{q\tau-(j+S)}$, for $0 \leq j < S$ is less than or equal to $\mu(G_{SU}(q))$.

Case (2.1): $\gcd(S, \tau) = 1$.

The MCM in this case will be $\mu(G_{SU}(q)) = \tau P$ (from Equation 5.56). We know that, in the current case, an actor w_i has an edge with an actor x_{i+S} (from Equation 4.15). Thus, there are two paths from an actor x_i to an actor x_{i+S} - one is direct path through the S x actors with path length S and other is through w_i of path length P . Since $P \geq S$, path length through w_i is always greater than the one through x actors. Using this fact, the longest path between x_0 and $x_{q\tau-1}$ will be $\max(q\tau, ((\tau - 1)P + S))$, where $q\tau$ is the path length through all x actors and $((\tau - 1)P + S)$ is the path length through maximum possible $(\tau - 1)$ x_i, w_i pairs and last S x actors. Thus, we get

$$\max(q\tau, (\tau - 1)P + S) = \max(S\tau, \tau P - P + S)$$

Since, for a TDM scheduler, $P \geq S$, $\tau P \geq \tau S$. Also, $\tau P - (P - S) \leq \tau P$. Both the terms above are less than τP . Hence, it is proved that $\max(S\tau, \tau P - P + S) \leq \mu(G_{SU}(q))$.

Let us try to find the longest path from x_j to $w_{(q\tau-(j+S))}$, for $0 \leq j < S$. The direct path from x_j to $w_{(q\tau-(j+S))}$ will be from x_j to $x_{(q\tau-(j+S))}$ to $w_{q\tau-(j+S)}$. Path length in this case will be $(\tau - 1)S + P$. The other path will be through x_j, w_j , and all possible x and w actors given by:

$$x_j \rightarrow w_j \rightarrow x_{((j+S)\%(q\tau))} \rightarrow w_{((j+S)\%(q\tau))} \rightarrow x_{((j+2S)\%(q\tau))} \rightarrow w_{((j+2S)\%(q\tau))} \rightarrow \dots w_{((j+\tau S)\%(q\tau))}$$

$$\text{But, } (j + \tau S)\%(q\tau) = (j + \tau S)\%(S\tau) = j$$

Thus, the path should end at $w_{((j+(\tau-1)S)\%(q\tau))} = w_{q\tau-(j+S)}$. In this case, the path length will be τP . Thus, the longest path is equal to $\max(\tau P, (\tau - 1)S + P)$ which is τP .

Thus, if $\gcd(s, \tau) = 1$, the longest path from x_0 to $x_{q\tau-1}$ and the longest paths from x_j to $w_{(q\tau-(j+S))}$, for $0 \leq j < S$ is less than or equal to $\mu(G_{SU}(q))$.

Case (2.2): $\gcd(S, \tau) > 1$.

Let $\gcd(S, \tau) = g$. The MCM in this case will be $\mu(G_{SU}(q)) = \frac{\tau P}{g}$ (from Equation 5.56).

We know that, in the current case, an actor w_i has an edge with an actor x_{i+S} . Thus, There are two paths from an actor x_i to an actor x_{i+S} - one is direct path through S actors of type x with path length S and other is through w_i of path length P . Since $P \geq S$, path length through w_i is always greater than the one through x actors. Using this fact, the longest path between x_0 and $x_{q\tau-1}$ will be $\max(q\tau, ((\tau/g - 1)P + S))$, where $q\tau$ is the path length through all x actors and $((\tau/g - 1)P + S)$ is the path length through maximum possible $(\tau/g - 1)$ x_i, w_i pairs and last S x actors. Thus, we get

$$\max(q\tau, (\tau/g - 1)P + S) = \max(S\tau/g, \tau P/g - P + S).$$

Since $P \geq S$, $\tau P/g \geq \tau S/g$. Also, $\tau P/g - (P - S) \leq \tau P/g$. Both the terms above are less than $\tau P/g$. Hence, it is proved that $\max(S\tau, \tau P/g - P + S) \leq \mu(G_{SU}(q))$.

Let us try to find the longest path from x_j to $w_{q\tau-(j+S)}$, for $0 \leq j < S$. The direct path x_j to $w_{q\tau-(j+S)}$ will be from x_j to $x_{q\tau-(j+S)}$ to $w_{q\tau-(j+S)}$. Path length in this case will be $(\tau/g - 1)S + P$. The path through x_j, w_j , and all possible x and sw actors will be

$$x_j \rightarrow w_j \rightarrow x_{((j+S)\%(q\tau))} \rightarrow w_{((j+S)\%(q\tau))} \rightarrow x_{((j+2S)\%(q\tau))} \rightarrow w_{((j+2S)\%(q\tau))} \rightarrow \dots w_{((j+\tau S)\%(q\tau))}$$

$$\text{But, } (j + \tau S)\%q\tau = (j + \tau S)\%S\tau = j.$$

Thus, the path should end at $w_{((j+(\tau/g-1)S)\%(q\tau))} = w_{q\tau-(j+S)}$. In this case, the path length will be $\tau P/g$. Thus, the longest path is equal to $\max(\tau P/g, (\tau/g - 1)S + P)$ which is clearly $\tau P/g$.

Thus, if $\gcd(s, \tau) > 1$, the longest path from x_0 to $x_{q\tau-1}$ and the longest paths from x_j to $w_{q\tau-(j+S)}$, for $0 \leq j < S$ is less than or equal to $\mu(G_{SU}(q))$.

Conclusion: We have proved for the case where $S \neq k\tau$, longest path between actors having an edge between them with a unit delay token along it is less than $\mu(G_{SU}(q))$.

We have shown that for all edges having unit delay token along them, the longest path from destination actor to source actor is always less than or equal to MCM of the unrolled graph.

Hence, the Theorem 7 is proved.

The theorem 7 shows that the longest path between the actors having an edge with a unit delay along them is always less than the MCM of the unrolled graph. Hence, by using our hypothesis in Section 5.2.4.6 and the theorem, we have proved that number of iterations of the unrolled graph $G_{SU}(q)$ in the transition phase i.e. K is equal to 1. The graph $G_{SU}(q)$ is an unrolled form of the graph G_S by a factor of q . This implies that the number of iterations for the graph G_S in the transition phase K is equal to q .

Thus, the TDM scheduler has q executions before it enters a periodic phase and the single rate form of the multi-rate model also has q iterations before it enters a periodic phase.

Hence, the Theorem 6 is proved.

5.2.4.9 Equations for the burst mode

We first describe the equations for the multi rate model for the first K or q iterations. In an unrolled graph $G_{SU}(q)$, there are a total of $q\tau$ copies of x , w actors and there are q copies of c , l and s actors. The condition for the burst response, $a(k) + P - S < f(c, k-1)$, implies that the finish time of an iteration depends upon the finish time of the previous iteration. This argument applies to all the q iterations. Equivalently, the finish time for all the q iterations depends upon the arrival of the first input token of the burst. In terms of the unrolled graph, it implies that the finish time of an iteration will be equal to the sum of the arrival of the first token of the burst and the longest forward path from the actor l_0 to actor c_k , $k \in [0, q)$. Let m^{th} token, $m \in [0, \text{inf})$ be the first token of the burst. Hence, we can write

$$\begin{aligned} f(c_k, m) &= a(m) + t(l_0) + \lambda_F(s_0, c_k) \\ f(c_k, m) &= s(l_0, m) + P - S + \lambda_F(s_0, c_k) \quad k \in [0, q) \end{aligned} \quad (5.59)$$

$$f(c_k, m) = s(l_0, m) + P - S + \lambda_F(x_0, x_{(k+1)\tau-1}) \quad k \in [0, q) \quad (5.60)$$

The Equation 5.60 can be written from Equation 5.59 due to the fact that $t(s_0) = t(c_k) = 0$. Substituting the values of the longest forward path in the Equation 5.60 for all values of k should give us the equations for the burst mode.

The first S copies of the x actors have delay tokens along their incoming (w, x) edges. Thus, the firing of the first S copies of the x actors can occur as soon as an input token arrives at their other input edge (from the preceding x actor). The firing of the first τ x actors, $(x_0$ to $x_{\tau-1})$ in $G_{SU}(q)$ marks the finish time of the first iteration of the input burst, firing of the x_τ to $x_{2\tau-1}$ in $G_{SU}(q)$ marks the finish time of the second iteration of the input burst and so on. Thus, the firing of every τ^{th} copy of x actor in $G_{SU}(q)$ marks the finish of an iteration of G_S . Equivalently, $\lfloor S/\tau \rfloor$ x actors in G_S can fire immediately leading to a completion of $\lfloor S/\tau \rfloor$ iterations of G_S .

$$\lambda_F(x_0, x_{(k+1)\tau-1}) = ((k+1)\tau - 1).t_x + t_x = (k+1).t_x.\tau \quad 0 \leq k < \lfloor \frac{S}{\tau} \rfloor \quad (5.61)$$

Substituting Equation 5.61 in Equation 5.60 we get,

$$f(c_k, m) = s(l_0, m) + P - S + (k+1).t_x.\tau \quad \text{for } 0 \leq k < \lfloor S/\tau \rfloor \quad (5.62)$$

There are no delay tokens on the incoming (w, x) edges of the remaining x actors, starting from x_S to $x_{q\tau-1}$. As a result, the longest forward path from the actor x_0 to actor $x_{(k+1)\tau-1}$ would be through the pairs of (x, w) actors, each contributing the path length of P . This is because the longest forward path from an actor $x_{(j-S)}$ to an actor x_j in $G_{SU}(q)$ is given by $\lambda_F(x_j, x_{(j-S)}) = \max((S+1).t_x, P + t_x)$. With $t_x = 1$ and $P \geq S$, we can say that $\lambda_F(x_j, x_{(j-S)}) = P + t_x$. Thus, for the x_0 actor in $G_{SU}(q)$,

the longest forward path to every $x_{(k+1)\tau-1}$ actor can be written with the help of the argument made for writing the Equation 5.2. The same equation is modified for the case under consideration. Thus, we get

$$\lambda_F(x_0, x_{(k+1)\tau-1}) = \left\lfloor \frac{(k+1)\tau}{(S\%(q\tau))} \right\rfloor P + ((k+1)\tau - \left\lfloor \frac{(k+1)\tau}{(S\%(q\tau))} \right\rfloor (S\%(q\tau))) t_x$$

for $\lfloor \frac{S}{\tau} \rfloor \leq k < q-1$ (5.63)

Substituting the value of Equation 5.63 in Equation 5.60, we get

$$\begin{aligned} f(c_k, m) &= s(l_0, m) + P - S + \left\lfloor \frac{(k+1)\tau}{(S\%(q\tau))} \right\rfloor P + ((k+1)\tau - \left\lfloor \frac{(k+1)\tau}{(S\%(q\tau))} \right\rfloor (S\%(q\tau))) t_x \\ &= s(l_0, m) + P - S + \left\lfloor \frac{(k+1)\tau}{S} \right\rfloor P + ((k+1)\tau - \left\lfloor \frac{(k+1)\tau}{S} \right\rfloor S) t_x \\ &\text{for } \lfloor S/\tau \rfloor \leq k < q-1 \end{aligned} \quad (5.64)$$

The argument used for the calculation of the longest forward path cannot be applied to the q^{th} iteration. This is because, the last S copies of x actors have to be traversed in the along the path through x actors, since there is no path through the last w actors (due to back edges) to reach actor $x_{q\tau-1}$. This can be seen in Figure 5.8. Hence, we calculate the longest forward path for the last iteration separately. For the last finish, once we reach the any one of the last x actors, we traverse along the chain of x actors. So, the number of (x, w) pairs have to be limited. Thus we get,

$$\lambda_F(x_0, x_{(q)\tau-1}) = \left\lfloor \frac{(k+1)\tau - S}{(S\%(q\tau))} \right\rfloor P + ((k+1)\tau - \left\lfloor \frac{(k+1)\tau - S}{(S\%(q\tau))} \right\rfloor (S\%(q\tau))) t_x \text{ for } k = q-1$$

Substituting Equation 5.65 in Equation 5.60 we get,

$$\begin{aligned} f(c_k, m) &= s(l_0, m) + P - S + \left\lfloor \frac{(k+1)\tau - S}{(S\%(q\tau))} \right\rfloor P + ((k+1)\tau - \left\lfloor \frac{(k+1)\tau - S}{(S\%(q\tau))} \right\rfloor (S\%(q\tau))) t_x \\ &= s(l_0, 0) + P - S + \left\lfloor \frac{(k+1)\tau - S}{S} \right\rfloor P + ((k+1)\tau - \left\lfloor \frac{(k+1)\tau - S}{S} \right\rfloor S) t_x \\ &\text{for } k = q-1 \end{aligned} \quad (5.65)$$

Since, $k = q-1$, we can write

$$\begin{aligned} f(c_k, m) &= s(l_0, m) + P - S + \left\lfloor \frac{q\tau - S}{S} \right\rfloor P + (q\tau - \left\lfloor \frac{q\tau - S}{S} \right\rfloor S) t_x \\ &\text{for } k = q-1 \end{aligned} \quad (5.66)$$

The three equations mentioned in Equations 5.62, 5.64 and 5.66 give the finish time of the first q iterations of a unrolled graph. But based on the equivalence between the SRDF graph and the unrolled graph discussed in Section 2.4, the start time $s(l_0, m)$ of

the unrolled graph is equal to $s(l, m)$ of the SRDF graph and the $f(c_k, m)$ of the unrolled graph is equal to $f(c, mq + k)$ of the SRDF graph. Thus,

$$\begin{aligned} s(l_0, 0) &= s(l, 0) \\ f(c_{\tau-1}, k) &= f(c, mq + k) \end{aligned} \quad (5.67)$$

Also, after the first q or K iterations, we have already seen that the graph enters periodic execution. The finish times during the periodic iterations can be represented by

$$f(c, k) = f(c, k - \frac{S}{g}) + \frac{\tau P}{g} \text{ for } k \geq q \text{ and } g = \gcd(S, \tau) \quad (5.68)$$

Due to the periodic nature of the finish times of the first q iterations, the arbitrary number m chosen can be also equal to zero. Thus, from Equations 5.62, 5.64, 5.66, 5.67 and 5.68 the equations for the multi rate model are as follows:

$$\begin{aligned} f(c, k) &= s(l, 0) + P - S + (k + 1).t_x.\tau \\ &\text{if } 0 \leq k < \lfloor S/\tau \rfloor \end{aligned} \quad (5.69)$$

$$\begin{aligned} &s(l, 0) + P - S + \lfloor \frac{(k+1)\tau}{S} \rfloor P + ((k+1)\tau - \lfloor \frac{(k+1)\tau}{S} \rfloor S)t_x \\ &\text{if } \lfloor S/\tau \rfloor \leq k < q - 1 \end{aligned} \quad (5.70)$$

$$\begin{aligned} &s(l, 0) + P - S + \lfloor \frac{q\tau - S}{S} \rfloor P + (q\tau - \lfloor \frac{q\tau - S}{S} \rfloor S)t_x \\ &\text{if } k = q - 1 \end{aligned} \quad (5.71)$$

$$\begin{aligned} &f(c, k - \frac{S}{g}) + \frac{\tau P}{g} \\ &\text{if } k \geq q \end{aligned} \quad (5.72)$$

$$\text{where, } g = \gcd(S, \tau) \quad (5.73)$$

5.2.5 Conservativity for the burst response

Theorem 8: The burst response of the SRDF equivalent graph of the multi-rate model is conservative i.e. $f(c, k) \geq F(k)$ if $a(k) + P - S < f(c, k - 1)$.

Proof:

The worst case finish time for a TDM scheduler is given by equation 3.5 in chapter 3 as

$$F(n) = \begin{cases} T(0) + (\lfloor \frac{n\tau}{S} \rfloor)P + (P - S) + (n\tau \% S); & \text{for } n\tau \% S \neq 0 \\ T(0) + (\lfloor \frac{n\tau}{S} \rfloor)P; & \text{for } n\tau \% S = 0 \end{cases} \quad (5.74)$$

Here, n is counted from one.

To prove the conservativity of the model, we have to show that the finish times given

by the equation (5.73) are equal to or greater than those given by (5.74). We have to prove it for all executions of n , i.e. from 1 to q . We prove it in three cases, namely $S < \tau$, $S > \tau$ and $S = \tau$. We are counting k from 0. Hence, the analogous k values will be from 0 to $q - 1$.

case (1): $S < \tau$

When $S < \tau$, $\lfloor S/\tau \rfloor = 0$. We are considering a situation where the longest path is governed by $w - x$ edges only. Thus, Equation 5.69 is not applicable. Using the definition of modulo operation in 5.70, and $t_x = 1$ we can write

$$f(c, k) = s(l, 0) + P - S + \lfloor \frac{(k+1)\tau}{S} \rfloor P + ((k+1)\tau \% S) \quad \text{for } 0 \leq k < q - 1$$

which is same as first condition of Equation 5.74, since k is counted from 0.

Using equation (3.10) and assuming $g = \gcd(S, \tau)$, we can write,

$$q\tau - S = \frac{S\tau}{g} - S = S(\frac{\tau}{g} - 1)$$

$$\text{Therefore, } \lfloor \frac{q\tau - S}{S} \rfloor = (\frac{\tau}{g} - 1)$$

Substituting these values in Equation 5.71 and with $t_x = 1$, we can write,

$$\begin{aligned} f(c, k) &= s(l, 0) + P - S + (\frac{\tau}{g} - 1)P + (\frac{S\tau}{g} - (\frac{\tau}{g} - 1)S) \\ &= s(l, 0) + P - S + \frac{P\tau}{g} - P + \frac{S\tau}{g} - \frac{S\tau}{g} + S \\ &= s(l, 0) + \frac{P\tau}{g} \\ &= s(l, 0) + \frac{SP\tau}{gS} \\ &= s(l, 0) + \frac{q\tau P}{S} \end{aligned}$$

which is same as the second condition of equation (5.74).

case (2): $S > \tau$

When slice time S is greater than τ , it is difficult to write equations for TDM scheduler using Equation 5.74. In order to simplify this, we consider a number m such that $\lfloor S/\tau \rfloor = m$. Then, $S > m\tau$ which means $\lfloor m\tau/S \rfloor = 0$ and $m\tau \% S = m\tau$.

From Equation 5.74, we can write another expression for the first m finishes

as

$$\begin{aligned}
 F(m) &= T(0) + P - S + \lfloor m\tau/S \rfloor P + (m\tau)\%S \\
 &= T(0) + P - S + 0 + (m\tau) \\
 &= T(0) + P - S + (m\tau)
 \end{aligned} \tag{5.75}$$

Here, m is counted from one. From (5.69) we can write the finish time for the first m finishes as

$$f(c, k) = s(l, 0) + P - S + m\tau \quad \text{for } 0 \leq k < m \tag{5.76}$$

which is same as Equation 5.75.

For the remaining $S - m\tau$ firings, the proof is exactly same as case (1).

case (3): $S = \tau$

After substituting $S = \tau$ in Equation (5.69), we get,

$$f(c, k) = s(l, 0) + P \quad \text{for } 0 \leq k < 1 \tag{5.77}$$

Similarly, from Equation (5.70), we get,

$$f(c, k) = s(l, 0) + (k + 1)P \quad \text{for } 1 \leq k < q - 1 \tag{5.78}$$

From Equation (5.71), we get,

$$f(c, k) = s(l, 0) + qP \quad \text{for } k = q - 1 \tag{5.79}$$

After substituting $S = \tau$ in Equation (5.74), we get,

$$F(n) = T(0) + nP$$

Substituting, $n = 1, k + 1$ and q and remembering that k is counted from zero while n from one in the above equation will give us the expressions which are same as Equations 5.77, 5.78 and 5.79.

Thus, the finish times obtained for the burst response of the multi-rate model are conservative towards the actual finish times.

Hence, the Theorem 8 is proved.

From theorem 2 and theorem 8, we have proven that, in all cases, the finish times obtained from the multi rate model are equal to those obtained from the TDM schedule in the worst case and hence the model is conservative.

Hence, Theorem 1 is proved.

5.3 Improvements over the LR model

Since, we have the equations for the LR model as well, we compare the finish times predicted by the LR model with those predicted by the multi-rate model and show that the finish times predicted by the multi-rate model are either same or less than those predicted by the LR model, for the same arrival times.

Theorem 9: For the same arrival times, the finish times predicted by the multi-rate model are less than or equal to those predicted by the latency rate model.

Proof:

We divide the proof into two sections depending upon the response of the model to the various type of inputs.

5.3.1 Comparison between the impulse responses

Consider the impulse response of the multi-rate model, given by Equation 5.3 and the response model for LR in Chapter 3. Section 3.5.1.2. We have to prove that

$$\begin{aligned} a(k) + (P - S) + \tau &\leq a(k) + (P - S) + \tau P/S && \text{for } S \geq \tau \\ a(k) + (P - S) + \lfloor \frac{\tau}{S \% \tau} \rfloor \cdot P + (\tau - (\lfloor \frac{\tau}{S \% \tau} \rfloor \cdot S)) &\leq a(k) + (P - S) + \tau P/S && \text{for } S < \tau \end{aligned}$$

Hence, we have to prove that

$$\begin{aligned} \tau &\leq \tau P/S && \text{for } S \geq \tau \\ \lfloor \frac{\tau}{S \% \tau} \rfloor \cdot P + (\tau - (\lfloor \frac{\tau}{S \% \tau} \rfloor \cdot S)) &\leq \tau P/S && \text{for } S < \tau \end{aligned} \quad (5.80)$$

Consider the first expression in Equation 5.80. We have to prove that

$$\begin{aligned} \tau &\leq \tau P/S \\ \Leftrightarrow \tau \cdot S &\leq \tau P \\ \Leftrightarrow S &\leq P \end{aligned} \quad (5.81)$$

$S \leq P$ is true for a TDM scheduler, as per our basic assumption in Section 5.1.

Consider the second expression in Equation 5.80:

$$\begin{aligned} \lfloor \frac{\tau}{S \% \tau} \rfloor \cdot P + (\tau - (\lfloor \frac{\tau}{S \% \tau} \rfloor \cdot S)) &\leq \tau P/S \\ \Leftrightarrow \lfloor \frac{\tau}{S} \rfloor \cdot P + (\tau - (\lfloor \frac{\tau}{S} \rfloor \cdot S)) &\leq \tau P/S \end{aligned} \quad (5.82)$$

If we could prove that the expression holds for maximum value of $\lfloor \frac{\tau}{S} \rfloor$, then it is proved for all values of $\lfloor \frac{\tau}{S} \rfloor$. The maximum value of $\lfloor \frac{\tau}{S} \rfloor = \frac{\tau}{S}$

$$\Leftrightarrow \tau \cdot P/S + (\tau - (\tau \cdot S/S)) \leq \tau P/S \quad (5.83)$$

Hence, both the sides are equal.

Thus, it is proved that the finish time predicted by the multi-rate model for the impulse response is less than or equal to that provided by the LR model.

5.3.2 Comparison between the burst responses

Consider the burst response of the multi-rate model, given by Equation 5.73. It is sufficient to we prove that the finish times predicted by the first q iterations of the burst by Equation 5.73 are less than or equal to those predicted by the LR model. This is because the remaining iterations during the periodic regime are related to the first q iterations and the period of both the graphs during the periodic regime is the same.

Based on the expression for the LR model, $f(x, k) = \max(a(k) + (P - S), f(x, k - 1)) + \frac{\tau P}{S}$, the burst response for the LR model can be written with $f(x, k) = f(x, k - 1) + \frac{\tau P}{S}$ and $f(x, 0) = a(0) + (P - S) + \frac{\tau P}{S}$ as $f(c, k) = a(0) + P - S + (k + 1) \cdot \frac{\tau P}{S}$.

For the multi-rate model, $s(l, k) = a(k)$.

Hence, we have to prove that

$$a(0) + P - S + (k + 1) \cdot t_x \cdot \tau \leq a(0) + P - S + (k + 1) \cdot \frac{\tau P}{S} \quad \text{if } 0 \leq k < \lfloor S/\tau \rfloor$$

$$a(0) + P - S + \lfloor \frac{(k + 1)\tau}{S} \rfloor P + ((k + 1)\tau - \lfloor \frac{(k + 1)\tau}{S} \rfloor S) t_x \leq a(0) + P - S + (k + 1) \cdot \frac{\tau P}{S} \quad \text{if } \lfloor S/\tau \rfloor \leq k < q - 1$$

$$a(0) + P - S + \lfloor \frac{q\tau - S}{S} \rfloor P + (q\tau - \lfloor \frac{q\tau - S}{S} \rfloor S) t_x \leq a(0) + P - S + (q) \cdot \frac{\tau P}{S} \quad \text{if } k = q - 1 \quad (5.84)$$

Hence, we have to prove that

$$(k + 1) \cdot t_x \cdot \tau \leq (k + 1) \cdot \frac{\tau P}{S} \quad \text{if } 0 \leq k < \lfloor S/\tau \rfloor \quad (5.85)$$

$$\lfloor \frac{(k + 1)\tau}{S} \rfloor P + ((k + 1)\tau - \lfloor \frac{(k + 1)\tau}{S} \rfloor S) t_x \leq (k + 1) \cdot \frac{\tau P}{S} \quad \text{if } \lfloor S/\tau \rfloor \leq k < q - 1 \quad (5.86)$$

$$\lfloor \frac{q\tau - S}{S} \rfloor P + (q\tau - \lfloor \frac{q\tau - S}{S} \rfloor S) t_x \leq (q) \cdot \frac{\tau P}{S} \quad \text{if } k = q - 1 \quad (5.87)$$

$$(5.88)$$

Case (1): In this case, we prove the condition in Equation 5.85.

Consider the expression in Equation 5.85. With $t_x = 1$, it is exactly same as Equation 5.81. Hence, we have proved first expression.

Case (2): In this case, we prove the condition in Equation 5.86.

Consider the expression in Equation 5.86. With $t_x = 1$, we have to prove that

$$\lfloor \frac{(k+1).\tau}{S} \rfloor .P + ((k+1).\tau - (\lfloor \frac{(k+1).\tau}{S} \rfloor .S)) \leq (k+1).\tau P/S$$

If we could prove that the expression holds for maximum value of $\lfloor \frac{(k+1).\tau}{S} \rfloor$, then it is proved for all values of $\lfloor \frac{(k+1).\tau}{S} \rfloor$. The maximum value of $\lfloor \frac{(k+1).\tau}{S} \rfloor = \frac{(k+1).\tau}{S}$.

$$\Leftrightarrow (k+1).\tau P/S + ((k+1).\tau - (k+1).(\tau S/S)) \leq (k+1).\tau P/S \quad (5.89)$$

Hence, both the sides are equal.

Case (3): In this case, we prove the condition in Equation 5.87.

Consider the expression in Equation 5.87. With $t_x = 1$, we have to prove that

$$\lfloor \frac{q\tau - S}{S} \rfloor P + (q\tau - \lfloor \frac{q\tau - S}{S} \rfloor S) \leq (q).\frac{\tau P}{S}$$

If we could prove that the expression holds for maximum value of $\lfloor \frac{q\tau - S}{S} \rfloor$, then it is proved for all values of $\lfloor \frac{q\tau - S}{S} \rfloor$. The maximum value of $\lfloor \frac{q\tau - S}{S} \rfloor = \frac{q\tau - S}{S}$.

$$\begin{aligned} &\Leftrightarrow (\frac{q\tau - S}{S})P + (q\tau - (\frac{q\tau - S}{S})S) \leq (q).\frac{\tau P}{S} \\ &\Leftrightarrow (\frac{q\tau - S}{S})P + S \leq (q).\frac{\tau P}{S} \\ &\Leftrightarrow S \leq (q).\frac{\tau P}{S} - (\frac{q\tau - S}{S})P \\ &\Leftrightarrow S \leq P \end{aligned} \quad (5.90)$$

$S \leq P$ is true for a TDM scheduler, as per our basic assumption in Section 5.1.

Thus, it is proved that the finish time predicted by the multi-rate model for the burst response is less than or equal to that provided by the LR model.

Hence, the Theorem 9 is proved.

The finish times predicted by the multi-rate model, under all conditions, are less pessimistic than the LR model.

5.4 Summary

In this chapter, we cover two important aspects, i.e. the multi-rate model is conservative and that it is less pessimistic than the LR model. In order to prove the conservativity

of the multi-rate model, we first classify the behaviour of the model based upon the rate of arrival of input tokens into impulse response and burst response. We also argue that these are only two possibilities for the input arrival. Later, based upon this classification, we provide a detailed outline of this proof. Based upon the outline, we propose a number of theorems and prove them, finally proving that the multi-rate model is conservative. In this process, we find the MCM and cyclicity of the multi-rate model. In order to prove that the LR model is more pessimistic, we use the equations developed for the burst and the impulse response in the proof for the conservativity.

In the previous chapter, we discussed the various properties and proofs related to the multi-rate model. In this chapter, we discuss the implementation details of the data-flow analysis and scheduling tool that we used and extended further in this thesis. We first describe the software as it existed before we started our work. We then discuss the features added into the software during the course of our work.

6.1 Data-flow analysis and scheduling tool - Heracles

The Data-flow analysis and scheduling tool called Heracles is written in OCaml [15]. OCaml is an open source functional programming language with imperative extensions. It is an object-oriented language. It has compile time type inferencing, which provides execution safety. The source code can be compiled into a byte coded version which is interpreted by a virtual machine, so that the software written in OCaml can be portable. There is also a native machine compiler for efficient performance. Hence software developed using OCaml supports both portability as well as performance. The actual development was done on the Linux platform using the open source tools for compiling and debugging. Heracles supports Graphviz dot format [1] and hence we can use the associated tool-chain for visualization of graphs. Heracles consists of a scheduler, analyser and a simulator. Figure 6.1 provides a simplified flowchart of the software tool after the addition of new features. The boxes in red colour show the features implemented in this thesis. However, it does not show the features related to optimizing the current schedules which are discussed in Section 6.2.4.

6.1.1 Inputs to Heracles

The primary inputs to Heracles are as follows: an application graph in CSDF format (MRDF and SRDF are sub-sets of CSDF and hence are considered valid inputs) and a file describing the MPSoC system. The file describing the MPSoC system consists of a list of processors. Each processor in the list has the following attributes:

1. name of the processor
2. scheduler type
3. cycle time (period of the scheduler)
4. maximum allowable utilization

Each actor in the application graph has the following compulsory attributes:

1. name of the actor
2. worst case execution time

There are some optional attributes for an actor like

1. type of processor on which the actor needs to be mapped
2. forced mapping

The name of the processor uniquely names a processor. The scheduler type tells the tool what type of scheduler is managing the processor time. The cycle time is the time taken for an iteration of a periodic scheduler. Maximum allowable utilization is defined as the ratio of slice time to period, expressed in percentage. Thus, utilization is just another way of describing slice time. For example, utilization of 80% for a processor with period as 1000 time units essentially means that the slice time is 800 time units.

The processor type provides Heracles with the information that the actor should be mapped to certain type of processor. For example, let us assume that we have an actor called *FIR* which models FIR filtering operation in some application graph. It should be mapped to a DSP (digital signal processor) for better timing performance. Hence, the actor will have an attribute processor type its value being some DSP.

With forced mapping, a designer can force an actor to be mapped onto a particular processor in the MPSoC system file. The value of this attribute has to be one of the processors from the MPSoC system file. For example, let us assume that we have an actor called *DECODE* which models a decoding operation. It should be mapped to dedicated decoder hardware. In such cases, forced mapping of an actor is helpful. Such attributes can also help the designer to use his or her knowledge of the system to prune the scheduler's search space.

6.1.2 Scheduler

The scheduler accepts a SRDF graph (expanded from the input CSDF graph) and the MPSoC system descriptor file provided by the user. The scheduler works in the following way. It maps an actor onto a processor either based on the forced mapping or on the processor type for that actor. While mapping the actors, the scheduler also creates a static order of the actors that are mapped on the same processor. After each mapping decision, it creates a new analysis graph. The analysis graph is a data-flow graph built by using a data-flow model (discussed in Chapter 3) for each actor in the application graph and connecting these data-flow model by the edges from the application graph as well as by adding edges to respect the static ordering between actors. An example with multi-rate model is shown in Figure 6.2. The application graph has two actors A and B, A mapped to ARM and B mapped to DSP. The two actors have been replaced by their respective multi-rate models and the edges of the application graph are reproduced in the analysis graph (shown in red colour). Another example of such conversion with latency rate model is shown in Figure 3.4 in Chapter 3.

```

let schedule graph =
  (*scheduling dag is a graph without edges with delays*)
  let dnodes = (nodes graph)
  and dedges = filter (fun e -> delay e < 1) (edges graph) in
  let dag = (dnodes, dedges)
  and binds = [] in

```

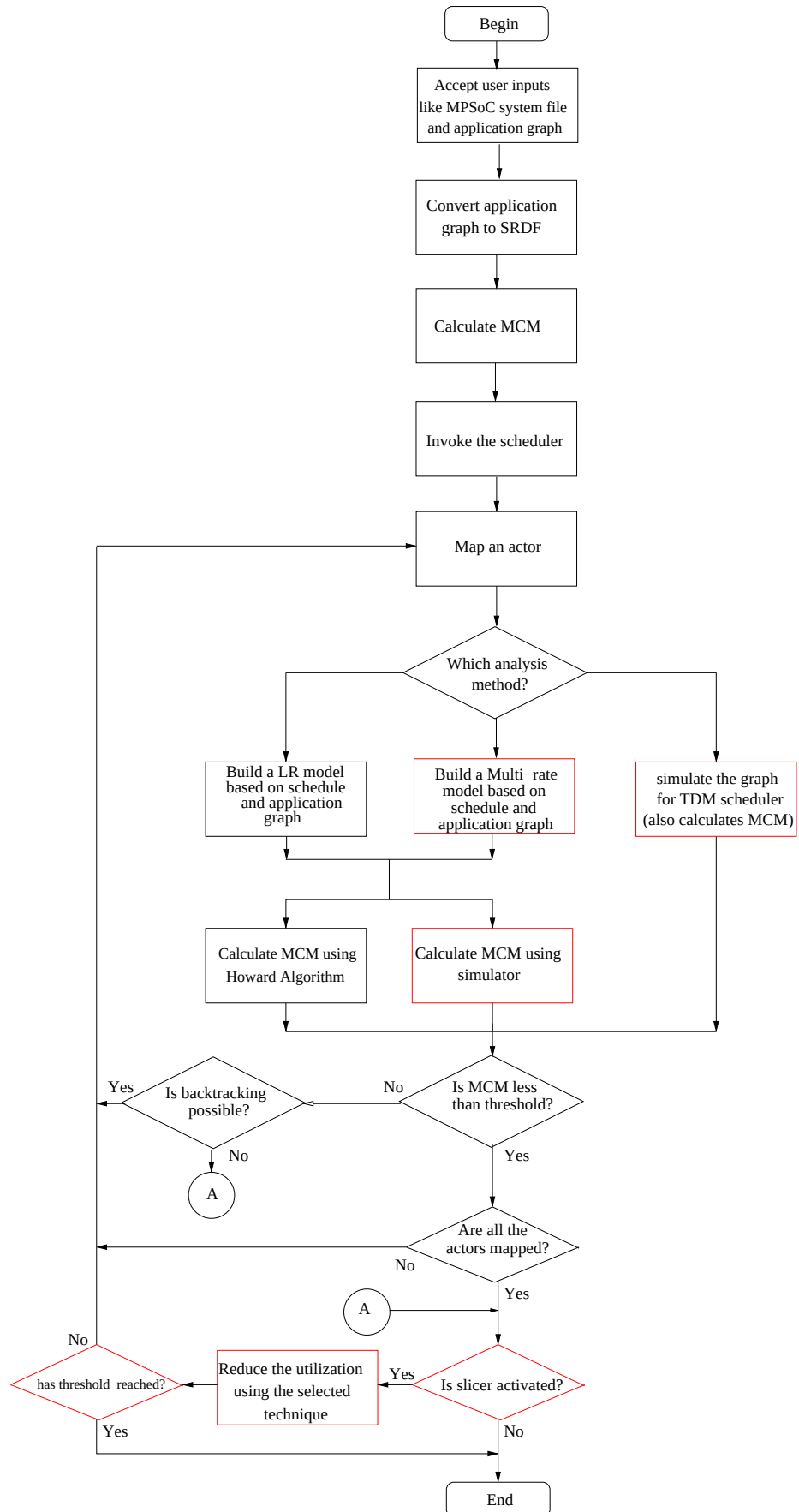


Figure 6.1: Simplified flowchart for the software tool

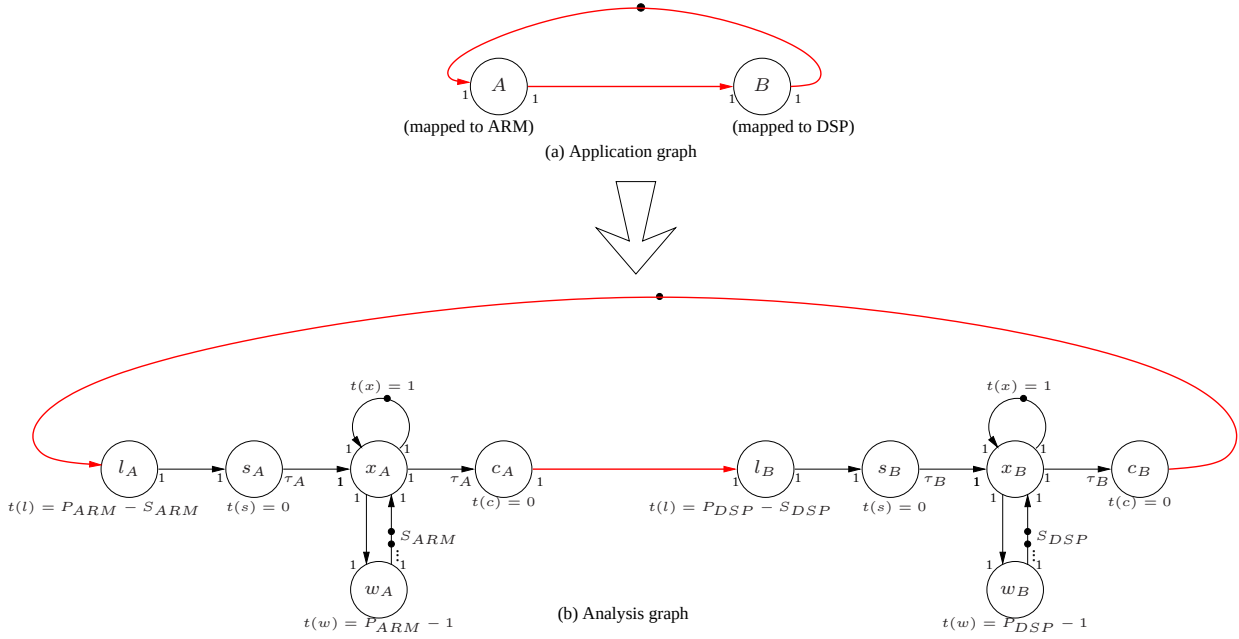


Figure 6.2: application graph to analysis graph conversion

(*a schedule is a pair of a dag with a list of binds*)

```

let empty_schedule = (dag, binds) in
scheduling_step empty_schedule;
raise No_Solution_Found

```

The scheduler employs a brute force algorithm which tries to map each actor on all possible processors. As a first step [20], the scheduler generates a directed acyclic graph (DAG) from the application graph by removing the edges with delays on them and initializes an empty schedule and then makes a call to the *scheduling_step* function. The *scheduling_step* function is implemented as a recursive function [20]. It goes through all the actors that can be mapped and tries to map each one of them to each processor where it can run. As a result, it performs exhaustive search. Hence, the scheduler has an exponential worst case complexity since it tries to map every actor on every processor. Then it generates an analysis graph as mentioned above.

With such an analysis graph, the scheduler invokes the analyser (discussed later) via *analyse* function. The analyser returns either *Valid* or *Invalid* depending upon whether the current set of mappings meets the temporal requirements. If the temporal requirements are not met, then the scheduler undoes the mapping and tries other possible mappings, creates another analysis graph, invokes the analyser and keeps repeating these steps until either all possible (actor, processor) pairs have been tried or a mapping is found which meets the temporal requirements. If the temporal requirements are met, then the scheduler continues with the next actor in the application graph and the process repeats. If all the possibilities have been tried with no success, then the scheduler stops with an exception *No_Solution_Found*. The scheduler also stops with an exception

Solution_found *s* if all the actors have been mapped successfully.

```

let scheduling_step schedule =
let (dag, binds) = schedule in
  (*an actor is fireable if it has no incoming edges in a dag*)
let fireable_nodes = get_sources dag in
if dag = ([], []) then raise (Solution_found schedule);
for each actor in fireable_nodes do
  for each processor in (possible_mappings actor) do
    let new_schedule = map actor processor schedule in
    let analysis_result = analyse new_schedule graph in
    match analysis_result with
    | Valid -> scheduling_step new_schedule
    | Invalid -> ()
  done
done

```

It is possible to find a number of solutions for a given MPSoC system and a application graph. When a solution is found, the scheduler keeps searching for better solutions (or for all solutions if requested to do so). Each solution is a feasible schedule, which will meet the temporal requirements. Please note that the pseudo-code [20] does not depict this part.

6.1.3 Analyser

The analyser uses the MCM (maximum cycle mean) of the application graph and calculates the MCM of the analysis graph. The analyser checks if the graph has delay-less cycles. It also calculates the MCM of the analysis graph. We have seen earlier that reciprocal of the MCM of the application graph G_{app} , $(1/\mu(G_{app}))$ is the maximum achievable throughput. Due to the scheduling of actors onto processors with TDM schedulers, the actual achievable throughput will be less than the value of $(1/\mu(G_{app}))$. In other words, the analysis graph models the timing behaviour of the application when it is mapped on processors that are using a TDM scheduler. Thus, the MCM of the analysis graph can never be lower than the MCM of the application graph. Thus, reciprocal of the MCM of the analysis graph is the actual achievable throughput. The MCM of the analysis graph is tested against a threshold MCM which comes from the throughput requirements of the system. Thus, the MCM of the analysis graph should be less than the threshold for a schedule that meets the throughput requirements. Thus, the analyser tells the scheduler whether the current schedule is temporally valid.

Let us extend the example of Figure 6.2. In this example, actor A is mapped to a ARM processor and actor B is mapped to a DSP. Let us assume that the execution times of A and B are $\tau_A = 3$ units $\tau_B = 2$ units. There is only one cycle in the application graph and hence the its cycle mean is $\frac{(3+2)}{1} = 5$. Thus, the maximum achievable throughput is $\frac{1}{5} = 0.2$. Let the periods of the TDM schedulers on ARM and DSP be $P_{ARM} = 5$ units and $P_{DSP} = 5$ units. Also, let the allocated slice times be $S_{ARM} = 4$ units and $S_{DSP} = 3$ units. MCM is defined only for SRDF graphs. The single rate equivalent of the analysis graph shown in Figure 6.2(b) has the MCM as 11. Hence the actual

achievable throughput is $1/11$. Now if we want to have a throughput of 0.1, then we set the threshold MCM as 10 during the analysis. In this case, it is not possible to have a temporally valid schedule with the current settings.

6.1.4 Simulator

The simulator takes a scheduled application graph as an input and simulates it. The simulator is an event-based simulator and it maintains the state of each actor and each buffer (edge). Whenever an event is created, the simulator is activated. It first checks if the current state was previously visited. If it was visited then the simulator stops the simulation and prints the results. If it is a new state, then it processes the events. Each event corresponds to an actor. The simulator uses three types of events - start event, finish event and update event. During the start event, it checks if the actor is fireable (i.e. all the incoming edges have sufficient tokens on them). If it is fireable, then it reserves the tokens to be consumed and produced and issues a finish event for that actor at a time which is equal to current time plus the execution time of the actor. If the actor is not fireable nothing is done. On a finish event, the tokens to be consumed are actually removed from the incoming edge buffer and the tokens to be produced are added on the outgoing edge buffers. Start events for consumers of the outgoing edges are issued. Usually, an actor with least repetitions (in the repetitions vector) is chosen to update the state and is called an updater. An update event is also set if the current finish event was that for an updater. On update event, the internal state is updated and checked it was visited before, so that the simulation can be stopped. The termination criterion is the end of an iteration of the graph being simulated.

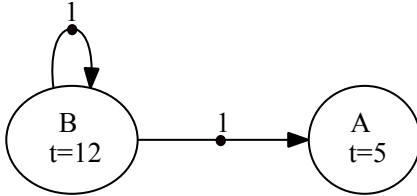


Figure 6.3: input graph

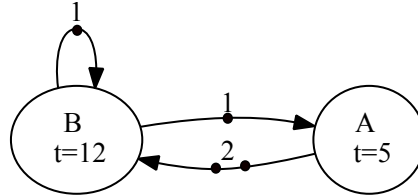


Figure 6.4: simulated graph

The graph which is given as an input to the simulator is converted into a strongly connected graph by adding back edges. A large number of delay tokens are populated along these back edges so that the cycles created by these back edges do not affect the MCM (throughput) of the input graph. Also, the number of these tokens can be considered as the limit on the buffer size of the actual graph edges. For example, please see the Figure 6.3 and Figure 6.4. The input graph in Figure 6.3 gets modified as Figure 6.4 inside the simulator and the graph in Figure 6.4 is simulated. The edge from actor A to B with two delay tokens is added in order to make the graph strongly connected.

Since, the simulator uses a strongly connected graph, convergence is guaranteed. The number of required for the convergence, however, will depend upon the entry in the repetition vector for the updater node.

As defined in [10], for every consistent and strongly connected MRDF graph, the throughput of MRDF graph is equal to the number of actor firings per time unit during

one period divided by the entry corresponding to that actor in the repetitions vector. This in turn is equal to the number of iterations executed in one period divided by the duration of one period. Using this definition, it is possible to calculate the throughput of the input graph using the simulator. The time difference between the same simulator state visited twice in the simulator is nothing but the period P of the periodic execution of the graph. The number of firings n_{fires} of any actor in the time interval in which the same state was detected during the simulation divided by the entry corresponding to this actor in the repetitions vector q would give us number of firings of this actor per cycle. Then the throughput can be calculated as $\frac{(\frac{n_{fires}}{q})}{P} = \frac{n_{fires}}{q.P}$. MCM is the reciprocal of the throughput (based on Section 2.3.2).

As per the monotonicity of the self-timed execution [20], the start times of the self-time schedule obtained for a graph (V, E) with actors represented by their worst case execution time provide an upper bound to all self-time schedules. $s_{STS}(i, k) \leq s_{WCSTS}(i, k) \quad \forall i \in V$. Thus, if we use a graph with actors represented with their worst case execution times for simulation, then we can say that the simulation is conservative.

6.2 Features added to Heracles

In this section, we describe the software features that were added to Heracles. We first discuss the implementation details of the multi-rate model based analysis graph builder. We then discuss about the simulation approach to analysis.

6.2.1 Analysis graph builder for multi-rate model

In Heracles, the analysis graph is built using the LR model (Section 3.5.1.2). We wanted to evaluate the multi-rate model against the existing LR model. Hence, we added an analysis graph builder to Heracles which builds the analysis graph using the multi-rate model. We can thus compare the performance of the two models. It also gives the user of this tool a choice of using either of the two models based upon his or her requirements. A command line option was added to the software tool to enable the user to select the analysis graph builder of his choice - either LR or multi-rate model. The new analysis graph builder takes an application graph and a schedule for the application graph actors as input. The analysis graph builder uses these inputs and generates an analysis graph. Let us now visit the internal details of the new analysis graph builder - the initiation phase and the execution phase.

6.2.1.1 Initiation phase of the analysis graph builder based on the multi-rate model

Heracles defines complex data structures to represent actors, edges, graphs, system of processors and schedules. As a initiation step, before invoking the scheduler, the analysis graph builder is also presented with the MPSoC system information and the application graph, so that the internal data structures are updated to store this information. During this phase, actors and edges internal to the individual multi-rate model of the actors are also created, so that the software can have faster execution during the actual scheduling

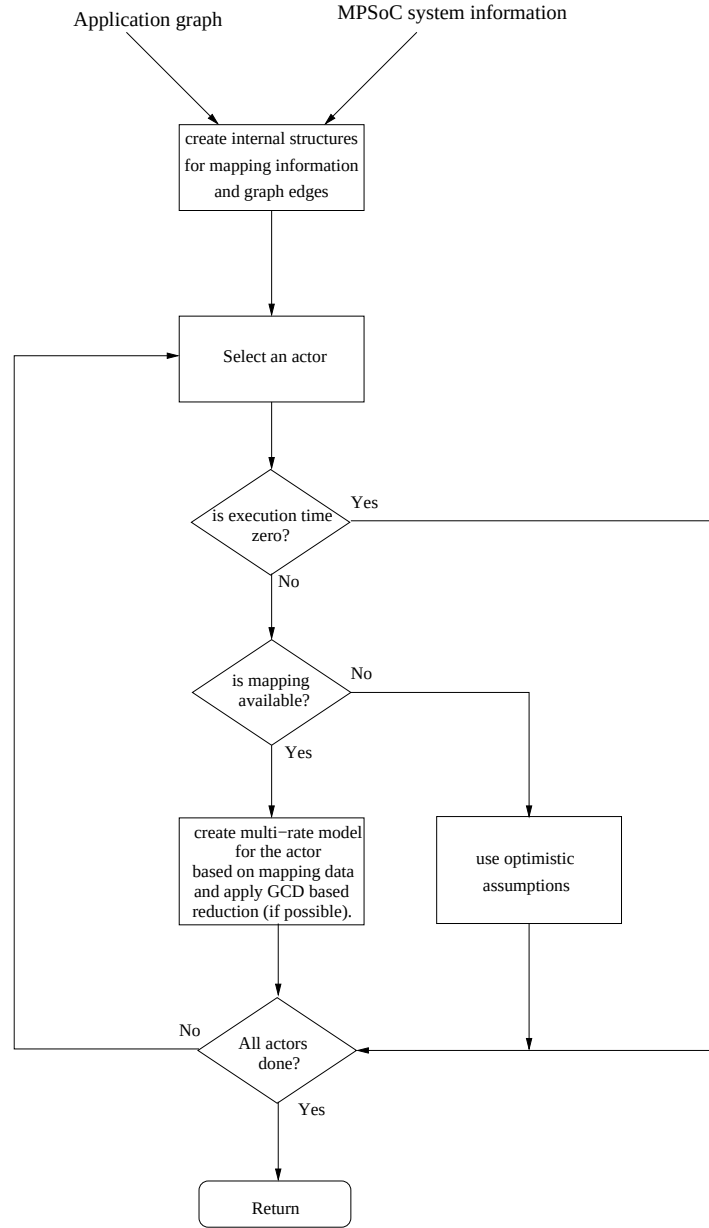


Figure 6.5: Simplified flowchart for the initiation phase

phase. For example, the actors and the edges shown in black colour in the Figure 6.2(b) are created and stored in the internal data structures. Figure 6.5 explains a detailed flow of the initiation phase.

6.2.1.2 Optimistic assumptions

Before we describe the execution of the new analysis graph builder, we present a problem and a key design choice that we had to make to overcome it and the reasons behind doing

so.

The multi-rate model is constructed based on three parameters: the execution time of the actor being modelled τ , the period of TDM scheduler P running on the processor, and the slice time S allotted to the actor. If an actor is unmapped, we do not have any information about the period P and the slice time S . How can we construct the multi-rate model in such cases? This problem arises during the initiation phase as well as execution phase. In the initiation phase, we cannot create actors and edges internal to the multi-rate model, if we do not have any mapping information about that actor. Similarly, during the execution phase, during each invocation of the graph builder, a complete graph has to be provided to the scheduler for analysis. The scheduler maps one actor at a time. Hence, the question arises as to how can we construct complete analysis graph when mapping information of some of the actors is still unavailable?

In order to overcome this problem, we expect that each actor to have at least the processor type defined. Once this is done, we have at least some information about the actor's mapping. Then, we make an optimistic assumption that the actor communicates without any communication overhead. This is a temporary adjustment made such that we do not restrict any solution due to pessimistic assumptions since we generate and analysis graph after each mapping. When the mapping for all actors becomes from the scheduler, we generate an accurate analysis graph without any assumption [20].

Let us see an example. Let us assume we have an unmapped actor B which gets an input token from actor A. Actor A is already mapped to processor 1. Both A and B have same processor type and there are two processors of that type. In this case, we assume that actor B is mapped such that the overhead for communication between A and B is zero. If we had mapped B such that communication overhead is not zero, we might miss some feasible solutions during the analysis phase due to such pessimistic assumption. [20].

6.2.1.3 Execution phase of the analysis graph builder based on the multi-rate model

Figure 6.6 explains a detailed flow of the execution phase. As a first step, the current schedule is checked in order to figure out if there is any backtracking done by the scheduler. Backtracking (also discussed in Section 6.1.2) means that the scheduler has changed the mapping of an already mapped actor and no new actor was mapped in the current schedule. Backtracking is done by the scheduler if the analysis had failed and there was a scope to change the mapping of an already mapped actor. The analysis graph builder detects the backtracking to clear its internal structures since it maintains active information about the previously generated analysis graph. If backtracking is detected then all the internal database is refreshed. If there is no backtracking, then no action is taken, except that the new mapping information is extracted from the schedule. The analysis graph builder maintains active information about the old analysis graph in order to make the execution of the common case faster. Backtracking is a relatively less probable phenomenon and hence, maintaining some information about the previous graph results in the faster execution of the analysis graph builder.

Later, the static ordering of actors is detected. Static order is nothing but an ordered

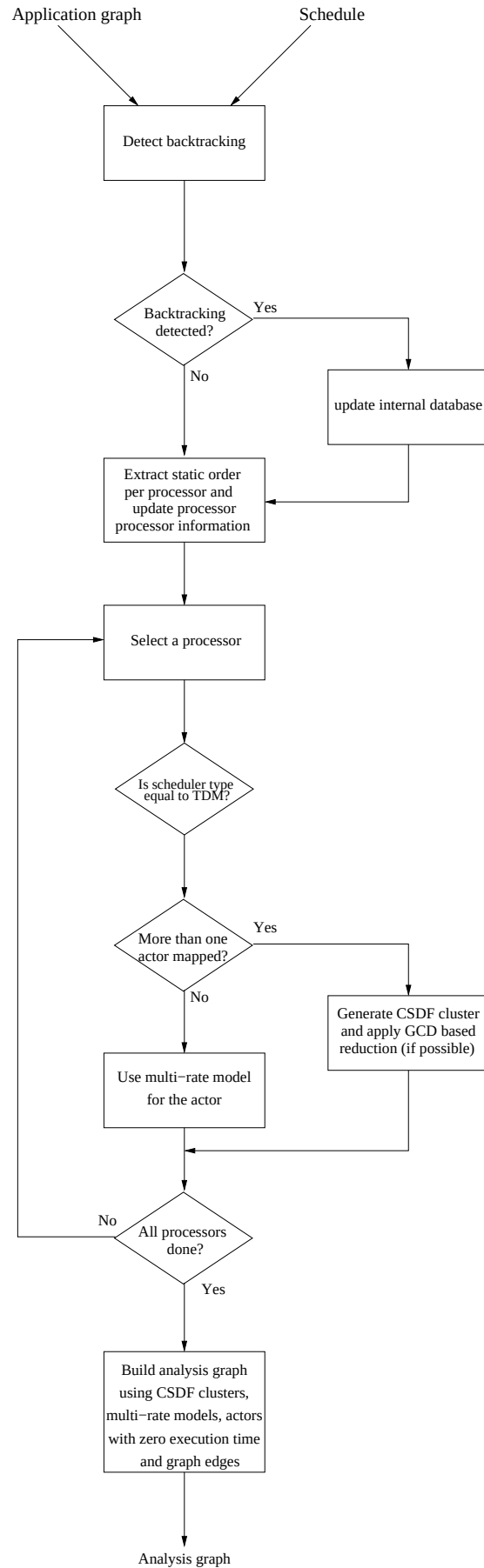


Figure 6.6: Simplified flowchart for the execution phase

list of actors mapped on the same processor. In any system, there will be multiple static orders, one for each processor. For each static order, the graph builder checks if the scheduler type is TDM. If the scheduler type for that processor is TDM, then it creates a CSDF graph cluster as shown in Figure 4.5. If the scheduler type is not TDM then, a simple model is created for each actor in the static order. For a processor having a single actor mapped onto it, it creates a multi-rate model as shown in Figure 4.3, if the processor is using a TDM scheduler, else a simple model is created corresponding to that actor. Also, for unmapped actors, models are created per actor with optimistic assumptions. Once all these sub-models are created per processor and per actor (for unmapped actors and actors with non-TDM schedulers), these sub-models are joined together using the edges in the original application graph, giving us a complete analysis graph.

The analysis graph obtained in the previous step is in-fact a CSDF graph. It is not analysable as in its existing form. It has to be converted to its equivalent SRDF graph before it can be analysed. The analysis is done (as seen in Section 6.1.3) using MCM algorithm. This algorithm works only with SRDF graphs. Hence, the conversion of the analysis graph to its equivalent SRDF graph is necessary.

Actors with zero execution time are treated differently. No model is created for such actors. Only while building the final analysis graph, actors with zero execution time are added in the analysis graph. It is an implicit requirement because the multi-rate model has the production rate of actor s and consumption rate of actor c equal to the actor execution time. The model is no longer a valid MRDF graph if the production or consumption rate of certain actors is zero.

We perform an optimization so that the size of the analysis graph i.e. number of actors and edges can be reduced. Some of the optimizations done while building the analysis graph are as follows: Greatest common divisor based model reduction is used, both for single actor models as well as for CSDF clusters. If $gcd(S, \tau) > 1$, then we use an equivalent model with parameters $\tau' = \frac{\tau}{gcd(S, \tau)}$, $S' = \frac{S}{gcd(S, \tau)}$ and $t'_x = t_x \cdot gcd(S, \tau)$ instead of using the conventional multi-rate model. This model, when converted to single rate by the analyser results into fewer actors and edges. This optimization is also applicable to CSDF static order model, if the $gcd(S, \tau_1, \tau_2, \tau_3, \dots, \tau_h)$ should be greater than 1, if h actors are in the static order. We provide an intuitive proof for the greatest common divisor based model reduction in the next section.

6.2.1.4 Greatest common divisor (GCD) based reduction

In this section, we prove that the greatest common divisor based reduction described as an optimization in the earlier section is exact. The symbols and definitions are as given in Section 5.1.

If $g = gcd(S, \tau) > 1$, then we can use an equivalent model with parameters $\tau' = \frac{\tau}{g}$, $S' = \frac{S}{g}$ and $t'_x = t_x \cdot g$. We now prove that the two models are equivalent. Equation 5.73 gives the finish times of all iterations for a multi-rate model in case of a burst input.

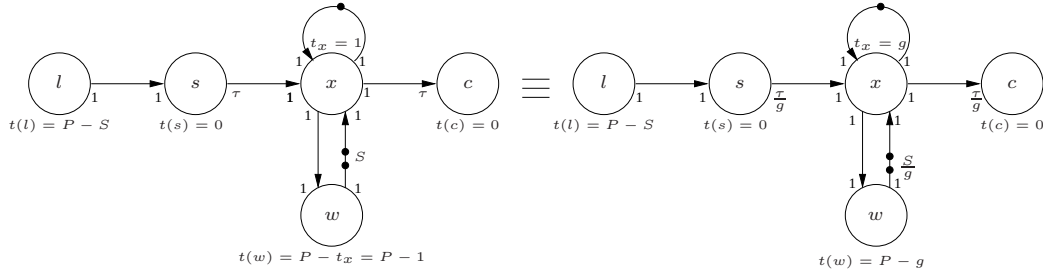


Figure 6.7: Equivalent model for GCD based reduction

The equation for the reduced multi-rate model can be written using Equation 5.73 as

$$f(c, k) = \begin{cases} s(l, 0) + P - S + (k + 1) \cdot t'_x \cdot \tau' & \text{if } 0 \leq k < \lfloor S'/\tau' \rfloor \\ s(l, 0) + P - S + \lfloor \frac{(k+1)\tau'}{S'} \rfloor P + ((k+1)\tau' - \lfloor \frac{(k+1)\tau'}{S'} \rfloor S') \cdot t'_x & \text{if } \lfloor S'/\tau' \rfloor \leq k < q' - 1 \\ s(l, 0) + P - S + \lfloor \frac{q'\tau' - S'}{S'} \rfloor P + (q'\tau' - \lfloor \frac{q'\tau' - S'}{S'} \rfloor S') \cdot t'_x & \text{if } k = q' - 1 \\ f(c, (k - (S'/g'))) + \tau' P/g' & \text{if } k \geq q' \end{cases} \quad (6.1)$$

where, $g' = \gcd(S', \tau')$.

Here, $g' = \gcd(S', \tau') = 1$, since the reduction by gcd of S and τ has already accounted for the common factors.

$q' = \frac{S'}{g'} = \frac{S/g}{g'} = \frac{S/g}{1} = q$. Substituting the values of t'_x , S' , τ' , g' and q' in (6.1), we get

$$f(c, 0) = \begin{cases} s(l, 0) + P - S + (k + 1) \tau \cdot t_x & \text{if } 0 \leq k < \lfloor S/\tau \rfloor \\ s(l, 0) + P - S + \lfloor \frac{(k+1)\tau}{S} \rfloor P + ((k+1)\tau - \lfloor \frac{(k+1)\tau}{S} \rfloor S) \cdot t_x & \text{if } \lfloor S/\tau \rfloor \leq k < q - 1 \\ s(l, 0) + P - S + \lfloor \frac{q\tau - S}{S} \rfloor P + (q\tau - \lfloor \frac{q\tau - S}{S} \rfloor S) \cdot t_x & \text{if } k = q - 1 \\ f(c, (k - (S/g))) + \tau P/g & \text{if } k \geq q \end{cases} \quad (6.2)$$

Comparing Equations 6.2 and 5.73, the reduced model gives exactly the same finish times as the original model. The MCM of the reduced model is $\frac{\tau'P}{S'} = \frac{\tau P}{S}$, which is same as the original model. Thus, the reduced model matches the original model exactly in latency and throughput and hence is accurate.

This technique is also applicable to the CSDF static order model. If $\gcd(S, \tau_1, \tau_2, \dots, \tau_h) > 1$, then each of the execution time of each of the h actors in a static order as well as the slice time can be divided by $\gcd(S, \tau_1, \tau_2, \dots, \tau_h)$.

During MRDF to SRDF conversion, we get τ copies of x and w actors with edges interconnecting them. When we use this reduction, we actually reduce the number of w and x actors by a factor of $\gcd(S, \tau)$ per MRDF actor. Also the edges in the SRDF equivalent graph are reduced by a factor of $2 \cdot \gcd(S, \tau)$. With reduced size, the MRDF to SRDF conversion time as well as the analysis time (MCM computation time) method is reduced.

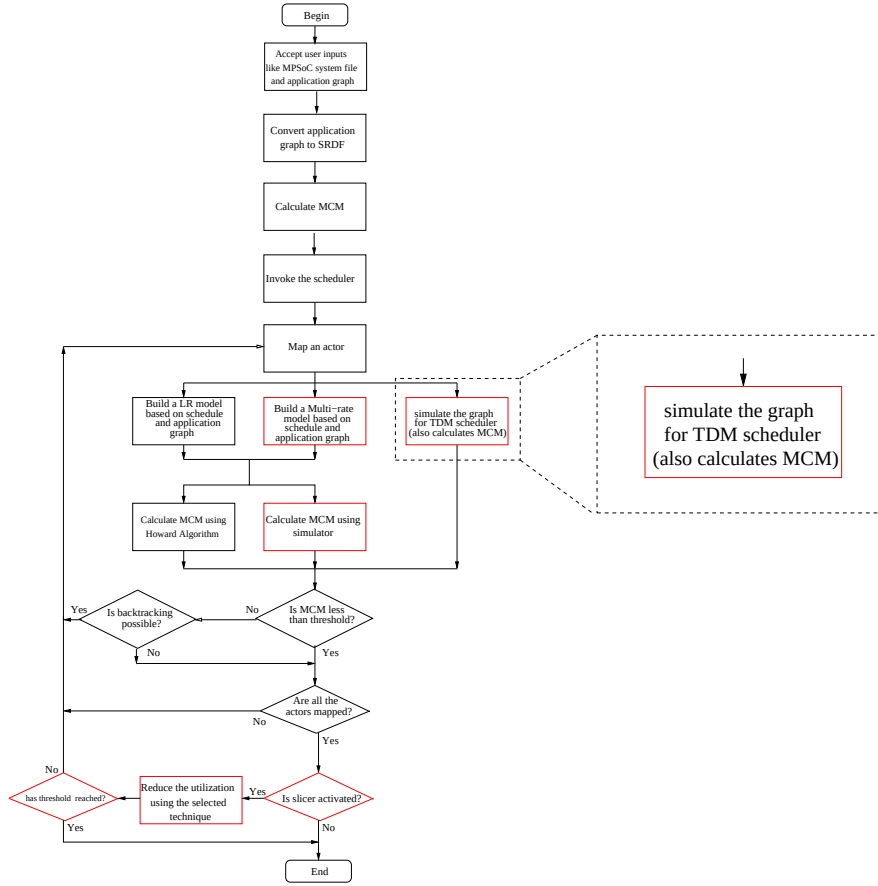


Figure 6.8: Position of Time wheel simulation in the Heracles

6.2.2 Time wheel simulation

Time wheel simulation is nothing but simulation of the application graph to predict the finish times of its actors for an iteration of application graph under constraints that the actors is scheduled on TDM schedulers. (Please note that an iteration can be number of firings of an actor, depending upon the type of application graph). The simulator mentioned in Section 6.1.4 is a graph simulator for self-time schedule. Time wheel simulation is a way to simulate the behaviour when actors are scheduled on a TDM scheduler.

This method can be used for temporal analysis in lieu of analysis graph builders and Howard algorithm because it is claimed that the model is exact in terms of predicting the finish times for the TDM scheduler [14]. This should lead to least utilization The TDM scheduler with period P , slice S and arrival time T and start of the slice ϕ can all be used to write equations to calculate the finish times of actors that are running on a processor with TDM scheduler. In Chapter 3, Section 3.4 we have seen such a way of representing a single execution of an actor on a TDM scheduler. These equations can be used to simulate the graph after each mapping of the scheduler instead of using the analysis model. A simulator was built with these equations.

A new command line option was added so that the time wheel simulation based analysis can be used. When this option is set, the scheduler invokes the time-wheel simulator which simulates the TDM behaviour the partially mapped application graph at every step. The simulator is same as the event based simulator discussed above. The only difference is that the finish events are issued based on the calculations of the finish time done using Equation 3.4. Figure 6.8 shows the position of the time wheel simulation in the Heracles's flow. It provides an alternate method of analysis and hence a simulator related to it was developed.

However, there is no proof to suggest that the simulation based approach is conservative.

6.2.3 Optimizing schedules by rescheduling

We have seen that the scheduler tries to find a schedule to fit the maximum utilization provided by the user. Existence of a schedule does not mean that a better schedule is not possible. A schedule is considered better if the utilization of processors can be reduced further. In this section we present some techniques that help optimizing schedule for less utilization of each processor. There are two methods of reducing the utilization. In the first method, we try to find alternate feasible schedule with lesser utilization of the processors. The other method can be used to reduce the utilization of processors with the existing schedule. These methods are called slicers [20] because they search for reduced (better) processor utilization (slice time). Binary slicer, randomized slicer and randomized weighted slicer are the three methods for the former technique [20]. Later, we will see the similar methods for the latter. This is discussed in Section 6.2.4. All the methods for finding optimized schedules can be used with any TDM model. Figure 6.9 shows the position of the slicers in the flow of Heracles.

6.2.3.1 Binary slicer

In the beginning, the user gives a desired maximum utilisation for each processor. Based on the given utilisation, the scheduler checks if a feasible schedule can be found or not. If a solution exists, then we perform a binary search for reducing the utilization of each processor with TDM schedule. Starting with a processor, we set its maximum utilization to half the sum of its current utilization value and a small threshold value and again invoke the scheduler. If a schedule is still found, then we further reduce the utilization to half the sum of its current utilization value and a small threshold value. If a schedule is not found, we set the utilization to half the sum of its current value and its value when the schedule was last found. This way, we prune the range of the search in each iteration and eventually stop when the difference between the two ends (terms used in the sum) is less than the threshold. The process is repeated for all processors with TDM schedule. We call this technique as binary slicer.

In this method, we invoke the scheduler of the Heracles in our search and hence, the schedules (actor processor mappings with static orders) found before and after the binary search may not be same. A point to note here is that the resolution of the slicer is configurable. The threshold value, mentioned above, can be set as a command line option. The resolution decides the runtime of the slicer. If the threshold value is very

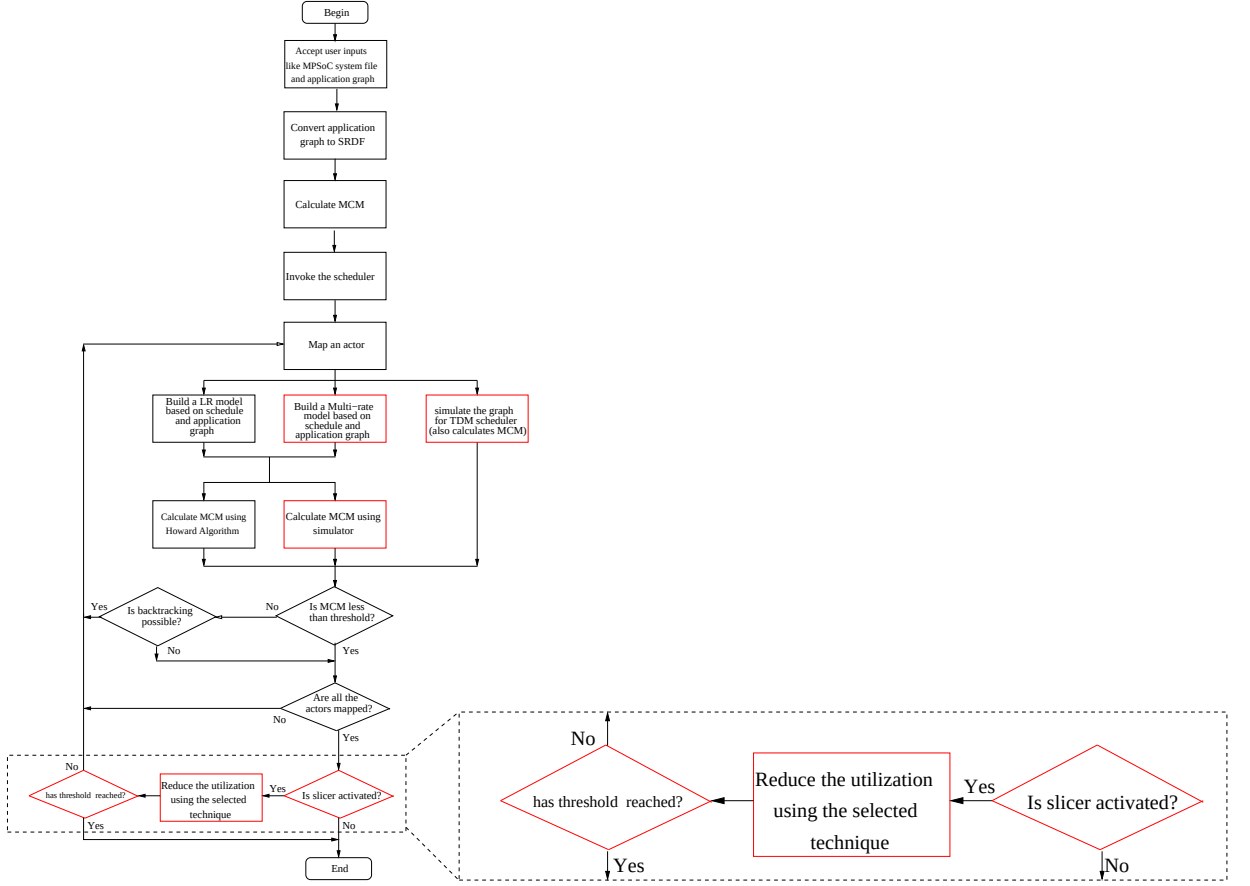


Figure 6.9: Position of slicers in Heracles

small, the number of iterations required to attain the desired accuracy will be high and hence the run time of Heracles will be high.

The schedule found after the binary search is better than the previous but need not be the best. Depending upon the order in which a processor is chosen for reducing its utilization, we reach a local minimum. The question of finding a global minimum is a difficult problem to solve. If there are n processors, then there will be $n!$ ways in which a solution can be achieved, each of which may be a different point in the search space. However, there can be many more points in the search space other than these $n!$ points. The other points cannot be reached using this technique. Hence, we propose a technique called randomized slicer.

6.2.3.2 Randomized slicer

The final set of processor utilizations obtained using a binary slicer is governed by the order in which the processors are selected for slicing down their utilizations. In randomized slicer, we select a processor randomly and then shrink its utilization. In the next iteration, we select again a processor randomly and shrink its utilization. Thus, the process of selection of processors is random with equal probability.

Since the selection of processors is random, it is possible that we reach different schedules or different levels of utilizations, each time this type of slicer is used. Thus, the randomness makes it possible to reach other points in the search space. A new command line option is added to use randomized slicer.

6.2.3.3 Randomized weighted slicer

A user may, sometimes, want that a certain processor should have more shrinkage of its utilization than others. In randomized weighted slicer, a user can assign a weight to a processor. The weight is an indication to the software tool that shrinking the utilization of this processor is more important than others. This attribute is used to bias the randomness of processor selection. If a processor has more weight, it has more probability of being selected for slicing its utilization. The probabilities are calculated in the following way.

$$p_i = \frac{w_i}{\sum_{i=1}^{|P_r|} w_i} \quad \text{where, } P_r \text{ is the set of processors} \quad (6.3)$$

A new command line option is also added to use randomized weighted slicer.

6.2.4 Optimizing the schedules without rescheduling

The three methods described above find new schedules while trying to reduce the processor utilization. In this section, we provide methods to reduce the utilizations of processors without creating new schedules.

We make use of the same three methods, namely, binary search, randomized search and weighted randomized search to reduce the utilizations of the processors with the existing schedules. As a first step, feasible schedules are found with the utilizations provided by the user. For the given schedule, utilization of a processor is reduced using one of the three methods mentioned above. Then the graph builder and analyser are invoked to check if temporal requirements are still met. If temporal requirements are still met, then the process is repeated until no further shrinkage is possible for all processors with a TDM schedule. This process is then repeated for all the feasible schedules.

Optimizing the existing schedules can be done by any one of the three methods by using the command line option.

6.3 Summary

In this chapter, we have discussed the implementation details of Heracles. We first saw the flow of Heracles before the addition of our work into it. We provided an overview of the existing features and the important components of Heracles tool. We also discussed the features that were added to Heracles as part of this thesis. These include software modules for multi-rate model based analysis graph builder, various slicing techniques and the time wheel based simulation. The multi-rate model based analysis technique will be useful to reduce the pessimism in the modelled finish times of actors scheduled on a TDM scheduler. The overall effect is that, we can reduce the utilization of the processors. The

slicing techniques enable us to achieve the reduced utilizations. Simulation of the time wheel provides another way of analysing the graphs. We implemented it to see the benefits of this technique over data flow model based analysis. In the next chapter, experimentally investigate correctness of our analysis presented in Chapter 5 and the conservativity of our model. We also try to show the benefits of lesser pessimism in the multi-rate model for some sample application graphs.

Experimental results

In the previous chapter, we have discussed the implementation details of the multi-rate model based analysis graph builder. In this chapter, we initially present the results of experiments with synthetic applications to demonstrate the basic working of the models. Then we present the result of experiments with real application graphs.

we present the results for a single application actor modelled using the multi-rate model in various cases to demonstrate the conservativity and accuracy of the multi-rate model. These results also indicate that our analysis of the multi-rate model is correct. Furthermore, we study the effect of reduced pessimism in the multi-rate model with the help of the TDS-CDMA and WLAN application graphs running on an assumed MPSoC platform consisting of ARM, EVP and SWC processors (the detailed setup is described in Section 7.2). We compare the arrival and finish times of various actors obtained using the time wheel simulation, simulation of the analysis graph obtained using the multi-rate analysis graph builder and simulation of the analysis graph obtained from the LR analysis graph builder to understand the effect of reduced pessimism. We also present the results of various slicers implemented in order to obtain optimized schedules in terms of reduced processor utilization.

7.1 Single actor response

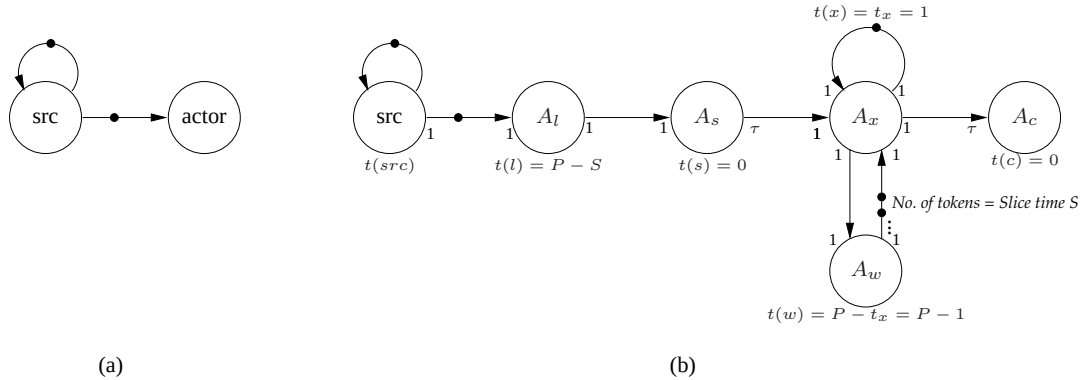


Figure 7.1: Setup for single actor response

In Chapter 5, we have seen that the behaviour of the multi-rate model on arrival of a single token at its input. We have also seen that, on arrival of a burst at its input, output tokens are produced by the multi-rate model, depending upon the rate of arrival of tokens and the its internal state. we did experiments in order to show some examples

and confirm our understanding of the behaviour of multi-rate model. These experiments are indicative but we cover various cases with respect to the relation between S and τ . All the experiments discussed in this section use the values of the arrival times such that the boundary conditions are also tested for accuracy along with the correctness and conservativity of the multi-rate model. We mention the boundary conditions that we test in each of the experiments below.

7.1.1 Setup

The data-flow graph with two actors called *src* (stands for source) and *A* (actor under test) (Figure 7.1) was used a test set-up. Execution time $t(src)$ of the *src* actor was varied to create an effect of variable input rates. The scheduler type of *src* was kept as off so that the *src* does not get modelled by the multi-rate model. The scheduler type for the actor *A* was selected as TDM. The period P , the slice time S of the scheduler and the actor execution time τ were varied to evaluate the model under various possibilities. The data flow graph in Figure 7.1(a) was used as a input to the multi-rate model based analysis graph builder section and the analysis graph in Figure 7.1(b) was obtained. In order to force the first input at time = 0, there is a delay token on the edge (*src*, *A*).

Then, the graph in Figure 7.1(a) was simulated to obtain the start times and the finish times of actor *A* by running time wheel simulation using the same values of P , S and τ as used with the multi-rate model. The timings obtained were later compared with the timings obtained from the multi-rate model.

7.1.2 Impulse response results

This section checks the equations derived in Section 5.2.3 using the simulation of the analysis graph mentioned above. It also provides a comparison between the exact arrival times predicted by the time wheel simulation. These experiments also verify if the boundary condition $a(k) \geq (c, k - 1)$ for the impulse response is correct.

7.1.2.1 Experiment 1: Impulse response for $S < \tau$

In this experiment, we create the condition for impulse response i.e. ($a(k) \geq f(c, k - 1)$) when $S < \tau$. Thus, we choose the arrival time $a(k) = f(c, k - 1)$. This is the boundary condition that we are testing in this experiment. We choose $S = 5$, $\tau = 7$, $P = 10$ so that we get $t(A_l) = 5$, $t(A_x) = 1$, $t(A_w) = 9$, $t(A_s) = t(A_c) = 0$. The boundary condition for this case will $a(k) \geq 17$. Hence, $t(src) = 17$. With $t(src) = 17$, tokens will be arriving at the input of actor *A_l* at times 0, 17, 34, 51, etc. The Table 7.1 shows the data collected using simulation of the analysis graph based on multi-rate model for the setup in Figure 7.1 and the time wheel simulation for the given values of S , P and τ

7.1.2.2 Experiment 2: Impulse response for $S \geq \tau$

In this experiment, we try to create a condition for impulse response i.e. ($a(k) \geq f(c, k - 1)$) when $S \geq \tau$. Thus, we choose the arrival time $a(k) = f(c, k - 1)$. This is the

arrival times	finish times with		
	simulated time wheel	Multi-rate model	LR model
0	17	17	19
17	29	34	36
34	47	51	53

Table 7.1: Impulse response experiment 1: $S < \tau$

boundary condition that we are testing in this experiment. We change the value of slice time to 17 and the value of period to 20 i.e. $S = 17$, $\tau = 7$, $P = 20$ so that we get $t(A_l) = 3$, $t(A_x) = 1$, $t(A_w) = 19$, $t(A_s) = t(A_c) = 0$. In this case, the condition for impulse response is such that the difference between the two consecutive arrivals should be greater than or equal to 10. Hence, we select $t(src) = 10$ as a border case. There will be tokens arriving at the input of the actor A_l at times 0, 10, 20, 30...and so on. The Table 7.2 shows the data collected using simulation of the analysis graph for the setup in Figure 7.1 and the results obtained time wheel simulation.

arrival times	finish times with		
	simulated time wheel	Multi-rate model	LR model
0	10	10	12
10	17	20	22
20	30	30	32
30	37	40	42

Table 7.2: Impulse response experiment 2: $S \geq \tau$

The values of the finish times obtained from the simulation of the analysis graph are greater than or equal to those obtained from the simulation of the time wheel, both in Experiment 1 (refer to Table 7.1) and Experiment 2 (refer to Table 7.2). Also, in some cases, these results match exactly to each other. With the two experiments above, we cover all the cases for single arrival response of the multi-rate model. We can also see the delay experienced by the token inside the multi-rate model is same irrespective of its arrival time when the condition $a(k) \geq f(c, k - 1)$ is satisfied. It can be also seen that the time predicted by the LR model are more pessimistic than the multi-rate model.

We ran independent simulations of the analysis graph generated by the multi-rate model based analysis graph builder. The finish times predicted by the multi-rate model were either same or greater than the ones predicted by the simulation of the time wheel. Hence, the impulse response and the condition for impulse response have been tested and appear to be correct.

7.1.3 Burst Response

This section demonstrates that the burst response of the multi-rate is conservative using the simulation of the analysis graph and comparing its results with simulation of the time wheel. We also demonstrate that the condition for the burst response $a(k) + P - S < f(c, k - 1)$ is correct.

arrival times	finish times with		
	simulated time wheel	Multi-rate model	LR model
0	17	17	19
11	29	29	33
22	46	46	47
33	58	58	61
44	70	70	75
55	87	87	89
66	99	99	103
77	116	116	117
88	128	128	131
99	140	140	145
110	157	157	159

Table 7.3: Burst response experiment 3: $S < \tau$

7.1.3.1 Experiment 3: Burst response for $S < \tau$

In this experiment, we use the same values of P , S and τ that were used in Experiment 1 in Section 7.1.2.1. Only the value of $t(src)$ is changed to reflect the bursty input. In this case, the condition for bursty input $a(k) + P - S < f(c, k - 1)$ becomes $a(k) < 12$.

Thus, $S = 5$, $\tau = 7$, $P = 10$ so that we get $t(A_l) = 5$, $t(A_x) = 1$, $t(A_w) = 9$, $t(A_s) = t(A_c) = 0$. We choose $t(src) = 11$. There will be tokens arriving at the input of actor A_l at times 0, 11, 22, 33, 44 etc. The Table 7.3 shows the data collected using simulation of the analysis graph for the setup in Figure 7.1 and the results obtained using time wheel simulation.

Here $q = \frac{S}{gcd(S, \tau)} = \frac{5}{1} = 5$. The finish times of the first 5 ($k = 0$ to $k = 4$) iterations are unique. The finish times for $k \geq 5$ can be seen as a periodic repetition of the first 5 iterations, the period being $\frac{\tau P}{gcd(S, \tau)} = 70$. For example, $f(c, 5) = 17 + 70 = 87$, $f(c, 6) = 29 + 70 = 99$. The finish times of the first q iterations obtained using the multi-rate model are exactly same as those obtained from the simulation of the time wheel. Thus, the finish times are not just conservative but also accurate in this case.

7.1.3.2 Experiment 4: Burst response for $S \geq \tau$

In this experiment, we use the same for P , S and τ as the ones used in Experiment 2 in Section 7.1.2.2. Only the value of $t(src)$ is changed to reflect the bursty input. In this case, the condition for bursty input $a(k) + P - S < f(c, k - 1)$ becomes $a(k) < 7$. We choose $t(src) = 6$.

For this example, $S = 17$, $\tau = 7$, $P = 20$, $t(A_l) = 3$, $t(A_x) = 1$, $t(A_w) = 19$, $t(A_s) = t(A_c) = 6$. Since $t(src) = 6$, tokens will be arriving at the input of actor A_l at times 0, 6, 12, 18, 24 etc. The Table 7.4 shows the data collected using simulation of the analysis graph for the setup in Figure 7.1 and the results obtained using time wheel simulation.

For this experiment, the value of $q = 17$. The finish times of the first q ($k = 0$ to $k = 16$) iterations are unique. We can see that the finish times of the latter iterations are a periodic repetition of the first q iterations, the period being $\frac{\tau P}{gcd(S, \tau)} = 140$. For example,

arrival times	finish times with		
	simulated time wheel	Multi-rate model	LR model
0	10	10	12
6	17	17	21
12	27	27	30
18	34	34	39
24	44	44	48
30	51	51	57
36	58	58	66
42	68	68	75
48	75	75	84
54	85	85	93
60	92	92	102
66	99	99	111
72	109	109	120
78	116	116	129
84	126	126	138
90	133	133	147
96	140	140	156
102	150	150	165
108	157	157	174

Table 7.4: Burst response experiment 4: $S \geq \tau$

$$f(c, 17) = 10 + 140 = 150, f(c, 18) = 17 + 150 = 157.$$

We ran independent simulations of the analysis graph generated by the multi-rate model based analysis graph builder and of the time wheel. The finish times obtained from the simulation of this analysis graph suggests they are conservative towards the finish times obtained from the time wheel simulation results, in the all the experiments for the burst response. From Figure 7.2 and Figure 7.3, we can see that the finish times predicted by multi-rate model are in fact same as that obtained from the time wheel simulation, both Experiment 3 and Experiment 4. From Figure 7.3, we can also see that using the LR model, the difference in the actual and the predicted finish time may increase linearly with time.

7.2 Application graphs used for evaluation

In this section, we describe the two actual application graphs that are used in the SDR domain to demonstrate the effect of reduced pessimism in the multi-rate model. Let us assume that we have a multiprocessor platform as shown in Figure 1.2 for baseband processing with an ARM core as a general purpose processor for handling the control part of the baseband processing, an EVP [28] as a vector processor for detection and demodulation and a application specific codec SWC for coding and decoding operations.

The platform mentioned above is used to handle the TDS-CDMA and WLAN applications. The application graphs in the Figure 7.15 and Figure 7.4 show the actual application graphs for TDS-CDMA and WLAN respectively. The actors in these graphs

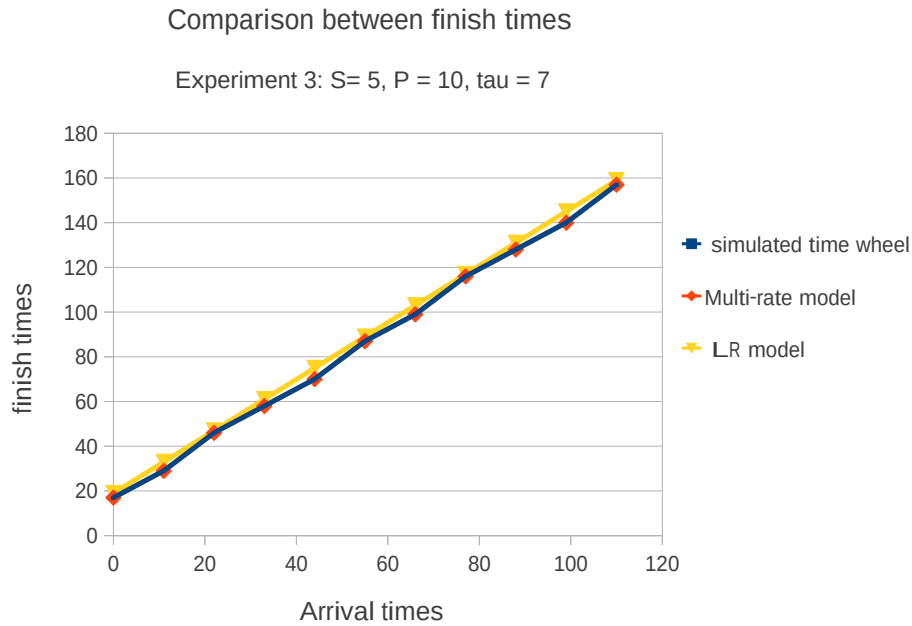


Figure 7.2: Arrival time vs Finish time plot for experiment 3

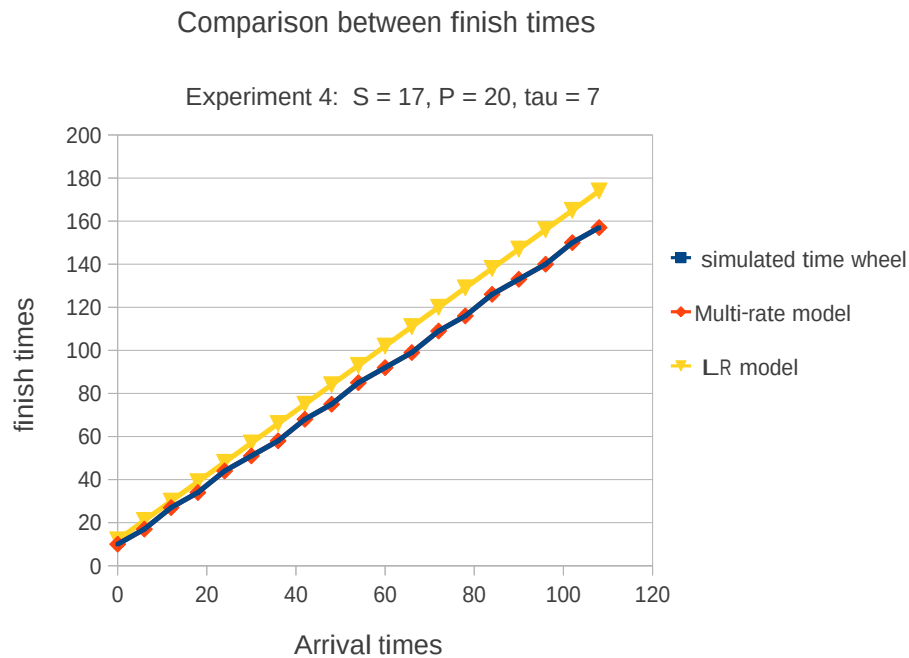


Figure 7.3: Arrival time vs Finish time plot for experiment 4

Actors	start time (ns)	finish time (ns)	Duration (ns)	start time (ns)	finish time (ns)	Duration (ns)
rx1	0	275000	275000	675000	950000	275000
rx2	275000	287500	12500	950000	962500	12500
rx3	287500	387500	100000	962500	1062500	100000
ce	387500	461200	73700	1062500	1135200	72700
rx4	387500	675000	287500	1062500	1350000	287500
mi	461200	554900	93700	1135200	1228900	93700
dass	554900	597000	42100	1228900	1271000	42100
latency1	597000	882680	285680	1271000	1556680	285680
jd1	597000	657500	60500	1271000	1331500	60500
decodecrc1	657500	713300	55800	1331500	1387300	55800
tfci	657500	662800	5300	1331500	1336800	5300
jd2	675000	709200	34200	1350000	1384800	34800
tpc	709200	711300	2100	1384800	1386900	2100
latency2	711300	996800	285500	1386900	1672400	285500
decodecrc2	713300	769100	55800	1387300	1443100	55800
latency3	769100	1164100	395000	1443100	1838100	395000

Table 7.5: Start times and finish times for two iterations of time wheel simulation for TDS-CDMA

also model the source and the maximum latency for a sporadic source [20]. The WLAN application graph is a simplified version of the true WLAN model. In a real WLAN model the PDeMode (Payload Demodulator) varies from 1 to 255. In our example, we are using only one PDeMode actor. The 'Rx' actors model the source. The 'Latency' actors convert the latency into throughput requirements [20]. This graph has a $MCM = 40000.00 \text{ ns}$. Similarly, for the TDS-CDMA application graph, The 'Rx' actors model the source. The 'Latency' actors convert the latency into throughput requirements. This graph has a $MCM = 675000.00 \text{ ns}$. These latency and source actors are not scheduled using any scheduler.

The period of the processors is chosen as 2000.00 ns . The choice of period is based on the assumption that we want to have the upper bound of 100 ns for the context-switch time for all processors and we would want to limit the time spent on context-switch as 10% of the total time [20].

7.3 Effect of reduced pessimism in multi-rate model

The application graphs and the processor platform description mentioned above are used as a input to Heracles. Temporal analysis (discussed in Chapter 6, Section 6.1.2 and Section 6.1.3) of the application graphs is carried out using three methods - LR model based analysis, multi-rate model based analysis and time wheel simulator based analysis. At each step, the MCM is calculated using the simulation of the analysis graph for the LR model and multi-rate model case. For time wheel simulation, the actual application graph is simulated. The flowchart in Figure 7.5 explains the flow used for obtaining these results. The three paths in the flowchart depict the flow for the three techniques mentioned above for obtaining the results.

The result of the last iteration step of the scheduler (the step when all the actors are scheduled) is logged. The start time, finish time and the duration (end time - start time) is given in the Tables 7.5, 7.6 and 7.7 for each actor in the TDS-CDMA application graph.

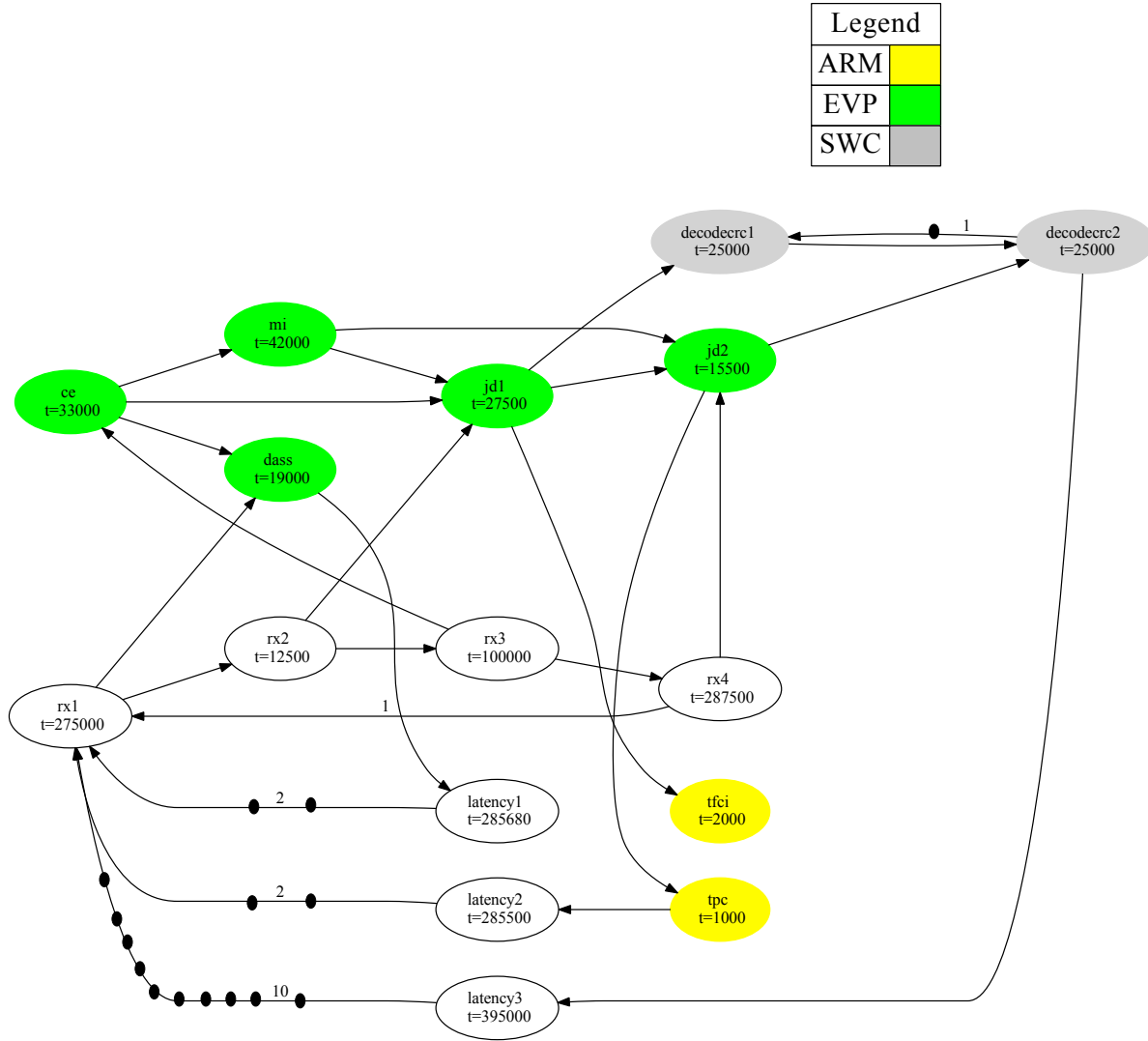


Figure 7.4: TDS-CDMA application graph

For the Table 7.6, only the start time of the splitter node s and finish time of the collector node c are tabulated in front of the actor. The latency node l for each actor is a separate entry in the table.

In order to compare these results visually, we present the Gantt chart depicting the modelled execution time of various actors obtained using the three methods in Figure 7.16, Figure 7.17 and Figure 7.18 for Table 7.5, Table 7.6 and Table 7.7 respectively. The Gantt chart were obtained using TimeDoctor, a open source tool for task traces visualization [2].

For all actors, the modelled execution time (duration of time for which the actor is shown as executing) by the LR model is higher than the modelled execution time for an actor by the other two methods i.e. multi-rate model and time wheel simulation. This is the effect of higher pessimism in the LR model.

Actors in multi-rate analysis graph	start time (ns)	finish time (ns)	Duration (ns)	start time (ns)	finish time (ns)	Duration (ns)
rx1_L	0	0	0	999000	999000	0
rx1	0	275000	275000	675000	950000	275000
dass_L	275000	276100	1100	950000	951100	1100
rx2	275000	287500	12500	950000	962500	12500
jd1_L	287500	288600	1100	962500	963600	1100
rx3	287500	387500	100000	962500	1062500	100000
ce_L	387500	388600	1100	1062500	1063600	1100
rx4	387500	675000	287500	1062500	1350000	287500
ce	388600	461200	72600	1063600	1136200	72600
mi	461200	554900	93700	1136200	1229900	93700
dass	554900	597000	42100	1229900	1272000	42100
jd1	597000	657500	60500	1272000	1332500	60500
latency1_L	597000	597000	0	1272000	1272000	0
latency1	597000	882680	285680	1272000	1557680	285680
decodecrc1_L	657500	658600	1100	1332500	1333600	1100
tfci_L	657500	658600	1100	1332500	1333600	1100
decodecrc1	658600	713300	54700	1333600	1388300	54700
tfci	658600	662800	4200	1333600	1337800	4200
jd2_L	675000	676100	1100	1350000	1351100	1100
jd2	676100	710300	34200	1351100	1385300	34200
decodecrc2_L	710300	711400	1100	1385300	1386400	1100
tpc_L	710300	711400	1100	1385300	1386400	1100
tpc	711400	713500	2100	1386400	1388500	2100
decodecrc2	713300	769100	55800	1388300	1444100	55800
latency2_L	713500	713500	0	1388500	1388500	0
latency2	713500	999000	285500	1388500	1674000	285500
latency3_L	769100	769100	0	1444100	1444100	0
latency3	769100	1164100	395000	1444100	1839100	395000

Table 7.6: Start times and finish times for two iterations of multi-rate analysis graph for TDS-CDMA

Actors in LR analysis graph	start time (ns)	finish time (ns)	Duration (ns)	start time (ns)	finish time (ns)	Duration (ns)
mi_L	0	1100	1100	555268	556368	1100
rx4_L	0	0	0	675000	675000	0
rx3_L	0	0	0	387500	387500	0
rx2_L	0	0	0	287500	287500	0
rx1_L	0	0	0	999368	999368	0
rx1_R	0	275000	275000	675000	950000	275000
dass_L	275000	276100	1100	950000	951100	1100
rx2_R	275000	287500	12500	950000	962500	12500
rx3_R	287500	387500	100000	962500	1062500	100000
jd1_L	287500	288600	1100	962500	963600	1100
ce_L	387500	388600	1100	1062500	1063600	1100
rx4_R	387500	675000	287500	1062500	1350000	287500
ce_R	388600	461934	73334	1063600	1136934	73334
mi_R	461934	555268	93334	1136934	1230268	93334
dass_R	555268	597491	42223	1230268	1272491	42223
latency1_L	597491	597491	0	1272491	1272491	0
jd1_R	597491	658603	61112	1272491	1333603	61112
latency1_R	597491	883171	285680	1272491	1558171	285680
decodecrc1_L	658603	659703	1100	1333603	1334703	1100
tfci_L	658603	659703	1100	1333603	1334703	1100
decodecrc1_R	659703	715259	55556	1334703	1390259	55556
tfci_R	659703	664148	4445	1334703	1339148	4445
jd2_L	675000	676100	1100	1350000	1351100	1100
jd2_R	676100	710545	34445	1351100	1385545	34445
decodecrc2_L	710545	711645	1100	1385545	1386645	1100
tpc_L	710545	711645	1100	1385545	1386645	1100
tpc_R	711645	713868	2223	1386645	1388868	2223
latency2_L	713868	713868	0	1388868	1388868	0
latency2_R	713868	999368	285500	1388868	1674368	285500
decodecrc2_R	715259	770815	55556	1390259	1445815	55556
latency3_L	770815	770815	0	1445815	1445815	0
latency3_R	770815	1165815	395000	1445815	1840815	395000

Table 7.7: Start times and finish times for 2 iterations of LR analysis graph for TDS-CDMA

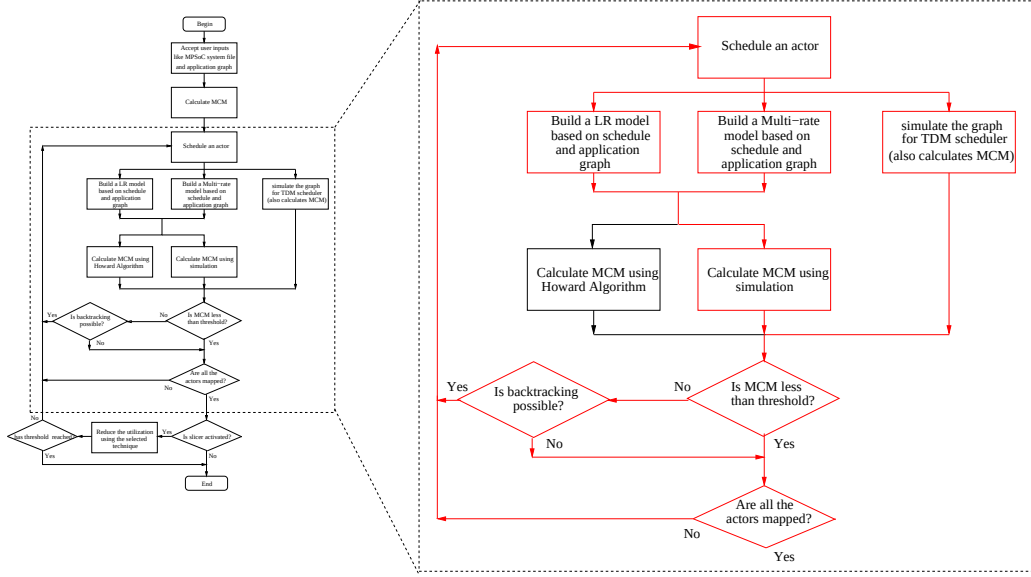


Figure 7.5: Flowchart used for obtaining pessimism results

For all actors, the start times are greater or equal to the start times of the actor obtained from the time wheel simulation. Also for all the actors, the finish times obtained from the time wheel simulation are lower or equal to those obtained from the multi-rate model. Thus, the finish times obtained from the multi-rate model (as well as CSDF static order model) are conservative. This can also be inferred from Figure 7.6, where all the modelled finish times are greater than the finish times obtained from the time wheel simulation. In-fact all the finish times except for the actor 'ce' in the second iteration are exactly equal to the time wheel simulation. The effect of pessimism can be seen from the Figure 7.6. The finish times of the first iteration are closely spaced, but the finish times for the second iteration show increased spacing from the finish times obtained from time wheel simulation for 'dass' and 'latency1' actors. The difference between actual and the predicted start time of first iteration is zero for the multi-rate model 491 ns for the LR model. The difference increases to 1000 ns for the multi-rate model and about 2491 ns for the second iteration.

Similar experiments were carried out for WLAN but the tables and their corresponding Gantt charts are not presented in this thesis due to their exhaustive size. However, we present the graph in Figure 7.7 showing finish times of various iterations of few actors in the WLAN application obtained with the three methods.

The start times as well as the finish times of the actors get delayed due to the pessimistic predictions in both the models. However, the amount of pessimism or the delay in the start and finish times is less in multi-rate model than in LR model.

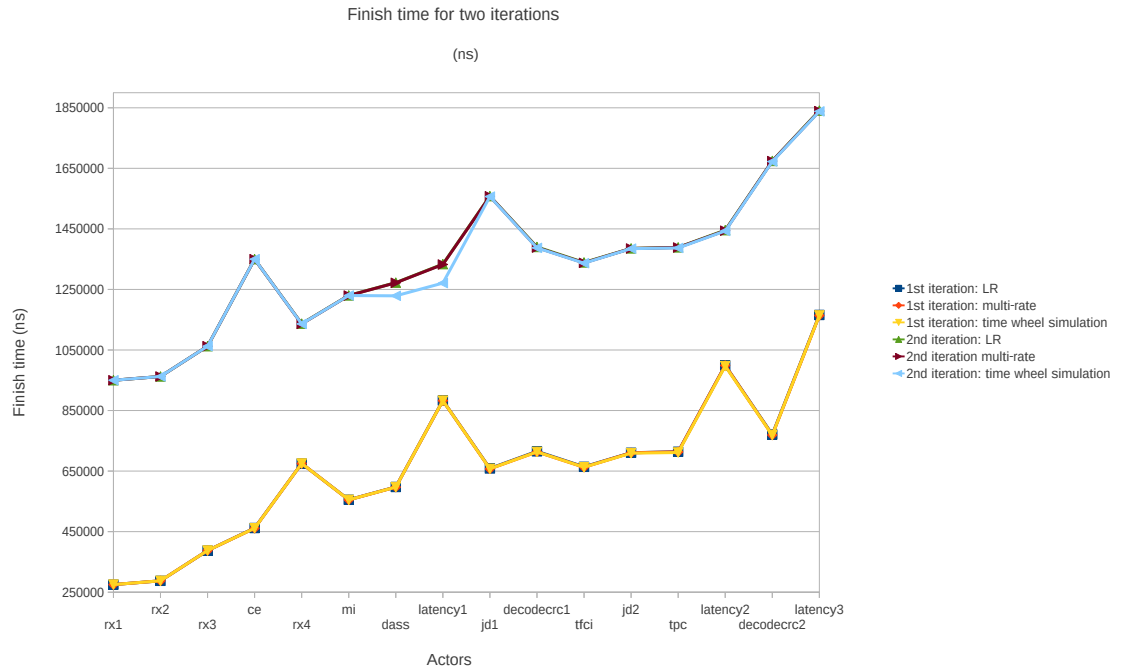


Figure 7.6: Effect of pessimism in the models on the finish times for TDSCDMA

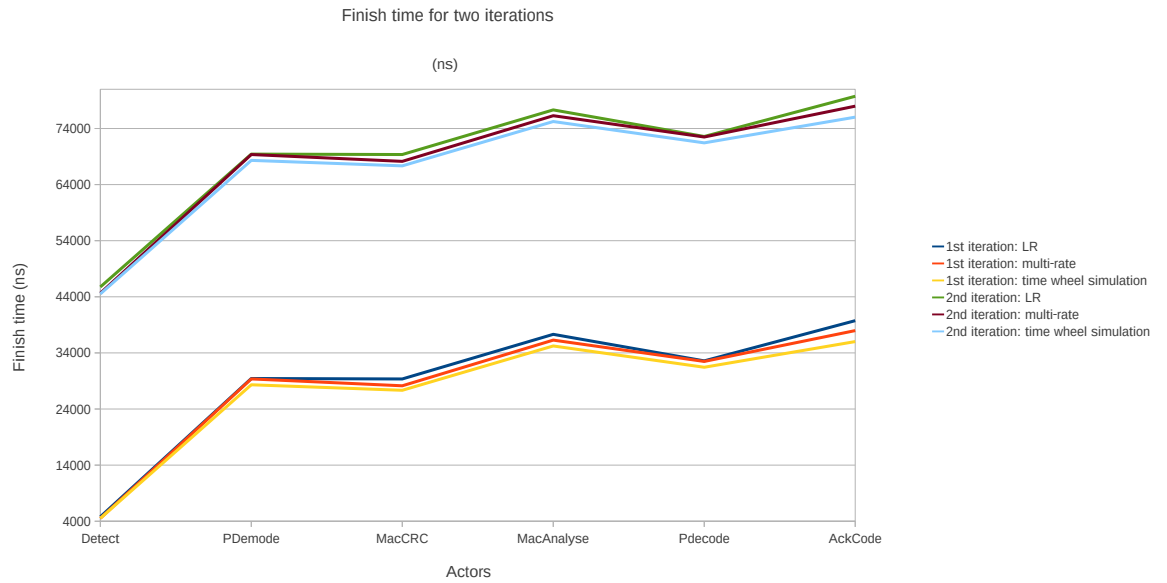


Figure 7.7: Effect of pessimism in the models on the finish times for WLAN

iteration	LR model actors	LR model edges	multi-rate model actors	multi-rate model edges	SRDF actors	SRDF edges
1	39	60	75	115	11975	29865
2	39	60	74	113	11974	29863
3	39	60	73	110	11967	29845
4	39	60	72	107	11922	29732
5	39	62	72	107	11922	29732
6	39	62	70	105	11922	29733
7	39	63	68	103	12922	32234
8	39	65	67	100	5781	14381
9	39	66	65	98	5781	14382
10	39	68	65	98	5781	14382
11	39	70	65	98	5781	14382
12	39	70	63	95	5781	14382
13	39	71	61	94	5781	14384
14	39	71	59	92	5781	14385
15	39	73	58	89	4640	11532
16	39	75	57	86	3851	9559
total	624	1056	1064	1610	133562	332773

Table 7.8: Number of actors and edges handled by the analyser for TDS-CDMA

7.4 Time taken by modelling, conversion and analysis

As discussed in Section 6.2.1.3, the analysis graph obtained from the multi-rate model based analysis graph builder is a CSDF graph. It has to be converted to a SRDF graph before feeding it to the analyser. Hence, there is an additional step involved in using the multi-rate model based analysis. We had predicted that the conversion of the CSDF graph into a SRDF graph is going to take considerably more time. LR model based technique does not have this conversion step and hence no time is spent in the conversion. Once the analysis graph is converted into its SRDF equivalent, we will get large number of actors and edges as compared to LR model based analysis graph. As a result, the analysis time is also going to be higher than the analysis time for the LR model based analysis graph.

We measured the time taken for the three phases - actual modelling, conversion to SRDF and analysis for LR model based analysis and multi-rate model based analysis. We also measured the number of edges and actors in both cases to showcase the added time taken in the by the multi-rate model based analysis. The Table 7.8 shows the number of actors and edges handled by the analyser for the analysis of TDS-CDMA graph. The first column shows the iteration number of Heracles. There are 16 actors in TDS-CDMA application graph. Hence, there are 16 iterations internally, each mapping one actor. The next two columns show the number of actors and edges handled while using the LR model for analysis. The next two columns show the number of actors and edges handled while using the multi-rate model for analysis. These are CSDF actors and edges. The last two columns show the SRDF actors and SRDF edges handled by the analyser, while analysing the multi-rate model based analysis graph. The last row gives the total number of edges and actors handled for analysing the TDS-CDMA application graph once. Thus, the size of the analysis graph is 624 actors and 1056 edges for LR model while the size of the SRDF analysis graph for multi-rate model is 133562 actors

and 332773 edges. Thus, the total number of edges and actors is about 277 times more than the LR model analysis graph.

Similarly, the number of actors and edges handled while analysing the WLAN application graph is 72478 SRDF actors and 176606 SRDF edges, while analysing using the multi-rate model. The corresponding numbers for LR model are 2800 actors and 5222 edges.

TDS-CDMA and WLAN are relatively small application graphs with only 16 and 35 application actors respectively. The size of the SRDF analysis graphs for some other applications like DVB-T receiver presented in [27] can be enormous. The large size of the graph results in a significantly longer analysis time.

The time taken by the LR model based analysis as opposed to the time taken by multi-rate model based analysis is given in Table 7.9 and 7.10. The values are rounded off to 5 decimal places. The timing is measured using the *gettimeofday()* function. This is a standard function for accurate time measurement on Unix/Linux based systems. The results were also verified using the code profiling tools. The time given in these tables is the aggregated time over all iterations (16 for TDS-CDMA and 35 for WLAN).

Figure 7.8 and Figure 7.9 show the time taken by the three phases - modelling, conversion and analysis while using the multi-rate model based analysis. Each graph was analysed number of times. The graphs show the time for 6 such tests. The modelling time is not significant as compared to the conversion time as well as analysis time. The majority of the time is spent in either conversion or analysis.

The modelling time is larger for TDS-CDMA as compared to WLAN even when it has less number of actors. This is mainly because of the backtracking of the scheduler. While generating a schedule for TDS-CDMA, the scheduler backtracks number of times. When backtracking is detected by the analysis graph builder based on the multi-rate model, it cleans up all its internal data structures and starts afresh. As a result, more time is required per invocation of the analysis graph builder. In other words, the scheduler does not backtrack while generating a schedule for WLAN and hence we can see the benefits of our 'common case faster' design of the analysis graph builder.

One more reason for the higher modelling time seen in case of the TDS-CDMA is because of the time taken by the GCD calculation ($gcd(S, \tau_1, \tau_2, \dots, \tau_h)$). The application actors in TDS-CDMA have large execution times are compared to those in WLAN and hence the time taken to calculate the GCD is also high. The GCD is useful for applying the GCD based reduction. However, due to the reduced size of the analysis graph, we can see that the conversion time for the TDS-CDMA is lesser than the WLAN application. Hence, by paying a bit more in terms of the modelling time, we can save more on the conversion time.

Test case No	Modelling time(Sec)		Conversion time(Sec)		Analysis time(Sec)	
	LR	multi-rate	LR	multi-rate	LR	multi-rate
1	0.00325	4.68725	0.00000	16.53155	0.00255	52.13458
2	0.00322	4.26382	0.00000	14.71871	0.00231	51.36887
3	0.00317	4.25423	0.00000	15.76101	0.00217	54.99142
4	0.00319	4.39955	0.00000	15.06342	0.00234	57.51341
5	0.00319	4.22711	0.00000	22.00124	0.00218	61.24769
6	0.00315	4.04412	0.00000	14.86148	0.00227	55.57790

Table 7.9: Time taken by Heracles for analysing TDS-CDMA application graph

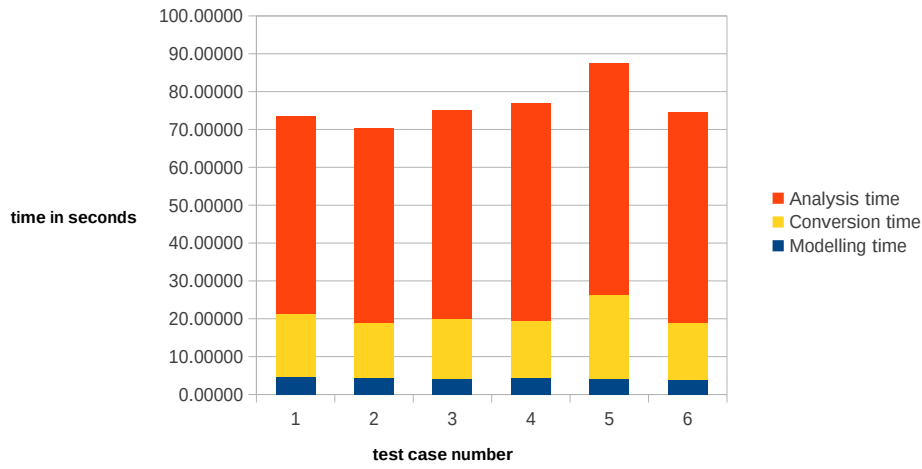


Figure 7.8: Modelling, conversion and analysis time for TDS-CDMA



Figure 7.9: Modelling, conversion and analysis time for WLAN

Test case No	Modelling time(Sec)		Conversion time(Sec)		Analysis time(Sec)	
	LR	multi-rate	LR	multi-rate	LR	multi-rate
1	0.04994	1.08542	0.00000	144.43951	0.01278	19.72460
2	0.05338	0.82499	0.00000	63.35466	0.01512	11.07283
3	0.05085	0.62372	0.00000	78.36283	0.01336	9.08371
4	0.05065	0.80025	0.00000	81.85164	0.01264	10.71035
5	0.04993	0.68109	0.00000	78.06110	0.01287	10.16881
6	0.05170	0.68340	0.00000	67.88635	0.01237	10.71269

Table 7.10: Time taken by Heracles for analysing WLAN application graph

Time taken for carrying out the analysis using the multi-rate model is very high as compared to the other techniques. For example, the analysis of a WLAN application using multi-rate model takes around 1000 to 2000 times more time as compared to the LR model. This is a clear drawback of this technique. Though, the amount of time taken is high, we get a lot more accurate results and the time is still not very high. We also suggest techniques to reduce the conversion time and the analysis time in the next chapter.

7.5 Binary slicer results

The benefits of the reduced pessimism can be seen in the binary slicer results. The WLAN application has three processors. As per Section 6.2.3.1, there are $3! = 6$ ways in which an optimized solution can be found out. We used the binary slicer with LR model as well as the multi-rate model to reduce the processor utilization. The results for WLAN are given in Table 7.11. The numbers in the first three columns give the priority of a processor. If the priority of a processor is 3, then it is chosen first for reducing its utilization, followed by the one with priority 2 and then the third processor with priority one. The utilization numbers are approximated to 5 decimal places. The slicing resolution was set to $(1/10000)$. The input utilization was 0.45 for all processors.

Priority (3 is highest)			minimum utilization with								
			Latency Rate model			multi-rate model			time wheel simulation		
ARM	EVP	SWC	ARM	EVP	SWC	ARM	EVP	SWC	ARM	EVP	SWC
1	2	3	0.45000	0.45000	0.24921	0.37501	0.26251	0.23000	0.32301	0.30651	0.15351
1	3	2	0.45000	0.24921	0.45000	0.45000	0.23000	0.32351	0.27401	0.18750	0.41250
2	1	3	0.45000	0.45000	0.24921	0.20701	0.45000	0.23000	0.25001	0.33901	0.15351
2	3	1	0.45000	0.24921	0.45000	0.25001	0.23000	0.40351	0.25001	0.18750	0.42450
3	2	1	0.17341	0.45000	0.45000	0.16702	0.28251	0.42751	0.12501	0.23001	0.43900
3	1	2	0.17341	0.45000	0.45000	0.16702	0.45000	0.26001	0.12501	0.30700	0.23001

Table 7.11: WLAN binary slicer results

Comparing the results for the LR model against the multi-rate model, we find that the processor utilization has improved (reduced) immensely in all cases for all the processors. It can be seen there is reduction in utilization only for the processor with highest priority with the LR model based approach. With multi-rate based approach, we could achieve reduction in the utilization of the remaining two processors as well.

This is the result of the reduced pessimism in the multi-rate model as opposed to the LR model. The immediate impact to reduced processor utilizations is that the processors can perform more operations in the given timing resource. This leads to better utilization of the available resources on the MPSoC.

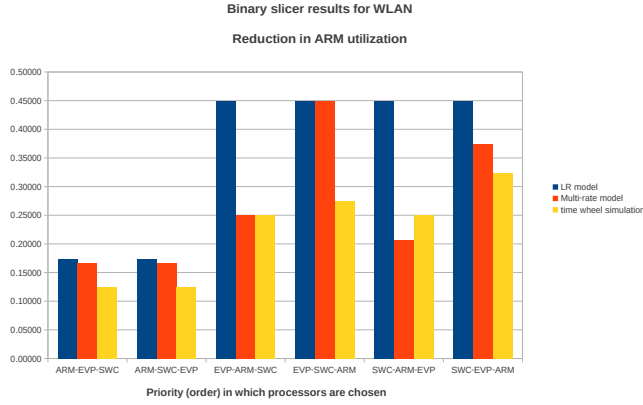


Figure 7.10: Reduced utilizations for ARM

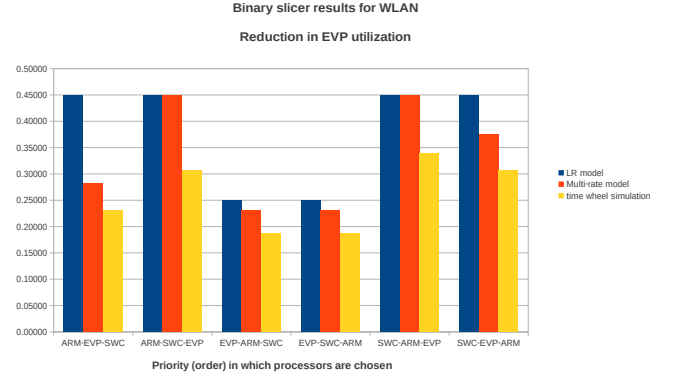


Figure 7.11: Reduced utilizations for EVP

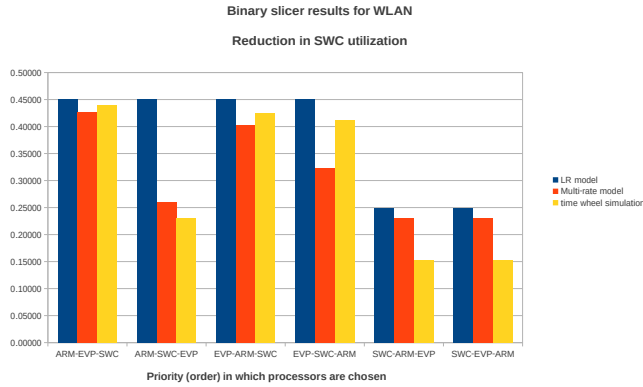


Figure 7.12: Reduced utilizations for SWC

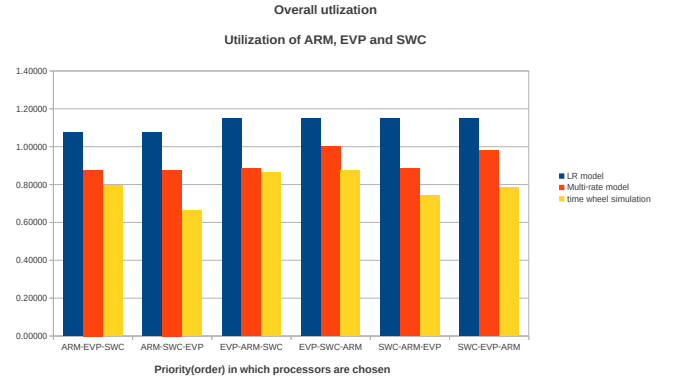


Figure 7.13: Total utilization

One thing worth noting is that the reduction in utilization is higher for the highest priority processor. As the priority becomes lower, we find that the reduction achieved in processor utilization also becomes lesser. This is because the range to reduce the utilization of the processors with lower priorities is less, since the processors with higher priority have used up most of the slack. The Figure 7.10, Figure 7.11 and Figure 7.12 show the reduced utilizations obtained using the three methods for all the three processors. The Table 7.11 also gives the processor utilizations obtained by using time wheel simulation instead of the multi-rate model. The results are a lot better than the multi-rate model in some cases while they are worse than multi-rate model in some other cases. Ideally, we expected that the results obtained using the time wheel simulation should have been better for all the processors in all the six experiments that we have performed. We analysed the problem to find out that there is a delay between the firing of actor 'Detect' and actor 'R321' during simulation of the time wheel. The

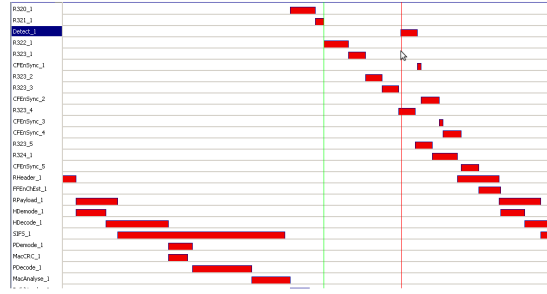


Figure 7.14: Delayed firing of the 'Detect' actor in time wheel simulation

actor 'Detect' is dependent only on 'R321' (please refer Figure 7.15) and should have fired immediately after the end of execution of 'R321'. But due to static ordering of the actors, a new dependency is created. A new edge is formed in the analysis graph between the 'Detect' and the 'ModHeader' actor. As a consequence of the cyclic dependencies between the actors mapped onto different processors, the delay in firing on one actor affects the firing of the next actor and so on. Hence, we see that the firing gets delayed. The delay in the firing is suspected to be due to the variable actor execution times between two iterations of the graph and the reduced utilizations of processors. Though the Figure 7.14 shows the delayed firing of 'Detect' for only one case when the priority for SWC was 3, EVP was 2 and ARM was 1, it has been observed in all four cases. The 'Detect' actor should have started at the green marker but starts the red marker. The total reduction achieved for all the three processors is the highest for the time wheel simulation. This can be seen in Figure 7.13.

The slicer results for lower priority processors would vary from one technique to other, since the amount of reduction achieved in the first processor affects the reductions in the remaining processors since the amount of slack that can be exploited is reduced. If more reduction is achieved for the the first processor, then the remaining processors cannot be reduced to a large extent. Also the search space is shown to be non-linear for the LR model in [20]. We also suspect that the search space for other techniques may also be non-linear. Hence, it is possible to have better results from multi-rate model as opposed to the LR model.

Time taken by the binary slicer to find the optimized schedules depends upon the analysis technique used. We have seen earlier than the time taken for a single multi-rate model based analysis is very high as compared to the time taken for LR model based analysis. In case of the binary (as well as other slicers), we are executing the scheduler code multiple times in a loop until a desired threshold is reached and for each invocation of the scheduler, the analyser is invoked multiple times. Hence, we expect that the time taken for obtaining one optimized schedule using the multi-rate model is very high. Table 7.12 shows the actual time taken to compute the utilizations presented in Table 7.11. These experiments were carried out using the default resolution of (1/10000).

Priority (3 is highest)			time taken in seconds	
ARM	EVP	SWC	LR model	Multi-rate model
1	2	3	0.38	117223
1	3	2	0.41	57445
2	1	3	0.40	3738
2	3	1	0.40	20228
3	2	1	0.32	67673
3	1	2	0.31	3644

Table 7.12: Time taken by binary slicer to compute reduced utilizations for WLAN application

Test case no.	Latency Rate model		
	ARM	EVP	SWC
1	0.22503	0.45000	0.35757
2	0.29985	0.45000	0.29842
3	0.33739	0.28118	0.45000
4	0.30923	0.29356	0.45000
5	0.30058	0.44617	0.30000
6	0.33751	0.44989	0.28118

Table 7.13: Randomized slicer results for WLAN using LR model

Test case no.	multi-rate model		
	ARM	EVP	SWC
1	0.17350	0.45000	0.25500
2	0.19600	0.44801	0.24002
3	0.18950	0.28101	0.41201
4	0.25001	0.25600	0.25351
5	0.25001	0.25602	0.25352
6	0.16800	0.41451	0.29451

Table 7.14: Randomized slicer results for WLAN using multi-rate model

These timings presented her are obtained after applying GCD based reductions to the analysis graphs. It is clear that the amount of time taken to find a solution for the binary slicer using the multi-rate model is orders of magnitude higher than the time taken using the LR model.

One way to reduce to the time taken for finding optimized schedules is by reducing the resolution of the slicer. This value can be increased to achieve better timing performance. However, increasing this value will compromise the accuracy of the obtained utilization. Also, the binary slicer is nothing but a binary search, the reduction in timing would not be much, due to the logarithmic relation of the binary search. So, reducing the slicer resolution is not a very good prospect for reducing the computation time of binary slicer.

7.6 Randomized slicer results

As discussed in Section 6.2.3.2, the randomness helps us to reach other points in the search space. Due to random selection of a processor, we can reach a different point in the processor utilization search space. Thus, potentially each run of the randomized slicer for the same input application graph might lead us to a different solution. For the same reason, we cannot compare results of one method with the another. The results with various techniques have been tabulated in Tables 7.13, 7.14, and 7.15. We see that different utilizations are indeed possible and can be found using randomized slicer and weighted randomized slicer. The time taken for obtaining randomized slicer results is also orders of magnitude higher for the multi-rate model than the LR model. The time taken by Randomized slicer is comparable with the time taken by the binary slicer.

Test case no.	time wheel simulation		
	ARM	EVP	SWC
1	0.25700	0.23001	0.28101
2	0.25001	0.24651	0.23901
3	0.16700	0.26251	0.44750
4	0.16700	0.29951	0.41051
5	0.22501	0.40400	0.26251
6	0.25001	0.25051	0.23150

Table 7.15: Randomized slicer results for WLAN using time wheel simulation

7.7 Summary

In this chapter, we presented the results for a single application actor modelled using the multi-rate model. We show with the help of examples that the equations presented for the impulse response and the burst response are correct. We also presented the results comparing the various analysis methods. We saw the effect of reduced pessimism with the help of simulations of TDS-CDMA and WLAN applications. We also present the results of slicers implemented in order to obtain optimized schedules in terms of reduced processor utilization. The processor utilizations obtained using the multi-rate model based analysis technique are superior over the latency rate model based technique. In certain cases, the results of the multi-rate model are superior even to the time wheel simulation. We were able to achieve about 40% reduction in processor utilization for EVP over the LR model using the multi-rate model in case where it is the lowest priority processor. We also investigate the reasons for the large computation time taken by the multi-rate model based analysis as compared to the LR model based analysis.

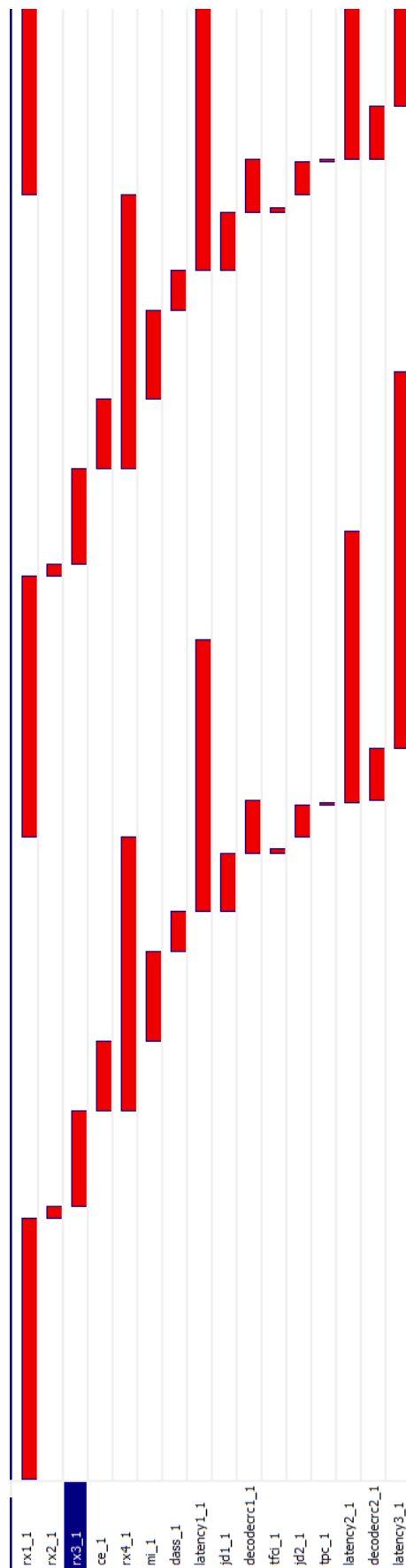


Figure 7.16: Gantt chart for TDS-CDMA with time wheel simulation

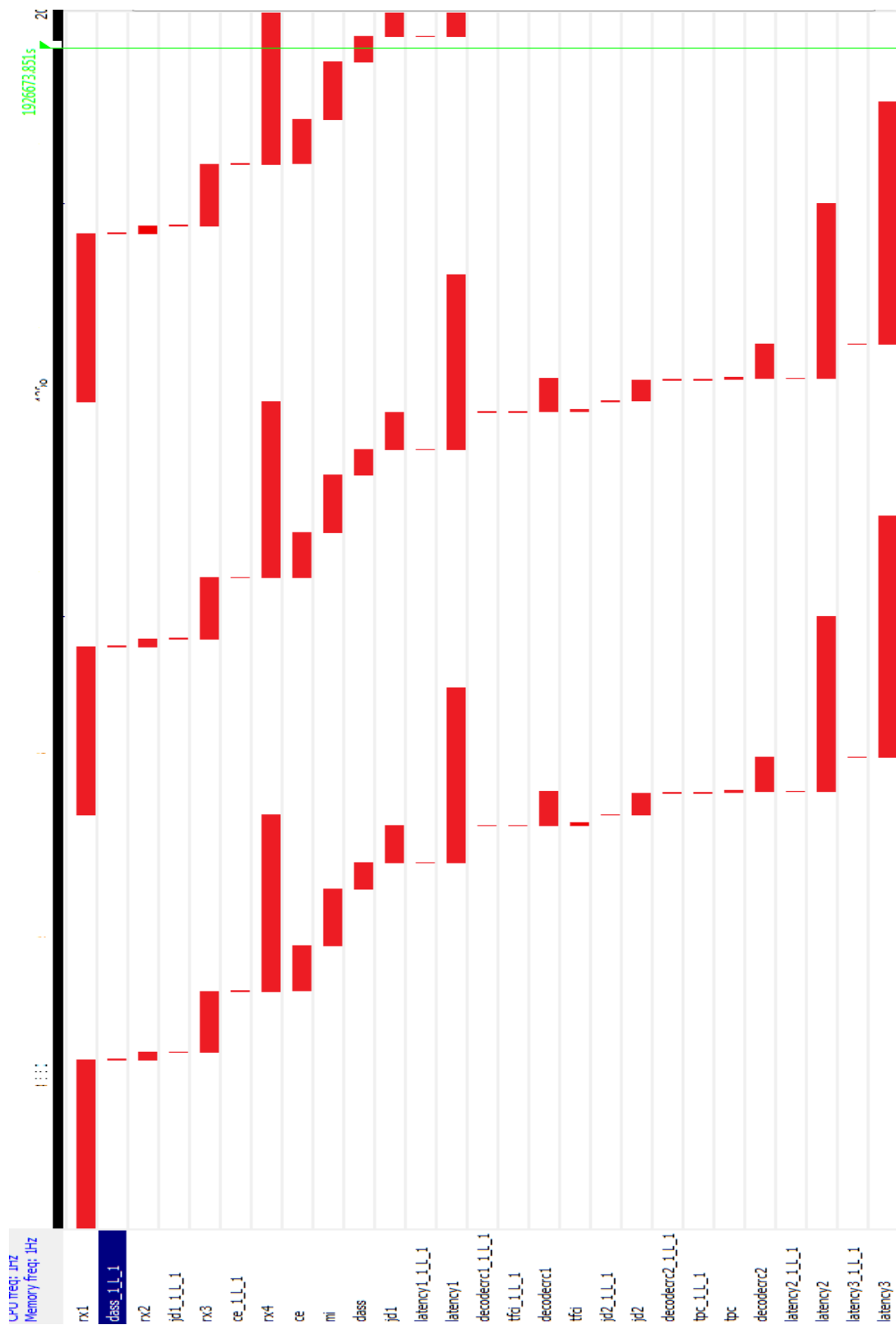


Figure 7.17: Gantt chart for TDS-CDMA with multi-rate model based analysis

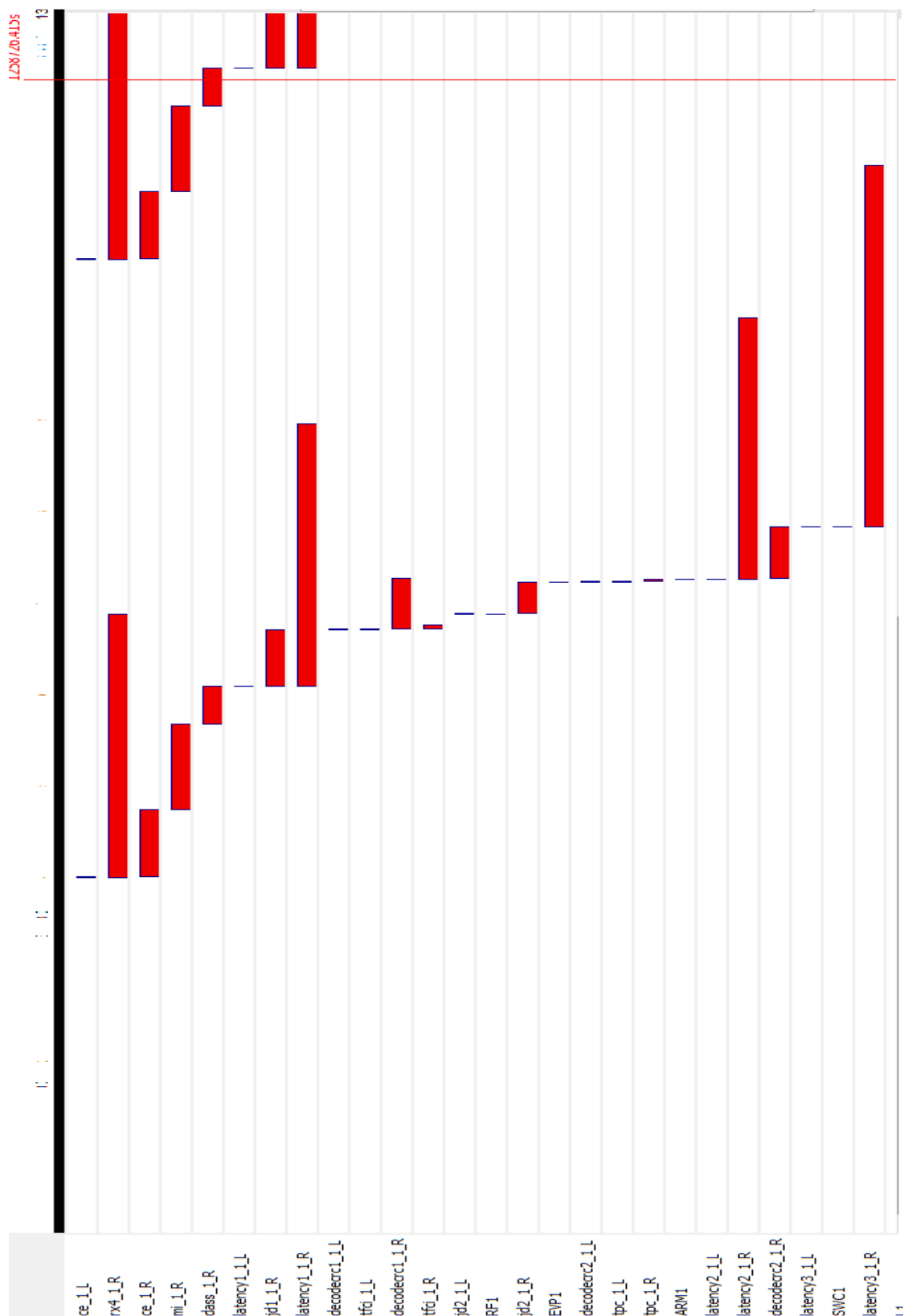


Figure 7.18: Gantt chart for TDS-CDMA with LR model based analysis

Future work and conclusions

In this chapter, we present our comments on the complexity of the analysis graphs generated using the multi-rate model. We summarize the ideas that we have implemented in our thesis for accurate reduction of the SRDF analysis graph generated from the multi-rate analysis graph. We also present an approximate technique for reduction of the SRDF analysis graph. We also present ideas for future work as well as conclusions of this thesis.

8.1 Discussion

In this section, we present our insights regarding the analysis of the multi-rate model graphs.

8.1.1 Complexity of analysis graphs

We have seen in the previous chapter that the time taken for analysis using the multi-rate model is very high as compared to the LR model. We have also seen that the size of the analysis graph is very large when we use the multi-rate model. In this section, we quantify the complexity of the generated analysis graphs in terms of the size of the generated analysis graph and the complexity associated with the analysis itself.

8.1.1.1 Size of the analysis graph

As discussed in the Section 6.2.1.3, the application graph generated by the multi-rate model based graph builder is a CSDF graph. We need to convert it to a SRDF graph in order to analyse it. There are 5 MRDF actors and 6 MRDF edges in the multi-rate model based analysis graph. If the application graph has n actors and r edges, then the worst case size of the application graph in MRDF form will be $5n$ MRDF actors and $5n + r$ MRDF edges, assuming that each actor is mapped on a different processor and hence static ordering is not possible and GCD based reduction is not implemented. Then, the size of the equivalent SRDF graph is given by $(3n + 2(\tau_1 + \tau_2 \dots \tau_n))$ actors ($3n$ l, c and s actors and $2(\tau_1 + \tau_2 \dots + \tau_n)$ w and x actors) and $5(\tau_1 + \tau_2 \dots + \tau_n)$ internal edges and other edges due to r MRDF edges in the application graph. The edges in the SRDF equivalent graph due to these r MRDF edges depends upon the production and consumption rates in the original application graph. In the worst case, the size of the SRDF application graph increases exponentially as compared to the size of the original application graph [9] [11] [22].

8.1.1.2 Accurate reduction techniques

We reduce the graph size by using the GCD based reduction. During the MRDF to SRDF conversion of the analysis graph, we get τ copies of x and w actors with edges interconnecting them. When we use this reduction, we actually reduce the number of w and x actors by a factor of $\gcd(S, \tau)$ per MRDF actor. This type of reduction is also applicable to CSDF static order model. Static ordering of actors also helps to reduce the number of x and w actors by having a common $x - w$ pair for all actors in a static order. These are all accurate techniques of graph size reduction. All these techniques have been implemented in Heracles, as a part of our work.

8.1.1.3 Analysis time complexity

We have seen earlier in the Section 6.1.2 that the worst case complexity of the scheduler is exponential. The analyser uses Howard algorithm for MCM computation. This algorithm has a pseudo-polynomial complexity [7] [8]. Since an exponential algorithm is invoking other algorithms of less complexity, the time complexity for the scheduling process will be exponential, in the worst case.

The time taken by the analysis can be improved by the MCM computation technique suggested in [8]. An efficient cycle detection algorithm has been proposed. The results in [8] show that the MCM of a graph with 185000 nodes can be found out using the Howard algorithm in combination of the efficient cycle detection algorithm in a second. The efficient cycle detection algorithm should be tried out to see if we get the same performance. If it works, then the multi-rate model based analysis can be used to analyse larger application graphs as well. Though, this will not improve the worst case timing complexity of the tool, it may make the computations faster than the current version.

Finally, the conversion of CSDF and MRDF graphs into SRDF graphs is currently a brute force implementation involving matrix computations. Timing improvements can be achieved by using faster matrix computation algorithms.

8.2 Conclusions

The real time streaming applications like SDR implemented on MPSoC platforms require timing guarantees with respect to the latency as well as throughput. The TDM scheduler is a basic type of scheduler which is easy to analyse and hence it is used to schedule tasks on the individual processors within the MPSoC platform. We provide a data-flow based model which is more accurate than the existing models. We also proposed a new multi-rate model for supporting the static ordering of actors with the CSDF static order model.

We proved that the multi-rate model represents the cyclo-static behaviour of the TDM scheduler. In this process, we provided a detailed analysis of the multi-rate model and introduced the two modes of the multi-rate model - single arrival response and burst response. We were also able to provide equations modelling the behaviour of the model in these two modes. We also proved and found out the expressions for the MCM and cyclicity of the multi-rate model. We presented an unique way of checking if the initial

number of iterations of the SRDF graph are equal to one, i.e. $K_{G_S} = 1$. We also proved that the multi-rate model is conservative and more accurate than the LR model.

We implemented the multi-rate model based analysis graph, various processor utilization reduction techniques in the existing software tool called Heracles. We also presented the experimental results for the various modelling techniques with real world examples showcasing the improvements that were achieved due to the multi-rate and static order model. The results for TDS-CDMA as well as WLAN application graph indicate that we were able to achieve the goals of providing an improved model for TDM scheduler analysis. The timing analyses of the two applications suggest that the idea of making the making the 'common case faster' seems to work with current scheduler of Heracles. We experimentally show that the finish times obtained for various actors of the real application graphs like TDS-CDMA and WLAN that are scheduled on processors running a TDM scheduler are conservative towards the actual finish times. We also show that these finish times are less pessimistic than the ones predicted by the LR model.

Due to more accurate modelling, we achieved up to 40% reduction in processor utilizations for the EVP processor, even when it was the lowest priority processor. In some cases, the reductions in processor utilizations were equal while some others were more than what could be achieved using time wheel simulation. In all experiments, our results showed either lesser or equal utilizations as compared to the LR model.

The time taken for analysis as well as finding reduced processor utilizations is higher as compared to the LR model. The time taken for binary slicing of the WLAN application in one case was slightly more than a day, which can be considered acceptable, given the amount of reductions achieved. We investigated the complexity of the generated analysis graphs and provided the reasons as to why the time taken is very large. We also presented various accurate and approximation based techniques to reduce the analysis graph size. Lastly, we also tried to check if the model can be generalized for other budget schedulers.

We would like to conclude that the multi-rate model and the CSDF static order model, jointly provide a more accurate model for TDM scheduler. The analysis of real graphs is possible using these techniques but the graph size of the generated analysis graphs can be an impeding factor.

8.3 Future work

Now, we present ideas that can be considered as a future based on our work as well as the insights provided in the previous section.

8.3.1 Formal proofs for CSDF static order model

We have used the CSDF static order model to represent the static ordering of application actors. However, we have not proved in this thesis that the CSDF static order model is conservative. This proof can be based in the proof for the multi-rate model proof flow except the fact that there will be variable number of x and w actors in the equivalent SRDF model, due to the different execution times of different actors in the static order.

Also, we have proved that GCD based reduction is correct for the multi-rate model. Similar reasoning may be used to prove that GCD based reduction is accurate for the

CSDF static order model. This work is left as future work.

8.3.2 Applicability of the multi-rate model to other budget schedulers

The budget and the waiting time are the parameters that differ between various budget schedulers. In the TDM scheduler, the budget is nothing but the slice time S and the waiting time is $P - S$. We expect that if these parameters of the multi-rate model can be changed then it can be used for modelling other budget schedulers as well. For example, the high priority task of the priority based scheduler (PBS) presented in [25] can be modelled using the multi-rate model by changing the waiting time to B as well as allocated budget to B . Thus, in the multi-rate model, the execution time of actor l will be B and the number of delays along the (w, x) edge will also be equal to B . This model can be explored further.

8.3.3 Reducing the analysis and conversion time

As suggested in the Section 8.1.1.3, the efficient cycle detection algorithm can be implemented to see if the analysis time can be reduced further. The matrix computation module used for matrix manipulation can be improved using known faster matrix computation techniques.

There is one more way of reducing the graph size, but at the expense of the accuracy of modelling (increased pessimism). Since it is not always possible to have a $gcd(S, \tau)$ greater than one, we cannot apply the GCD based reduction. However, we can create an approximate model with S' and τ' , where slice time S' is a closest even number smaller than S and τ' is a closest even number greater than the execution time τ . Then we have the $gcd(S', \tau')$ as 2. and we can apply the GCD based reduction to reduce the number of x and w actors by a factor of 2. After having done this, we can still guarantee that the estimate of finish time is conservative. In fact, this idea can be extended further. The values S and τ can be increased to a closest multiple of 4, 8, 16 and so on, which are greater their original values. Doing so will reduce the number of x and w actors considerably. We select slightly higher execution time and smaller slice time and hence the model would predict more pessimistic finish times. The processor utilization will also be greater than the value that we would have achieved without using the proposed approximation. With this idea, the complexity of the equivalent SRDF graph can be reduced by sacrificing the accuracy of the model. The approximation can be extended to higher powers of two till a point is reached where the results from the approximate model become as inaccurate as the LR model. This technique can be implemented as another reduction technique. It would be interesting to compare the results of the approximate model with the LR model.

Bibliography

- [1] *Graphviz: Graph visualization software*; <http://www.graphviz.org/>.
- [2] *Timedocor: execution traces visualization software*; <http://sourceforge.net/projects/timedocor/>.
- [3] Ghamarian A.H., *Timing analysis of synchronous data flow graphs*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2008.
- [4] Franois Baccelli, Guy Cohen, Geert Jan Olsder, and Jean pierre Quadrat, *Synchronization and linearity : an algebra for discrete event systems. wiley series in probability and mathematical statistics*, 1992.
- [5] Marco Bekooij, Rob Hoes, Orlando Moreira, Peter Poplavko, Milan Pastrnak, Bart Mesman, Jan Mol, Sander Stuijk, Valentin Gheorghita, and Jef Meerbergen, *Dataflow analysis for real-time embedded multiprocessor system design*, Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices (Peter Stok, ed.), Philips Research Book Series, vol. 3, Springer Netherlands, 2005, pp. 81–108.
- [6] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, *Cycle-static dataflow*, Signal Processing, IEEE Transactions on **44** (1996), no. 2, 397–408.
- [7] Jean Cochet-terrasson, Guy Cohen, Stephane Gaubert, Michael Mc Gettrick, and Jean pierre Quadrat, *Numerical computation of spectral elements in max-plus algebra*, 1998.
- [8] Ali Dasdan, *Experimental analysis of the fastest optimum cycle ratio and mean algorithms*, ACM Trans. Des. Autom. Electron. Syst. **9** (2004), no. 4, 385–418.
- [9] Marc Geilen, *Reduction techniques for synchronous dataflow graphs*, Proceedings of the 46th Annual Design Automation Conference (New York, NY, USA), DAC '09, ACM, 2009, pp. 911–916.
- [10] A.H. Ghamarian, M.C.W. Geilen, S. Stuijk, T. Basten, A.J.M. Moonen, M.J.G. Bekooij, B.D. Theelen, and M.R. Mousavi, *Throughput analysis of synchronous data flow graphs*, Application of Concurrency to System Design, 2006. ACSD 2006. Sixth International Conference on, june 2006, pp. 25–36.
- [11] E.A. Lee and D.G. Messerschmitt, *Synchronous data flow*, Proceedings of the IEEE **75** (1987), no. 9, 1235–1245.
- [12] Edward Ashford Lee and David G. Messerschmitt, *Static scheduling of synchronous data flow programs for digital signal processing*, IEEE Trans. Comput. **36** (1987), no. 1, 24–35.

- [13] Alok Lele, *Data-flow based temporal analysis for tdm arbitration*, Master's thesis, Technical University of Eindhoven, 2011.
- [14] Moreira Orlando Lele, Alok and Pieter Cuijpers, *A new data flow analysis for tdm*, Proceedings of the 12th ACM & IEEE international conference on Embedded software (New York, NY, USA), EMSOFT '12, ACM, 2012.
- [15] Frisch A. Garrigue J. Didier R. Leroy X., Doligez D. and Vouillon J., *The ocaml system release 3.12: Documentation and users manual*, release 3.12 ed., Institut National de Recherche en Informatique et en Automatique, France, July 29, 2011.
- [16] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Kriszti Flautner, *Soda: A high-performance dsp architecture for software-defined radio*, IEEE Micro **27** (2007), 114–123.
- [17] P. Marwedel, *Embedded system design*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [18] III Mitola, J., *Software radios-survey, critical evaluation and future directions*, Telesystems Conference, 1992. NTC-92., National, may 1992, pp. 13/15 –13/23.
- [19] Orlando Moreira, Frederico Valente, and Marco Bekooij, *Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor*, Proceedings of the 7th ACM & IEEE international conference on Embedded software (New York, NY, USA), EMSOFT '07, ACM, 2007, pp. 57–66.
- [20] Moreira O., *Temporal analysis and scheduling of hard real-time radios on a multiprocessor*, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, the Netherlands, 2012.
- [21] Ulrich Ramacher, *Software-defined radio prospects for multistandard mobile phones*, Computer **40** (2007), no. 10, 62–69.
- [22] Sundararajan Sriram and Shuvra S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*, 2nd ed., CRC Press, Inc., Boca Raton, FL, USA, 2009.
- [23] J.A. Stankovic and K. Ramamritham, *Tutorial hard real-time systems*, IEEE Computer Society Press, Computer Society Press of the IEEE, 1988.
- [24] Jan Staschulat and Marco Bekooij, *Dataflow models for shared memory access latency analysis*, Proceedings of the seventh ACM international conference on Embedded software (New York, NY, USA), EMSOFT '09, ACM, 2009, pp. 275–284.
- [25] Marcel Steine, Marco Bekooij, and Maarten Wiggers, *A priority-based budget scheduler with conservative dataflow model*, 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, DSD 2009, 27-29 August 2009, Patras, Greece (Antonio Nez and Pedro P. Carballo, eds.), IEEE Computer Society, 2009, pp. 37–44.

- [26] William Thies and Saman Amarasinghe, *Language and compiler support for stream programs*, Tech. report, 2009.
- [27] Wei Tong, O. Moreira, R. Nas, and K. van Berkel, *Hard-real-time scheduling on a weakly programmable multi-core processor with application to multi-standard channel decoding*, Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th, april 2012, pp. 151 –160.
- [28] Kees van Berkel, Frank Heinle, Patrick P. E. Meuwissen, Kees Moerman, and Matthias Weiss, *Vector processing as an enabler for software-defined radio in hand-held devices*, EURASIP J. Appl. Signal Process. **2005** (2005), 2613–2625.
- [29] Maarten H. Wiggers, Marco J. G. Bekooij, and Gerard J. M. Smit, *Modelling run-time arbitration by latency-rate servers in dataflow graphs*, Proceedings of the 10th international workshop on Software & compilers for embedded systems (New York, NY, USA), SCOPES '07, ACM, 2007, pp. 11–22.
- [30] Cognovo wireless, *Cognovo sdm*, www.cognovo.com.

