
Pós-Graduação em Ciência da Computação

# "Software Synthesis for Energy-Constrained Hard Real-Time Embedded Systems"

**Por**

***Eduardo Antônio Guimarães Tavares***

**Tese de Doutorado**


Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, NOVEMBRO/2009


*Universidade Federal de Pernambuco*

*CENTRO DE INFORMÁTICA*

*PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO*

***Eduardo Antônio Guimarães Tavares***

# *"Software Synthesis for Energy-Constrained Hard Real-Time Embedded Systems"*

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Doutorado em Ciência da Computação.

*ORIENTADOR:* ***Prof. Dr. Paulo Romero Martins Maciel***

*RECIFE, NOVEMBRO/2009*

Tavares, Eduardo Antônio Guimarães
Software synthesis for energy-constrained hard real-time embedded systems / Eduardo Antônio Guimarães Tavares. - Recife: O Autor, 2009.
xvii, 241 folhas : il., fig., tab.

Tese (doutorado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2009.

Inclui bibliografia e apêndice.

1. Processamento em tempo real. 2. Engenharia de software. 3. Sistemas Operacionais. I. Título.

004.3 CDD (22. ed.) MEI2009 - 150

Tese de Doutorado apresentada por **Eduardo Antônio Guimarães Tavares** a Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Software Synthesis for Energy-Constrained Hard Real-Time Embedded Systems"**, orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Paulo Roberto Freire Cunha
Centro de Informática / UFPE

Prof. Nelson Souto Rosa
Centro de Informática / UFPE

Prof. Ricardo Massa Ferreira Lima
Centro de Informática / UFPE

Prof. Sing Wun Song
Instituto de Matemática e Estatística / USP

Prof. Ricardo Augusto da Luz Reis
Departamento de Informática / UFRGS

Visto e permitida a impressão.
Recife, 20 de novembro de 2009.

**Prof. Nelson Souto Rosa**
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

*This thesis is dedicated to my mother Fátima and my sister Kátia*

# ACKNOWLEDGEMENTS

First of all, I would like to thank God for all the things I have conquered. Also, I am very grateful to my mother, who has always encouraged me to do what I dreamed. Many thanks to my sister, who has been at my side at all difficult moments. Thanks to my father.

Many thanks to my advisor and friend professor Paulo Maciel. He gave me the chance to be part of his research group, and has helped me at all moments. A special thanks to professor Meuse Oliveira Jr. who provided the hardware implementation to validate the case studies; to Bruno Silva and Pedro Dallegrave for implementing, testing and validating the experiments; to professor Ricardo Massa who helped with valuable reviews; to all members of our research group (MODCS - Modeling of Distributed and Concurrent Systems); and to Raimundo Barreto who started the research about software synthesis in our group.

Thanks to several colleagues I made at CIn/UFPE. Furthermore, I would like to express my gratitude to all professors and staff in the Center for Informatics. Finally, I would like to thank CNPq for the financial support provided in this thesis.

# RESUMO

A grande expansão do mercado de dispositivos digitais tem forçado empresas desenvolvedoras de sistemas embarcados em lidar com diversos desafios para prover sistemas complexos nesse nicho de mercado. Um dos desafios prominentes está relacionado ao consumo de energia, principalmente, devido aos seguintes fatores: (i) mobilidade; (ii) problemas ambientais; e (iii) o custo da energia. Como consequência, consideráveis esforços de pesquisa têm sido dedicados para a criação de técnicas voltadas para aumentar a economia de energia.

Na última década, diversas técnicas foram desenvolvidas para reduzir o consumo de energia em sistemas embarcados. Muitos métodos lidam com gerenciamento dinâmico de energia (DPM), como, por exemplo, *dynamic voltage scaling* (DVS), cooperativamente com sistemas operacionais especializados, a fim de controlar o consumo de energia durante a execução do sistema. Entretanto, apesar da disponibilidade de muitos métodos de redução de consumo de energia, diversas questões estão em aberto, principalmente, no contexto de sistemas de tempo real crítico.

Este trabalho propõe um método de síntese de software, o qual leva em consideração relação entre tarefas, overheads, restrições temporais e de energia. O método é composto por diversas atividades, as quais incluem: (i) medição; (ii) especificação; (iii) modelagem formal; (vi) escalonamento; e (v) geração de código. O método também é centrado no formalismo redes de Petri, o qual define uma base para geração precisa de escalas em tempo de projeto, adotando DVS para reduzir o consumo de energia. A partir de uma escala viável, um código customizado é gerado satisfazendo as restrições especificadas, e, dessa forma, garantindo previsibilidade em tempo de execução. Para lidar com a natureza estática das escalas geradas em tempo de projeto, um escalonador simples em tempo de execução é também proposto para melhorar o consumo de energia durante a execução do sistema. Diversos experimentos foram conduzidos, os quais demonstram a viabilidade da abordagem proposta para satisfazer restrições críticas de tempo e energia. Adicionalmente, um conjunto integrado de ferramentas foram desenvolvidas para automatizar algumas atividades do método de síntese de software proposto.

**Palavras-chave:** sistemas de tempo real crítico; síntese de software; geração de código; escalonamento; restrições de energia; *dynamic voltage scaling* (DVS); redes de Petri;

# ABSTRACT

The widespread expansion of digital device market has forced embedded system companies to deal with several additional challenges in order to provide complex systems in this market niche. One of the most important challenges is related to energy consumption, mainly, due to the following factors: (i) mobility issues; (ii) environmental problems and (iii) energy costs. As a consequence, considerable researching efforts have been devoted to create techniques for increasing energy saving.

Over the last decade, several techniques have been developed to reduce energy consumption in embedded systems. Many of them deal with dynamic power management (DPM), such as dynamic voltage scaling (DVS), cooperatively with specialized operating systems in order to control energy consumption during system runtime. However, in spite of the availability of many energy reduction methods, several issues are still open, mainly, in the context of hard real-time systems.

This work proposes a software synthesis method for hard real-time systems, which takes into account intertask relations, overheads, timing as well as energy constraints. The method is composed of several activities, which include: (i) measurement; (ii) specification; (iii) formal modeling; (vi) scheduling; and (v) code generation. The method is also centered on Petri net formalism, which lays down a basis for precise pre-runtime schedule generation, adopting DVS for reducing energy consumption. From a feasible schedule, a customized code is generated satisfying the specified constraints and, so, assuring runtime predictability. To tackle the static nature of pre-runtime schedules, a lightweight runtime scheduler is also proposed to improve energy consumption during system execution. Several experiments were conducted, which demonstrate the feasibility of the proposed approach to satisfy strigent timing constraints as well as reducing energy consumption. Additionally, a set of integrated tools has been developed to automate some activities of the proposed software synthesis method.

**Keywords:** hard real-time systems; software synthesis; code generation; scheduling; energy constraints; dynamic voltage scaling; Petri nets;

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

CHAPTER 1

# INTRODUCTION

Embedded systems are present in most of our daily activities, ranging from simple household appliances (e.g., microwave ovens) to complex medical devices (e.g., pulse oximeter). In other words, embedded systems are ubiquitous nowadays. Because of the varied applications that may be implemented as embedded systems, diverse constraints may have to be taken into account in design-time for satisfying system requirements, such as reliability, energy consumption, timing, and so on. Lately, considerable attention has been devoted to energy consumption in embedded systems, mainly due to the following factors:

- Advances in microelectronics have allowed the development of small-sized embedded systems with several complex features, making possible the rapid emerging of powerful mobile devices. These devices generally rely on constrained energy sources (e.g., battery), in such a way that if the energy source is depleted, the system stops functioning. Hence, energy saving becomes of utmost importance in order to prolong battery lifetime in such devices;

- Energy costs have been rising through the years [1] due to the disproportional increase of energy consumption and availability of new energy sources. Thus, energy saving in embedded systems may provide less operational costs for the final users of the respective systems;

- Global warming has increased environmental concerns related to energy consumption, since energy production is one of the greatest sources of air pollution [2]. As embedded systems are ubiquitous, they have a considerable contribution in the utilization of energy sources. Therefore, reducing energy consumption in such systems may contribute to a lesser environmental pollution;

- The reliability of integrated systems is somewhat impacted by the energy consumption due to the influence on operational temperature [3]. Reducing energy consumption in such systems may provide benefits in terms of heat dissipation and, so, system reliability.

Since software accounts for more than 70% of system functionalities in many embedded systems nowadays [4], several techniques have been developed to reduce energy consumption at application and behavioral level. Many deal with dynamic power management (DPM), such as dynamic voltage scaling (DVS), cooperatively with specialized operating systems to control energy consumption during system runtime. Nevertheless, the indiscriminate use of DPM/DVS technologies may generate undesirable issues due to the conflict with other non-functional requirements, more specifically, timing constraints [5]. For ordinary embedded systems, those techniques may be directly applied, as the issues

do not provide major concerns. However, in real-time systems, such techniques need to be adopted with caution, since responsiveness may be greatly affected. In the context of hard real-time embedded systems, which have stringent timing constraints that must be met, catastrophic issues may occur (e.g., loss of human life).

It is not an easy task to develop hard real-time software, even more when considering the minimization of energy consumption. Thus, appropriate methods and tools are required, such that a project scheduling and, in consequence, the final budget are not jeopardized. Several energy reduction methods, mostly based on runtime techniques implemented as operating system services, have been developed in order to cope with energy reduction in hard real-time systems. However, simplified system specifications are generally considered, mainly, neglecting precedence and exclusion relations. Additionally, there are situations in which finding a feasible schedule using runtime approaches is not possible, even if such a schedule exists [6]. Furthermore, overheads, for instance, preemptions and frequency/voltage switchings, are disregarded by most works. Indeed, if overheads are neglected, tasks' constraints may be affected and even the gains obtained with such techniques may be significantly reduced [7]. In this context, it is worth stating that operating systems incorporate, during system execution, a significant amount of overheads, which may impact timing as well as energy constraints. In addition to the previous statements, it is important to point out that formal models are not adopted by most energy saving methods, which may complicate, in some cases, the verification and analysis of quantitative as well as qualitative properties.

Although diverse methods have been developed to deal with energy consumption in embedded systems, several issues are still open, mainly, in context of hard real-time systems. This thesis is related to embedded systems with stringent timing and energy constraints, focusing on the software part. More specifically, this work is concerned with the adoption of formal models for modeling hard real-time systems with energy constraints as well as the utilization of energy saving technologies, such as DVS, for improving the software energy consumption in such systems.

## 1.1 MOTIVATION

Hard real-time embedded systems have stringent timing constraints that must be met for the correct functioning of the system. In other words, not only the result of the computation is important to the correct system behavior, but also the time when such result was acquired [8]. Thus, it is important to differentiate fast computing and real-time computing. While real-time computing aims at providing results within the defined timing constraints, fast computing aims at getting results as quickly as possible. Therefore, energy saving methods may be adopted in hard real-time systems, inasmuch as timing constraints are met.

Despite the large availability of many energy reduction methods [9], previous section described some open issues related to energy savings in hard real-time embedded systems. In other words, there is a lot of effort to be performed in order to properly deal with energy constraints in those systems, opening several possibilities for new energy reduction methods. For instance, a feasible alternative to operating system usage may

be the adoption of automatic code generation (e.g., software synthesis), in such a way that customized code is obtained only with the required services, providing considerably less overheads than specialized operating systems. Additionally, system constraints, for instance, intertask relations, time and energy, can be represented using formal models to allow the verification/analysis of qualitative and quantitative properties [10] as well as to provide a basis for precise schedule generation [11].

## 1.2 OBJECTIVES

Considering the problems stated previously, this thesis proposes a software synthesis method for hard real-time systems [8] with energy constraints, adopting dynamic voltage scaling (DVS) [12] for reducing energy consumption as well as a formal model for representing system constraints. The specific objectives are:

1. to create a specification model that captures system constraints [6]. Hard real-time systems are composed of several concurrent tasks with stringent timing constraints and intertask relations, in such a way that if these non-functional requirements [11] are violated, catastrophic issues can occur. Besides, several time-critical systems [8] are also subjected to energy constraints nowadays. Thus, a specification model plays an important role in capturing all required system attributes that may affect system behavior;

2. to provide a formal model based on time Petri net (TPN) [13] to represent hard real-time systems with energy constraints. Formal models are of great importance in hard real-time system design, since they can precisely model system constraints, and also provide a basis for verification/analysis of quantitative and qualitative properties [10]. Petri net is an appropriate family of formalisms very well suited for modeling real-time embedded systems, since concurrency, synchronization and communication mechanism - usual features of such systems - are naturally represented. The adoption of TPN also allows representing timing constraints, such as deadline, release, as well as computation time;

3. to propose a scheduling approach that satisfies the specified requirements. Scheduling is an important part of a hard real-time system, since this activity must guarantee that all tasks execute according to the respective constraints. This work proposes a pre-runtime (off-line) [6] scheduling approach, which is more predictable than runtime counterparts (see Chapter 3). In order to reduce energy consumption, dynamic voltage scaling (DVS) is adopted, as it provides better results than other DPM techniques [14];

4. to implement an automatic code generator, such that, from a feasible schedule, customized code [15] is obtained. In other words, only the required services will be present in the final code for a given application. The code generator considers a runtime support, namely dispatcher, to control each task during system execution. This approach together with a lightweight runtime [6] support decreases consider-

ably the overheads that may be incurred during system runtime while guaranteeing the specified constraints;

## 1.3 CONTRIBUTIONS

This work extends the method proposed by Barreto [16] to consider energy consumption as well as dynamic voltage scaling. Besides, some issues not addressed in that work are considered in this thesis, such as: (i) composition rules that assure some quantitative/qualitative properties; (ii) algorithms for parsing the feasible schedule; (iii) measurement techniques and tools; and (iv) hybrid scheduling. The following items summarize the contributions:

- **Measurement Approach.** To provide the required data for the system specification concerning tasks' timing information and the hardware energy consumption, some measurements may be required in the hardware platform. This work adopts a set of statistical techniques to obtain such values as well as an environment to automate the measuring process;

- **Specification Model.** This work proposes a specification model that considers a set of concurrent tasks with timing constraints, intertask relations and behavioral descriptions. In addition, information regarding the hardware platform as well as the system energy constraint are also taken into account. These requirements are adopted as an input to an environment that allows the automatic generation of customized scheduled code;

- **Formal Modeling.** The proposed software synthesis method adopts a bottom-up approach, in which a set of composition rules are considered for combining basic building block models. The building blocks are combined in such a way that all generated models possess some fundamental qualitative/quantitative properties. Such an approach allows the generation of TPN models from a system specification using tools that automate the whole process;

- **Pre-runtime Scheduling.** Scheduling plays an important role in the design of hard real-time systems. This work adopts a pre-runtime scheduling approach, which is a depth-first search algorithm that utilizes the TPN model to find a feasible schedule that meets the specified constraints. The algorithm adopts a set of techniques to generate a reduced state space of the Petri net model, in the sense that only the states related to a feasible schedule should be reached;

- **Runtime Scheduling.** In order to further improve energy consumption, a runtime scheduler is proposed to take advantage of slack times that may occur at system runtime. The scheduler overhead minimally impacts the system execution, as it may contain a table with pre-calculated values for usual variations of some tasks' execution cycles.

- **Code Generation.** To the best of the present author's knowledge, there is no similar work to date that generates customized code for hard real-time systems considering energy constraints, intertask relations, runtime overheads and DVS.

Chapter 4 details the proposed software synthesis method and the methodology, in which it is inserted.

The proposed software synthesis method has been published in some journals as well as conference proceedings, and the reader is referred to [17, 18, 19, 20, 21, 22, 23, 24] for such publications.

## 1.4 OUTLINE

This work is organized as follows. Chapter 2 describes some related works. Chapter 3 presents basic concepts for understanding this thesis. Afterwards, Chapter 4 describes the proposed software synthesis method as well as the methodology, in which it is inserted. Next, Chapter 5 explains the proposed measurement activity and the adopted specification model. Chapter 6 initiates the presentation of the formal modeling approach adopted in this thesis, focusing on the adopted petri net extension and building block models. Chapter 7 continues the presentation, focusing on the composition rules, some examples, as well as the analysis and verification of prominent properties. After that, Chapter 8 describes the pre-runtime scheduling approach, the code generation mechanism, and the runtime scheduler. Finally, Chapter 9 presents some case studies and Chapter 10 concludes this thesis, including a presentation of future works.

Since this work adopts some abbreviations in several chapters, Appendix C provides a list of abbreviations to allow a more pleasant reading of this thesis.

CHAPTER 2

# RELATED WORKS

This chapter presents related works in the context of the proposed software synthesis method. Since scheduling is a critical activity in software synthesis methods that deal with energy-constrained hard real-time systems, this chapter also describes scheduling approaches that take into account stringent timing constraints as well as DVS for reducing energy consumption.

Firstly, representative scheduling approaches are presented, followed by related works concerning software synthesis methods.

## 2.1 SCHEDULING

Although DVS can provide considerable energy savings, this technology needs to be adopted with caution in hard real-time systems due to the stringent timing constraints. As a consequence, several scheduling approaches have been developed in order to cope with DVS in hard real-time systems. Although intertask relations and runtime overheads are usually disregarded in those approaches, the proposed software synthesis method takes into account such constraints. Next paragraphs describe some representative scheduling methods.

Yao et al. [25] propose an optimal off-line voltage allocation algorithm considering continuously variable voltage and a set of independent tasks. The algorithm searches for a critical interval, in which a subset of task instances are scheduled with highest speed using EDF (Earliest-Deadline First) policy. The instances contained in such interval are removed from the task instance set, and, similarly, the algorithm continues to search for other critical intervals considering the remaining instances. The algorithm stops when all task instances have been scheduled. Due to the adoption of EDF policy, Yao's approach is also called Low-Power Earliest-Deadline First (LPEDF). Despite the importance of such an algorithm, precedence and exclusion relations are not taken into account.

In [12], Ishihara and Yasuura describe an optimal off-line voltage allocation approach considering discrete set of voltages and assuming for single task problem. That work provides an important theorem, which states that if an ideal voltage is unavailable, the two immediately neighboring CPU voltages can be adopted to reduce energy consumption. Although Ishihara's work assumes a single task problem, the proposed software synthesis method adopts the associated theorem to simulate unavailable voltage levels during the modeling and scheduling activities.

LPEDF extensions have been proposed in order to consider new issues. In [26], Kwon and Kim extend LPEDF to take into account CPUs with discrete voltage levels by adopting Ishihara's theorem. Mochocki et al. [27] propose an extension that considers overheads related to voltage/frequency switching. In [28], the authors adjust Yao's algorithm to re-

gard scheduling policies based on static priorities, such as rate-monotonic policy. In all presented extensions, intertask relations are not considered.

In [29], Leung et al. present an off-line scheduling method that deals with dynamic workload variation by taking into account the average-case execution cycles (ACEC) of each task. The method aims at obtaining the best slack distribution to improve energy consumption in the average scenario, and still guaranteeing no deadline violation in the worst-case situation. The approach relies on rate monotonic scheduling policy and assumes a CPU with continuously variable voltage. Besides, a set of independent tasks is considered, in other words, precedence and exclusions are not taken into account.

Many other works are based on runtime scheduling policies, which can greatly improve energy consumption, as shown by their experimental results. Some of them apply a preprocessing for defining an initial voltage for each task before runtime. Somewhat, this can be viewed as a hybrid approach, which mixes runtime and pre-runtime approaches.

Aydin et al. [30] propose a scheduling method composed of 3 components: (i) an off-line mechanism to compute an optimal constant speed; (ii) an online technique to adjust CPU speed according to the actual workload; and (iii) an online adaptive mechanism that predict early completions in order to adjust CPU speed. Experimental results depict the effectiveness of that method, in the sense that it outperforms other DVS algorithms. However, that work does not take into account intertask relations, assumes continuously variable voltage/frequency levels, and incorporates overheads into the work-case workload of each system task. The latter assumption may be too pessimistic and may affect schedule generation.

In [31], the authors describe a scheduling approach considering precedence relations, assuming that all tasks are non-preemptable. Initially, a number of solutions are obtained off-line and stored into lookup tables. At runtime, one of the pre-calculated solutions is chosen taking into account the actual values of time and energy. Despite the feasibility to reduce overheads due to pre-calculated solutions, only part of a task is guaranteed to finish before the respective deadline and the method considers that the voltage level can continuously be varied.

In [32], Jejurikar and Gupta present a non-preemptive scheduling method based on Earliest-Deadline First (EDF) policy in order to deal with shared resources in hard real-time systems with DVS. However, the strict adoption of non-preemptive scheduling restricts the domain of applications that may adopt the approach.

Some of the presented works do not properly tackle runtime overheads related to voltage/frequency switching, preemption, and runtime calculations. A common approach in dealing with runtime overheads is considering them in tasks' worst-case execution cycles (WCEC). Nevertheless, this approach may be too pessimistic, since the total overhead is not known before schedule generation. In this context, [33] and [34] explicitly take into account overheads related to voltage/frequency switching during scheduling generation without relying on the previous statement. Nevertheless, dispatcher/scheduler and preemption overheads are disregarded. In [35], the authors propose a technique for reducing the impact of preemptions in system energy consumption. Although interesting results are provided, the technique does not consider intertask relations and assumes CPUs with continuously variable voltage.

Besides, although multiprocessor and distributed systems are not the focus of this thesis, it is important to state that some scheduling approaches have been developed considering stringent timing constraints and DVS for both systems. In [36], the authors describe a DVS scheduling method for a distributed environment considering: (i) precedence relations; and (ii) a technique to deal with the leakage power. Despite the effectiveness of such a method, they ignore mutual exclusions and consider the scheduler overhead in tasks' worst-case execution time. [37] also takes into account precedence relations in a distributed environment. However, that method adopts continuous voltage variation and does not present details about tackling overheads (e.g., voltage/frequency switching).

As an alternative, the proposed software synthesis method adopts a pre-runtime DVS scheduling approach, which takes into account runtime overheads (preemptions and voltage/frequency switching), intertask relations (precedence and exclusion), and, also, utilizes a formal model based on time Petri nets. The scheduling algorithm is a depth-first search method, which seeks for a feasible schedule that satisfies stringent timing constraints and does not surpass an upper bound in terms of energy consumption. Different from other approaches (e.g., [25]), the algorithm does not necessarily generate an optimal solution due to the size of the state space (see Chapter 8), but it looks for a solution that meets all non-functional requirements provided in the system specification. Besides, the assumed constraints allow the practical utilization of feasible schedules in the implementation of real systems. To take advantage of slack times that may occur at system runtime, a lightweight runtime scheduler is adopted to improve energy consumption without violating the constraints previously met by the pre-runtime schedule.

## 2.2 SOFTWARE SYNTHESIS

Several software synthesis methods have been proposed to address the problem of code generation for real-time embedded systems. However, few works consider energy constraints.

Cornero et al. [15] present a software synthesis method for real-time information processing systems. As stated by the authors, such systems have the distinct characteristic of the coexistence of two different types of functionalities: digital signal processing and control functions. In that work, the specification is composed of concurrent processes with their respective timing constraints, data dependencies and communications. Such specification is translated into a set of program threads, which may or may not start with a non-deterministic operation. A constraint graph models the program threads, in which the nodes represent the threads and the edges capture data dependency, control precedence and timing constraints between threads. Initially, constraint graphs are partitioned into disjoint clusters, called thread frames. Next, a static scheduling is carried out for indicating the relative arranging of threads in the same thread frame. Lastly, the static information is adopted at runtime by the dynamic scheduler, which combine distinct thread frames based on runtime system evolution. Although Cornero's work seems an interesting approach, the method can not be adopted to hard real-time systems, since it considers non-deterministic delays. In safety time-critical systems, predictability is

essential.

Balarin and Chiodo [38] present the software synthesis approach adopted in POLIS co-design framework [39], which focuses on reactive embedded systems. In that work, systems are specified as networks of communicating processes, called Codesign Finite State Machines (CFMS), which are finite state machines with arithmetic and relational operators. Moreover, an intermediary data structure, namely, s-graph (software graph), is adopted to describe the reactive behavior. Such structure is translated into a C code jointly with a Real-Time Operating System (RTOS), which is responsible for performing the runtime scheduling (e.g., Rate-Monotonic or Deadline-Monotonic). Although the approach seems to be very interesting, the authors do not show how intertask relations are handled, and no code example is shown.

Bokhnoven et al. [40] propose a software synthesis method for system level design using process execution trees. The approach aims to translate a specification described in a process algebra language, namely, Parallel Object-Oriented Specification Language (POOSL), into an imperative programming language (C++). Besides, the process execution trees are adopted to capture the real-time and concurrent behaviour of processes. Nevertheless, the work proposed by Bokhoven et al. has some drawbacks: (i) it does not describe how mutual exclusions are tackled; and (ii) nothing is stated about preemptions.

Sgroi et. al. [41] propose a software synthesis method based on a Petri net subclass, namely, Free Choice Petri Nets (FCPN). The system specification is represented by a FCPN model, which is taken as an input in a quasi-static scheduling algorithm for partitioning the model into a set of tasks. Basically, the algorithm decomposes the model into a set of conflict-free nets with the purpose of finding possible solutions for each non-deterministic choice. Each solution is represented by a cyclic firing sequence, which is utilized to compose a feasible schedule, in a such way that memory constraints are met. After obtaining a feasible schedule, a C code is synthesized by traversing the schedule and replacing transitions with the corresponding code. Although the approach seems very promising, it does not directly deal with real-time constraints, which are left to a real-time operating system (RTOS).

Hsiung [42] proposes an extension of Sgroi's work [41] in order to take into account timing constraints. The new approach adopts a formal model, namely, Time Free-Choice Petri Net (TFCPN), which is a Free-Choice Petri Net extended with time. As described by the author, the TFCPN time semantics is equal to time Petri nets [13]. In that work, two scheduling are carried out: (i) quasi-static scheduling (for coping with task generation with limited memory), and (ii) dynamic fixed-priority scheduling (for meeting stringent timing constraints). Firstly, the quasi-static scheduling is performed, which is the same as in [41]. For each finite complete cycle of the conflict-free subnets, the execution time interval is calculated by summing up all earliest firing time and latest firing time values, respectively, of each transition in the sequence. Among all the execution time intervals of conflict-free subnets, the maximum latest firing time is selected as the worst-case execution time of the TFCPN. In this way, a real-time scheduling algorithm, such as rate monotonic or deadline monotonic, may be utilized to schedule the TFCPN. Lastly, the code generation is performed. For each TFCPN, a real-time process is generated, and, in each process, a task is created for each transition with independent firing rate.

Although the approach deals with stringent timing constraints, there are drawbacks: (i) no real-world experiments are presented; and (ii) it is not shown how to add preemption in the proposed method.

Amnell et al. [43] propose a framework for the development of real-time embedded systems based on timed automata extended with a notion of real-time tasks. The authors detail the translation process from the design model to executable programs, such that predictability is taken into account. In addition, the approach relies on a real-time operating system for controlling task executions during system runtime. More specifically, the framework utilizes a fixed-priority scheduling, which is appropriate for independent tasks. However, this scheduling policy may not provide feasible schedules when considering intertask relations, for instance, precedence and exclusion.

In [26], the authors describe a code generation approach for multitasking embedded systems with real-time characteristics. Tasks are modeled considering extended dataflow models as well as finite-state machines, and operating systems are in charge of controlling tasks during system execution. That work does not present how stringent timing constraints are guaranteed, and may incur considerable overheads due to the usage of operating systems.

Nácul and Givargis [44] present a technique for embedded software synthesis considering time-constrained applications . Such an approach adopts a specialized compiler that generates a monolithic code from a multitasking C application, in such a way that the utilization of an operating system is discarded. In spite of presenting interesting results, the authors do not give details how precedence and exclusion relations are handled.

Previous works propose methods that regard (stringent) timing constraints, but nothing is stated about energy consumption. As follows, some works are presented in the context of energy utilization.

Wang et al. [45] propose a software synthesis method that receives as input an embedded program to be optimized and generates improved code in terms of performance and energy consumption. The approach assumes that the computational complexity of a program is related to the values of input and intermediate program variables. Thus, for a given program, program parts (e.g., subprogram) are optimized for some input subspaces and the final code is augmented with such optimizations. Experimental results demonstrate that such method significantly reduces energy consumption as well as enhancing runtime performance. Nevertheless, nothing is stated about how timing constraints are considered in such method.

In [46], the authors describe a code generation method for a specific processor (Motorola DSP56K) in order to provide energy-efficient code. The approach packs a set of specific data transfer instructions into a single instruction word for simultaneous execution in order to provide less energy consumption than the equivalent sequential execution. Besides, address generation instructions are reduced as minimal as possible for shrinking the code size, and hence the energy consumption. Despite the efficiency of the method in terms of energy savings, the approach does not tackle timing constraints.

Sober et al. [47] propose a programming language as well as a runtime environment to allow the development of energy constrained applications. The language provides several features, for instance, it allows a programmer to specify program parts that are

eligible for energy reduction in order to provide different functionality or data quality considering the energy availability. Besides, the approach may generate code to several hardware platforms as well as operating systems. However, the authors do not present how time-critical applications can benefit from the method in terms of meeting stringent timing constraints.

Instead of what has been previously considered, this thesis proposes a software synthesis method for energy-constrained hard real-time systems, such that, from a feasible schedule, customized predictable code is obtained satisfying stringent timing constraints as well as an energy consumption upper bound enforced by the designer. More specifically, a pre-runtime scheduling is responsible for dealing with intertask relations, overheads, timing and energy constraints, while the code generation mechanism provides a customized runtime support to assure that all tasks are executed in accord with the pre-runtime schedule.

## 2.3 SUMMARY

This chapter summarized the representative works related with the proposed scheduling approach as well as the software synthesis method. Many works address the problem of scheduling hard real-time systems in conjunction with DVS for reducing energy consumption. Nevertheless, as presented, intertask relations and runtime overheads are usually disregarded. In the context of software synthesis, the scientific community has developed prominent methods for generating predictable code for real-time systems. However, not all methods consider stringent timing constraints and intertask relations. On the other hand, few works take into account energy constraints, but they do not deal with real-time systems. As a conclusion, there are several open issues in the research area of software synthesis (including scheduling), mainly, regarding hard real-time systems with energy constraints.

CHAPTER 3

# BACKGROUND

This chapter aims at presenting fundamental concepts for a better understanding of the proposed software synthesis method. Firstly, real-time systems and the representative scheduling policies (in the context of time-critical systems) are presented. Afterward, this chapter describes some concepts regarding energy consumption in embedded systems, focusing on DVS (Dynamic Voltage Scaling). As demonstrated further, DVS provides interesting results in comparison to other power-aware technologies. Next, software synthesis concepts are presented. Software synthesis is an interesting alternative to specialized operating systems concerning hard real-time applications, since such method provides customized code as well as services with lower overheads. Finally, discrete event models are introduced and the adopted formal model is detailed, namely, Petri nets.

Petri nets [10] are a family of formal models very well suited for modeling real-time embedded systems, since concurrency, synchronization and communication mechanisms - usual features of such systems - are naturally represented. More specifically, the proposed method adopts a Petri net extension, namely, time Petri net (TPN), which allows representing timing constraints such as deadline, release, as well as computation time. Besides, TPN provides a mathematical basis for precise development of scheduling methods, and also contains a set of well-established methods for structural and behavioral property analysis and verification. In comparison with other formal models, Petri nets may be classified as hybrid models, since they take into account states as well as actions, allowing the capture of causality relations, concurrent events and conflict conditions in an explicit manner. On the other hand, state-based models (e.g., finite-state machines), capture such information implicitly [48], in such a way that a complex system may lead to an explosion in the number of possible states that can be represented using such models. Causality relations, concurrent events and conflict conditions are useful for the designers, and, in some systems, the explicit representation can not be discarded. In relation to event-based models (e.g., process algebras), states are represented implicitly or even suppressed, leading to several difficulties for properly representing system resources (e.g., the number of CPUs). The explicit representation is mandatory in several situations in order, for instance, to simulate system behavior considering the reduction or increasing of available resources. Besides, in [49], the author provides a concrete example demonstrating that Petri nets can easily model the problem, whereas process algebras may not represent the same example easily.

## 3.1 REAL-TIME SYSTEMS

Real-time applications are presented in most of our daily activities, ranging from MP3 players to complex medical devices. These applications are a special class of systems in which not only the result of the computation is important to the correct system behavior,

but also the time when such result was acquired [8]. Another way to conceptualize such systems is the capability to react to external stimuli in a timely manner. Based on such concepts, it is worth explaining that real-time computing is not fast computing. Fast computing aims at getting the results as fast as possible, while real-time computing aims at obtaining the results within prescribed timing constraints. In general, these systems group many common characteristics, which are described as follows [8]:

- *Timely Response*. Real-time systems must respond to some external stimuli within prescribed time constraints.
- *Predictability*. Each system execution should run in a more or less similar manner, and the users should be able to deterministically say when each of the tasks is going to be executed.
- *Robustness*. The system should be immune to minor changes in its state and should be able to run without degradation as when it was originally designed.
- *Accuracy*. The system should provide accurate results. Sometimes it is impossible to compute accurate results in the given timing constraints. In this way, a trade-off between computation time and accuracy results is very important.
- *Concurrency*. Real-time systems should be distributed and provide parallel processing, since a system may have multiple independent sensors, providing stimuli to the system and requiring response from the system within a given time frame.

Real-time systems can be classified into two categories: hard real-time systems and soft real-time systems. Basically, the difference lies in the stringency of the predictability requirements of each class. Hard real-time systems require guaranteed predictable responses and behaviors. In these systems, if timing constraints are not met, catastrophic issues may occur, such as loss of human life. Examples of hard real-time systems are aircraft navigation, medical devices, and so on. Differently, soft real-time systems have a trade-off between computation time and the accuracy of the desired results. In such systems, if a timing constraint is not met, nothing critical happens, the system may only have its performance degraded. Examples of soft real-time systems are multimedia applications, electronic games, and so on.

Whenever designing real-time systems, two different design approaches can be adopted: event-triggered and time-triggered. In *event-triggered*, an activity (e.g., communication) is initiated whenever a significant change of state occurs. In *time-triggered*, all activities are initiated at predetermined points in time, in other words, they are synchronized with a periodic clock interrupt.

The focus of this thesis is on hard real-time systems, therefore, next subsections are devoted to this particular category.

### 3.1.1 Timing Constraints and Intertask Relations

Hard real-time systems are composed of a set of concurrent real-time tasks that must execute their respective activities before or at the specified deadlines. According to [6],

in order to provide predictability in hard-real-time systems, the major characteristics of the tasks must be known beforehand, otherwise it would be impossible to assure that all timing constraints will be satisfied during system execution. Therefore, the timing constraints presented in this section are only related to tasks that can provide the timing characteristics previously. As follows, timing constraints are presented using a task classification:

- **Periodic Task.** A periodic task performs a computation that it is repeated after a fixed period of time. This task is defined by a tuple $\tau_i = (ph_i, r_i, c_i, d_i, p_i)$, in which:
  - $ph_i$ is the initial phase (delay related to the first request after the start of the system);
  - $r_i$ is the release time (interval between the beginning of a period and the earliest time that an execution of task $\tau_i$ can be started in each period);
  - $c_i$ is the worst-case execution time (WCET) required for executing task $\tau_i$. In some works, $c_i$ is the worst-case execution cycles (WCEC) of task $\tau_i$. Considering a cpu speed $f$ (in terms of frequency) and a task WCEC, WCET = WCEC/$f$;
  - $d_i$ is the deadline (interval between the beginning of a period and the time by which an execution of task $\tau_i$ must be completed in each period);
  - and $p_i$ is the period;
- **Sporadic Task.** A sporadic task is executed randomly, but the minimum interval between two consecutive activations is known beforehand. This task is defined by a tuple $\tau_k = (c_k, d_k, min_k)$, in which:
  - $c_k$ is the worst-case execution time (WCET) (or WCEC) required for execution of task $\tau_k$;
  - $d_k$ is the deadline;
  - and $min_k$ is the minimum period between two activations of task $\tau_k$.

In addition to timing constraints, tasks may have relations with other tasks, more specifically, intertask relations, such as precedence and exclusion relations. A task $\tau_i$ *precedes* task $\tau_j$, if $\tau_j$ can only start executing after $\tau_i$ has finished. Precedence relations may exist when a task requires the information produced by another task. A task $\tau_i$ *excludes* task $\tau_j$, if no execution of $\tau_j$ can start while task $\tau_i$ is executing. If a single processor is considered, then task $\tau_i$ could not be preempted by task $\tau_j$. Exclusion relations may exist when some tasks must prevent simultaneous access to shared resources, such as I/O devices.

### 3.1.2 Scheduling

Scheduling is one of the most active research areas in real-time systems and it is related with policies by which tasks are given access to resources, most notably, CPU time. Considering hard real-time systems, scheduling policies play an important role, since stringent timing constraints must be met, otherwise catastrophic issues may occur. As consequence, several scheduling policies for hard real-time systems have been developed over the years, each one providing a particular advantage. These policies can be broadly classified as runtime (dynamic) or pre-runtime (static) approaches, and they are described as follows:.

- *Runtime approaches.* Runtime approaches make decisions at run time by selecting one of the current ready tasks for execution. The selection is based on priorities, in such a way that the task with highest priority is selected. In general, a schedulability analysis is performed before runtime using certain equations. If the equations are satisfied, it is assumed that all timing constraints will be met at runtime. Several runtime scheduling methods have been developed over the years. Some representatives are: (i) Rate Monotonic Scheduling [50], (ii) Earliest-Deadline First [50] and (iii) Priority Ceiling Protocol [51];

- *Pre-runtime approaches.* Pre-runtime approaches perform scheduling decisions offline, in other words, before runtime. It aims at generating a schedule table for a runtime component, namely, dispatcher, which is responsible for controlling the tasks during system execution. In order to adopt such approach, the major characteristics of the tasks must be known in advance. In general, this approach only consider periodic tasks. Nevertheless, it is possible to translate a sporadic task into a periodic task using the technique described in [52, 53]. In pre-runtime approaches, tasks are scheduled considering a schedule period ($P_S$) that corresponds to the least common multiple (LCM) between all periods in the task set. Within this new period, there are several *task instances* of the same task, in which $\mathcal{S}(\tau_i) = P_S/p_i$ gives the number of instances of each task $\tau_i$. For the $n$th task execution $\tau_{i_n}$ corresponding to the $n$th period, the release time is $r_{i_n} = r_i + p_i \times (n-1)$ and the respective deadline is $d_{i_n} = d_i + p_i \times (n-1)$. Several pre-runtime algorithms and techniques have been proposed. Some representatives are Xu and Parnas' algorithm [54], and Barreto's approach [16].

Besides, a scheduling approach can also be characterized whether an executing task may be interrupted (*preemptive scheduling*) or not (*non-preemptive scheduling*).

### 3.1.3 Runtime versus Pre-runtime Scheduling

Runtime as well as pre-runtime approaches have advantages and disadvantages. However, pre-runtime approaches have become more suitable for embedded hard real-time systems, since these approaches provide more predictable behavior than runtime counterparts. This section aims at providing a brief comparison between both scheduling classes. For more details, the reader should refer to [6].

When adopting a runtime approach, the amount of system resources required (e.g., memory) is greater than a pre-runtime approach, since a schedule is computed entirely online. Moreover, the runtime scheduling takes time, which leads to overheads that directly affect the system predictability. In other words, tasks may miss their respective deadlines. On the other hand, pre-runtime approaches compute the schedule in advance. When using this approach, overheads are greatly reduced, since just a tiny dispatcher will be executing in addition to the real-time tasks.

Real-world time-critical applications are composed of several tasks with their respective timing constraints and intertask relations (e.g., precedence and exclusion relations). In a runtime approach, it is usually difficult to extend schedulability analysis to consider additional constraints, such as intertask relations, because additional application constraints are likely to conflict with existing priorities. In general, it may be unfeasible to map application constraints into a fixed hierarchy of priorities. In contrast, a pre-runtime approach can compute an off-line schedule considering additional constraints without being restricted by any priority scheme.

The runtime scheduling approach requires complex mechanisms for providing task synchronization and prevent simultaneous access to shared resources. In addition to the overheads, such mechanisms may conduct the system to a deadlock state. In contrast, in pre-runtime scheduling, there is no concern related to deadlocks, since a feasible schedule is guaranteed to be deadlock-free whenever it is found.

Another drawback of runtime approaches is that they have less chance of finding a feasible schedule than a pre-runtime scheduling algorithm. For instance, let us consider the task set consisting of two tasks, $A$, $B$, and the respective timing constraints (release, computation, and deadline): $A = (0, 10, 12)$; $B = (1, 1, 2)$. This specification also considers that $B$ can not preempt $A$. Figure 3.1(a) shows that a runtime approach could not find a feasible schedule, since task $B$ misses its deadline. However, a pre-runtime approach finds a feasible schedule (Figure 3.1(b)). It is worth observing that the processor must be left idle between time 0 and 1, even though $A$'s release time is 0.

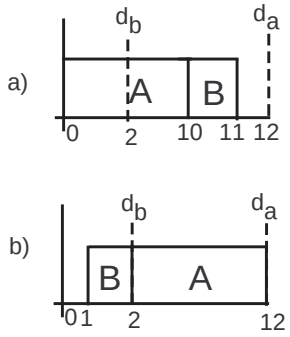


**Figure 3.1** Comparison between runtime and pre-runtime scheduling

Although pre-runtime approaches can provide more predictability than runtime counterparts, the former also have drawbacks. The main drawback is the static nature of

schedules, which can not be changed during system execution. However, some flexibility may be obtained by mixing runtime and pre-runtime approaches. Such technique is usually denominated as hybrid approach. For instance, some alternative schedules may be generated off-line, and, during system execution, a small scheduler can be adopted to switch schedules according to changes in the environment [55]. Another alternative is related to sporadic tasks. A runtime scheduler can be adopted to allocate resources for sporadic tasks in a dynamic manner [56, 57], instead of relying only on the pre-runtime way.

## 3.2 ENERGY CONSUMPTION IN EMBEDDED SYSTEMS

Several factors have propelled researching efforts for increasing energy saving in embedded systems. One of the major factors is the widespread expansion of mobile devices market, which has forced embedded systems companies to deal with several additional challenges in order to provide complex systems in this market niche. As portable devices generally rely on constrained energy sources (e.g., battery), energy consumption is an important challenge that needs to be tackled.

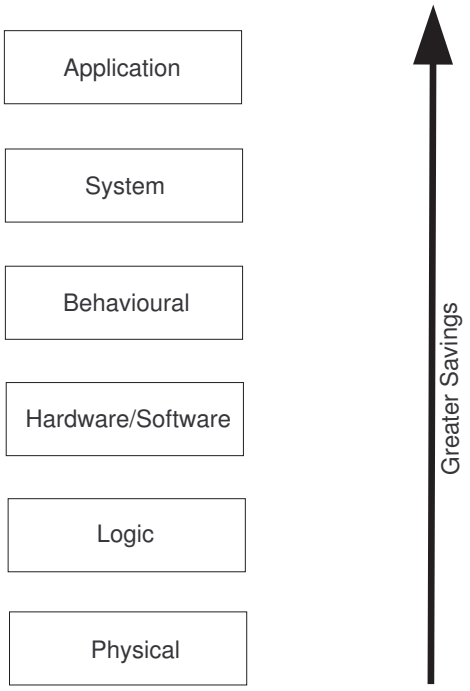


**Figure 3.2** Energy savings at different design levels

Considering the design levels of embedded systems [58] (Figure 3.2), techniques for reducing energy consumption can be applied from physical to application level. Without loss of generality, techniques at application level concern in providing functionalities with different modes, such that each mode performs the same job with a lower energy cost. For instance, a portable MP3 player may provide a playback mode with lower quality

(due to the reduced precision of the decompression technique) to provide better energy savings. At the system level, designers analyze the trade-off between hardware and software implementations not only considering energy savings, but also flexibility. Regarding behavioral level, application specific computational kernels, such as algorithms, are optimized in the context of power efficiency. The hardware design also plays an important role in determining how power efficient an algorithm implementation will be, for instance, the trade-off between parallelism against the core size. At the structural hardware-design and physical levels, various techniques can be used to reduce energy consumption. Although energy savings at such levels do not seem significant in comparison with other abstraction levels, if careful design is not taken into account in those lower levels, the benefits at higher ones can be destroyed.

It is important to bear in mind the impact of software in energy consumption, since, in many embedded systems, more than 70% of functionalities are implemented as software nowadays [4]. Thus, this section focuses on a well-established technique for reducing energy consumption from application to behavioral level, namely, dynamic voltage scaling (DVS). Firstly, dynamic power management (DPM) is explained, which is the category the DVS technology is inserted.

### 3.2.1 Dynamic Power Management - DPM

In recent years, dynamic power management (DPM) mechanisms have been extensively adopted in the context of managing energy consumption in embedded devices. As stated in [59], DPM is a design methodology that focuses on runtime reconfiguration of electronic systems in order to provide a minimum number of active components or reduce the performance of such components for increasing energy savings. DPM mechanisms rely on the basic assumption that systems have nonuniform workload during execution. Therefore, these methods can manage system components for increasing energy savings, in such a way that system constraints are still met. Usually, DPM mechanisms are implemented as part of a service in operating systems.

DPM mechanisms are broadly classified in two groups [60]. The first group is related to shut-off system components, such as hard disks, displays, and even CPUs, when they are idle or underutilized. This group is also termed as *DPM*. The second group refers to the dynamic control of supply voltage level of system components, in such a way that the performance as well as the energy consumption are decreased. This group is denominated as *dynamic voltage scaling (DVS)*, and it has been adopted, most notably, to reduce energy consumption in CPUs. Indeed, several DVS-capable processors are available on the market nowadays, such as Intel XScale [61], AMD Mobile Athlon [62] and Transmeta Crusoe processor [63]. Additionally, some research groups have developed their own DVS platforms, such as the platform based on Philips LPC2106 described in [64]. Ideally, a DVS CPU would operate at any supply voltage level considering a specific range and switch from one voltage to another without incurring overheads. Nevertheless, overheads (time as well as energy) always occur during a voltage switching in DVS-capable processors [27]. Besides, the DVS CPUs available on the market operate at discrete voltage levels [27].

When considering ordinary mobile embedded systems, DVS and DPM techniques may be directly applied without arising important issues. Nevertheless, in real-time systems, such techniques need to be adopted with caution, since responsiveness may be greatly affected. In hard real-time systems, catastrophic issues may occur due to timing constraint violations, which can lead to equipment damage or even loss of human life. Thus, several scheduling approaches have been developed in order to cope with DVS and DPM in time-critical systems.

For the rest of this work, the focus is the adoption of DVS technologies for reducing CPU energy consumption in the context of hard real-time systems. CPU is a major source of energy consumption [30], and, when both DVS and DPM are available for CPUs, it is always advantageous to exploit DVS first [14]. Next section describes how DVS works and, afterwards, a comparison between approaches for hard real-time systems scheduling, taking into account DVS, is presented.

### 3.2.2 Dynamic Voltage Scaling - DVS

As aforementioned, CPUs significantly impact the energy consumption of several systems and dynamic voltage scaling can provide meaningful energy savings during system runtime. The following paragraphs explain the rationale behind DVS technology.

The energy consumption of MOS microprocessors (in active status) can be described as [65, 66]

$$P_{active} = P_{dynamic} + P_{static} \tag{3.1}$$

in which $P_{dynamic}$ and $P_{static}$ are the dynamic and static power consumption, respectively. $P_{dynamic}$ is defined as[2, 67]

$$P_{dynamic} = A.V^2.f_{clk} \tag{3.2}$$

in which $A$ is the average switched capacitance per clock cycle, $V$ is the voltage supplied, and $f_{clk}$ is the clock frequency. Additionally, $P_{static}$ is represented as [65, 68]

$$P_{static} = I_{leak}.V \tag{3.3}$$

where $I_{leak}$ is the leakage current.

DVS relies on the premise that power consumption in MOS microprocessors is most influenced by the dynamic power, in other words, the dynamic power is dominant ($P_{active} \propto P_{dynamic}$). As dynamic power consumption has a quadratic dependence on voltage, lowering the supply voltage is the most effective way to reduce power consumption. Considering energy, equation 3.2 can be expressed as [69]

$$E_{dynamic} = A.V^2.N \tag{3.4}$$

in which $N$ is the number of cycles for the execution of a given task.

However, lowering the supply voltage increases circuit delay, as described in the following equation [12]

$$T_d = \frac{kV}{(V - V_t)^2} \propto \frac{V}{(V - V_t)^2} \tag{3.5}$$

where $T_d$ is the delay, $V$ is the supply voltage, $V_t$ is the threshold voltage and $k$ is a constant related to the technology. Considering previous equation, the maximum operating frequency is linearly proportional to the supply voltage ($f \propto V$). Therefore, DVS may be seen as a technique for trading off energy consumption and performance. It is important to state that concerns have emerged in the context of leakage current, but this thesis assumes the dynamic power is a major source of energy consumption.

For a better visualization about the gains that can be obtained using DVS, consider the following example, which was extracted from [12]. In this example, it is assumed a CPU that consumes 10nJ/cycle, 25nJ/cycle and 40nJ/cycle at 2.5V, 4.0V and 5.0V, respectively. Additionally, the maximum clock frequencies are 50MHz at 5.0V, 40MHz at 4.0V, and 25MHz at 2.5V. These assumptions are based on equations (3.4) and (3.5). Figure 3.3 [12] shows three kinds of voltage schedules for a given program whose worst-case execution cycles are $1000 \times 10^6$ cycles and the deadline constraint is 25 seconds.

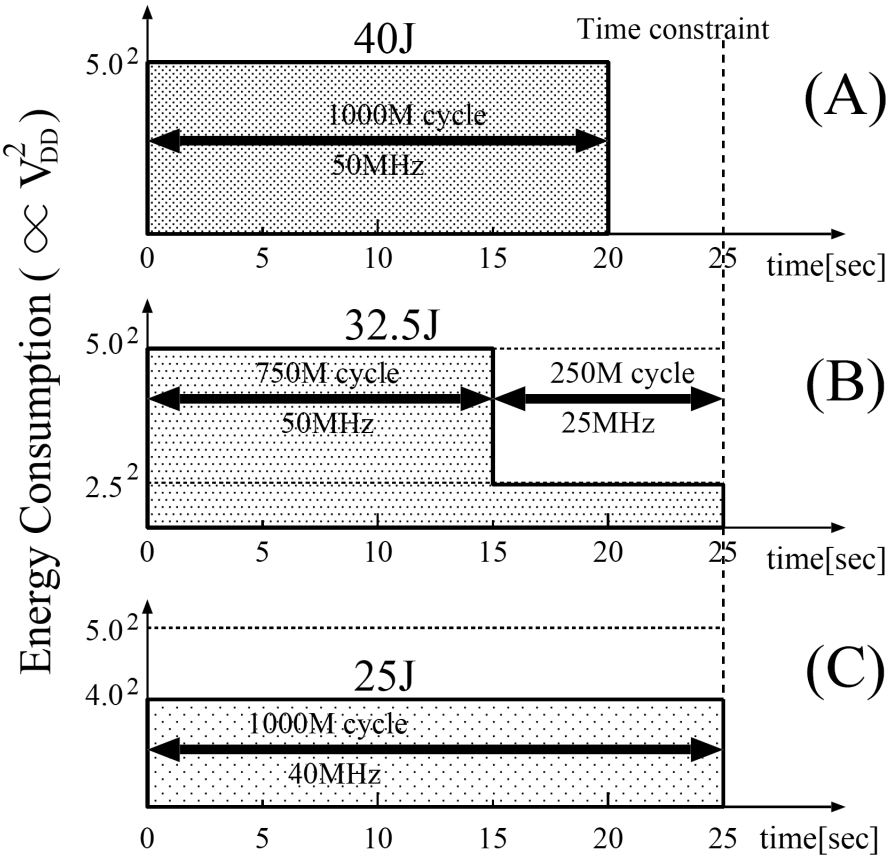


**Figure 3.3** DVS example

In Figure 3.3(A) [12], the total energy consumption is 40J at 5.0V, even if the power supply is turned off after finishing the program. Figure 3.3(B) depicts an alternative schedule combining 2.5V and 5.0V, in such a way the program execution is stretched to the deadline. In this case, energy consumption is reduced from 40J to 32.5J. Figure 3.3(C)

shows the ideal case for this example. If the processor uses a single supply voltage in which the program execution time is equal to the deadline, the total energy consumption is minimized.

**Classification** Several DVS algorithms have been proposed over the last decade, in such a way that categories are necessary to classify those algorithms. According to [30], DVS algorithms may fall into two categories:

- *Intertask DVS algorithms.* Intertask DVS algorithms [19, 64] assign a CPU supply voltage (and the respective maximum operating frequency) for a given task instance at the task level. In other words, once a CPU voltage is assigned to a task instance, the voltage is not changed until the instance is preempted or completed;
- *Intra-task DVS algorithms.* Intra-task DVS algorithms [70, 71] adjust CPU voltage (and the respective maximum frequency) within the boundaries of a task instance. In this case, the CPU voltage is gradually increased, in such a way that the task deadline is not violated.

Intra-task approaches commonly rely on compiler support to insert power management points inside the application code for allowing voltage scaling. On the other hand, intertask algorithms do not perform any intrusive modifications, since these algorithms are usually implemented as part of an operating system service, such as scheduling service. Since intertask DVS algorithms are more practical than intra-task algorithms due to the absence of changes in the application code [30], next section describes a comparison between some intertask techniques for hard real-time systems scheduling considering DVS.

### 3.2.3 Hard Real-Time Systems Scheduling Considering DVS

Although DVS can provide considerable energy savings, this technology needs to be adopted with caution in hard real-time systems due to the stringent timing constraints. As consequence, several (runtime and pre-runtime) scheduling approaches have been developed in order to cope with DVS in time-critical systems. As follows, a comparison is provided only assuming the impact of dynamic power.

In real systems, tasks often need to communicate between themselves or access shared resources in a mutual exclusive way. As shown in Section 3.1, the former is modeled as precedence relations and the latter as exclusion relations. Ignoring such constraints can lead to undesirable results during system execution, and, in the context of hard real-time systems, catastrophic issues can occur. As far as the constraints are available, pre-runtime approaches can provide feasible schedules, whereas runtime methods may fail. Previous affirmation still holds for DVS-capable systems, as depicted by the following example. Assume the following task set $\mathcal{T} = \{\tau_1 = (0, 0, 150 \times 10^6, 6, 13), \tau_2 = (0, 2, 50 \times 10^6, 3, 13), \tau_3 = (0, 0, 100 \times 10^6, 13, 13), \tau_4 = (0, 7, 60 \times 10^6, 9, 13)\}$. In this example, each task is represented by a tuple $\tau_i = (ph_i, r_i, c_i, d_i, p_i)$, where $ph_i$ is the initial phase; $r_i$ is the release time; $c_i$ is the worst-case execution cycles (WCEC) required for executing

task $\tau_i$; $d_i$ is the deadline; and $p_i$ is the period. In addition to timing constraints, the specification contains the following relations: $\tau_1$ *excludes* $\tau_2$, $\tau_1$ *precedes* $\tau_3$, $\tau_2$ *excludes* $\tau_1$, and $\tau_2$ *precedes* $\tau_4$. For this example, a DVS platform based on [64] is adopted, taking into account the following supply voltage/frequency levels: 1.21V/20MHz, 1.39V/30MHz and 1.76V/50MHz. Moreover, considering an average switching capacitance of 0.28nF per clock cycle [72], the energy consumption is 0.87nJ/cycle at 50MHz, 0.54nJ/cycle at 30Mhz, and 0.41nJ/cycle at 20MHz. These values were obtained from Equation 3.4.

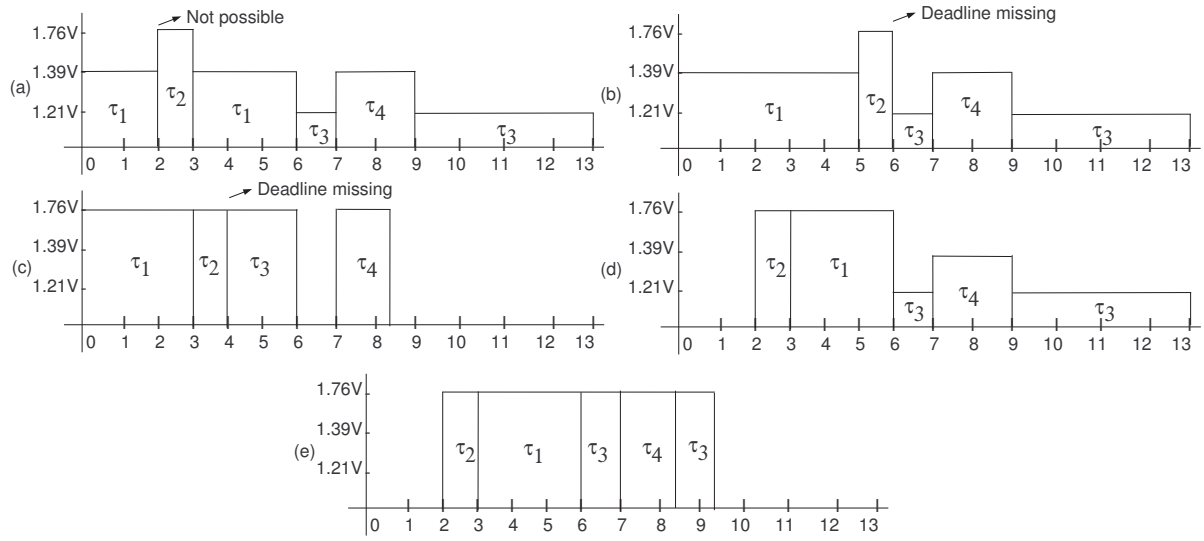


**Figure 3.4** Schedules generated according to different scheduling methods

Figure 3.4(a) shows a schedule obtained by adopting the optimal off-line DVS algorithm defined by [25], which relies on the optimal runtime scheduling method Earliest-Deadline First(EDF). The schedule is invalid, since $\tau_1$ cannot be preempted by $\tau_2$. As an alternative, Figure 3.4(b) shows a schedule obtained by blocking the execution of $\tau_2$ while $\tau_1$ is executing. Again, the schedule is infeasible, since $\tau_2$ misses its deadline due to the earlier release of $\tau_1$. Even discarding DVS, in other words, running every task at the maximal voltage/frequency level, a schedule could not be found if EDF is adopted (or other runtime scheduling algorithm) (see Figure 3.4(c)). On the other hand, Figure 3.4(d) depicts a schedule found using the proposed pre-runtime scheduling algorithm. All tasks meet their deadlines, and additional energy savings can be obtained comparing to a pre-runtime schedule without DVS (Fig. 3.4(e)). Fig. 3.4(d) presents a schedule that consumes 0.2474J, while the schedule depicted in Fig. 3.4(e) utilizes 0.3132J. Additionally, note that the processor needed to be left idle to find a valid schedule [6]. For the sake of simplicity, overheads have not been taken into account and the CPU was assumed to have a halt instruction that avoids energy consumption on idle state.

## 3.3 SOFTWARE SYNTHESIS

An embedded system is a system whose principal function is not only computational, but which is controlled by a computer embedded within it. Such computer may be a microprocessor or microcontroller. The meaning of the word embedded implies that the computer lies inside the overall system, hidden from view, forming an integral part of a greater whole. As a result, the user may be unaware of the computer's existence. Unlike a general-purpose personal computer, embedded system computer is usually purpose designed, or at least customized, for the single function of controlling its system. Besides, embedded systems do not terminate, unless it fails [73].

In order to provide flexibility, lower costs as well as to reduce time-to-market, most part of embedded system functionalities is implemented as software components nowadays. Nevertheless, the development of embedded softwares is not an ordinary activity. For instance, some (non-functional) requirements, generally considered as secondary to the functional correctness of PC-like software applications, are crucial for embedded software correctness. Additionally, while conventional PC softwares generally abstract the physical world, embedded software engage with it.

Taking into account the previous issues, *correct-by-construction* techniques play an important role in embedded software development, as software can be partially or fully generated satisfying several constraints and properties. In this context, software synthesis methods are an feasible solution. According to Cornero et.al. [15], software synthesis is the task of converting automatically a specification (typically composed of concurrent and communicating tasks) into a software considering:

- the specified functionalities;
- the required runtime support;

In other words, software synthesis methods translate a high-level specification into a program source code including the operational support code for allowing the software execution.

Software synthesis is necessary since specifications have special characteristics which are not found in traditional programming languages. For instance, specifications are generally composed of several concurrent tasks, so, scheduling and synchronization of multiple tasks are important issues. Considering multiple processors, additional issues should be considered, such as inter-processor communications. In these particular situations, software synthesis should provide an appropriate scheduler, in such a way that all specified constraints are satisfied. Thus, software synthesis consists of two main activities [15](see Figure 3.5):

- task handling;
- code generation.

Task handling takes into account tasks' scheduling, resource management, and intertask communication. Code generation is responsible for static generation of system source code.

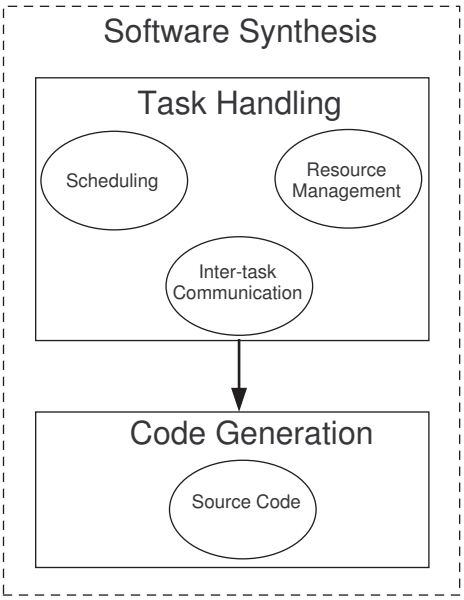


**Figure 3.5** Software synthesis activities

In general, complex systems adopt a specialized operating system in order to provide a runtime support to the respective software application. However, operating systems are very general and usually introduce overheads, mainly in execution time and memory requirements. On the other hand, software synthesis methods are an alternative to operating system usage, since these methods automatically generate customized codes, in such a way that all constraints are met and overheads are minimized. Considering embedded applications, software synthesis is of great importance, since constrained resources can be utilized in an efficient manner. Besides, due to greater overheads, operating systems may consume more energy than software synthesis approaches. In the context of hard real-time time systems with stringent energy requirements, the overheads related to operating system usage may compromise system constraints.

Although software synthesis methods may provide better results than operating systems in the context of hard real-time systems, these methods also have drawbacks. Since software synthesis methods are generally designed considering a specific type of application, the adoption to other types may not be so straightforward as operating systems. In relation to portability, operating systems may be more portable. However, techniques such as retargetable compilers can solve the portability problem in software synthesis [74].

## 3.4 DISCRETE EVENT MODELS

Hard real-time systems require careful design in order to avoid (catastrophic) issues during system execution. Regarding these systems, discrete event models provide several benefits, in the sense that system behaviour (and properties) can be reasoned at design-time. Prevalently, those models have been adopted to represent systems with the following

characteristics:

1. The state space is a discrete set;

2. The state transition mechanism is event-driven.

Systems with these characteristics are denominated as *discrete event systems (DES)*, and the hard real-time systems of interest belong to such category. As follows, the concept of model is presented, and, next, representative formal models in the context of DES systems are described. This section is based on [75] and [48].

### 3.4.1 Models

Scientists as well as engineers are usually concerned with the quantitative analysis of systems, and the development of techniques for design, control, and the explicit measurement of system performance based on well-defined criteria. As consequence, a model is needed for assisting in such tasks. In general, a model may be viewed as a device that simply duplicates the behavior of the system itself. In many cases, mathematical models are adopted for representing such behavior.

In order to perform the modeling process, it is necessary to define a set of *measurable variables* associated with a given system. By measuring such variables over a period of time, data can be collected. Next, a subset of these variables is selected and it is assumed that they can vary over time. This defines a set of time functions, which are called *input variables*. Then, another set of variables, namely, *output variables*, is selected for measurements, while varying the input variables. Output variables may be interpreted as the response to the stimulus provided by the selected input functions. Additionally, it is worth stating that not all system variables may be associated with either the input or output. Such variables are generally identified as *suppressed* output variables.

Strictly speaking, system is the real thing, whereas model is an abstraction, such as a set of mathematical equations. In other words, a model is an approximation of the real behavior of the system. Nevertheless, once a good model is obtained, the terms system and model may be used in an interchangeable manner. This situation is depicted on Figure 3.6. Besides, it is worth pointing out that it is always possible (in principle) to obtain a model from a system, but the converse is not true. As an analogy, since systems may be modeled as mathematical functions, an inverse function is not always possible to obtain.

Depending on the desired view of the system, several combinations of input and output variables may be selected during the modeling process. It is up to the modeler to identify the variables of interest and consider them in the final model. This flexibility may provide several models with different aspects of the same system.

### 3.4.2 Formal Models

Whenever designing embedded real-time systems, models play an important role, since system parts can be visualized together with their respective interactions. Nevertheless, a model to be useful should possess certain qualities, which are discussed as follows. The

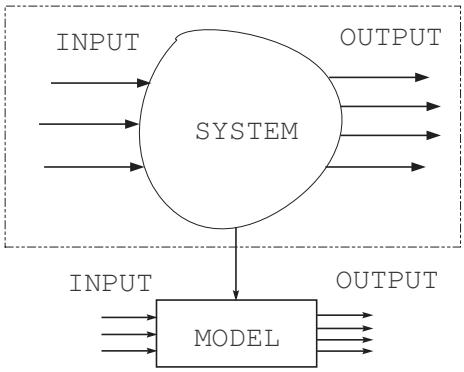


**Figure 3.6** Modeling process

model should be *complete* in the sense that it describes the parts of interest. Additionally, the model should be *comprehensible* to the designers, in the sense that it should be easy to understand. Finally, the most important characteristic is related to *unambiguity*. The model must describe the system without ambiguity in order to capture precisely the features of interest. Such characteristic is of utmost importance for hard real-time systems, since if timing constraints are not properly represented, undesirable results may occur with the final system, such as equipment damage or even loss of human lives (see Section 3.1).

Taking into account the previous requirements, formal models can provide most of the desired features, mainly the possibility of describing system functionalities precisely. In fact, formal models are based on mathematical foundations, which allow the verification and the analysis of qualitative as well as quantitative properties. Such characteristics are of great importance for designing time-critical systems.

Due to the diversity of formal models, a classification is necessary for visualizing the main ideas of each model class. Hence, this work adopts the classification described in [76], which groups formal models as state-based models and event-based models. Besides, a supplementary class is considered, namely, heterogeneous models. The classification is described below:

1. **State-based models.** The model contemplates all possible states that the system under review could be throughout execution. For instance, a state may include the values of all program variables, e.g.: Automata and Finite-State Machines;

2. **Event-based models.** The model comprises all event sequences that can occur during system execution. Such events represent system actions, where an action can be, for instance, a variable assignment or a function call, e.g.: Process Algebras.

3. **Heterogeneous models.** The model considers states as well as events that can occur in the system. Differently from previous models, a state actually represents a local state (e.g., the value of a program variable), whereas an event represents

an action taking into account a specific local state(s) as precondition(s), e.g.: Petri nets.

Although state-based models as well as event-based models have been adopted for formal specification and verification of several systems, such models capture causality relations, concurrent events and conflict conditions in an implicit manner [48]. Nevertheless, such information is useful for the designers, and, in some systems, the explicit representation is desirable. Additionally, without explicit support for concurrency, a complex system will lead to an explosion in the number of possible states that can be represented. In the case of heterogeneous models (e.g., Petri nets), such information can be explicitly described, since these formal models provide more structure than the other classes. On the other hand, depending on the application, heterogeneous models may not possess the same analytical power as state-based models [75]. Furthermore, although Petri nets (heterogeneous model) are usually adopted for purposes of system validation and simulation, process algebras (event-based models) are often adopted in system verification [77].

In the beginning, this chapter explains the adoption of Petri nets in the proposed work, however, it is important to provide more details regarding the representative models of each class, for instance, to better visualize the benefits provided by Petri nets in the modelling of real-time systems. As follows, the representative models of each class are presented, more specifically, Finite-State Machines and Process Algebras. Petri nets are detailed in Section 3.5

### 3.4.3 Finite-State Machine

A finite-state machine (FSM) is a good example of a state-based model that has been largely adopted in the modeling of control systems. The suitability is based on the behavior of control systems, which are naturally represented by states and the transitions between states. Basically, a FSM model consists of a set of states, a set of transitions between states, and a set of actions associated with these states or transitions. As follows, a formal definition of FSM is presented.

**Definition 3.1** (Finite-state Machine)**.** *A FSM is a tuple ($S$, $I$, $O$, $f$, $g$, $s_0$), where $S$ is a finite set of states, $I$ is a finite input alphabet, $O$ is a finite output alphabet, $f : S \times I \to S$ is the transition function that assigns to each state and input pair a new state, $g : S \times I \to O$ is an output function that assigns to each state and input pair an output, and $s_0 \in S$ is the initial state.*

In addition to the formal definition, a FSM can be graphically represented using a state diagram, which is a directed graph with labeled edges. In the graph, each node represents a state, and each labeled edge depicts the input and output pair of a given transition. As an example, Figure 3.7 [78] shows an elevator controller modeled using FSM. The respective algebraic representation is ($S = \{s_1, s_2, s_3\}$, $I = \{r_1, r_2, r_3\}$, $O = \{u_1, u_2, d_1, d_2, n\}$, $f = \{(s_1, r_1, s_1),(s_1, r_2, s_2),(s_1, r_3, s_3),(s_2, r_1, s_1),(s_2, r_2, s_2),(s_2, r_3, s_3),(s_3, r_1, s_1),(s_3, r_2, s_2),(s_3, r_3, s_3)\}$,$g = \{(s_1, r_1, n),(s_1, r_2, u_1),(s_1, r_3, u_2),(s_2, r_1, d_1),(s_2, r_2, n),(s_2, r_3, u_1),(s_3, r_1, d_2),(s_3, r_2, d_1),(s_3, r_3, n)\}$,$s_1$). It is worth noting that functions $f$ and $g$ are defined as relations. For a better comprehension, each state represents the floor where the elevator is

located. The set of inputs $I$ represents the floor desired. For instance, $r2$ means that floor 2 is requested. The set of outputs $O$ represents the direction as well as the number of floors the elevator needs to move. More specifically, $u_x$ (e.g., $u_1$) indicates that the elevator needs to move up $x$ floors, $d_x$ (e.g., $d_2$) represents that the elevator needs to move down $x$ floors, and $n$ represents that the elevator stays at same floor.

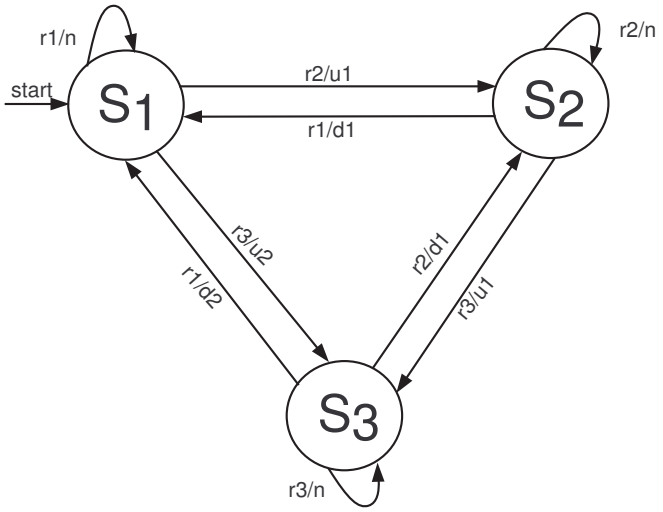


**Figure 3.7** Elevator controller model

Over the years, finite-state machines have been extended to model different kinds of systems. The type of FSM presented so far is named as transition-based or Mealy machine, which is characterized by the output value be associated with the state and input values ($g : S \times I \rightarrow O$). Another important type of finite-state machine is state-based or Moore machine. In this case, the output value depends only on the state of the FSM ($g : S \rightarrow O$). In short, the main difference between both models is related to the number of states. State-based FSM may require more states than transition-based FSM, since the latter allows multiple arcs pointing to a single state, each arc having a different output value. On the other hand, each output value would require its own state in state-based FSM. In order to provide a better visualization, Figure 3.8 [78] depicts the elevator controller model using Moore's approach.

Besides system modeling, another fundamental application of FSM is in the construction of language recognizers. Such application have a prominent role in the design of compilers for programming languages. For this case, there is a suitable type of finite-state machine, namely, finite automata or finite-state automata. Without loss of generality, instead of generating outputs, finite automata have final states, in such a way that a string is recognized if it takes the starting state to one of these final states. Considering the behaviour of a system, strings may also be interpreted as event sequences (state transitions) of the form $e_1e_2...e_n$, in such a way that they represent the evolution of the respective system.

**Definition 3.2** (Finite Automaton)**.** *A finite automaton is a tuple ($S$, $I$, $f$, $s_0$, $F$), where*

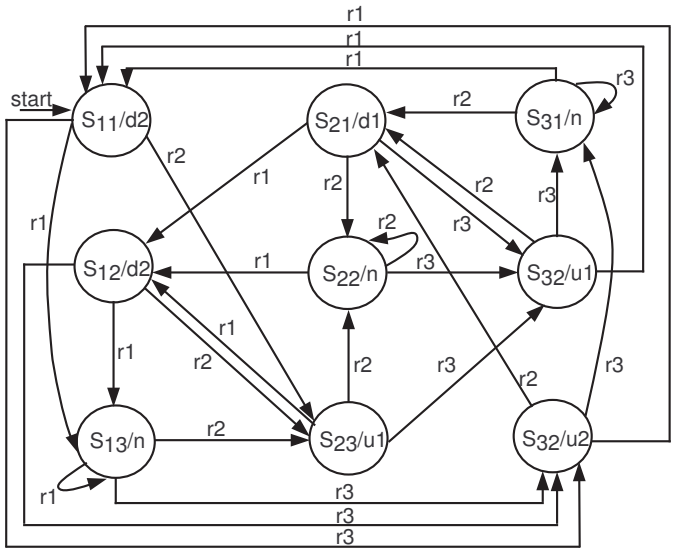


**Figure 3.8** Moore FSM model for the elevator controller

*$S$ is a finite set of states, $I$ is a finite input alphabet, $f : S \times I \to S$ is the transition function that assigns to each state and input pair a new state, $s_0 \in S$ is the initial state, and $F \subseteq S$ is the set of final states.*

**Definition 3.3** (Language)**.** *A language defined over an alphabet $I$ is the set of finite-length strings generated from the alphabet $I$.*

As an example, take a look at the following tuple, which is a finite automaton: $(S = \{s_0, s_1\}$,$I = \{0, 1\}$,$f = \{(s_0, 1, s_0), (s_0, 0, s_1), (s_1, 0, s_1), (s_1, 1, s_1)\}$,$F = \{s_0\})$. The respective graphical representation is depicted in Figure 3.9. Before presenting the language recognized by this automaton, the transition function $f$ is extended to consider all pairs of states and strings: $f : S \times I^* \to S$, where $I^*$ represents all possible strings generated from $I$. Considering $x = x_1x_2...x_k$ a string in $I^*$, $f(s_1, x)$ is the state obtained by using each successive symbol of x as input, from left to right, starting from state $s_1$. For a better understanding, from $s_1$, state $s_2 = f(s_1, x_1)$ is reached, then $s_3 = f(s_2, x_2)$, and so on, with $f(s_1, x) = f(s_k, x_k)$

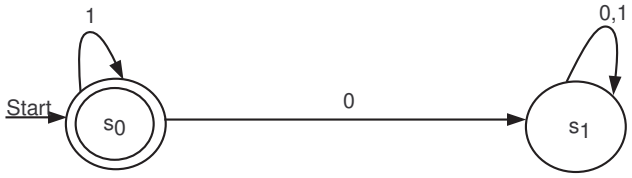


**Figure 3.9** Finite automaton

**Definition 3.4** (Language accepted by an automaton)**.** *A language accepted or recognized by an automaton is defined as* $\mathcal{L}(M) = \{x \in I^* | f(s_0, x) \in F\}$*, where* $M$ *is an automaton,* $x$ *is a string,* $f : S \times I^* \to S$ *is the transition function of* $M$*,* $s_0$ *is the initial state of* $M$*, and* $F$ *is the set of final states of* $M$*.*

For the automaton depicted in Figure 3.9, the language recognized is $\mathcal{L}(M) = \{1^n | n \geq 0\}$. It is worth stating that if two finite-state automata recognize the same language, they are considered equivalent.

The finite automata presented so far is defined as deterministic, since each pair of state and input value has only one next state ($f : S \times I \to S$). Another important type of FSM is nondeterministic finite automata, which may have for each pair of state and input value more than one next state.

**Definition 3.5** (Nondeterministic Finite Automaton)**.** *A nondeterministic finite automaton is a tuple* $(S, I, f, s_0, F)$*, where* $S$ *is a finite set of states,* $I$ *is a finite input alphabet,* $f : S \times I \to P(S)$ *is the transition function that assigns to each state and input pair a set of states,* $s_0 \in S$ *is the initial state, and* $F \subseteq S$ *is the set of final states. In this definition, P(S) means the power set of S.*

A comparative example between deterministic and nondeterministic finite automata is presented in Figure 3.10 [78]. The problem in question is related to the construction of a language recognizer that accepts all strings over {0,1} containing a 1 in the third position from the end (e.g., 000100). Figure 3.10(a) depicts a nondeterministic automaton, whereas Figure 3.10(b) shows an equivalent deterministic automaton. It is worthwhile noting that Figure 3.10(a) is much smaller than its deterministic counterpart, and the respective behaviour is easier to understand. To conclude, if a language $\mathcal{L}$ is recognized by a nondeterministic finite automaton, the same language can be recognized by a deterministic automaton.

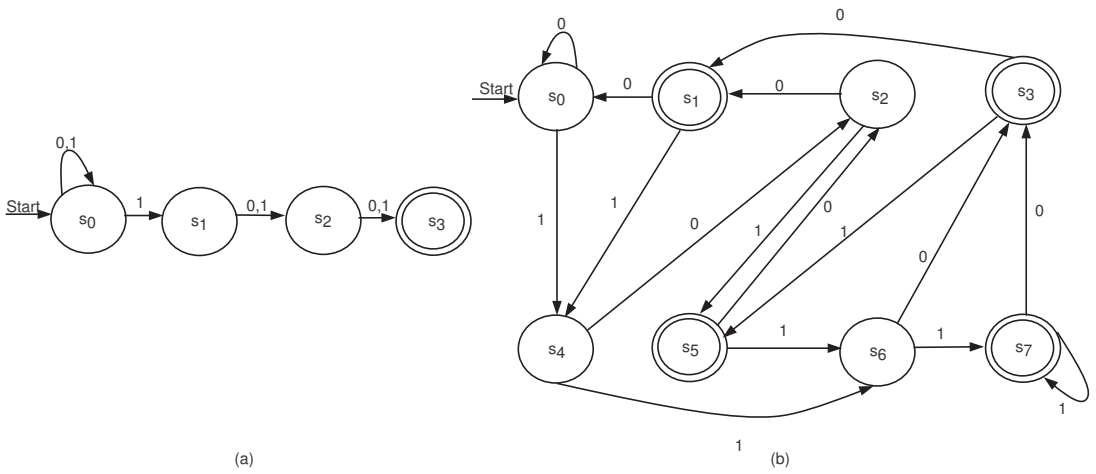


**Figure 3.10** Deterministic and nondeterministic finite automata

### 3.4.4 Process Algebra

This section provides information about process algebras, and the respective content is based on [79]. Process algebras have been adopted as a mathematical framework for reasoning about the behavior as well as the structure of distributed and reactive systems. The term *process algebra* denotes the adoption of an algebraic approach for describing the behavior of a system. Indeed, the word *process* refers to the behavior, which is the total of events and actions that a system can execute, the sequence in which they can be performed, and, possibly, other characteristics, such as timing or probabilities. The main idea of a process algebra model is to provide an observation of the behavior, where an action is the unit of observation. Generally, the actions are considered discrete, in the sense that they occur at some time instant, and different actions are separated in time. Considering the previous statement, this is the reason that a process is also named as *discrete event system*.

The simplest model of behavior is the function model, in the sense that a value (input) is given at the beginning of the process, and at some moment a result (output) is obtained. This model was adopted to model the behavior of a computer program from the start of the subject in the middle of the twentieth century. Such approach was of great help in the development of automata theory (presented in previous section). However, this model lacks an important feature, namely, interaction. This concept is needed to describe parallel or distributed systems, since a system may interact with another during the execution from the initial state to the final state.

Process algebra is the study of parallel or distributed systems by algebraic methods. Process algebra provides resources to describe or specify such systems, and, thus, it has means to allow parallel composition, alternative composition and sequential composition. Additionally, the system can be reasoned through equational reasoning, which can allow verification.

Several process algebras have been developed over the years, each one providing some differentiated features. The three most well-known process algebras are cited below [80]:

- **Calculus of Communicating Systems(CCS)** [81];
- **Communicating Sequential Processes(CSP)** [82];
- **Algebra of Communicating Processes(ACP)** [79].

The basic rules of process algebras are usually named as structural or static laws, since no action execution is explicitly described. From a given set of atomic actions, basic operators are adopted to compose such actions into more complex processes. In general, the basic operators [80] are + or | representing alternative composition, ; or $\rightarrow$ representing sequential composition, and || representing parallel composition. The basic rules are presented as follows (+ binding weakest, ; binding strongest).

- x + y = y + x (commutativity of alternative composition)
- x + (y + z) = (x + y) + z (associativity of alternative composition)

- x + x = x (idempotency of alternative composition)
- (x + y); z = x; z + y; z (right distributivity of + over ;)
- (x; y); z = x; (y; z) (associativity of sequential composition)
- x || y = y || x (commutativity of parallel composition)
- (x || y) || z = x || (y || z) (associativity of parallel composition)

The formal semantics of process algebras are defined in term of states. More specifically, such algebras have a formally defined structured operational semantics that map process algebra terms onto an automaton, called labeled transition system, in a compositional manner. While in the automata theory the equivalence is done through language equivalence, in process algebras, the equivalence is generally related with *bisimulation*.

In order to demonstrate examples of process algebra models, let us initially consider the behavior of a printer. The printer actions can be represented by accept, print and shutdown. Using the syntax presented previously, the printer specification can be written as follows:

PRINTER = (accept → print → PRINTER) | (shutdown → STOP)

Figure 3.11 depicts the respective FSM of the printer specification.

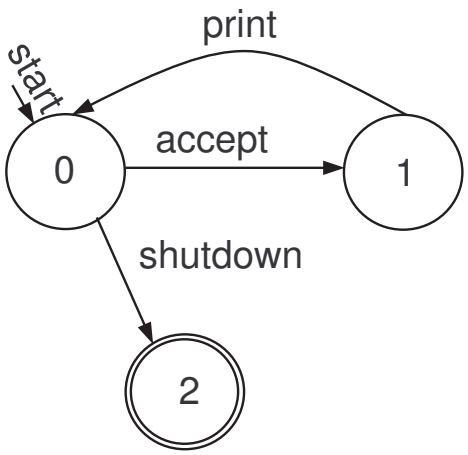


**Figure 3.11** Finite-state machine of printer specification

For demonstrating the utilization of parallel composition, consider the following specification of a client-server application.

CLIENT = request → response → read → CLIENT

SERVER = request → process → response → SERVER

CLIENT_SERVER = CLIENT || SERVER

In this specification, the client and the server processes need to synchronize in the actions *request* and *response*, since they are common in both processes. For a better visualization, Figure 3.12 depicts the FSM of each process and the parallel composition.

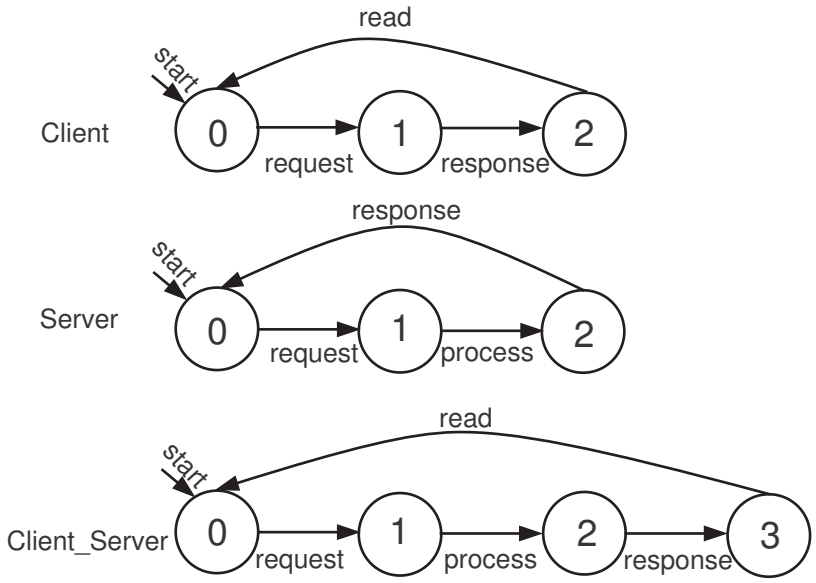


**Figure 3.12** Finite-state machines of client-server processes

It is important to bear in mind that if the parallel composition is applied in processes without common actions, all possible interleavings have to be taken into account. Such situation is described by the following specification and depicted in Figure 3.13.

$\mathsf{ITCH} = \mathsf{scratch} \rightarrow \mathsf{STOP}$

$\mathsf{SERVER} = \mathsf{think} \rightarrow \mathsf{talk} \rightarrow \mathsf{STOP}$

$\mathsf{CONVERSE\_ITCHR} = \mathsf{ITCH} \parallel \mathsf{CONVERSE}$

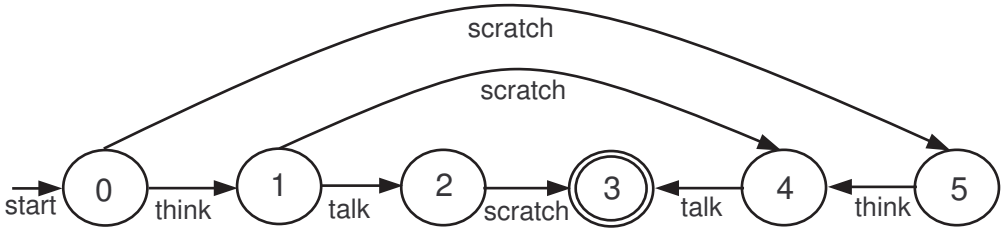


**Figure 3.13** Finite-state machine considering all possible interleavings

### 3.4.5 Timed Models

Whenever dealing with real-time systems, timing constraints are of utmost importance in order to guarantee system responsiveness. In hard real-time systems, if timing constraints are not met, catastrophic issues may occur. Therefore, in addition to model causalities and concurrency, formal models must provide means to capture timing constraints. FSMs, process algebras as well as Petri nets have extensions that deal with time. This section just provides a brief overview how deterministic (not probabilistic) time may be considered in those models. Next section details the model of interest, namely, Petri nets, with their respective timed extensions. For now, the overview is presented below:

- **Timed FSMs or Automata**. In those formal models, the underlying definition is extended with a finite set of clocks, which are synchronized and can be reset due to the transition from one state to another. Clocks are also adopted to guard transitions, in such a way that a transition can not be performed before a specified time. For more details, the reader should refer to [83, 84].
- **Timed Process Algebras**. Over the years, process algebras with time have been developed to allow the modeling of real-time distributed systems. A common approach associates actions with numbers that represent time stamps (i.e., labels indicating the time of execution). For more details, the reader should refer to [85, 86, 87].
- **Petri Nets with Time**. Several approaches have been developed to consider time in Petri net models. Most of them [13, 88] associate transitions with timing constraints, in such a way that a transition is firable if it stays enabled until the time elapsed reaches the respective timing constraint. Next section provides the details.

## 3.5 PETRI NETS

In the sixties, Carl Adams Petri proposed the Petri net theory in his Ph.D. thesis [89] at Technical University of Darmstadt, Germany. In general, Petri net is a term adopted for a whole class of net-based models, which provide a graphical and mathematical modeling tool applicable to many systems, such as distributed systems, parallel systems, and so on [10]. For a better understanding, the following definition describes Place/Transition nets, which are usually called Petri nets [90].

**Definition 3.6** (Petri net)**.** *A Place/Transition net (Petri net) is a bipartite directed graph represented by a tuple $(P, T, F, W, m_0)$, where $P$ (set of places) and $T$ (set of transitions) are non-empty disjoint sets of nodes ($P \cap T = \emptyset$). The edges are represented by $F$, where $F \subseteq A = (P \times T) \cup (T \times P)$. $W : A \to \mathbb{N}$ represents the weight of the edges, such that*

$$W(f) = \begin{cases} x \in \mathbb{N}, & if\ (f \in F) \\ 0, & if\ (f \notin F) \end{cases}$$

*A marking $m_i$ is a function ($m_i : P \to \mathbb{N}$), and $m_0$ is the initial marking.*

**Table 3.1** Interpretation for places and transitions

| input places | transitions | output places |
|---|---|---|
| pre-conditions | events | post-conditions |
| input data | computation step | output data |
| input signals | signal processor | output signals |
| resource granting | tasks | resource releasing |
| conditions | logical clauses | conclusions |
| buffers | processor | buffers |

Considering the previous definition, places represent local states and transitions denote local actions. The set of arcs $F$ represents the relationships between places and transitions, in such a way that arcs connect places to transitions and vice-versa. Function $W$ assigns to each arc a natural number, which may be interpreted as the amount of parallel arcs. A marking function $m_i$ associates to each place a natural number, which represents the number of tokens in the respective place. Thus, the initial marking $m_0$ defines the initial number of tokens in each place. Places and transitions may have several interpretations [10], some of which are shown in Table 3.1. Typical interpretations assume marked places as the truth of a condition or the availability of resources, and transitions are considered events or computational steps.

Taking into account graphical representation, Figure 3.14 depicts the basic components of a Petri Net. Places (a) are represented by circles, transitions (c) are depicted as bars or rectangles, arcs (b) are represented by arrows, and (d) tokens - the marking - are generally represented by filled small circles.

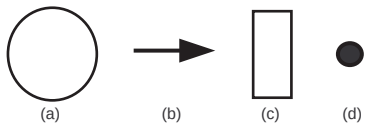


**Figure 3.14** Basic components of a Petri net: (a) place, (b) arc, (c) transition, and (d) token

There are other notations for representing elements of a Petri net. For instance, the set of input transitions of a place $p_i \in P$ (pre-set) may be represented as:

$$\bullet p_i = \{t_j \in T \mid (t_j, p_i) \in F\}$$

and the set of output transitions of a place $p_i \in P$ (post-set) may be denoted as:

$$p_i \bullet = \{t_j \in T \mid (p_i, t_j) \in F\}$$

Likewise, the set of input places of a transition $t_j \in T$ may be described as :

$$\bullet t_j = \{p_i \in P \mid (p_i, t_j) \in F\}$$

and the set of output places of a transition $t_j \in T$ may be represented as:

$$t_j \bullet = \{p_i \in P \mid (t_j, p_i) \in F\}$$

### 3.5.1 Transition Enabling and Firing

In general, the behavior of systems can be described in terms of system states and their changes. In Petri nets, a state is represented by a marking, and the change is performed by transition firing rule.

**Definition 3.7** (Enabled Transitions)**.** *A set of enabled transitions at marking $m_i$ is denoted by: $ET(m_i) = \{t \in T \mid m_i(p_j) \geq W(p_j, t)\}$, $\forall p_j \in P$.*

**Definition 3.8** (Transition Firing Rule)**.** *The firing of transition $t \in ET(m_i)$ generates a new marking (state) $m_j$, which is obtained by applying the following formula: $m_j(p) = m_i(p) - W(p,t) + W(t,p)$, $\forall p \in P$.*

Consider the following net $\mathcal{N} = (P = \{p_0, p_1, p_2\}, T = \{t_0, t_1\}, F = \{(p_0, t_0), (p_1, t_0), (t_0, p_2), (p_2, t_1), (t_1, p_0), (t_1, p_1)\}, W = \{(p_0, t_0, 2), (p_1, t_0, 1), (t_0, p_2, 1), (p_2, t_1, 1), (t_1, p_0, 2), (t_1, p_1, 1)\}, m_0 = \{(p_0, 2), (p_1, 1), (p_2, 0)\})$, which is depicted in Figure 3.15(a). In order to demonstrate the transition firing rule, assume the firing of transition $t_0$ (Figure 3.15(b)). A new marking $m_1 = \{(p_0, 0), (p_1, 0), (p_2, 1)\}$ is reached, such that, $m_1(p_0) = 2 - 2 + 0$, $m_1(p_1) = 1 - 1 + 0$, and $m_1(p_2) = 0 - 0 + 1$.

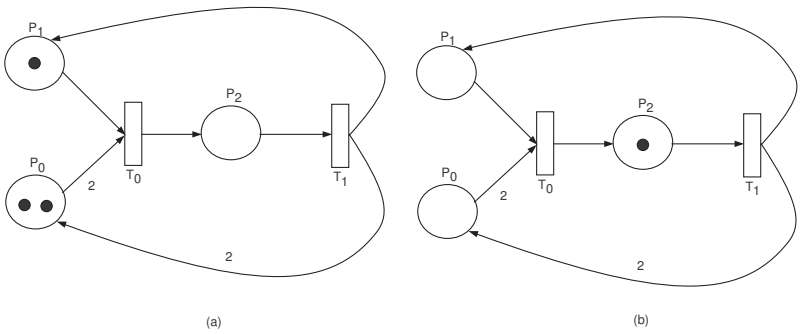


**Figure 3.15** Petri net example

### 3.5.2 Reachability graph

Generally, a labeled directed graph is adopted to represent all possible states (markings) that a Petri net can reach. Such graph is named Reachability Graph.

**Definition 3.9** (Reachability Graph)**.** *A reachability graph is defined by a tuple (V,E), where $V$ is the set of vertices represented by the reachable markings, and $E \subseteq (V \times V)$ is the set of labeled edges.*

As an example, consider $M = \{m_0 = \{(p_0, 2), (p_1, 1), (p_2, 0)\}, m_1 = \{(p_0, 0), (p_1, 0), (p_2, 1)\}\}$ the set of reachable markings of the net depicted in Figure 3.15(a). The respective reachability graph is depicted in Figure 3.16.

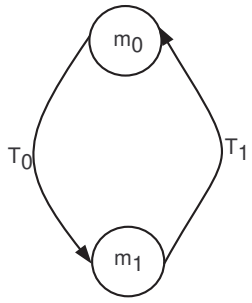


**Figure 3.16** Reachability graph

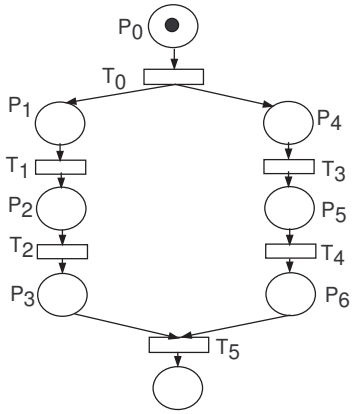


**Figure 3.17** Parallel processes

### 3.5.3 Examples

This section presents some classical problems and their respective Petri net models.

**Parallel Processes** Figure 3.17 depicts a net composed of two independent processes. As can been seen, both processes become eligible to execute their activities after firing transition $t_0$. When both processes end their independent activities (transitions $t_1, t_2, t_3, t_4$), they are synchronized (transition $t_5$).

**Mutual exclusion** Sometimes, parallel processes need to cooperate in order to achieve a joint solution. Therefore, shared resources may be required and accessed in a mutual exclusive manner for avoiding undesirable results.

Figure 3.18 shows an example of two processes sharing a common resource in a mutual exclusive manner. The resource is represented by a token in place $p_8$, and it is accessed by only one process at a time.

**Dining Philosophers** The dining philosophers is a classical problem that was proposed by Dijsktra in [91]. Briefly, three philosophers are arranged in a ring with one

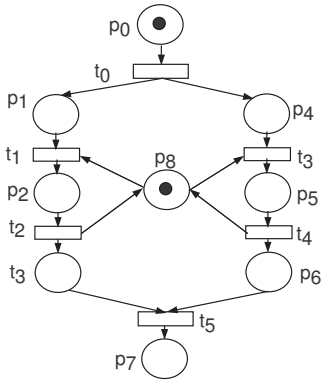


**Figure 3.18** Mutual exclusion

fork (resource) between each pair of neighbors, and for eating, a philosopher must have exclusive access to both of its adjacent forks. If all philosophers take at the same the right fork and wait the left fork to be freed, the system enters in a *deadlock* state.

Figure 3.19 [92] presents a solution for this problem. The resources (forks) are represented by tokens in places $\text{fork}_1$, $\text{fork}_2$, and $\text{fork}_3$. The state of each philosopher is represented by the places eating ($\text{pc}_i$), hungry ($\text{pcf}_i$), and thinking ($\text{pp}_i$). The event start-to-eat is represented by transition $tcc_i$, and is-hungry as well as start-to-think are represented by $ttf_i$ and $tcp_i$, respectively.

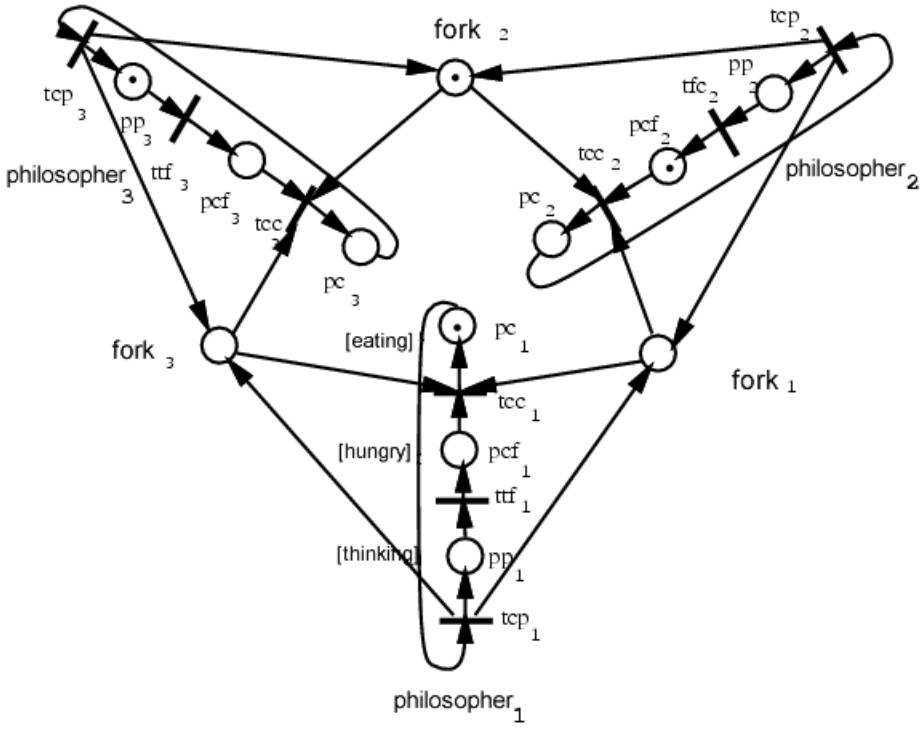


**Figure 3.19** Dining Philosophers

### 3.5.4 Petri nets Properties

Petri nets are not just a modeling tool for describing systems. A major strength of Petri nets is their support for analysis of many interesting properties. In general, two types of properties may be found in Petri net models: behavioral properties (those which depend on the marking) and structural properties (those which do not depend on the marking).

### 3.5.5 Behavioral Properties

Behavioral properties are also known as quantitative properties. This section, based on [10], describes the behavioral properties of interest.

**Reachability** Reachability property indicates the possibility of reaching a given marking through the finite firing of transitions from the initial marking. A marking $m_i$ is said to be reachable from marking $m_0$ (initial marking), if there exists a sequence of firings that transforms $m_0$ to $m_i$. A firing (or occurrence) sequence is denoted by $\sigma = t_1 t_2 \cdots t_i$. In this case, $m_i$ is reachable from $m_0$ by $\sigma$.

**Boundedness** A Petri net is considered *bounded* or *k-bounded*, if the number of tokens in each place does not exceed a finite number $k$ for any reachable marking. A Petri net is said to be *safe*, if in each place the number of tokens does not surpass 1 (*1-bounded*).

**Liveness** A Petri net is denominated as *live*, if, no matter what marking has been reached from $m_0$, it is possible to fire any transition of the net by some firing sequence. In other words, liveness is related to the concept of deadlock-free, a desirable property for many systems, such as operating systems.

Although liveness is an ideal property for many real systems, it is impractical to verify this property for some systems due to their size. Thus, different levels of liveness are adopted [10], as described below:

- A transition $t$ is said to be dead (L0-Live), if it can never be fired in any firing sequence.
- L1-Live (potentially firable) is denoted to transitions that can be fired at least once in some firing sequence.
- A transition $t$ is considered in the level L2-Live, if, given any positive integer $k$, $t$ can be fired at least $k$ times in some firing sequence.
- L3-Live is adopted to transitions that can infinitely fire, considering the existence of an infinite-length firing sequence.
- A transition is said to be L4-Live (or simply live), if it is L1-Live for every marking $m$ reachable from $m_0$.

A Petri net is said to be at level $i$, if every transition is at the same level $i$. It is worth noting that transition live at level 4, is also live at levels 3, 2, 1. Obviously, this principle is not applied to level 0.

**Reversibility and Home State** A Petri net is said to be *reversible*, if, for each marking $m$, $m_0$ is reachable from $m$. Thus, in a reversible net one can always get back to the initial marking. A marking $m_k$ is said to be a home state, if, for each marking $m$, $m_k$ is reachable from $m$.

**Coverability** Coverability is strongly related to liveness and reachability. A marking $m$ in a Petri net is said to be *coverable*, if there exists a reachable marking $m_i$, such that $m_i(p) \geq m(p)$, for each place $p$ in the net. In order to see the relation with liveness, consider $m$ the minimum marking needed to enable a transition $t$. $t$ is only firable (L1-Live) if $m$ is coverable.

### 3.5.6 Structural Properties

This section aims at describing structural properties, which provide characteristics independent of the marking. It is worth stating that structural properties are also known as qualitative properties.

The structural properties of interest are presented below. For more information, the reader should refer to [10].

**Structural Liveness** A Petri net is considered *structurally live*, if there is a live initial marking $m_0$.

**Structural Boundedness** A net is *structurally bounded*, if it is bounded for any finite initial marking $m_0$.

**Conservativeness** A Petri net is said to be *strictly conservative* whether any transition firing does not change the number of tokens. Nevertheless, there are nets that are not classified as strictly conservative, but they can be converted into strictly conservative nets. Such nets are said to be conservative.

**Repetitiveness** A net is classified as repetitive, if there is marking $m_0$ and a firing sequence from this marking, such that every transitions fires infinitely. If only some of these transitions are fired infinitely often in the sequence, the net is called *partially repetitive*.

**Consistency** A Petri net $\mathcal{N}$ has the *consistency* property, if there exists a marking $m_0$ as well as a firing sequence from $m_0$ back to $m_0$, such that every transition fires at least once in the firing sequence. Consistency can also be partial, in the sense that some transition occurs in the firing sequence.

### 3.5.7 Incidence Matrix

Assume a Petri net with $m$ places and $n$ transitions. Its incidence matrix $A = [a_{ij}]$ is an $m \times n$ matrix of integers such that $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$. $a_{ij}^{+} = W(t_j, p_i)$ is the weight of the arc from transition $t_j$ to its output place $p_i$ and $a_{ij}^{-} = W(p_i, t_j)$ is the weight of the arc to transition $t_j$ from its input place $p_i$. As an example, the incidence matrix for the model depicted in Figure 3.15 is presented below:

$$A = \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \end{array} \begin{array}{c} \begin{array}{cc} t_0 & t_1 \end{array} \\ \begin{bmatrix} -2 & 2 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \end{array}$$

Although the incidence matrix allows the structural representation of Petri net models, there are situations in which the matrix does not properly represent the structure. For instance, this situation may occur when a Petri net model contains a *self-loop*, which occurs when a place is pre-condition and post-condition of a transition. Nevertheless, *self-loops* can be removed utilizing *dummy pairs* [10].

The incidence matrix is adopted in some analysis methods (e.g., fundamental equation [10]), which are very useful to indicate properties in Petri net models. In this work, incidence matrixes are utilized in the context of invariants.

### 3.5.8 Invariants

In Petri nets, invariants are related to the conservative and repetitive stationary components, which are denoted by place invariants and transition invariants, respectively. Both invariants are defined below, but the focus is on places invariants.

**Definition 3.10** (Place Invariant - P-semiflow or P-Invariant)**.** *Assume a Petri net $\mathcal{N}$ with $m$ places and its incidence matrix $A$. A vector of non-negative integers* $\mathcal{I}_p^{\mathrm{T}} = \begin{bmatrix} p_0 & p_1 & \dots & p_m \\ x_0 & x_1 & \dots & x_m \end{bmatrix}^T$ *is a P-semiflow or place invariant (P-invariant), if and only if $\mathcal{I}_p{}^T \times A = 0$. A value $x_m$ is the weight associated with place $p_m$.*

**Definition 3.11** (Transition Invariant - T-semiflow or T-invariant)**.** *Assume a Petri net $\mathcal{N}$ with $n$ transitions and its incidence matrix $A$. A vector of non-negative integers* $\mathcal{I}_t = \begin{bmatrix} t_0 & t_1 & \dots & t_n \\ y_0 & y_1 & \dots & y_n \end{bmatrix}^T$ *is a T-semiflow or transition invariant (T-invariant), if and only if $A \times \mathcal{I}_t = 0$. A value $y_n$ is the weight associated with transition $t_n$.*

The support $P(\mathcal{I}_p)$ of an invariant $\mathcal{I}_p$ [93] is the set of places (or transitions for $T(\mathcal{I}_t)$) whose values in the vector are not zero. The reader should observe that $P(\mathcal{I}_p) \subseteq P$ (and $T(\mathcal{I}_t) \subseteq T$). Throughout this work, only non-negative vectors are considered, namely, P-semiflow or T-semiflow. In other words, all weights are equal to or greater than 0. Besides, the terms P-semiflow (or T-semiflow) and P-invariant (or T-invariant) are used interchangeably throughout this thesis.

If there is a P-invariant (or T-invariant) such that all weights are different from 0 (or equivalently $|P(\mathcal{I}_p)| = |P|$), a Petri net is said to be *covered* with P-invariants (or

T-invariants). Besides, an invariant $\mathcal{I}_{p_1}$ is a *minimal support invariant*, if there is no invariant $\mathcal{I}_{p_2}$, in which $P(\mathcal{I}_{p_2}) \subset P(\mathcal{I}_{p_1})$ (or $T(\mathcal{I}_{t_2}) \subset T(\mathcal{I}_{t_1})$). When there is no other invariant $\mathcal{I}_2$ such that $\mathcal{I}_1 \geq \mathcal{I}_2$, $\mathcal{I}_1$ is said to be *minimal* ($\geq$ means for every $i$, $a(i) \geq b(i)$ and at least one $i$ such that $a(i) > b(i)$). In this work, a minimal invariant with minimal support is denominated *basic invariant*. Moreover, new invariants can be generated from the combination of others invariants, for instance: (i) $\mathcal{I}_1$'$= \alpha\mathcal{I}_1 (\alpha \in \mathbb{N}^*)$; or (ii) $\mathcal{I}_3 = \alpha\mathcal{I}_1 + \beta\mathcal{I}_2 (\alpha, \beta \in \mathbb{N}^*)$.

For a better understanding, the basic P-invariants of the Petri net depicted in Figure 3.15 are:

- $\mathcal{I}_{p_1} = \overset{\begin{matrix} p_0 & p_1 & p_2 \end{matrix}}{\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}^T}$;
- $\mathcal{I}_{p_2} = \overset{\begin{matrix} p_0 & p_1 & p_2 \end{matrix}}{\begin{bmatrix} 1 & 0 & 2 \end{bmatrix}^T}$.

$P(\mathcal{I}_{p_1}) = \{p_1, p_2\}$ and $P(\mathcal{I}_{p_2}) = \{p_0, p_2\}$ are the respective supports. Additionally, new invariants can be constructed using $\mathcal{I}_{p_1}$ and $\mathcal{I}_{p_2}$, for instance:

- $\mathcal{I}_{p_3} = \mathcal{I}_{p_1} + \mathcal{I}_{p_2} = \overset{\begin{matrix} p_0 & p_1 & p_2 \end{matrix}}{\begin{bmatrix} 1 & 1 & 3 \end{bmatrix}^T}$;
- $\mathcal{I}_{p_4} = 5\mathcal{I}_{p_1} = \overset{\begin{matrix} p_0 & p_1 & p_2 \end{matrix}}{\begin{bmatrix} 0 & 5 & 5 \end{bmatrix}^T}$;
- As well as using $\mathcal{I}_{p_3}$, $\mathcal{I}_{p_5} = 3\mathcal{I}_{p_1} + 5\mathcal{I}_{p_3} = \overset{\begin{matrix} p_0 & p_1 & p_2 \end{matrix}}{\begin{bmatrix} 5 & 8 & 18 \end{bmatrix}^T}$.

It is important to state that one prominent application of invariants is in the analysis of structural properties [10, 93]. A Petri net $\mathcal{N}$ is *conservative* if only if there is a P-invariant $\mathcal{I}_p$, such that $\mathcal{I}_p > 0$ (all elements are positive integers) and ${\mathcal{I}_p}^T \times A = 0$. If there is a vector $\mathcal{I}_p > 0$ and ${\mathcal{I}_p}^T \times A \leq 0$, $\mathcal{N}$ is *structurally bounded*. As an example, the net in Figure 3.15 is structurally bounded and conservative, since $\mathcal{I}_{p_3} \times A = 0$ ($A$ is the respective incidence matrix). Regarding T-invariants, a net is *consistent* if exists a T-invariant $\mathcal{I}_t > 0$ and $A \times \mathcal{I}_t = 0$. If there is a vector $\mathcal{I}_t > 0$ and $A \times \mathcal{I}_t \geq 0$, the net is *structurally repetitive*.

**Juxtaposition of Invariants** The composition of Petri net models from basic building blocks is an efficient approach to tackle the issues involved in the representation of complex and large-scale systems. For instance, tools may be adopted to automate the modeling process, which may assure (by construction) that the generated models possess some behavioral and structural properties.

As presented, invariants are a powerful analysis method that can reason about the structural properties in Petri net models. The following definition presents a technique for obtaining the P-invariants of a Petri net composed by merging places of basic models [94, 95, 92, 96]. Although attention is devoted to P-invariants, equivalent results may be obtained with T-invariants [94, 95, 92].

**Definition 3.12** (Juxtaposition of P-invariants - $J$)**.** *Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two Petri nets, as well as $P_1 = P^1_{not} \cup P^1_{shared}$ and $P_2 = P^2_{not} \cup P^2_{shared}$ be the respective set of places, such that $P^1_{not} = \{p^1_0, ..., p^1_n\}$, $P^1_{shared} = \{p_{k_0}, ..., p_{k_q}\}$, $P^2_{not} = \{p^2_0, ..., p^2_m\}$, $P^2_{shared} = \{p_{k_0}, ..., p_{k_q}\}$, $P^1_{not} \cap P^2_{not} = \emptyset$, and $P^1_{shared} = P^2_{shared}$. Additionally, consider that*

$$\mathcal{I}_{p(1)} = \begin{bmatrix} p^1_0 & \cdots & p^1_n & p_{k_0} & \cdots & p_{k_q} \\ x^1_0 & \cdots & x^1_n & x^1_{k_0} & \cdots & x^1_{k_q} \end{bmatrix}^T$$

*is a P-invariant of net $\mathcal{N}_1$ and*

$$\mathcal{I}_{p(2)} = \begin{bmatrix} p_{k_0} & \cdots & p_{k_q} & p^2_0 & \cdots & p^2_m \\ x^2_{k_0} & \cdots & x^2_{k_q} & x^2_0 & \cdots & x^2_m \end{bmatrix}^T$$

*is a P-invariant of net $\mathcal{N}_2$. Finally, assume a Petri net $\mathcal{N}_3$ generated by merging places of subnets $\mathcal{N}_1$ and $\mathcal{N}_2$, more specifically, $P^1_{shared}$ and $P^2_{shared}$. A P-invariant $\mathcal{I}_{p(3)}$ of net $\mathcal{N}_3$ may be obtained from $\mathcal{I}_{p(1)}$ and $\mathcal{I}_{p(2)}$ via juxtaposition, $\mathcal{I}_{p3} = J(\mathcal{I}_{p(1)}, \mathcal{I}_{p(2)})$, if the following condition holds: $\forall p \in P_1 \cap P_2, \mathcal{I}_{p(1)}(p) = \mathcal{I}_{p(2)}(p)$. Thus, assuming $x^1_{k_0} = x^2_{k_0} = x_{k_0}$, ... ,$x^1_{k_q} = x^2_{k_q} = x_{k_q}$,*

$$\mathcal{I}_{p_3} = \begin{bmatrix} p^1_0 & \cdots & p^1_n & p_{k_0} & \cdots & p_{k_q} & p^2_0 & \cdots & p^2_m \\ x^1_0 & \cdots & x^1_n & x_{k_0} & \cdots & x_{k_q} & x^2_0 & \cdots & x^2_m \end{bmatrix}^T.$$

Note that, for the condition stated in Definition 3.12 be satisfied, there are cases the invariants need to be multiplied by non-negative integers. For instance, assume a net $\mathcal{N}_3$ composed by merging places of subnets $\mathcal{N}_1$ and $\mathcal{N}_2$. Additionally, consider the P-invariant $\mathcal{I}_{p(1)} = [x_1 \ x_{k_1}]$ of net $\mathcal{N}_1$, in which $x_{k_1}$ represents the weights associated to the merged places and $x_1$ the weights associated to the untouched ones. Similarly, $\mathcal{I}_{p(2)} = [x_{k_2} \ x_2]$ represents a P-invariant of net $\mathcal{N}_2$, in which $x_{k_2}$ represents the weights associated to the merged places. If $x_{k_1} \neq x_{k_2}$, it is necessary to find $a, b \in \mathbb{N}$ such that $a.x_{k_1} = b.x_{k_2}$, which will result in $\mathcal{I}_{p(3)} = J(a.\mathcal{I}_{p(1)}, b.\mathcal{I}_{p(2)})$. If is not possible to find $a, b \in \mathbb{N}$ that satisfy the condition, the juxtaposition of these P-invariants can not be performed.

To demonstrate the application of juxtaposition technique, assume a net $\mathcal{N}_c$ composed by merging a common place of subnets $\mathcal{N}_a$ and $\mathcal{N}_b$ ($P_a \cap P_b = \{p_{shared}\}$). Additionally, consider the following basic P-invariants for each subnet:

- $\mathcal{N}_a$: $\mathcal{I}_{p(a)(1)} = \begin{bmatrix} p_0 & p_1 & p_{shared} \\ 1 & 1 & 0 \end{bmatrix}^T$ and $\mathcal{I}_{p(a)(2)} = \begin{bmatrix} p_0 & p_1 & p_{shared} \\ 1 & 0 & 1 \end{bmatrix}^T$;
- $\mathcal{N}_b$: $\mathcal{I}_{p(b)(1)} = \begin{bmatrix} p_{shared} & p_2 & p_3 & p_4 \\ 5 & 1 & 1 & 0 \end{bmatrix}^T$ and $\mathcal{I}_{p(b)(2)} = \begin{bmatrix} p_{shared} & p_2 & p_3 & p_4 \\ 5 & 1 & 0 & 1 \end{bmatrix}^T$.

The following lines demonstrate the P-invariants obtained through juxtaposition for net $\mathcal{N}_c$ :

- $\mathcal{I}_{p(c)(1)} = J(\mathcal{I}_{p(a)(1)}, 0.\mathcal{I}_{p(b)(1)}) = \begin{bmatrix} p_0 & p_1 & p_{shared} & p_2 & p_3 & p_4 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$;

- $\mathcal{I}_{p(c)(2)} = J(\mathcal{I}_{p(a)(1)}, 0.\mathcal{I}_{p(b)(2)}) = \overset{p_0}{\big[}\ 1 \quad \overset{p_1}{1} \quad \overset{p_{shared}}{0} \quad \overset{p_2}{0} \quad \overset{p_3}{0} \quad \overset{p_4}{0}\ \big]^T$;
- $\mathcal{I}_{p(c)(3)} = J(5.\mathcal{I}_{p(a)(2)}, \mathcal{I}_{p(b)(1)}) = \overset{p_0}{\big[}\ 5 \quad \overset{p_1}{0} \quad \overset{p_{shared}}{5} \quad \overset{p_2}{1} \quad \overset{p_3}{1} \quad \overset{p_4}{0}\ \big]^T$;
- $\mathcal{I}_{p(c)(1)} = J(5.\mathcal{I}_{p(a)(2)}, \mathcal{I}_{p(b)(2)}) = \overset{p_0}{\big[}\ 5 \quad \overset{p_1}{0} \quad \overset{p_{shared}}{5} \quad \overset{p_2}{1} \quad \overset{p_3}{0} \quad \overset{p_4}{1}\ \big]^T$.

A general P-invariant for $\mathcal{N}_c$ (as well as for $\mathcal{N}_a$ and $\mathcal{N}_b$) can be obtained as follows:

- $\mathcal{I}_{p(a)} = \alpha.\mathcal{I}_{p(a)(1)} + \beta.\mathcal{I}_{p(a)(2)} = \overset{p_0}{\big[}\ \alpha+\beta \quad \overset{p_1}{\alpha} \quad \overset{p_{shared}}{\beta}\ \big]^T$, where $\alpha, \beta \in \mathbb{N}$;
- $\mathcal{I}_{p(b)} = \gamma.\mathcal{I}_{p(b)(1)} + \delta.\mathcal{I}_{p(b)(2)} = \overset{p_{shared}}{\big[}\ 5\gamma+5\delta \quad \overset{p_2}{\gamma+\delta} \quad \overset{p_3}{\gamma} \quad \overset{p_4}{\delta}\ \big]^T$, where $\gamma, \delta \in \mathbb{N}$;
- Assuming $\beta = 5\gamma+5\delta, \mathcal{I}_{p(c)} = J(\mathcal{I}_{p(a)}, \mathcal{I}_{p(b)}) = \overset{p_0}{\big[}\ \alpha+5\gamma+5\delta \quad \overset{p_1}{\alpha} \quad \overset{p_{shared}}{5\gamma+5\delta} \quad \overset{p_2}{\gamma+\delta} \quad \overset{p_3}{\gamma} \quad \overset{p_4}{\delta}\ \big]^T$.

Since $\exists \alpha, \gamma, \delta \in \mathbb{N}^*, {\mathcal{I}_{p(c)}}^T \times A_c = 0$, in which $A_c$ is the incidence matrix of net $\mathcal{N}_c$ , $\mathcal{N}_c$ is conservative as well as structurally bounded, in other words, for any initial marking, the state space size is finite.

### 3.5.9 Time Petri Nets

Several Petri net extensions have been proposed in order to consider timing information. This section describes the extension of interest, namely, time Petri nets, which is a feasible model for describing real-time systems with preemption mechanisms.

Time Petri nets were initially proposed by Melin and Faber in [13]. Nevertheless, for the sake of this work, a definition based on [16] is adopted, which assumes a discrete time domain.

**Definition 3.13** (Time Petri net)**.** *A time Petri net is defined by a tuple ($\mathcal{N}$, $I$), where $\mathcal{N}$ is the underlying Petri net, and $I : T \to \mathbb{N} \times \mathbb{N}$ represents the timing constraints, such that $I(t) = (EFT(t), LFT(t))\ \forall t \in T$, $EFT(t) \leq LFT(t)$. $EFT(t)$ is the Earliest Firing Time, and $LFT(t)$ is the Latest Firing Time.*

In the previous definition, each transition $t$ has timing constraints represented by $I(t) = (EFT(t), LFT(t))$. For a better understanding, an enabled transition $t$ (see Definition 3.7) can not fire before $EFT(t)$, and must fire before or at its $LFT(t)$. The time elapsed, since the respective transition enabling, is represented by a clock vector $c \in (\mathbb{N} \cup \{\#\})^{|T|}$, where # represents the undefined value for disabled transitions. Additionally, it is worth stating that the situation related to $LFT$, in the sense that a transition must fire before or at its LFT, is denominated *strong firing mode* [97]. Another alternative is *weak firing mode*, which does not force any enabled transition to fire. Nevertheless, *strong firing mode* is adopted in this work.

It is important to separate the definitions of *enabled* transitions from *firable* transitions. Without loss of generality, enabled transitions are only related to the marking (Definition 3.7), and firable transitions take into account the marking and their respective clock values (the time elapsed of each enabled transition). The following paragraphs lay the groundwork for firable transitions. Firstly, the difference between static and dynamic firing intervals associated with transitions is required. The dynamic firing interval of transition $t$, $I_D(t) = (DLB(t), DUB(t))$, is dynamically modified whenever the respective clock variable $c(t)$ is incremented, and $t$ does not fire. $DLB(t)$ is the Dynamic Lower Bound, and $DUB(t)$ is the Dynamic Upper Bound. The dynamic firing interval is computed in the following way: $I_D(t) = (DLB(t), DUB(t))$, where $DLB(t) = max(0, EFT(t) - c(t))$, $DUB(t) = LFT(t) - c(t)$. Whenever $DLB(t) = 0$, $t$ can fire, and, when $DUB(t) = 0$, $t$ must fire, since *strong firing mode* is assumed. Initially, at the moment transition $t$ becomes enabled, $I(t) = I_D(t)$.

**Definition 3.14** (States)**.** *Let $\mathcal{N}_\mathcal{T}$ be a time Petri net, $M \subseteq P \times \mathbb{N}$ be the set of reachable markings of $\mathcal{N}_\mathcal{T}$, and $C \subseteq (\mathbb{N} \cup \{\#\})^{|T|}$ be the set of clock vectors. The set of states $S$ of $\mathcal{N}_\mathcal{T}$ is given by $S \subseteq (M \times C)$, that is, a state is defined by a marking, and the respective clock vector.*

Different from Petri nets (Definition 3.6), the state of time Petri nets is composed of a marking and the clocks of each enabled transition in that marking.

**Definition 3.15** (Firable Transitions)**.** *Let $\mathcal{N}_\mathcal{T}$ be a time Petri net, the set of firable transitions at state $s \in S$ is defined by: $FT(s) = \{t_i \in ET(m) |\ \ DLB(t_i) \leq \min(DUB(t_k)), \forall t_k \in ET(m)\}$.*

This definition enforces the *strong firing mode*. Besides, $FT \subseteq ET \subseteq T$.

**Definition 3.16** (Firing Domain)**.** *The firing domain for a transition $t$ at state $s$, is defined by the interval: $FD_s(t) = [DLB(t), \min(DUB(t_k))], \forall t_k \in ET(m)$.*

A transition $t$ at state $s$ is only fireable in the interval denoted by $FD_s(t)$.

**Definition 3.17** (Reachable States)**.** *Let $\mathcal{N}_\mathcal{T}$ a time Petri net, and $s_i = (m_i, c_i)$ a reachable state. $s_j = \texttt{fire}(s_i, (t, \theta))$ denotes that firing a transition $t \in FT(s_i)$ at time $\theta \in FD_{s_i}(t)$ from the state $s_i$, the reached state $s_j = (m_j, c_j)$ is obtained from:*

- $\forall p \in P,\ m_j(p) = m_i(p) - W(p,t) + W(t,p)$, *as usual in Petri nets;*
- $\forall t_l \notin ET(m_j), c_j(t_l) = \#$;
- $\forall t_k \in ET(m_j),\ c_j(t_k) = \begin{cases} 0, & if(t_k = t) \\ 0, & if(t_k \in ET(m_j) - ET(m_i)) \\ c_i(t_k) + \theta, & else \end{cases}$

When changing from a state to another, the marking as well as the clocks need to be updated. Firstly, the marking is changed using the transition firing rule (Definition 3.8), and, next, the value # is assigned for disable transitions in the clock vector. After that, the clocks for the enabled transitions are updated using *enabling memory* mechanism, in the sense that if a transition was not fired in the previous state and continues enabled in the new reached state, the respective clock value is incremented. However, when a transition $t$ is fired and its enabling degree is larger than one, *single-server semantics* is adopted [98]. In other words, the clock value of transition $t$ is set equal to zero when $t$ is first enabled, and the clock is again reset to zero after firing $t$, if $t$ is still enabled in the new marking. Additionally, the above definition does not consider the state change only due to time elapsing, but also it takes into account a transition firing. Indeed, time elapsing states increase the state space size, and they are not of interest for most real-time systems.

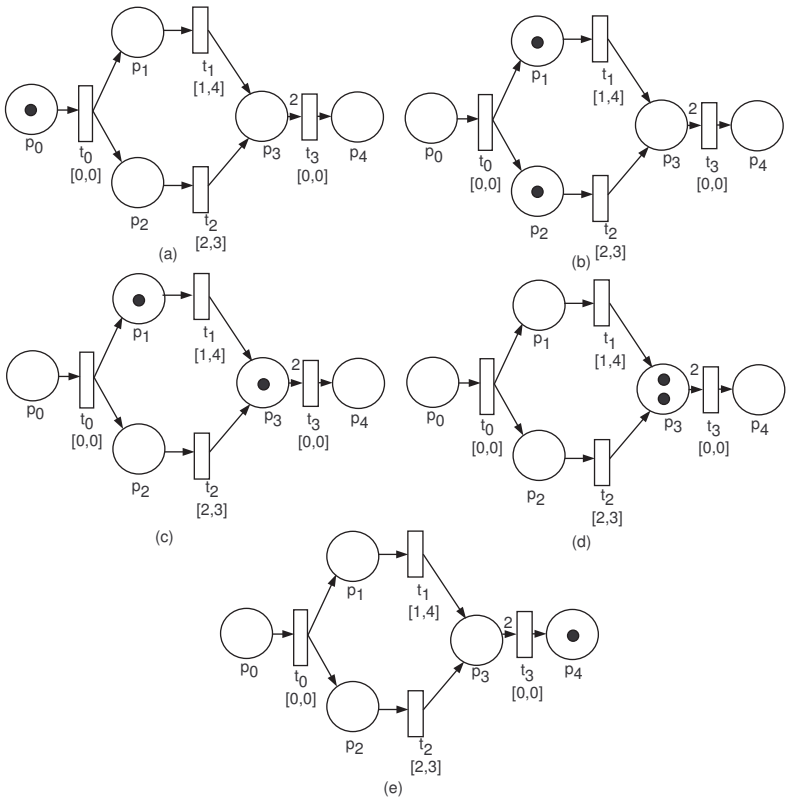


**Figure 3.20** Time Petri net example

For a better comprehension of concepts related to time Petri nets, consider the net $\mathcal{N}_{\mathcal{T}} = (P = \{p_0, p_1, p_2, p_3, p_4\}, T = \{t_0, t_1, t_2, t_3\}, F = \{(p_0, t_0), (t_0, p_1), (t_0, p_2), (p_1, t_1), (p_2, t_2), (t_1, p_3), (t_2, p_3), (p_3, t_3), (t_3, p_4), W = \{(p_0, t_0, 1), (t_0, p_1, 1), (t_0, p_2, 1), (p_1, t_1, 1), (p_2, t_2, 1), (t_1, p_3, 1), (t_2, p_3, 1), (p_3, t_3, 2), (t_3, p_4, 1)\}, m_0 = \{(p0, 1), (p1, 0), (p2, 0), (p3, 0), (p4, 0)\}, I = \{(t_0, 0, 0),(t_1, 1, 4), (t_2, 2, 3), (t_3, 0, 0)\})$, which is depicted in Figure 3.20(a). The initial state is represented by $s_0 = (\{(p0, 1), (p1, 0), (p2, 0), (p3, 0), (p4, 0)\},$

$\overset{t_0}{[\ 0}\ \overset{t_1}{\#}\ \overset{t_2}{\#}\ \overset{t_3}{\#}\ ]^T$), which defines that transition $t_0$ is the only enabled transition ($ET(m_0)$=$\{t_0\}$) due to the marking. The set of firable transitions is $FT(s_0) = \{t_0\}$. Since $I_D(t_0) = [0, 0]$, $t_0$ must fire because of strong firing mode. Figure 3.20(b) shows the new state $s_1 = (\{(p0, 0), (p1, 1), (p2, 1), (p3, 0), (p4, 0)\}, \overset{t_0}{[\ \#}\ \overset{t_1}{0}\ \overset{t_2}{0}\ \overset{t_3}{\#}\ ]^T)$ reached due to the firing of $t_0$ at $\theta = 0$. The dynamic firing intervals of each enabled transitions are $I_D(t_1) = [1, 4]$ and $I_D(t_2) = [2, 3]$, which imply that both transitions are firable ($FT(m_1) = \{t_1, t_2\}$). This can be easily seen, because the minimum dynamic upper bound of all enabled transitions is $DUB(t_2) = 3$, and $DLB(t_1) \leq DUB(t_2)$, $DLB(t_2) \leq DUB(t_2)$. Besides, the firing domains of each transition are $FD_{s_1}(t_0) = [1, 3]$ and $FD_{s_1}(t_1) = [2, 3]$. Let us consider two time units elapsed ($\theta = 2$) without any transition firing, the new values of the dynamic intervals are $I_D(t_1) = [min(0, 1-2), 4-2] = [0, 2]$ and $I_D(t_2) = [min(0, 2-2), 3-2] = [0, 1]$. At this moment, assume the firing of transition $t_2$, which leads to the state $s_7 = (\{(p0, 0), (p1, 1), (p2, 0), (p3, 1), (p4, 0)\}, \overset{t_0}{[\ \#}\ \overset{t_1}{2}\ \overset{t_2}{\#}\ \overset{t_3}{\#}\ ]^T)$ (Figure 3.20(c)). Although $t_1$ had a lower $DLB$, any firable transition can fire, inasmuch as the respective firing domain is respected. After that, considering the firing of transition $t_1$ at $\theta = 0$ in $s_7$, state $s_3 = (\{(p0, 0), (p1, 0), (p2, 0), (p3, 2), (p4, 0)\}, \overset{t_0}{[\ \#}\ \overset{t_1}{\#}\ \overset{t_2}{\#}\ \overset{t_3}{0}\ ]^T)$ is obtained (Figure 3.20(d)). Lastly, state $s_4 = (\{(p0, 0), (p1, 0), (p2, 0), (p3, 0), (p4, 1)\}$, $\overset{t_0}{[\ \#}\ \overset{t_1}{\#}\ \overset{t_2}{\#}\ \overset{t_3}{\#}\ ]^T)$ (Figure 3.20(e)) is reached due to the firing of transition $t_3$ in $s_3$ at $\theta = 0$.

It is important to bear in mind that other states may be reached by firing transitions at other time instants in the respective firing domains. For instance, in state $s_1$, $t_1$ can be fired at $\theta = 1$ instead of firing $t_2$ at $\theta = 2$, reaching state $s_2$. For a better visualization, Figure 3.21 depicts the reachability graph for the net in question, generated using the Integrated Net Analyzer (INA) [99]. In this graph, the nodes (states) are labeled with the marking and the clock vector, whereas the edges are labeled with the fired transition and the time elapsed ($\theta$) in the previous state.

## 3.6 SUMMARY

This chapter presented concepts related to the proposed software synthesis method, ranging from the definition of real-time systems to the Petri net formalism. Initially, real-time systems were presented focusing on a specific class, namely, hard real-time systems. Afterward, DPM (Dynamic Power Management) and DVS (Dynamic Voltage Scaling) technologies were described, and an example was provided to show the feasibility of DVS technology in hard real-time systems. Next, software synthesis was conceptualized in the context of embedded systems. After that, attention was devoted to discrete-event models, giving particular focus to Petri nets. Petri nets are a family of formalisms very suitable for modeling real-time systems, and several techniques (as well as tools) are available for analysis and verification of properties in Petri net models.

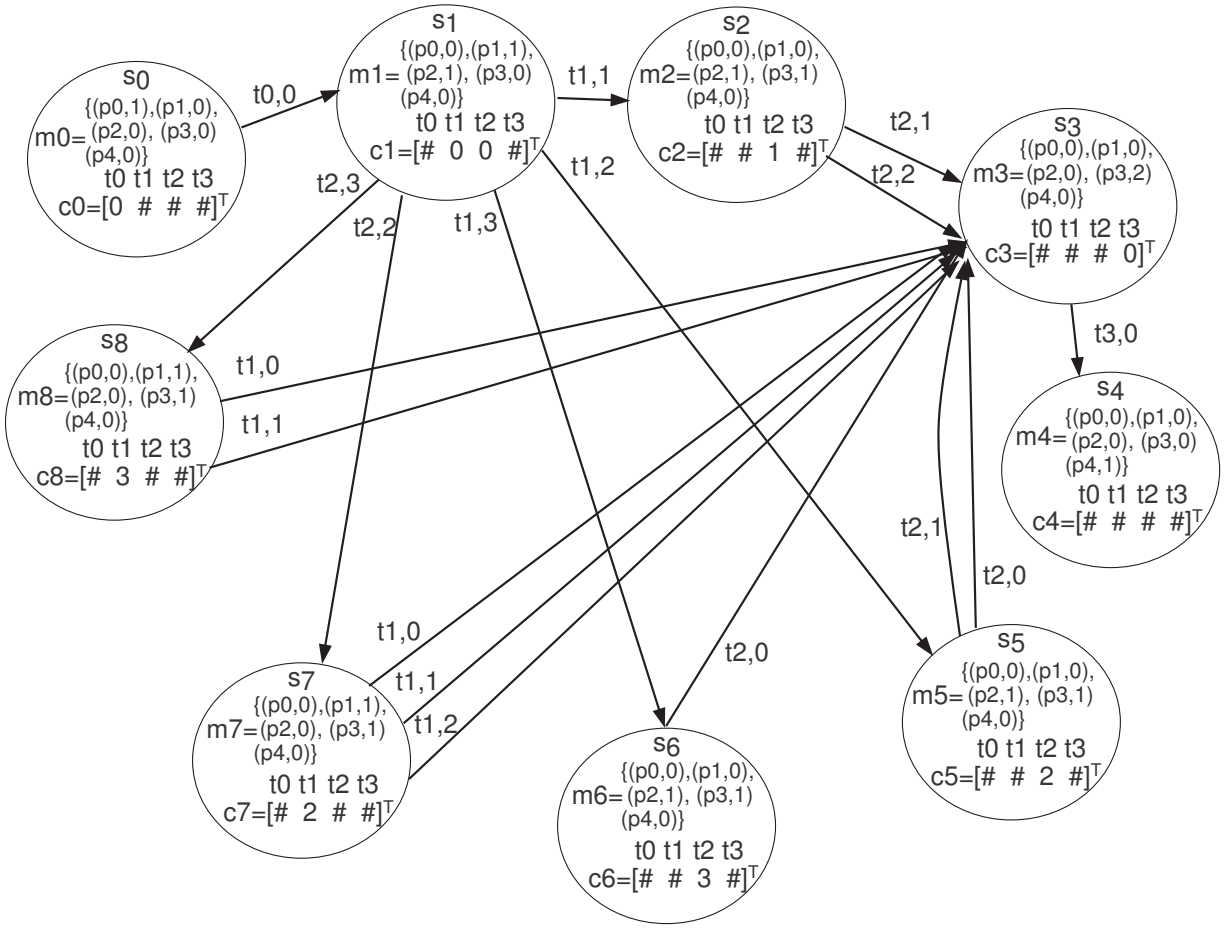


**Figure 3.21** Reachability graph for Figure 3.20(a)

CHAPTER 4

# SOFTWARE SYNTHESIS METHOD

The challenges of embedded software development have evolved through the years due to the necessity of handling even more complex functional and non-functional requirements. In this context, model-driven methods have been quite effective for reducing the intricacies of software development, since they allow designers to reason about a system by focusing on details of interest (and ignoring extraneous ones). Nevertheless, concerning hard real-time embedded systems with energy constraints, few model-driven methods are available, and, in general, most disregard formal models. It is important to bear in mind the benefits that formal models can provide in embedded software development, in the sense that they lay down a mathematical foundation for property analysis/verification as well as correct-by-construction techniques (e.g., automatic generation of customized code satisfying timing constraints).

This chapter describes the proposed software synthesis method for hard real-time embedded systems with energy constraints and provides an overview of the methodology in which the proposed method is inserted. The methodology is named **MEMBROS** - A **M**ethodology for **EMB**edded C**R**itical S**O**ftware Con**S**truction, and it is centered on the Petri net formalism, which is very suited for modeling energy constrained time-critical systems. It should be emphasized that timing and energy predictability are fundamental non-functional requirements of those systems, and Petri nets' mathematical foundation substantially helps in achieving these and other important goals.

The focus of this work is a software synthesis method for energy-constrained hard real-time systems, which is depicted in Figure 4.1 (see the box with bold lines). Nevertheless, an overview of MEMBROS methodology is presented firstly.

## 4.1 OVERVIEW

Figure 4.1 depicts the core activities of MEMBROS methodology, which organizes the activities in three groups: (i) Requirements Validation; (ii) Performance Evaluatation; and (iii) Software Synthesis. As follows, an overview of the methodology is provided.

Initially, the activities regarding requirements validation are performed. After carrying out the requirement analysis, the system requirements are modeled using a set of SysML diagrams (SDs), which represent the functionalities of the embedded software to be developed. The SDs provide to the designer an intuitive language for modeling the requirements without knowing the details of the Petri net formalism, which will be utilized in further activities for reasoning about quantitative/qualitative properties. SysML [100] is an abbreviation for Systems Modeling Language, an extension of UML for systems engineering. Since timing and energy constraints are of utmost importance in the systems of interest, the SDs are annotated with timing and energy consumption information (e.g., initial estimates) using MARTE [101]. MARTE is an UML profile that stands

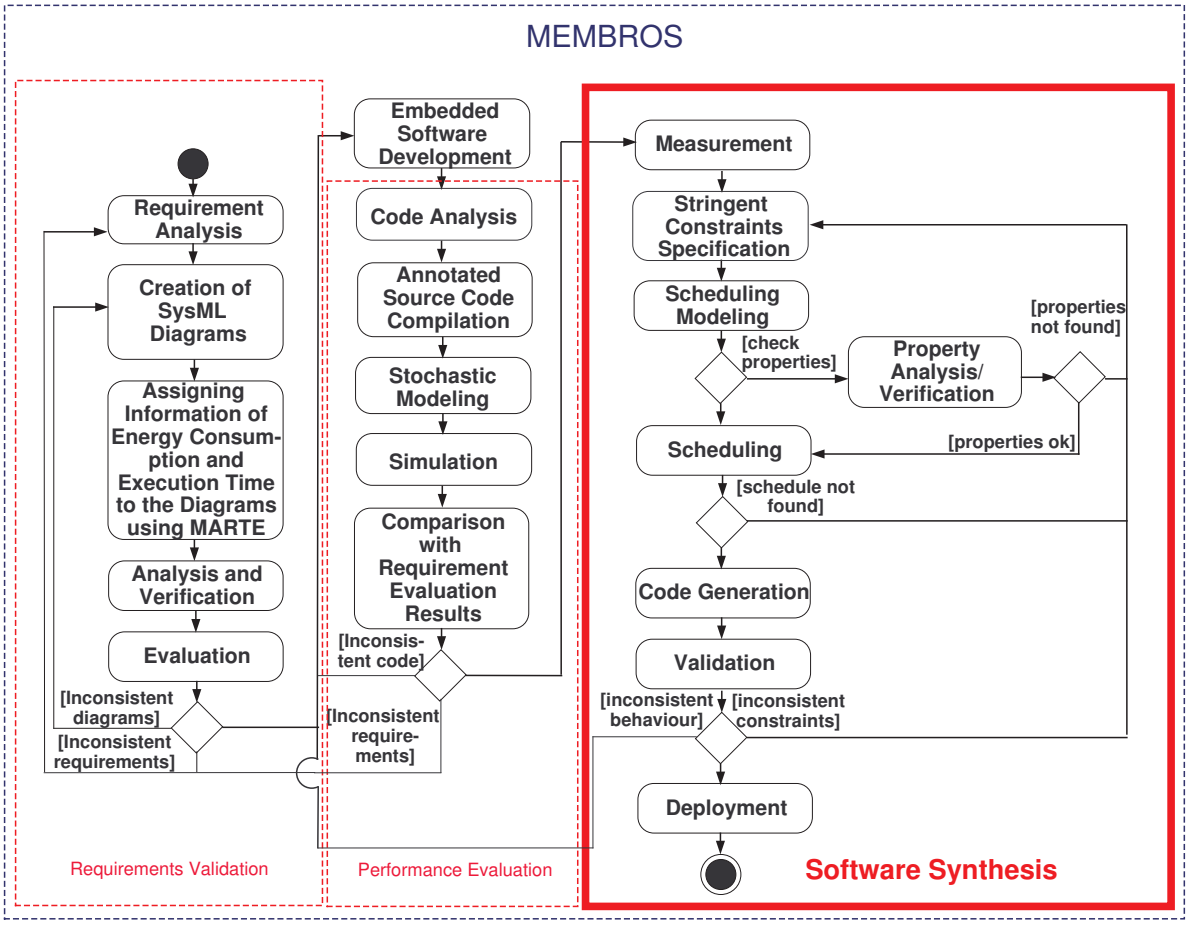


**Figure 4.1** MEMBROS activity diagram

for Modeling and Analysis of Real-Time and Embedded Systems. Next, the annotated SDs are automatically mapped into time Petri net (TPN) models in order to lay down a mathematical basis for analysis and verification of properties (e.g., absence of deadlock conditions between requirements). This activity also concerns to obtain best and worst-case execution times and the respective energy consumptions, in such a way that the requirements are also evaluated whether timing and energy constraints can be met. As the SDs are constructed by the designer, undesirable results in the evaluation activity may be not only related to inconsistent requirements, but also to inconsistent SDs.

Afterwards, the embedded software is implemented taking into account the results obtained in previous activities. Once the source code implementation is concluded, the designer analyzes the code in order to assign probability values to conditional and iterative structures. The probability annotations allow the compiled code be evaluated in the context of time and energy consumption, in such a way that these costs may be estimated before running the code on the hardware platform. Next, the compiled code is automatically translated into a coloured Petri net (CPN) model [102], a high-level Petri net extension, in order to provide a basis for the stochastic simulation of the embedded

software. Although not depicted in Figure 4.1, an architecture characterization activity is also considered to permit the construction of a library of CPN basic building blocks, which provide the foundation for the automatic generation of CPN stochastic models. From the CPN model (generated by the composition of basic blocks), a stochastic simulation of the compiled code is carried out considering the characteristics of the target platform. If the simulation results are in agreement with the requirements, the software synthesis is performed.

Software synthesis activities are concerned with the stringent constraints (e.g., time and energy), and, in the general sense, it is composed of two subgroups of activities: (i) tasks' handling; and (ii) code generation. Tasks' handling is responsible for tasks' scheduling, resource management, and intertask communication, whereas code generation deals with the static generation of the final source code, which includes a customized runtime support, namely, dispatcher. It is important to state that the concept of task is similar to *process*, in the sense that it is a concurrent unit activated during system runtime. For the following activities, it is assumed that the embedded software has been implemented as a set of concurrent hard real-time tasks.

Initially, a measurement activity is performed to obtain the tasks' timing information as well as the information regarding the hardware energy consumption. Next, the designer defines the specification of system stringent constraints, which consists of a set of concurrent tasks with their respective constraints, behavioral descriptions, information related to the hardware platform (e.g., voltage levels and energy consumption) as well as the energy constraint. Afterward, the specification is translated into an internal model able to represent concurrent activities, timing information, intertask relations, such as precedence and mutual exclusion, as well as energy constraints. The adopted internal model is a time Petri net extension (TPNE), labeled with energy consumption values and code annotations. After generating the internal model (TPNE), the designer may firstly choose to perform property analysis/verification or carry out the scheduling activity. This work adopts a pre-runtime scheduling approach in order to find out a feasible schedule that satisfies timing, intertask and energy constraints. Next, the feasible schedule is adopted as an input to the automatic code generation mechanism, such that a tailored code is obtained with the respective runtime control, namely, dispatcher. Finally, the application is validated on a DVS platform in order to check system behaviour as well as the respective constraints. Once the system is validated, it can be deployed to the real environment.

Henceforward, the focus is on the software synthesis activities. For more information about the requirements validation and performance evaluation activities, the reader should refer to [103] and [104], respectively.

## 4.2 PROPOSED METHOD

Although the software synthesis activities have been presented in the context of MEMBROS methodology, the proposed method can be utilized in other methodologies as well as a self-contained method. In the latter situation, the designer should provide all data required in the specification activity. The reader should bear in mind that the contribution

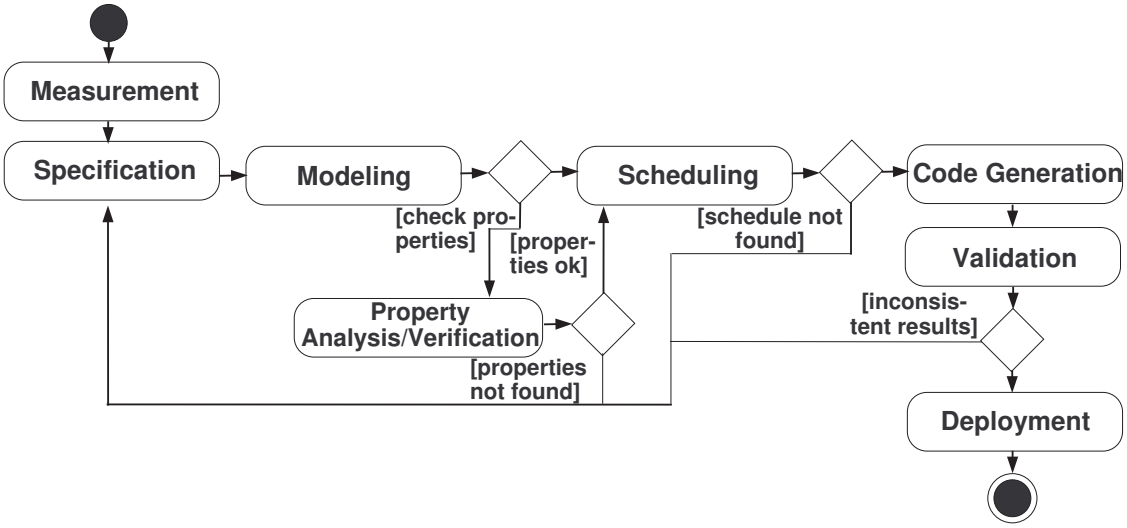


**Figure 4.2** Software synthesis activity diagram

of this thesis is the software synthesis method, thus, for a better visualization, Figure 4.2 depicts the proposed method activities detached from MEMBROS methodology.

The following items provide an overview of each activity, the associated artifacts and roles, leaving for the forthcoming chapters the in-depth details:

- **Measurement.** This activity concerns the measurement of each task and dispatcher (runtime support) worst-case execution cycles (WCEC) and the hardware energy consumption. In the proposed method, the hardware energy consumption is related with the mean value of the energy consumption per clock cycle in each voltage level (and the respective maximum frequency) of a DVS processor. This approach allows the calculation of a task worst-case execution time (WCET) as well as the respective energy consumption when the task is executing in a specific voltage level (and the associated maximum CPU frequency);
  *Artifacts*: The worst-case execution cycles of each software component (i.e., tasks, dispatcher and scheduler) and the energy per clock cycle of each voltage level (and the respective maximum frequency)
  *Role*: designer and software engineer

- **Specification.** Using the information from previous activity, the non-functional specification is carried out. The proposed method adopts a specification model that contemplates:

  1. *Hard Real-Time Tasks.* This work assumes an embedded software implemented as a set of concurrent periodic time-critical tasks. Hence, the designer specifies each task considering: (i) the stringent timing constraints, for instance, phase, release, WCEC, deadline and period; (ii) the behavioral description, which is

represented by a function implemented in C programming language; and (iii) the intertask relations, such as precedence and exclusion relations. Although this thesis assumes tasks implemented in C language, the method is not programming language dependent. Moreover, it is possible to have situations that the proposed method may receive as an input a monolithic embedded software (for instance, when the method is adopted in a different methodology). In this case, it is up to designer to split the software in concurrent tasks, being careful in specifying the relations that may exist between them;

2. *Runtime Support.* The dispatcher execution may affect the time-critical tasks during system runtime. Thus, the proposed specification model also takes into account the overheads (e.g., time and energy) related with the dispatcher for managing the tasks throughout system execution.

3. *Scheduling Type.* The scheduling type specifies whether the tasks are preemptable or non-preemptable;

4. *Hardware Architecture.* The hardware architecture is assumed to adopt a DVS-capable processor, and the designer should provide a specification that includes: (i) a discrete set of voltage/frequency levels; and (iii) the energy consumption per clock cycle in each level (which was obtained in the measurement activity);

5. *Energy Constraint.* The system energy constraint also needs to be defined, which sets an upper bound in terms of energy consumption that a schedule must not surpass;

*Artifacts*: a non-functional specification that includes stringent timing constraints, intertask relations, the source code of each task, overheads and details about the DVS platform
*Role*: designer

- **Modeling.** From the non-functional specification, the system modeling can be performed. This work adopts a bottom-up approach, in which a set of formal composition rules are considered for combining basic building block models. These building blocks, represented as TPNE models, represent each aspect of a hard real-time system, more specifically, the tasks' timing constraints (e.g., release time), intertask relations, overheads, the energy consumption during a task computation as well as the processor availability. Such an approach generates a TPNE model from the system specification, so that tools can be adopted to automate the modeling and scheduling processes. Besides, the proposed modeling approach is also adopted for analysis/verification of behavioral and structural properties;
  *Artifacts*: a time Petri net model representing the system
  *Role*: designer

- **Property Analysis/Verification.** This activity allows the designer to reason about a system by analyzing and verifying qualitative as well as quantitative properties in the TPNE models. It is worth mentioning that the generated models are assured to contain some structural and behavioral properties, which are of utmost importance, for instance, in the scheduling activity;
  *Artifacts*: a report stating the qualitative and quantitative properties of the Petri net model
  *Role*: designer

- **Scheduling.** Adopting the TPNE model, the scheduling activity provides the ordering of task executions at design-time, such that timing and energy constraints are met as well as system resources are properly allocated before system execution. More specifically, the proposed method adopts a pre-runtime scheduling approach, which is a depth-first search method on the TPNE model. The result of the scheduling activity is a feasible schedule represented as a time labeled transition system (TLTS);
  *Artifacts*: a feasible schedule that satisfies the specified constraints
  *Role*: designer

- **Code Generation.** In the proposed method, the code is generated by traversing the TLTS (feasible schedule), and detecting the times when each task will execute, ensuring that the constraints are also met during system execution. The code generation activity includes not only the code of each task, but also a customized runtime support and a schedule table that contains all information about each task execution;
  *Artifacts*: a customized predictable code
  *Role*: designer and software engineer

- **Validation** Finally, the embedded software is validated. In this work, attention is devoted to the verification of timing and energy constraints;
  *Artifacts*: a report about the test procedure and bugs found
  *Role*: test engineer

- **Deployment.** Once all constraints are met, the system can be deployed to the real environment. This activity concerns all steps required to the deployment of the embedded system in the final environment.
  *Artifacts*: a report about the deployment procedure
  *Role*: deployment team

Some activities are automated by support tools, and, thus, those activities are only performed by the designer. Next chapters present the activities and the associated tools.

## 4.3 SUMMARY

Embedded software development provides several challenges that are not usually found in traditional PC-like application construction, for instance, challenges related to stringent timing and energy constraints. To cope with these and other issues, this chapter presented the activities related with the proposed software synthesis method as well as the methodology in which the software synthesis is inserted. The proposed method and MEMBROS methodology are centered on the Petri net formalism, which substantially helps in mitigating the intrinsic complexities of embedded software development as well as in achieving the desired goals.

CHAPTER 5

# MEASUREMENT AND SPECIFICATION

This chapter presents the measurement and specification activities, which are responsible for capturing and describing the characteristics of the hardware architecture and the embedded software to be coped. More specifically, the measurement activity is responsible for measuring the tasks' worst-case execution cycles (WCEC), the runtime support WCEC and the energy consumption at each voltage level of a DVS processor. The specification activity concerns the specification of each task constraints, timing information and behavioral description as well as the information regarding the DVS hardware platform (which includes the energy consumption values measured previously). As follows, the activities are detailed.

## 5.1 MEASUREMENT

The proposed measurement activity contemplates two steps in order to provide the necessary data for carrying out the specification activity: (i) the measurement of tasks and dispatcher WCEC; and (ii) the hardware characterization, which concerns the measurement of the energy consumption per clock cycle at each voltage level (and the associated maximum CPU frequency). Firstly, the approach for measuring each task and dispatcher WCEC is presented, and, next, details regarding the estimation of energy consumption are provided. Afterwards, the measurement scheme and the adopted equations are described, which supply the necessary foundation for obtaining the desired values using a hardware platform and an oscilloscope. Finally, the statistical methods adopted in this activity are presented as well as the software tool implemented to automate the measuring process.

### 5.1.1 Tasks and Dispatcher WCEC

In order to obtain the tasks and dispatcher WCEC, this work assumes that the designer (or developer) adjusts the source codes to always incur the worst-case execution times (WCET) during the measurement activity. More specifically, the expressions, which guard the execution of iterative and conditional commands (as well as recursive functions), are tuned to always impose the worst-case situations. As an example, Figure 5.1 depicts an iterative and a conditional command adjusted to force the WCET. In Figure 5.1(a), variable $n$ is set to an upper bound value, and, in Figure 5.1(b), the guard expression is adjusted to be always true as the execution of *then* block provides the worst-case situation. Additionally, the designer may provide input data representing the worst-case scenario, whether the code requires some data to generate the WCET.

After adjusting each task and dispatcher code (assumed to be implemented as functions in C language), the designer places each individual function between two commands,

```
for(i = 0; i<= n; i++) {
            ...
}
```

a)

```
if((guard expression) | TRUE) {
        //takes 100ms
            ...
} else {
       //takes 20ms
            ...
}
```

b)

**Figure 5.1** Adjusting a task code to impose the WCET

which manipulate the hardware I/O port to indicate the code start and end times. For instance, Figure 5.2 depicts an example of a pseudocode for measuring a task $T_1$. This mechanism allows the measurement of a task WCET directly on the hardware platform using an oscilloscope. More specifically, the oscilloscope detects a pulse from the hardware I/O port and provides the data regarding the pulse length, which actually represents the code execution time.

```
while(TRUE) {

      IOPort = IOPort | 0x1;
      codeT1();
      IOPort = IOPort & ~0x1;

      for(i = 0;i < 300;i++); //delay

}
```

**Figure 5.2** Adjusting a task function to measure the WCET

The presented approach is plausible, since this work is concerned with hard real-time embedded systems with energy constraints, in which timing and energy predictability are of utmost importance. In this case, the code must be predictable, hence, the designer (or developer) is assumed to have sufficient knowledge of the code structures that may affect each task WCET. Although the code adjustment seems a burdensome activity, the approach can be automatized by software tools. An alternative is the adoption of code analyzers for estimating the tasks WCET/WCEC [105]. Nevertheless, those analyzers also have drawbacks [105] (e.g., WCET overestimation) and a comparison between the adopted approach and the analyzers is beyond the scope of this thesis.

### 5.1.2 Hardware Characterization

Hardware characterization contemplates the measurement of the energy consumption per clock cycle at each voltage level of a DVS processor, since the duration of a task execution is specified as execution cycles in the proposed software synthesis method. Thus, multiplying a task WCEC by the energy consumption per clock cycle at a voltage level (and the associated maximum CPU frequency), results in the worst-case energy consumption of this task at the selected level.

The energy consumption per clock cycle is not fixed for any embedded software, but software dependent. Previous assertion is based on the intrinsic characteristics of an embedded software, which may adopt instructions that utilize different CPU internal components (e.g., multipliers). In other words, the energy consumption varies from one software to another due to the different usage of the CPU core. Considering Equation 3.4 presented in Chapter 3 (Section 3.2.2), the characteristics of the instructions adopted by an embedded software directly affect the average switched capacitance per clock cycle ($A$) [26], which may increase or decrease the energy consumption.

```
void characterization(void){
   unsigned int d, i;
   float a;
   i++;
   for(d = 0; d < 0x9FFFF;d++){
       a = 50.3433/2424.22242  * 0.343;
   }
   i--;
}
```

**Figure 5.3** A code example for hardware characterization

Taking into account the previous issues, the hardware characterization step focuses on obtaining a *mean value* of the energy consumption per clock cycle at each voltage level (and the associated maximum CPU frequency) considering the characteristics of the embedded software. More specifically, a function in C language is implemented using the set of instructions common in the software application as well as the number of times that each instruction occurs. The power consumption of this C function is then measured at each voltage level and, next, divided by the respective CPU frequency, such that the energy consumption per clock cycle is obtained. For instance, Figure 5.3 depicts a code adopted to perform the characterization of a DVS processor, assuming the bulk of the software computation is related with $FOR$-loops and division of decimal numbers. Besides, the same approach as the one depicted in Figure 5.2 is adopted to measure the power consumption in this step.

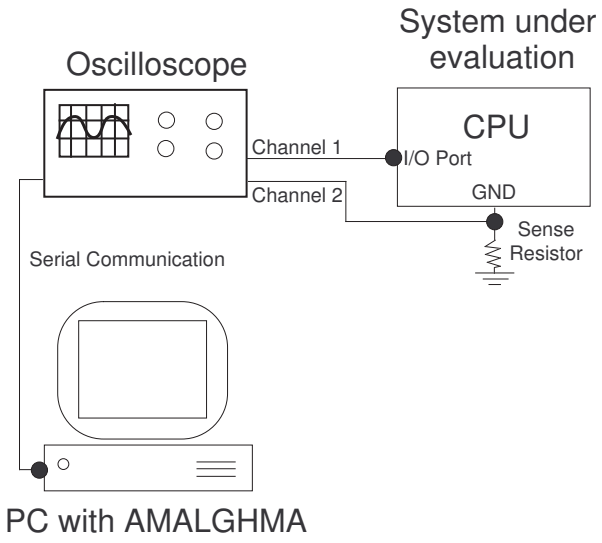


**Figure 5.4** Measurement scheme

### 5.1.3 Scheme and Equations

Figure 5.4 shows the adopted measurement scheme, which is based on the technique described in [106]. To measure a task WCEC, a PC is connected to an oscilloscope (Agilent DSO03202A), which captures the task start and end times by monitoring an I/O port of the target CPU. As demonstrated previously in Section 5.1.1, the task code is adjusted to indicate the respective execution, which is recognized by an oscilloscope as a pulse (see Figure 5.5). The data acquired by the oscilloscope are then transmitted to a PC, in such a way that a software tool calculates the task WCEC as follows

$$WCET = (x_{end} - x_{start}) \times TU \tag{5.1}$$

$$WCEC = WCET \times f \tag{5.2}$$

$x_{start}$ and $x_{end}$ are, respectively, the pulse start and end points, $TU$ is the adopted time unit in the oscilloscope, and $f$ is the CPU clock frequency selected during the measuring process.

Regarding the measurement of power consumption, a small sense resistor (1 Ohm) is attached to the CPU GND pin, such that, in the interval of the task execution, the oscilloscope captures the CPU current draw ($I$) by measuring the average voltage drop ($V_{sr}$) across the sense resistor. Since the supply voltage ($V_{cc}$) and the resistance of the sense resistor ($R_{sr}$) are known, the power consumption ($P$) in a voltage/frequency level can be calculated using the following equations:

$$I = V_{sr}/R_{sr} \tag{5.3}$$

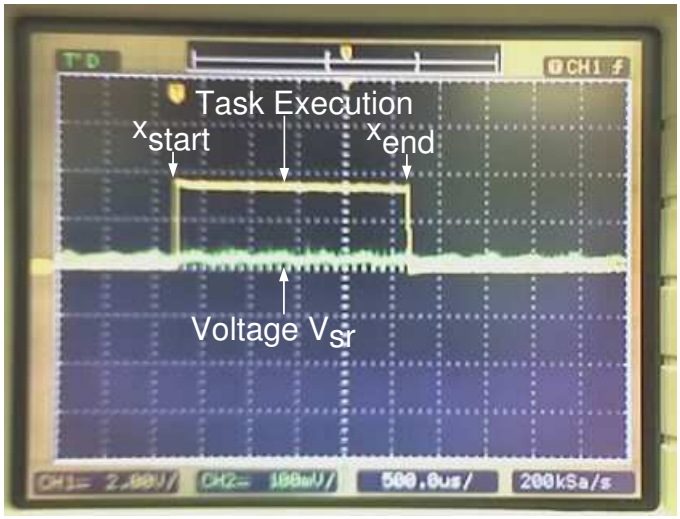


**Figure 5.5** Oscilloscope detecting a task execution and the voltage drop across a sense resistor

$$P = V_{cc} \times I \tag{5·4}$$

As the CPU clock frequency ($f$) is also known, the energy consumption per clock cycle ($E$) is obtained as follows:

$$E = P/f \tag{5·5}$$

### 5.1.4 Statistical Methods

This work adopts a set of statistical techniques to reduce the impact of errors (e.g., noise) in the measuring process, which is somehow affected by: (i) the resistor error; and (ii) the oscilloscope resolution. Considering the metrics of interest, execution time and energy consumption, the impact of noises has influenced more the energy consumption measurements, according to experiments conducted. Regarding execution times, the noises are less perceptible, as a code behaviour does not vary (it was previously adjusted to always generate the WCEC). Nevertheless, in both metrics, statical techniques are applied to guarantee a proper treatment of the collected data. Besides, for the adopted metrics, the desired statistic is the mean value.

In the proposed measurement activity, two statistical methods are available to the designer, one based on bootstrap [107] and other based on parametric methods [108, 109]. More specifically, bootstrap may not assume previous knowledge of the population distribution, whereas the parametric method assumes, in this work, a normally distributed population and a relative precision given by the designer. Comparing both approaches, bootstrap provides fast results, whilst parametric method provides flexibility in the sense that the designer can specify some parameters regarding the population. These methods are detailed below.

**Bootstrap** Bootstrap is a resampling method, in the sense that obtains samples within a previously measured sample. Since only one sample is performed on the real hardware, bootstrap method generates rapid results. Details are presented as follows:

1. Initially, a *random sample* $x = (x_1, x_2, ..., x_n)$ with $n$ measurements are obtained from the DVS processor.

2. Next, $B$ *bootstrap samples* $(x^{*1}, x^{*2}, ..., x^{*B})$ are generated. A *bootstrap sample* $x^* = (x_1^*, x_2^*, ..., x_n^*)$, is obtained by randomly sampling $n$ times, with replacement, the original data contained in $x$. The star notation indicates that $x^*$ is a resampled version of $x$. As an example, assuming $n = 4$, a typical bootstrap sample might be $x^* = (x_4, x_2, x_2, x_3)$, in which $x_1^* = x_4$, $x_2^* = x_3^* = x_2$ and $x_4^* = x_3$. It is important to state that each element in $x$ has the same probability to be selected: $1/n$.

3. For each bootstrap data set $x^{*b}$, $b = 1, 2, ..., B$, there is a *bootstrap replication* $\hat{\theta}^*(b) = s(x^{*b})$, in which $s$ is the statistic of interest, more specifically, the mean of the bootstrap data set:

$$s(x^{*b}) = \overline{x}^{*b} = \frac{(\sum_{i=1}^{n} x_i^{*b})}{n} \tag{5.6}$$

4. Afterwards, the replications are ordered, such that $\hat{\theta}^*_{(1)}$ is the smallest replication and $\hat{\theta}^*_{(B)}$ is the largest one. Considering a confidence degree of $1 - \alpha$, in which $\alpha$ is the significance level, the confidence interval is denoted by $[\hat{\theta}^*_{(B\alpha/2)}, \hat{\theta}^*_{(B[1-\alpha/2])}]$. For a better understanding, assuming $B = 1000$ and a confidence degree of 95% or 0.95 ($\alpha = 0.05$), the confidence interval is represented by $[\hat{\theta}^*_{(25)}, \hat{\theta}^*_{(975)}]$;

5. Finally, the midrange value is calculated, which represents the final estimated value:

$$mr = \frac{\hat{\theta}^*_{(B\alpha/2)} + \hat{\theta}^*_{(B[1-\alpha/2])}}{2} \tag{5.7}$$

The values for $n$ and $B$ are, respectively, 60 and 1000 in this work. Since these values are considerably large and the adopted bootstrap method takes into account sample means, central limit theorem [110] estimates the population mean.

**Parametric Method** Parametric method assumes a relative precision specified by the designer as well as the collected data fit on the normal distribution. Since all the measurements are directly performed on the DVS processor, this method is considerably slower than bootstrap (despite the flexibility to indicate the desired precision). The parametric method is detailed below:

1. Initially, a set of samples (or *replications*) are obtained from the DVS processor. The amount of initial samples is denoted by $B$ and each sample $x = (x_1, x_2, ..., x_n)$ contains $n$ measurements (or *observations*);

2. For each sample $x^b$, $b = 1, 2, ..., B$, the mean ($\overline{x}^b$) is calculated:

$$\overline{x}^b = \frac{(\sum_{i=1}^{n} x_i^b)}{n} \tag{5.8}$$

3. Next, the mean of all samples ($\overline{\overline{x}}$) is obtained as well as the respective standard deviation ($s$) and error ($se$):

$$\overline{\overline{x}} = \frac{(\sum_{b=1}^{B} \overline{x}^b)}{B} \tag{5.9}$$

$$s = \sqrt{\frac{\sum_{b=1}^{B} (\overline{x}^b - \overline{\overline{x}})^2}{B-1}} \tag{5.10}$$

$$se = t_{1-\alpha/2,n-1} \times \frac{s}{\sqrt{B}} \tag{5.11}$$

In previous equation, $t_{1-\alpha/2,n-1}$ is the critical value taken from student $t$ distribution for $1 - \alpha/2$ and $n - 1$ degrees of freedom;

4. The relative precision is then calculated ($rp$). If the relative precision is equal to or less than the precision specified by the designer ($dp$), the measuring process stops. Otherwise, an additional number of samples is obtained ($B'$), and steps from 2 to 4 are repeated. It is important to state that, at step 3, previous samples are also considered in the calculation. The equations are presented below:

$$rp = \frac{se}{\overline{\overline{x}}} \tag{5.12}$$

$$B' = \left( \frac{t_{1-\alpha/2,n-1} \times s}{dp \times \overline{\overline{x}}} \right)^2 \tag{5.13}$$

Although a normal distribution is assumed in this method for the adopted metrics, in fact, this assumption is valid for any data set measured by the described method. Previous assertion is grounded on: (i) the large values adopted for variables $B$ and $n$ (10 and 40, respectively); (ii) the distribution of sample means; and (iii) the central limit theorem [110]. Besides, as this method provides the mean ($\overline{\overline{x}}$) of all sample means, $\overline{\overline{x}}$ estimates the population mean, according to the central limit theorem.

### 5.1.5 AMALGHMA Tool

It is important to highlight that AMALGHMA tool (Figure 5.6) - **A**dvanced **M**easurement **A**lgorithms for **H**ardware **A**rchitectures - has been developed for automating the measuring process. AMALGHMA [24] implements the described statistical methods, such that the designer only needs to concern with the scheme (Figure 5.4) and the tasks' code. Besides, individuals not familiar with statistical techniques can take advantage of this tool, since the details (e.g., equations) are hidden from non-specialized users. Indeed, AMALGHMA tool is a contribution of this thesis. Besides, AMALGHMA provides the following functionalities and features:

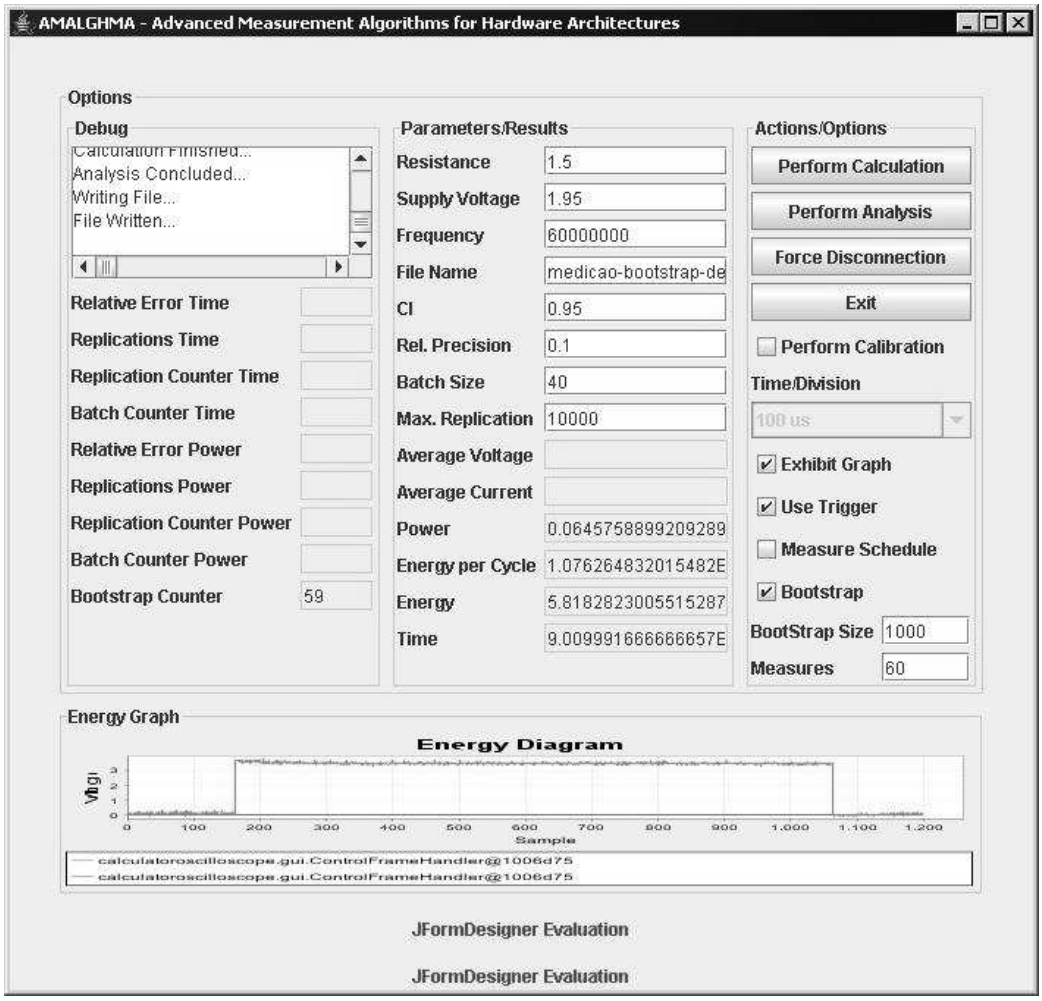


**Figure 5.6** AMALGHMA tool

- *Perform Calculation.* In this functionality, AMALGHMA acquires a single measurement in order to provide initial insights to the user about a metric of interest (e.g., the energy consumption per clock cycle at a voltage/frequency level);

- *Perform Analysis.* This functionality allows the execution of bootstrap or parametric method. Using the intuitive graphical interface of AMALGHMA, the user may provide the desired number of samples ($B$) as well the respective amount of measurements ($n$);

- *Graphical Interface.* AMALGHMA allows the graphical visualization of the data acquired by the oscilloscope as well as the execution of a measuring method, such that the designer can supervise the measuring process.

AMALGHMA tool has been validated using a NXP LPC2106 processor [72], a 32-bit microcontroller with ARM7 core [24, 104, 103]. Moreover, although AMALGHMA has

been implemented for the proposed software synthesis method, the tool can be adopted in any method or methodology, inasmuch as the scheme depicted in Figure 5.4 is provided.

## 5.2 NON-FUNCTIONAL SPECIFICATION

As described in Chapter 4, the proposed software synthesis method is responsible for dealing with the stringent timing and energy constraints of an embedded software as well as with the management of system resources utilized by it. Therefore, the proposed method assumes the embedded software was implemented previously, more specifically, as a set of concurrent tasks, which will have the respective constraints properly met after the software synthesis process.

In this context, the specification activity plays an important role, since it provides all information required for the realization of further activities, such as modeling, scheduling and code generation. In the general sense, this activity is concerned with the specification of system constraints and adopts a specification model that contemplates: (i) hard real-time tasks with the corresponding constraints; (ii) the runtime support information; (iii) the scheduling type; (iv) the information regarding the hardware platform; and (v) the energy constraint. In the following sections, the specification model is detailed.

### 5.2.1 Hard Real-Time Tasks

This work assumes an embedded software implemented as a set of concurrent periodic time-critical tasks, which perform a computation repeatedly, once in each fixed period of time [6]. Assuming $\mathcal{T}$ is the set of all tasks in a system specification and $\mathcal{ST}$ the set of task source codes, the definition of periodic task is presented as follows.

**Definition 5.1** (Periodic Task)**.** *A periodic task $\tau_i \in \mathcal{T}$ is defined by $\tau_i = (ph_i, r_i, c_i, d_i, p_i, code_i)$, in which $ph_i$ is the initial phase; $r_i$ is the release time; $c_i$ is the worst-case execution cycles (WCEC) required for executing task $\tau_i$; $d_i$ is the deadline; $p_i$ is the period; and $code_i \in \mathcal{ST}$ is the task C code (behavioral description). Besides, $ph_i, r_i, c_i, d_i, p_i \in \mathbb{N}$.*

In previous definition, the timing constraints are denoted by some multiple of a specific time unit, namely, *task time unit* (TTU). For instance, if one TTU corresponds to $100\mu s$, a task $\tau_i$ with a periodic execution of $3ms$ has a period of 30 TTU ($p_i = 30$). Regarding a task worst-case execution time (WCET), one TTU represents the smallest indivisible granule, in the sense that a task cannot be preempted during the execution of one TTU. In other words, a task can only be preempted between the occurrence of two TTUs. Recalling that $WCET = WCEC/f$, in which $f$ represents a CPU clock frequency (see Chapter 3). As the reader should note, the selection of a TTU is a design decision, and it is up to the designer to verify the best trade-off. For instance, a fine granularity implies more possibilities to be analyzed during the scheduling activity (e.g., an increase of the state space size), whereas a coarse granularity leads to less preemption points (which may affect the possibility of finding a feasible schedule).

Furthermore, in many applications, there are tasks that are not periodic, but are executed asynchronously as a response to internal or external events (e.g., a response

to an operator request). Although the request times are not known beforehand, the minimum interval between two consecutive requests can usually be obtained in advance. These tasks are denominated sporadic tasks.

**Definition 5.2** (Sporadic Task)**.** *A sporadic task $\tau_k \in \mathcal{T}$ is defined by $\tau_k = (c_k, d_k, min_k, code_k)$, in which $c_k$ is the worst-case execution cycles (WCEC) required for executing task $\tau_k$; $d_k$ is the deadline; $min_k$ is the minimum period between two activations of task $\tau_k$; and $code_k \in \mathcal{ST}$ is the task C code (behavioral description);*

To allow the scheduling of sporadic tasks, the proposed method translates these tasks into equivalent periodic tasks using a small variation of the technique described in [111, 52, 53, 11] in order to consider the effects of DVS in the computation time. Since a sporadic task is now executed periodically, the associated external or internal events need to be buffered until the periodic task deals with the request. Assuming a sporadic task $\tau_s = (c_s, d_s, min_s)$, the respective periodic task $\tau_p = (ph_p, r_p, c_p, d_p, p_p, code_p)$ can be obtained satisfying the following conditions:

1. $ph_p = 0$;
2. $c_p = c_s$;
3. $d_s \geq d_p \geq C_{p_{max}}$, in which $C_{p_{max}} = \lceil c_p / f_{max} \rceil$ and $f_{max}$ is the maximum operating frequency available on the CPU;
4. $C_{p_{max}} \leq p_p \leq min(d_s - d_p + 1, min_s)$;
5. $r_p = 0$.

In the above technique, there is a range of possible values for $d_p$ and $p_p$, and it is up to designer to select the best trade-off for a given application. In [11], details are provided about this technique and [53] depicts a thorough example.

### Intertask Relations

The proposed software synthesis method consider two types of relations between tasks: (i) precedence and (ii) exclusion relations.

**Definition 5.3** (Precedence Relation)**.** *A task $\tau_i$ PRECEDES task $\tau_j$, if $\tau_j$ can only start executing after $\tau_i$ has finished.*

Precedence relations may exist between tasks when a task requires information produced by other task.

**Definition 5.4** (Exclusion Relation)**.** *A task $\tau_i$ EXCLUDES task $\tau_j$, if no execution of $\tau_j$ can start while task $\tau_i$ is executing. In other words, task $\tau_i$ can not be preempted by task $\tau_j$.*

In the proposed method, the exclusion relation is symmetric, more specifically, if $\tau_i$ *EXCLUDES* task $\tau_j$, then $\tau_j$ *EXCLUDES* task $\tau_i$. Exclusion relations may occur between tasks when concurrent access to shared resources must be avoided, such as access to data and I/O devices.

There are situations that a task needs to be splited into smaller tasks in order to separate the critical section from the rest of the task computation. In general, a critical section is the task code responsible for accessing a shared resource and requires exclusion relations with other tasks that utilize the same resource. Such an approach is not only adopted to guarantee mutual exclusion to shared resources, but, also, to increase the possibilities of finding a feasible schedule, since only the critical section will be executed in a mutual exclusive way (not the whole task). Besides, the smaller tasks generated from the split have precedence relations between them in order to ensure the proper execution order.

For a better understanding, assume two tasks: $A$ and $B$. Both have a critical section and these sections access the same shared resource. Thus, A is divided into tasks $A_0$, $A_1$, $A_2$, and B into tasks $B_0$, $B_1$, $B_2$, such that the critical section is isolated from the rest of the code. For the sake of this example, the execution order of each subtask is denoted by the number subscripted in the task name. Considering that $A_1$ and $B_1$ are the critical sections, the relation $A_1$ *EXCLUDES* $B_1$ is defined. Additionally, for the proper execution of both tasks A and B, the following relations are required: $A_0$ *PRECEDES* $A_1$, $A_1$ *PRECEDES* $A_2$, $B_0$ *PRECEDES* $B_1$ and $B_1$ *PRECEDES* $B_2$. Considering the specification of each subtask, the timing constraints are the same, except the WCEC.

### 5.2.2 Runtime Support

Overheads, such as context-switching and voltage/frequency tuning, take place during system runtime and may affect the timing and energy constraints of a hard real-time system. Indeed, if overheads are neglected, catastrophic issues may occur and even the gains obtained with DVS may be significantly reduced [27].

The proposed method takes into account overheads by associating them with the runtime support, since the dispatcher is the only responsible for adjusting the voltage/frequency level, managing context-switching and for starting each system task.

**Definition 5.5** (Dispatcher - WCET)**.** *The dispatcher WCET is represented by variable $o$ and the respective energy consumption by variable $o_{\mathcal{E}}$.*

It is important to state that the proposed method assumes the dispatcher execution at a fixed voltage/frequency level, and it is up to the designer to select the appropriate one.

Additionally, there are situations that the dispatcher only needs to perform a simple voltage/frequency switching, which is less costly than the dispatcher WCET. This situation is required during the modeling and scheduling activity in order to improve the estimation of the system energy consumption as well as to reduce the impact of the dispatcher WCET in tasks' executions.

**Definition 5.6** (Dispatcher - Voltage/Frequency Switching)**.** *When considering just a voltage/frequency switching, the dispatcher execution is denoted by $a_v$ and the associated energy consumption by $a_{v\varepsilon}$*

### 5.2.3 Scheduling Type

Additionally, the designer needs to select the scheduling type in order to indicate whether preemption is allowed or not. In a preemptive scheduling (P), a task can be preempted by another task at any time, whereas a non-preemptive scheduling (NP) does not allow a task be interrupted by other tasks. In other words, a new task can only run if there is no task executing on the CPU.

### 5.2.4 Hardware Architecture

The proposed method considers a hardware architecture containing a DVS-capable processor, which permits the concomitant tuning of the CPU supply voltage and clock frequency for reducing energy consumption. Therefore, the designer must specify the available voltage/frequency levels on the DVS processor as well as the energy consumption per clock cycle at each level. The model for specifying these parameters is presented as follows.

**Definition 5.7** (Voltage-Frequency Function - $vff$)**.** *Let $\mathcal{V}_{cpu}$ and $\mathcal{F}_{cpu}$ be two sets. $\mathcal{V}_{cpu}$ is the discrete set of CPU supply voltage levels and $\mathcal{F}_{cpu}$ the respective discrete set of CPU frequencies, such that $|\mathcal{V}_{cpu}| = |\mathcal{F}_{cpu}|$; and $vff : \mathcal{V}_{cpu} \rightarrow \mathcal{F}_{cpu}$ (voltage-frequency function) an bijective function that maps each voltage level to one, and only one, processor execution frequency, which is the maximum operating frequency in that voltage level.*

**Definition 5.8** (Voltage-Energy Function - $vef$)**.** *Let $\mathcal{V}_{cpu}$ be the discrete set of CPU supply voltage levels. $vef : \mathcal{V}_{cpu} \rightarrow \mathbb{R}$ (voltage-energy function) is an function that maps each voltage level to a real number, which represents the energy consumption per clock cycle in that voltage level.*

In the proposed method, voltage/frequency levels that do not provide energy saving due to the leakage current are not considered in the specification.

### 5.2.5 Energy Constraint

The proposed software synthesis method is concerned with energy-constrained time-critical systems, in which timing and energy constraints are of utmost importance. In addition to timing constraints, the designer needs to specify the system energy constraint, which defines the maximum energy consumption that a schedule cannot violate.

**Definition 5.9** (System Energy Constraint - $e_{max}$)**.** *The system energy constraint $e_{max}$ is an upper bound in terms of energy consumption that a schedule must not surpass.*

$e_{max}$ is adopted by the scheduling algorithm (Chapter 8) to search for a feasible schedule, in which the energy consumption of all tasks executions (and the respective

overheads) does not violate such an upper bound. Moreover, since the designer previously knows the schedule period (Section 6.2) and, for instance, may have some insights about the battery limitation in the final system, he can enforce a desired maximum energy consumption related to the execution of all tasks in a feasible schedule.

## 5.3 SUMMARY

This chapter detailed the measurement and specification activities, which are, respectively, responsible for capturing and describing the characteristics of a hard real-time system. As presented, the measurement activity provides a set of steps to measure tasks' WCEC and the hardware energy consumption as well as a software tool to automate the process. Additionally, statistical techniques are taken into account in order to reduce the impact of noises in the measuring process and, thus, to provide more accurate estimates of the metrics of interest. Concerning the specification activity, several issues not usually considered in similar works, for instance, overheads and intertask relations, are taken into account in the proposed specification model, assuring that system constraints are met in the final generated code.

CHAPTER 6

# MODELING - BUILDING BLOCKS

Hard real-time systems (HRTS) have stringent timing constraints that must be met in order to guarantee the correct system functioning. If these constraints are not met, catastrophic issues can occur, such as equipment damage or even loss of human lives. Considering energy-constrained HRTS, the challenges considerably increase in the development of such systems, as timing and energy constraints are usually conflicting requirements. In order to tackle those issues, model-driven methods seem a promising solution, since: (i) they allow designers to reason about a system by focusing on details of interest (and ignoring extraneous ones); and (ii) lay down a (mathematical) foundation for property analysis/verification as well as *correct-by-construction* techniques (e.g., automatic generation of customized code satisfying timing constraints).

This chapter initiates the presentation of the modeling approach adopted for representing hard real-time systems with energy constraints, focusing on the adopted Petri net extension and the proposed building block models. In short, from the non-functional specification, the proposed software synthesis method adopts a bottom-up modeling approach, in which a set of composition rules are considered for combining basic building block models. These basic blocks are modeled using a time Petri net extension and provide a basis for generating larger models that properly represent energy-constrained hard real-time systems. The generated models are not only adopted for scheduling purposes, but, also, for property analysis and verification. Firstly, this chapter presents the computational model and comments about the scheduling period. Next, the building blocks are detailed.

In this chapter, the bullet is replaced by an open diamond in the itemized lists to avoid confusion with Petri net notation for post-sets and pre-sets (Section 3.5).

## 6.1 COMPUTATIONAL MODEL

The computational model syntax is given by a time Petri net extended with energy consumption values and code annotations, and its semantics by a timed labeled transition system. As presented in Chapter 3, Petri nets are a family of formalisms suitable for modeling real-time systems, as concurrency, communication mechanism and synchronization - usual features of such systems, are naturally represented. Since the adopted Petri net extension is based on time Petri nets (TPN) and this formal model was presented in Chapter 3 (Section 3.5.9), some common concepts are not detailed. However, all definitions related to TPN is presented in order to provide a self-contained section.

A time Petri net (TPN) is a bipartite directed graph represented by a tuple $\mathcal{N}_{\mathcal{T}} = (P, T, F, W, m_0, I)$, in which $P$ (set of places) and $T$ (set of transitions) are non-empty disjoint sets of nodes. The edges are represented by $F$, in which $F \subseteq A = (P \times T) \cup (T \times P)$. $W : A \to \mathbb{N}$ represents the weight of the edges, such that

$$W(f) = \begin{cases} x \in \mathbb{N}, & if\ (f \in F) \\ 0, & if\ (f \notin F) \end{cases}$$

A TPN marking $m_i$ is a function ($m_i : P \rightarrow \mathbb{N}$), and $m_0$ is the initial marking. $I : T \rightarrow \mathbb{N} \times \mathbb{N}$ represents the timing constraints, in which $I(t) = [EFT(t), LFT(t)]\ \forall t \in T$, $EFT(t) \leq LFT(t)$. $EFT(t)$ is the Earliest Firing Time, and $LFT(t)$ is the Latest Firing Time.

**Definition 6.1** (Time Petri Net with Energy Consumption and Code Annotations - TPNE $\mathcal{N}_\mathcal{E}$)**.** *An extended time Petri net with energy consumption values and code annotations is represented by* $\mathcal{N}_\mathcal{E} = (\mathcal{N}_\mathcal{T}, \mathcal{E}, \mathcal{CS})$. $\mathcal{N}_\mathcal{T}$ *is the underlying time Petri net and* $\mathcal{E}$*:*$T \rightarrow \mathbb{R}_+ \cup \{0\}$ *is a function that assigns transitions to energy consumption values.* $\mathcal{CS}$*:*$T \nrightarrow \mathcal{ST}$ *is a partial function that assigns transitions to behavioral source code, in which* $\mathcal{ST}$ *is a set of task source codes.*

In the adopted computational model, time Petri net is extended with energy consumption values and code annotations, in such a way that each transition $t \in T$ has an energy consumption value and may be associated with a task source code. Figure 6.1 depicts a TPNE model example.

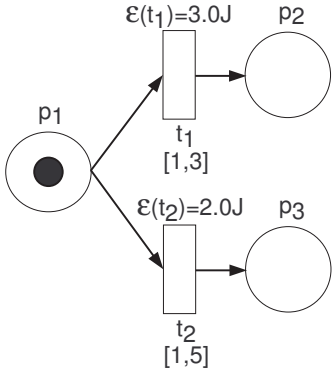


**Figure 6.1** TPNE example

As usual in Petri nets, a set of enabled transitions, at marking $m_i$, is denoted by: $ET(m_i) = \{t \in T \mid m_i(p_j) \geq W(p_j, t),\ \forall p_j \in P\}$. The time elapsed, since the respective transition enabling, is represented by a clock vector $c \in (\mathbb{N} \cup \{\#\})^{|T|}$, in which $\#$ represents the null value for disabled transitions. Besides, associated to each enabled transition, there is a dynamic firing interval $I_D(t) = [DLB(t), DUB(t)]$, which is dynamically modified whenever the respective clock variable $c(t)$ is incremented, and $t$ does not fire. $DLB(t)$ is the Dynamic Lower Bound, and $DUB(t)$ is the Dynamic Upper Bound. The dynamic firing interval is computed in the following way: $DLB(t) = max(0, EFT(t) - c(t))$, $DUB(t) = LFT(t) - c(t)$.

**Definition 6.2** (States)**.** *Let* $\mathcal{N}_\mathcal{E}$ *be a time Petri net extended with energy consumption values and code,* $M \subseteq P \times \mathbb{N}$ *be the set of reachable markings (e.g., all possible markings) of* $\mathcal{N}_\mathcal{E}$*,* $C \subseteq (\mathbb{N} \cup \{\#\})^{|T|}$ *be the set of clock vectors, and* $E \subseteq \mathbb{R}_+ \cup \{0\}$ *be the set of*

*accumulated energy consumptions. The set of states S of* $\mathcal{N}_\mathcal{E}$ *is given by* $S \subseteq (M \times C \times E)$*, that is, a state is defined by a marking, the respective clock vector, and the accumulated energy consumption from the initial state up to this state.*

In the adopted Petri net extension, a state ($s \in S$) is composed of a marking ($m \in M$), the clocks of each transition ($c \in C$), and the accumulated energy consumption ($e \in E$) from the initial state up to this state ($s$). Considering the TPNE model in Figure 6.2, the initial state is $s_0 = (m_0 = \{(p_1, 1),(p_2, 0),(p_3, 0)\}, c_0 = [0, 0], e_0 = 0)$.

**Definition 6.3** (Firable Transitions)**.** *The set of firable transitions at state* $s \in S$ *is defined by:* $FT(s, e_{max}) = \{t_i \in ET(m) |\ (e+\mathcal{E}(t_i) \leq e_{max}) \wedge (DLB(t_i) \leq \min(DUB(t_k))), \forall t_k \in ET(m)\}$*.* $e_{max}$ *is the system energy constraint (Chapter 5 - Section 5.2.5) and* $e$ *is the accumulated energy consumption from the initial state up to state* $s$*.*

Note that, in addition to timing constraints, a transition is only firable if the respective firing does not surpass the system energy constraint ($e_{max}$). Assuming $e_{max} = 2.5$J, only $t_2$ is firable in the net depicted in Figure 6.1. Moreover, a firable transition $t$ at state $s$ can only fire in the firing domain, which is denoted by $FD_s(t) = [DLB(t), \min(DUB(t_k))], \forall t_k \in ET(m)$.

**Definition 6.4** (Reachable States)**.** *Let* $\mathcal{N}_\mathcal{E}$ *be a TPNE and* $s_i = (m_i, c_i, e_i)$ *a reachable state.* $s_j = \texttt{fire}(s_i, (t, \theta))$ *denotes that firing a transition* $t \in FT(s_i, e_{max})$ *at time* $\theta \in FD_{s_i}(t)$ *from the state* $s_i$*, the reached state* $s_j = (m_j, c_j, e_j)$ *is obtained from:*

- $\forall p \in P,\ m_j(p) = m_i(p) - W(p, t) + W(t, p)$*, as usual in Petri nets;*
- $e_j = e_i + \mathcal{E}(t)$*;*
- $\forall t_l \notin ET(m_j), c_j(t_l) = \#$*;*
- $\forall t_k \in ET(m_j)$,

$$c_j(t_k) = \begin{cases} 0, & if (t_k = t) \\ 0, & if (t_k \in ET(m_j) - ET(m_i)) \\ c_i(t_k) + \theta, & else \end{cases}$$

Differently from time Petri nets, TPNE considers the energy consumption due to the firing of a transition for reaching a new state. For a better understanding, let us assume the transition $t_2$ firing at state $s_0$, in which $s_0 = (m_0 = \{(p_1, 1), (p_2, 0)(p_3, 0)\}, c_0 = [0, 0], e_0 = 0)$, $\theta$=1 and $e_{max} = 2.5$J. The new reached state is $s_1 = (m_1 = \{(p_1, 0),(p_2, 0), (p_3, 1)\}, c_1 = [\#, \#], e_1 = 2.0)$ (see Figure 6.2), such that: (i) $m_1(p_1) = 1 - 1 + 0$, $m_1(p_2) = 0 - 0 + 0$, $m_1(p_3) = 0 - 0 + 1$; (ii) $e_1 = 0 + 2.5$; and (iii) $c_1(t_1) = c_1(t_2) = \#$.

**Definition 6.5** (TLTS)**.** *A timed labeled transition system (TLTS) is a quadruple* $\mathcal{L}= (S, \Sigma, \rightarrow, s_0)$*, where* $S$ *is a finite set of discrete states,* $\Sigma$ *is an alphabet of labels representing actions,* $\rightarrow \subseteq S \times \Sigma \times S$ *is the transition relation, and* $s_0 \in S$ *is the initial state.*

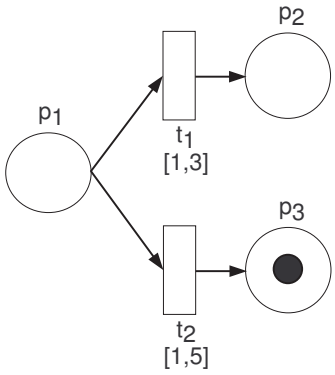


**Figure 6.2** TPNE after firing $t2$

The TPNE semantics is defined by associating a TLTS $\mathcal{L}_{\mathcal{N}_\mathcal{E}} = (S, \Sigma, \rightarrow, s_0)$, where (i) $S$ is the set of states of TPNE $\mathcal{N}_\mathcal{E}$; (ii) $\Sigma \subseteq (T \times \mathbb{N})$ is a set of labels $(t, \theta)$ corresponding to the transition $t$ firing at time $\theta$ in the firing interval $FD_s(t)$, $\forall s \in S$; (iii) $\rightarrow \subseteq S \times \Sigma \times S$ is the state transition relation; and (iv) $s_0$ is the initial state of $\mathcal{N}_\mathcal{E}$. Considering the firing of transition $t_2$ in Figure 6.2, the TLTS is $s_0 \xrightarrow{(t_1,1)} s_1$.

**Definition 6.6** (Feasible Firing Schedule)**.** *Let $\mathcal{L}_{\mathcal{N}_\mathcal{E}}$ be a timed labeled transition system of a extended time Petri net $\mathcal{N}_\mathcal{E}$, $s_0$ its initial state, $s_n = (m_n, c_n, e_n)$ a final state, and $m_n = M^F$ is the desired final marking.*

$$s_0 \xrightarrow{(t_0,\theta_0)} s_1 \xrightarrow{(t_1,\theta_1)} s_2 \rightarrow \ldots \rightarrow s_{n-1} \xrightarrow{(t_k,\theta_{n-1})} s_n$$

*is defined as a feasible firing schedule, such that $s_{i+1} = \texttt{fire}(s_i, (t_k, \theta_i))$, $i \geq 0$, $t_k \in FT(s_i, e_{max})$, and $\theta_i \in FD_{s_i}(t_k)$.*

The system modeling of the proposed methodology guarantees that the final marking $M^F$ (see Section 6.3.9) is well-known since it is explicitly specified. Assuming that a token in place $p_3$ (Figure 6.2) represents the desired final marking, $s_0 \xrightarrow{(t_1,1)} s_1$ is a feasible firing schedule (or just feasible schedule).

## 6.2 SCHEDULING PERIOD

The proposed modeling approach has been conceived for automatic pre-runtime schedule generation, where the *schedule period* ($P_S$) corresponds to the least common multiple (LCM) of all tasks' periods. Within this period, several task instances (of the same task) might be carried out, in which $\mathcal{S}(\tau_i) = P_S/p_i$ gives the number of instances for each task $\tau_i$. For the $n$th instance of task $\tau_i$ ($1 \leq n \leq \mathcal{S}(\tau_i)$), the release time is defined by $r_i^n = r_i + p_i \times (n-1)$ and the deadline is denoted by $d_i^n = d_i + p_i \times (n-1)$.

For a better understanding, consider the following task set $\mathcal{T} = \{\tau_1 = (0, 0, 100 \times 10^6, 3, 3), \tau_2 = (0, 2, 50 \times 10^6, 4, 4)\}$. As presented in Chapter 5, each task is represented by a tuple $\tau_i = (ph_i, r_i, c_i, d_i, p_i)$, in which $ph_i$ is the initial phase; $r_i$ is the release time;

$c_i$ is the worst-case execution cycles (WCEC) required for executing task $\tau_i$; $d_i$ is the deadline; and $p_i$ is the period. For this specification, the LCM is equal to 12, which points out the existence of 7 task instances ($\mathcal{S}(\tau_1) = 4$ and $\mathcal{S}(\tau_2) = 3$). Thus, the timing constraints of each instance need to be delineated and they are depicted in Table 6.1.

**Table 6.1** Timing constraints for each task instance

| | $\tau_1^1$ | $\tau_1^2$ | $\tau_1^3$ | $\tau_1^4$ | $\tau_2^1$ | $\tau_2^2$ | $\tau_2^3$ |
|---|---|---|---|---|---|---|---|
| r | 0 | 3 | 6 | 9 | 2 | 6 | 10 |
| c | $100 \times 10^6$ | $100 \times 10^6$ | $100 \times 10^6$ | $100 \times 10^6$ | $50 \times 10^6$ | $50 \times 10^6$ | $50 \times 10^6$ |
| d | 3 | 6 | 9 | 12 | 4 | 8 | 12 |
| p | 12 | 12 | 12 | 12 | 12 | 12 | 12 |

## 6.3 BASIC BUILDING BLOCKS

The proposed method adopts a bottom-up approach, in which a set of composition rules are considered for combining basic building block models. These building blocks represent each aspect of a hard real-time system, more specifically, the tasks' timing constraints (e.g., release time), intertask relations, overheads, the energy consumption during a task computation as well as the processor availability. Such approach generates a time Petri net model from the system specification, so that tools can be adopted to automate the modeling and scheduling activities.

In the proposed modeling process, the basic building blocks are: (i) Fork; (ii) Periodic Task Arrival; (iii) Voltage Selection; (iv) Non-preemptive Task Structure; (v) Preemptive Task Structure; (vi) Non-preemptive Task Structure with 2 Voltages; (vii) Preemptive Task Structure with 2 Voltages; (viii) Deadline Checking; (ix) Task Instance Conclusion; (x) Join; (xi) Preemptive Task Structure with Overheads; and (xii) Preemptive Task Structure with 2 Voltages and Overheads. Without loss of generality, the building blocks are combined via place merging and the places with dashed lines represent the connections with other building blocks. Besides, since all basic building block models are structurally conservative and structurally bounded, all generated models are also assured to be bounded structurally conservative and structurally bounded (a proof is presented in later sections).

Before presenting the building blocks, it is important to state that each place and transition of each basic model have a suffix *spec* in the respective name in order to indicate for which specification the block is being instantiated. This suffix is adopted to avoid instances of building blocks of the same type with elements (e.g., transitions) having equal names but with different constraints (e.g., computation time). This restriction is of utmost importance during the generation of TPNE models using the proposed composition rules (Section 7.1). However, to avoid long names, the suffix is omitted in the building blocks, except for the fork and join blocks. As follows, the building blocks are presented.

### 6.3.1 Fork Block

Supposing that the system has $n$ tasks, the fork block (Figure 6.3) is responsible for starting all tasks in the system. This block models the creation of $n$ concurrent tasks as well as it represents the initial marking. The fork block is modeled by a TPNE $N_f = (P_f, T_f, F_f, W_f, M_{0_f}, I_f, \mathcal{E}_f, \mathcal{CS}_f)$, in which:

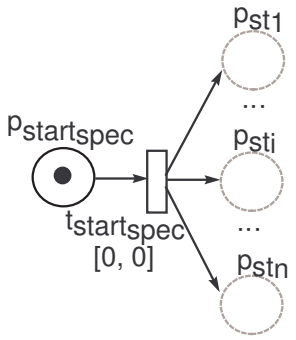


**Figure 6.3** Fork block

♢ $P_f = \{p_{start_{spec}}, p_{st_1}, \cdots, p_{st_i}, \cdots, p_{st_n}\}$. These places model the following situations:

- $p_{start_{spec}}$: waiting for the start of the system. *spec* is the specification name;
- $p_{st_i}$: starting of the $i_{th}$ task, $1 \leq i \leq n$.

♢ $T_f = \{t_{start_{spec}}\}$. This transition models the following action:

$t_{start_{spec}}$: starting of all tasks of the system. *spec* is the specification name.

♢ Pre and post-conditions of the transition are (flow relation $F_f$):

- $\bullet t_{start_{spec}} = \{p_{start_{spec}}\}$;
- $t_{start_{spec}}\bullet = \{p_{st_1}, \cdots, p_{st_i}, \cdots, p_{st_n}\}$.

♢ $\forall (x, y) \in F_f, W_f(x, y) = 1$;

♢ $M_{0_f}(p_x) = 0, \forall p_x \in P, p_x \neq p_{start_{spec}}; M_{0_f}(p_{start_{spec}}) = 1$;

♢ $I_f(t_{start_{spec}}) = [0, 0]$;

♢ $\mathcal{E}_f(t_{start_{spec}}) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_f$ as a set, $\mathcal{CS}_f = \emptyset$.

The fork block is covered by $n$ basic P-invariants, which are depicted as follows:

$$\diamondsuit\ \mathcal{I}_{(f)(1)} = \overset{\begin{matrix} p_{start_{spec}} & p_{st_1} & \cdots & p_{st_i} & \cdots & p_{st_n} \end{matrix}}{\begin{bmatrix} st_1 & st_1 & \cdots & 0 & \cdots & 0 \end{bmatrix}^T}$$
$$\vdots$$
$$\diamondsuit\ \mathcal{I}_{(f)(i)} = \overset{\begin{matrix} p_{start_{spec}} & p_{st_1} & \cdots & p_{st_i} & \cdots & p_{st_n} \end{matrix}}{\begin{bmatrix} st_i & 0 & \cdots & st_i & \cdots & 0 \end{bmatrix}^T}$$
$$\vdots$$
$$\diamondsuit\ \mathcal{I}_{(f)(n)} = \overset{\begin{matrix} p_{start_{spec}} & p_{st_1} & \cdots & p_{st_i} & \cdots & p_{st_n} \end{matrix}}{\begin{bmatrix} st_n & 0 & \cdots & 0 & \cdots & st_n \end{bmatrix}^T}$$

in which $st_1, \cdots, st_i \cdots, st_n \in \mathbb{N}^*$. Using these basic invariants, a P-invariant covering all places ($\mathcal{I}_{(f)}$) can be obtained:

$\mathcal{I}_{(f)} = \mathcal{I}_{(f)(1)} + \cdots + \mathcal{I}_{(f)(i)} + \cdots + \mathcal{I}_{(f)(n)}$

$$\mathcal{I}_{(f)} = \overset{\begin{matrix} p_{start_{spec}} & & p_{st_1} & \cdots & p_{st_i} & \cdots & p_{st_n} \end{matrix}}{\begin{bmatrix} st_1 + \cdots + st_i + \cdots + st_n & st_1 & \cdots & st_i & \cdots & st_n \end{bmatrix}^T}$$

Since $\mathcal{I}_{(f)}^T \times A_f = 0$, in which $A_f$ is the respective incidence matrix, and $\mathcal{I}_{(f)} > 0$, the fork block is structurally conservative as well as structurally bounded.

### 6.3.2 Periodic Task Arrival Block

This block (Figure 6.4) models the periodic invocation for all task instances in the schedule period ($P_S$). A transition $t_{ph_i}$ models the initial phase of the task first instance. Similarly, transition $t_{a_i}$ models the periodic arrival (after the initial phase) for the remaining instances and transition $t_{r_i}$ represents a task instance release. The reader should note the weight ($\alpha_i = \mathcal{S}(\tau_i) - 1$) of the arc ($t_{ph_i}, p_{wa_i}$), which models the invocation of all remaining instances after the first task instance. The timing intervals of transitions $t_{ph_i}$ and $t_{a_i}$ are the timing constraints depicted in the specification, in this case, $ph_i$ (phase) and $p_i$ (period). Considering transition $t_{r_i}$, the timing interval is $[r_i, d_i - C_{min}]$, where $r_i$ is the release time, $d_i$ is the deadline constraint, and $C_{min}$ is the computation time of task $\tau_i$ at the highest voltage level (and the respective maximum CPU frequency).

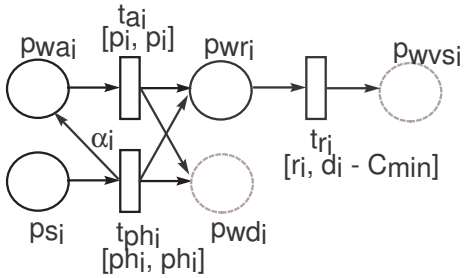


**Figure 6.4** Arrival block

The building block periodic task arrival is a TPN $N_a = (P_a, T_a, F_a, W_a, M_{0_a}, I_a, \mathcal{E}_a, \mathcal{CS}_a)$, such that:

♢ $P_a = \{p_{st_i}, p_{wa_i}, p_{wd_i}, p_{wr_i}, p_{wvs_i}\}$. These places model the following conditions:

- $p_{st_i}$: starting of task;
- $p_{wa_i}$: waiting for the arrival of another task instance;
- $p_{wd_i}$: waiting for deadline missing;
- $p_{wr_i}$: waiting for release time;
- $p_{wr_i}$: waiting for voltage selection.

♢ $T_a = \{t_{a_i}, t_{ph_i}, t_{r_i}\}$. These transitions model the following actions:

- $t_{a_i}$: arriving of a new task instance;
- $t_{ph_i}$: elapsing of the task initial phase; and
- $t_{r_i}$: releasing of a new task instance for execution.

♢ Pre and post-conditions of the transitions are (flow relation $F_a$):

- $\bullet t_{a_i} = \{p_{wa_i}\}$;
- $t_{a_i}\bullet = \{p_{wr_i}, p_{wd_i}\}$;
- $\bullet t_{ph_i} = \{p_{st_i}\}$;
- $t_{ph_i}\bullet = \{p_{wa_i}, p_{wr_i}, p_{wd_i}\}$;
- $\bullet t_{r_i} = \{p_{wr_i}\}$;
- $t_{r_i}\bullet = \{p_{wvs_i}\}$.

♢ $\forall (x,y) \in F_a, W_a(x,y) = \begin{cases} \alpha_i \in \mathbb{N}, & if\ (x,y) = (t_{ph_i}, p_{wa_i}) \\ 1, & otherwise. \end{cases}$

♢ $\forall p \in P, M_{0_a}(p) = 0$

♢ $I_a(t_{ph_i}) = [ph_i, ph_i]$; $I_a(t_{a_i}) = [p_i, p_i]$; and $I_a(t_{r_i}) = [r_i, C_{min} - d_i]$, such that $C_{min} = \lceil c_i / f_{max} \rceil$ and $f_{max} = max(vff(v)), \forall v \in \mathcal{V}_{cpu}$;

♢ $\forall t \in T_a, \mathcal{E}_a(t) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_a$ as a set, $\mathcal{CS}_a = \emptyset$.

The Periodic Task Arrival block is covered by 2 basic P-invariants, which are presented as follows:

♢ $\mathcal{I}_{(a)(1)} = \overset{\quad p_{st_i} \quad p_{wa_i} \quad p_{wd_i} \quad p_{wr_i} \quad p_{wvs_i}}{\left[ \begin{array}{ccccc} (\alpha_i+1)(wd_i) & wd_i & wd_i & 0 & 0 \end{array} \right]^T}$;

♢ $\mathcal{I}_{(a)(2)} = \overset{\quad p_{st_i} \quad p_{wa_i} \quad p_{wd_i} \quad p_{wr_i} \quad p_{wvs_i}}{\left[ \begin{array}{ccccc} (\alpha_i+1)(wvs_i) & wvs_i & 0 & wvs_i & wvs_i \end{array} \right]^T}$.

in which $wvs_i, wd_i \in \mathbb{N}^*$. A P-invariant covering all places ($\mathcal{I}_{(a)}$) can be obtained in the following way:

$\mathcal{I}_{(a)} = \mathcal{I}_{(a)(1)} + \mathcal{I}_{(a)(2)}$

$$\mathcal{I}_{(a)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i+1)(wvs_i+wd_i)} & \overset{p_{wa_i}}{wvs_i+wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wvs_i}}{wvs_i} \end{bmatrix}^T$$

Since $\mathcal{I}_{(a)}^T \times A_a = 0$, in which $A_a$ is the respective incidence matrix, and $\mathcal{I}_{(a)} > 0$, the periodic task arrival block is structurally conservative as well as structurally bounded.

### 6.3.3 Voltage Selection Block

For each available voltage level, this block (Figure 6.5) represents every possible voltage selection for executing a task $\tau_i$ ($\mathcal{V}_{total_i} = \mathcal{V}_{CPU} \cup \mathcal{V}_{ideal_i}$). More specifically, this block includes not only the voltage levels (and the respective maximum frequencies) available on the CPU ($\mathcal{V}_{CPU}$), but also other levels that can be simulated ($\mathcal{V}_{ideal_i}$).

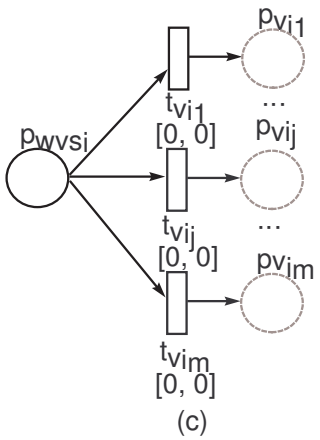


**Figure 6.5** Voltage selection block

The voltage selection block is modeled by a TPN $N_v = (P_v, T_v, F_v, W_v, M_{0_v}, I_v, \mathcal{E}_v, \mathcal{CS}_v)$, such that:

♢ $P_v = \{p_{wvs_i}, p_{v_{i_1}}, \cdots, p_{v_{i_j}}, \cdots, p_{v_{i_m}}\}$. These places model the following situations:

- $p_{wvs_i}$: waiting for voltage selection;
- $p_{v_{i_j}}$: voltage level $v_j$ selected, such that $1 \leq j \leq m$ and $m = |\mathcal{V}_{total_i}|$.

♢ $T_v = \{t_{v_{i_1}}, \cdots, t_{v_{i_j}}, \cdots, t_{v_{i_m}}\}$. These transitions model the following action:

$t_{v_{i_j}}$: selection of voltage level $v_j$, such that $1 \leq j \leq m$ and $m = |\mathcal{V}_{total_i}|$.

♢ Pre and post-conditions of the transitions are (flow relation $F_v$):

- $\forall t_{v_{i_j}} \in T_f, {\bullet t_{v_{i_j}}} = \{p_{wvs_i}\}$;

– $t_{v_{i_1}}\bullet = \{p_{v_{i_1}}\}; \cdots; t_{v_{i_j}}\bullet = \{p_{v_{i_j}}\}; \cdots; t_{v_{i_m}}\bullet = \{p_{v_{i_m}}\}$.

♢ $\forall (x,y) \in F_v, W_v(x,y) = 1$;

♢ $\forall p \in P_v, M_{0_v}(p) = 0$;

♢ $\forall t \in T_v, I_v(t) = [0,0]$;

♢ $\forall t \in T_v, \mathcal{E}_v(t) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_v$ as a set, $\mathcal{CS}_v = \emptyset$.

The voltage selection block is covered by one basic P-invariant:

$$\mathcal{I}_{(v)} = \begin{bmatrix} p_{wvs_i} & p_{v_{i_1}} & \cdots & p_{v_{i_j}} & \cdots & p_{v_{i_m}} \\ wvs_i & wvs_i & \cdots & wvs_i & \cdots & wvs_i \end{bmatrix}^T$$

in which $wvs_i \in \mathbb{N}^*$. As $\mathcal{I}_{(v)}$ covers all places ($\mathcal{I}_{(v)} > 0$), and $\mathcal{I}_{(v)}^T \times A_v = 0$ ($A_v$ is the incidence matrix), the voltage selection block is structurally conservative as well as structurally bounded.

### 6.3.4 Non-Preemptive Task Structure Block

Considering a non-preemptive scheduling method, the processor is just released after the entire computation is finished. This block models (Figure 6.6) a non-preemptive task computation adopting a specific voltage. In this block, processor granting and task computation are represented by transition $t_{g_{i_j}}$ and $t_{c_{i_j}}$, respectively. Only after the entire task computation, the processor is released by transition $t_{c_{i_j}}$. Assuming a voltage $v_j \in \mathcal{V}_{cpu}$ and the respective maximum frequency $f_j = vff(v_j)$, task computation time ($C_{i_j}$) can be obtained by $C_{i_j} = \lceil c_i/f_j \rceil$, where $c_i$ is the task ($\tau_i$) WCEC. Figure 6.6 shows that time interval of computation transition $t_{c_{i_j}}$ has bounds equal to the task computation time at a specific voltage ($[C_{i_j}, C_{i_j}]$). The timing intervals for transition $t_{vs_{i_j}}$, $t_{g_{i_j}}$ and $t_{fv_{i_j}}$ are equal to [0,0]. Furthermore, computation transitions have energy consumption values greater than zero.

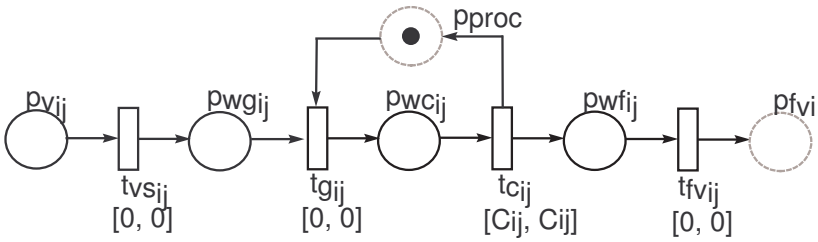


**Figure 6.6** Non-preemptive task structure block

Formally, the building block non-preemptive task structure is a TPN $\mathcal{N}_{np} = (P_{np}, T_{np}, F_{np}, W_{np}, M_{0_{np}}, I_{np}, \mathcal{E}_{np}, \mathcal{CS}_{np})$, such that:

♢ $P_{np} = \{p_{v_{i_j}}, p_{wg_{i_j}}, p_{wc_{i_j}}, p_{wf_{i_j}}, p_{fv_i}, p_{proc}\}$. These places model the following conditions:

- $p_{v_{i_j}}$: voltage level $v_j$ selected;
- $p_{wg_{i_j}}$: waiting for processor granting;
- $p_{wc_{i_j}}$: waiting for task computation;
- $p_{wf_{i_j}}$: waiting for task instance conclusion;
- $p_{fv_i}$: conclusion of a task instance;
- $p_{proc}$: processor.

♢ $T_{np} = \{t_{v_{i_j}}, t_{g_{i_j}}, t_{c_{i_j}}, t_{fv_{i_j}}\}$. These transitions model the following actions:

- $t_{vs_{i_j}}$: starting task execution at voltage $v_j$;
- $t_{g_{i_j}}$: processor granting;
- $t_{c_{i_j}}$: executing task;
- $t_{fv_{i_j}}$: finalizing task execution at voltage $v_j$.

♢ Pre and post-conditions of the transitions are (flow relation $F_{np}$):

- $\bullet t_{v_{i_j}} = \{p_{v_{i_j}}\}; \quad t_{v_{i_j}}\bullet = \{p_{wg_{i_j}}\};$
- $\bullet t_{g_{i_j}} = \{p_{wg_{i_j}}, p_{proc}\}; \quad t_{g_{i_j}}\bullet = \{p_{wc_{i_j}}\};$
- $\bullet t_{c_{i_j}} = \{p_{wc_{i_j}}\}; \quad t_{c_{i_j}}\bullet = \{p_{fw_{i_j}}, p_{proc}\};$
- $\bullet t_{fv_{i_j}} = \{p_{wf_{i_j}}\}; \quad t_{fv_{i_j}}\bullet = \{p_{fv_i}\}.$

♢ $\forall (x, y) \in F_{np}, W_{np}(x, y) = 1$;

♢ $M_{0_{np}}(p_x) = 0, \forall p_x \in P_{np}, p_x \neq p_{proc}$; and $M_{0_{np}}(p_{proc}) = 1$;

♢ $I_{np}(t_x) = [0, 0], \forall t_x \in T_{np}, t_x \neq t_{c_{i_j}}$; and $I_{np}(t_{c_{i_j}}) = [C_{i_j}, C_{i_j}]$;

♢ $\mathcal{E}_{np}(t_x) = 0.0\text{J}, \forall t_x \in T_{np}, t_x \neq \{t_{c_{i_j}}\}$; and $\mathcal{E}_{np}(t_{c_{i_j}}) = vef(v_j) \times c_i$ ($vef$ is the voltage-energy function. See Chapter 5 - Section 5.2.4);

♢ $\mathcal{CS}_{np}(t_{c_{i_j}}) = code_i$, in which $code_i \in \mathcal{ST}$ ($\mathcal{ST}$ is the set of task source codes. See Chapter 5 - Section 5.2.1).

The non-preemptive task structure block is covered by 2 basic P-invariants:

♢ $\mathcal{I}_{(np)(1)} = \begin{bmatrix} p_{v_{i_j}} & p_{wg_{i_j}} & p_{wc_{i_j}} & p_{wf_{i_j}} & p_{fv_i} & p_{proc} \\ 0 & 0 & proc & 0 & 0 & proc \end{bmatrix}^T$;

♢ $\mathcal{I}_{(np)(2)} = \begin{bmatrix} p_{v_{i_j}} & p_{wg_{i_j}} & p_{wc_{i_j}} & p_{wf_{i_j}} & p_{fv_i} & p_{proc} \\ wg_{i_j} & wg_{i_j} & wg_{i_j} & wg_{i_j} & wg_{i_j} & 0 \end{bmatrix}^T$.

where $wg_{i_j}, proc \in \mathbb{N}^*$. A P-invariant covering all places is presented below:

$\mathcal{I}_{(np)} = \mathcal{I}_{(np)(1)} + \mathcal{I}_{(np)(2)}$

$$\mathcal{I}_{(np)} = \begin{bmatrix} p_{v_{i_j}} & p_{wg_{i_j}} & p_{wc_{i_j}} & p_{wf_{i_j}} & p_{fv_i} & p_{proc} \\ wg_{i_j} & wg_{i_j} & wg_{i_j} + proc & wg_{i_j} & wg_{i_j} & proc \end{bmatrix}^T$$

As $\mathcal{I}_{(np)}^T \times A_{np} = 0$, in which $A_{np}$ is the respective incidence matrix, and $\mathcal{I}_{(np)} > 0$, the non-preemptive task structure block is structurally conservative and structurally bounded.

### 6.3.5 Preemptive Task Structure Block

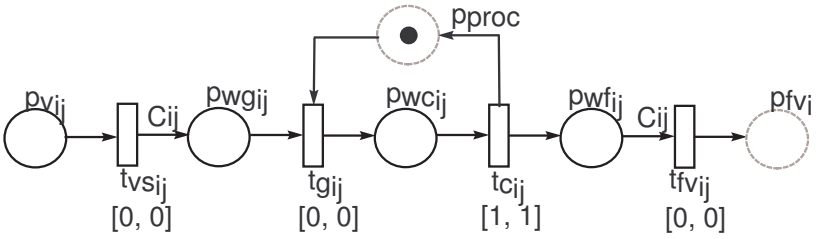


**Figure 6.7** Preemptive task structure block

In this particular scheduling method (Figure 6.7), tasks are implicitly split into subtasks, in which the computation time of each subtask is exactly equal to one task time unit (TTU). This method allows running other conflicting tasks, in this case, meaning that one task may preempt another task. This is modeled by the time interval of computation transitions ([1,1]), and the entire computation is modeled through the arc weights. Considering $C_{i_j}$ the task computation time at a specific voltage, $C_{i_j}$ tokens are stored in place $p_{wg_{i_j}}$, and the same amount of tokens in place $p_{wf_{i_j}}$ is needed for firing transition $t_{fv_{i_j}}$. In the same way as the non-preemptive task structure block, processor granting and task computation are represented by transition $t_{g_{i_j}}$ and $t_{c_{i_j}}$, respectively. However, the processor is released by transition $t_{c_{i_j}}$ just after the execution of one task time unit related to the computation time.

The preemptive task structure block is a TPN $\mathcal{N}_p = (P_p, T_p, F_p, W_p, M_{0_p}, I_p, \mathcal{E}_p, \mathcal{CS}_p)$ such that:

- $P_p = \{p_{v_{i_j}}, p_{wg_{i_j}}, p_{wc_{i_j}}, p_{wf_{i_j}}, p_{f_{i_j}}, p_{proc}\}$. These places model the following conditions:
  - $p_{v_{i_j}}$: voltage level $v_j$ selected;
  - $p_{wg_i}$: waiting for processor granting;
  - $p_{wc_i}$: waiting for task computation;
  - $p_{wf_i}$: waiting for task instance conclusion;
  - $p_{fv_i}$: conclusion of a task instance;
  - $p_{proc}$: processor.

♢ $T_p = \{t_{v_{i_j}}, t_{g_{i_j}}, t_{c_{i_j}}, t_{fv_{i_j}}\}$. These transitions model the following actions:

- $t_{vs_{i_j}}$: starting task execution at voltage $v_j$;
- $t_{g_{i_j}}$: processor granting;
- $t_{c_{i_j}}$: executing one TTU; and
- $t_{fv_{i_j}}$: finalizing task execution at voltage $v_j$.

♢ Pre and post-conditions of the transitions are (flow relation $F_p$):

- $\bullet t_{vs_{i_j}} = \{p_{v_{i_j}}\}$; $t_{vs_{i_j}}\bullet = \{p_{wg_{i_j}}\}$;
- $\bullet t_{g_{i_j}} = \{p_{wg_i}, p_{proc}\}$; $t_{g_{i_j}}\bullet = \{p_{wc_{i_j}}\}$;
- $\bullet t_{c_{i_j}} = \{p_{wc_i}\}$; $t_{c_{i_j}}\bullet = \{p_{wf_i}, p_{proc}\}$;
- $\bullet t_{fv_{i_j}} = \{p_{wf_{i_j}}\}$; $t_{fv_{i_j}}\bullet = \{p_{fv_i}\}$.

♢ $\forall (x,y) \in F_p, W_p(x,y) = \begin{cases} C_{i_j} & if\ (x,y) = (t_{c_{i_j}}, p_{wf_{i_j}}) \vee (x,y) = (t_{vs_{i_j}}, p_{wg_{i_j}}) \\ 1 & otherwise \end{cases}$

♢ $M_{0_p}(p_x) = 0, \forall p_x \in P_p, p_x \neq \{p_{proc}\}$; and $M_{0_p}(p_{proc}) = 1$;

♢ $I_p(t_x) = [0,0], \forall t_x \in T_p, t_x \neq \{t_{c_{i_j}}\}$; and $I_p(t_{c_{i_j}}) = [1,1]$;

♢ $\mathcal{E}_p(t_x) = 0.0\text{J}, \forall t_x \in T_p, t_x \neq \{t_{c_{i_j}}\}$; and $\mathcal{E}_p(t_{c_{i_j}}) = (vef(v_j) \times c_i)/C_{i_j}$. $vef$ is the voltage-energy function. See Chapter 5 - Section 5.2.4;

♢ $\mathcal{CS}_p(t_{c_{i_j}}) = code_i$, in which $code_i \in \mathcal{ST}$.

The preemptive task structure block is covered by 2 basic P-invariants, which are described as follows:

♢ $\mathcal{I}_{(p)(1)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{0} & \overset{p_{wg_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{proc} & \overset{p_{wf_{i_j}}}{0} & \overset{p_{fv_i}}{0} & \overset{p_{proc}}{proc} \end{bmatrix}^T$;

♢ $\mathcal{I}_{(p)(2)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} & \overset{p_{proc}}{0} \end{bmatrix}^T$.

in which $proc, wg_{i_j} \in \mathbb{N}^*$. Using these basic invariants, a P-invariant covering all places ($\mathcal{I}_{(p)}$) can be obtained in the following way:

$\mathcal{I}_{(p)} = \mathcal{I}_{(p)(1)} + \mathcal{I}_{(p)(2)}$

$\mathcal{I}_{(p)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$

Since $\mathcal{I}_{(p)} > 0$ and $\mathcal{I}_{(p)}^T \times A_p = 0$ ($A_p$ is the incidence matrix), the preemptive task structure block is structurally conservative as well as structurally bounded.

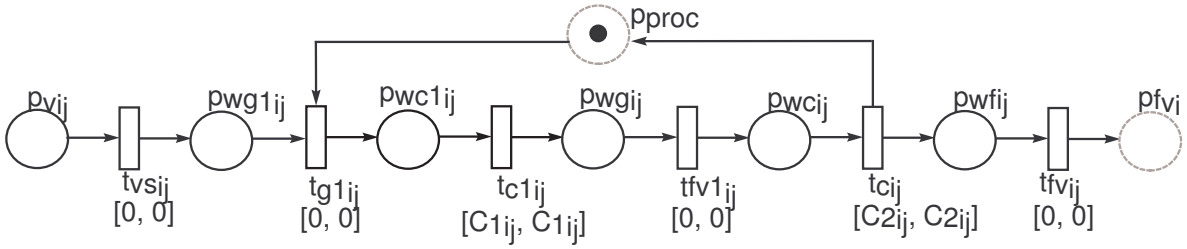


**Figure 6.8** Non-preemptive task structure block with 2 voltages

### 6.3.6 Non-Preemptive and Preemptive Task Structure with 2 Voltages Blocks

If the CPU provides a small number of discrete voltage levels (and the respective maximum frequencies) and an ideal voltage is not available ($v_{ideal} \notin \mathcal{V}_{cpu} \wedge v_{ideal} \in \mathcal{V}_{ideal_i}$), the two immediate neighbor voltages ($v_{ideal_L}$,$v_{ideal_H} \in \mathcal{V}_{cpu}$) to the ideal one can be adopted for reducing energy consumption [12]. For a better understanding, a task may be divided in two parts. The first part is executed at the immediate higher voltage in relation to the ideal one ($v_{ideal_H} = min\{v \in \mathcal{V}_{cpu} | v_{ideal} < v\}$), and the second part is executed at the immediate lower voltage ($v_{ideal_L} = max\{v \in \mathcal{V}_{cpu} | v < v_{ideal}\}$). When the ideal voltage is smaller than any available voltage level, the smallest CPU voltage level is adopted. However, when the ideal voltage is higher than any available voltage level, the task instance cannot be scheduled.

The proposed modeling approach allows the modeling of a task instance executing at two different voltage levels (and the respective maximum CPU frequencies) considering non-preemptive (Figure 6.8) and preemptive (Figure 6.9) executions. Assume that $f_{ideal}$ is the maximum operating frequency related to the ideal voltage $v_{ideal}$. $C_{1_{i_j}} = c_1/f_{ideal_H}$ represents the computation time of the first part of the task executing at $v_{ideal_H}$, and $C_{2_{i_j}} = c_2/f_{ideal_L}$ represents the computation time of the second part of the task executing at $v_{ideal_L}$, such that: (i) $c_i = c_1 + c_2$; and (ii) $c_i/f_{ideal} = C_{1_{i_j}} + C_{2_{i_j}}$. Without loss of generality, these block resemble the task structure blocks presented previously and the formal definitions are presented as follows.

The building block non-preemptive task structure with 2 voltages is a TPN $\mathcal{N}_{np2v} = (P_{np2v}, T_{np2v}, F_{np2v}, W_{np2v}, M_{0_{np2v}}, I_{np2v}, \mathcal{E}_{np2v}, \mathcal{CS}_{np2v})$, such that:

- ♢ $P_{np2v} = \{p_{v_{i_j}}, p_{wg1_{i_j}}, p_{wc1_{i_j}}, p_{wg_{i_j}}, p_{wc_{i_j}}, p_{wf_{i_j}}, p_{fv_i}, p_{proc}\}$. These places model the following conditions:
  - $p_{vs_{i_j}}$: voltage level $v_j$ selected;
  - $p_{wg1_{i_j}}$: waiting for the processor granting considering the execution of the first part;
  - $p_{wc1_{i_j}}$: waiting for task computation considering the execution of the first part;
  - $p_{wg_{i_j}}$: waiting for the processor granting considering the execution of the second part. Indeed, this a virtual situation, since the processor was already

granted. Besides, this place has an additional purpose when modeling overheads that may occur during system execution. Section 6.3.10 provides the details;

- $p_{wc_{i_j}}$: waiting for task computation considering the execution of the second part;
- $p_{wf_{i_j}}$: waiting for task instance conclusion;
- $p_{fv_i}$: conclusion of a task instance;
- $p_{proc}$: processor.

♢ $T_{np2v} = \{t_{v_{i_j}}, t_{g1_{i_j}}, t_{c1_{i_j}}, t_{fv1_{i_j}}, t_{c_{i_j}}, t_{fv_{i_j}}\}$. These transitions model the following actions:

- $t_{vs_{i_j}}$: starting task execution at voltage $v_j$;
- $t_{g1_{i_j}}$: processor granting;
- $t_{c1_{i_j}}$: executing task first part;
- $t_{fv1_{i_j}}$: finalizing first part execution at voltage $v_{ideal_H}$;
- $t_{c_{i_j}}$: executing task second part ;
- $t_{fv_{i_j}}$: finalizing second part execution at voltage $v_{ideal_L}$.

♢ Pre and post-conditions of the transitions are (flow relation $F_{np2v}$):

- $\bullet t_{v_{i_j}} = \{p_{v_{i_j}}\}$; $t_{v_{i_j}}\bullet = \{p_{wg1_{i_j}}\}$
- $\bullet t_{g1_{i_j}} = \{p_{wg1_{i_j}}, p_{proc}\}$; $t_{g1_{i_j}}\bullet = \{p_{wc1_{i_j}}\}$;
- $\bullet t_{c1_{i_j}} = \{p_{wc1_{i_j}}\}$; $t_{c1_{i_j}}\bullet = \{p_{wg_{i_j}}\}$;
- $\bullet t_{fv1_{i_j}} = \{p_{wg_{i_j}}\}$; $t_{fv1_{i_j}}\bullet = \{p_{wc_{i_j}}\}$;
- $\bullet t_{c_{i_j}} = \{p_{wc_{i_j}}\}$; $t_{c1_{i_j}}\bullet = \{p_{wf_{i_j}, p_{proc}}\}$;
- $\bullet t_{fv_{i_j}} = \{p_{wf_{i_j}}\}$; $t_{fv_{i_j}}\bullet = \{p_{fv_i}\}$.

♢ $\forall (x, y) \in F_{np2v}, W_{np2v}(x, y) = 1$;

♢ $M_{0_{np2v}}(p_x) = 0, \forall p_x \in P_{np2v}, p_x \neq p_{proc}$; and $M_{0_{np2v}}(p_{proc}) = 1$;

♢ $I_{np2v}(t_x) = [0, 0], \forall t_x \in T_{np2v}, t_x \notin \{t_{c1_{i_j}}, t_{c_{i_j}}\}$;
$I_{np2v}(t_{c1_{i_j}}) = [C_{1_{i_j}}, C_{1_{i_j}}]$;
and $I_{np2v}(t_{c_{i_j}}) = [C_{2_{i_j}}, C_{2_{i_j}}]$;

♢ $\mathcal{E}_{np2v}(t_x) = 0.0\text{J}, \forall t_x \in T_{np2v}, t_x \notin \{t_{c1_{i_j}}, t_{c_{i_j}}\}$; $\mathcal{E}_{np2v}(t_{c1_{i_j}}) = (vef(v_{ideal_H}) \times c_1)/C_{1_{i_j}}$; and $\mathcal{E}_{np2v}(t_{c_{i_j}}) = (vef(v_{ideal_L}) \times c_2)/C_{2_{i_j}}$. $vef$ is the voltage-energy function. See Chapter 5 - Section 5.2.4;

♢ $\mathcal{CS}_{np2v}(t_{c1_{i_j}}) = \mathcal{CS}_{np2v}(t_{c_{i_j}}) = code_i$, in which $code_i \in \mathcal{ST}$.

Additionally, the non-preemptive task structure block with 2 voltages is covered by 2 basic P-invariants, which are depicted as follows:

$$\diamondsuit\ \mathcal{I}_{(np2v)(1)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{0} & \overset{p_{wg1_{i_j}}}{0} & \overset{p_{wc1_{i_j}}}{proc} & \overset{p_{wg_{i_j}}}{proc} & \overset{p_{wc_{i_j}}}{proc} & \overset{p_{wf_{i_j}}}{0} & \overset{p_{fv_i}}{0} & \overset{p_{proc}}{proc} \end{bmatrix}^T;$$

$$\diamondsuit\ \mathcal{I}_{(np2v)(2)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{wg1_{i_j}} & \overset{p_{wg_{i_j}}}{wg1_{i_j}} & \overset{p_{wc_{i_j}}}{wg1_{i_j}} & \overset{p_{wf_{i_j}}}{wg1_{i_j}} & \overset{p_{fv_i}}{wg1_{i_j}} & \overset{p_{proc}}{0} \end{bmatrix}^T.$$

in which $proc, wg1_{i_j} \in \mathbb{N}^*$. A P-invariant covering all places ($\mathcal{I}_{(f)}$) is presented as follows:
$\mathcal{I}_{(np2v)} = \mathcal{I}_{(np2v)(1)} + \mathcal{I}_{(np2v)(2)}$

$$\mathcal{I}_{(np2v)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{wg1_{i_j} + proc} & \overset{p_{wg_{i_j}}}{wg1_{i_j} + proc} & \overset{p_{wc_{i_j}}}{wg1_{i_j} + proc} & \overset{p_{wf_{i_j}}}{wg1_{i_j}} & \overset{p_{fv_i}}{wg1_{i_j}} \\ \overset{p_{proc}}{proc} \end{bmatrix}^T$$

As $\mathcal{I}_{(np2v)} > 0$ and $\mathcal{I}^T_{(np2v)} \times A_{np2v} = 0$ ($A_{np2v}$ is the incidence matrix), $\mathcal{N}_{(np2v)}$ is structurally conservative as well as structurally bounded.

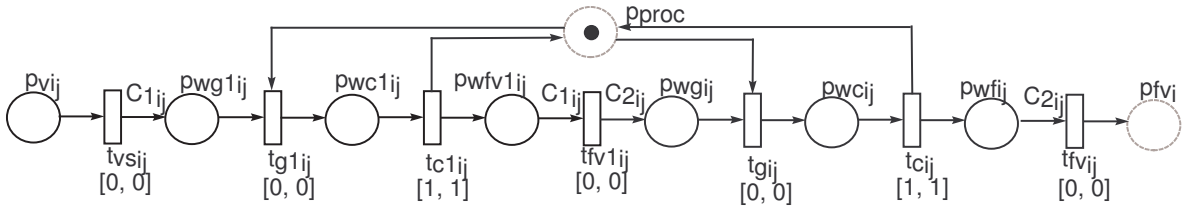


**Figure 6.9** Preemptive Task structure with 2 voltages block

The building block preemptive task structure with 2 voltages is a TPN $\mathcal{N}_{p2v} = (P_{p2v}, T_{p2v}, F_{p2v}, W_{p2v}, M_{0_{p2v}}, I_{p2v}, \mathcal{E}_{p2v}, \mathcal{CS}_{p2v})$, such that:

- $P_{p2v} = \{p_{v_{i_j}}, p_{wg1_{i_j}}, p_{wc1_{i_j}}, p_{wfv1_{i_j}}, p_{wg_{i_j}}, p_{wc_{i_j}}, p_{wf_{i_j}}, p_{fv_i}, p_{proc}\}$. These places model the following conditions:
  - $p_{v_{i_j}}$: voltage level $v_j$ selected;
  - $p_{wg1_{i_j}}$: waiting for the processor granting considering the execution of the first part;
  - $p_{wc1_{i_j}}$: waiting for task computation considering the execution of the first part;
  - $p_{wfv1_{i_j}}$: waiting for the conclusion of the first part;
  - $p_{wg_{i_j}}$: waiting for the processor granting considering the execution of the second part;
  - $p_{wc_{i_j}}$: waiting for task computation considering the execution of the second part;

- $p_{wf_{i_j}}$: waiting for task instance conclusion;
- $p_{fv_i}$: task instance conclusion;
- $p_{proc}$: processor.

♢ $T_{p2v} = \{t_{vs_{i_j}}, t_{g1_{i_j}}, t_{c1_{i_j}}, t_{fv1_{i_j}}, t_{g_{i_j}}, t_{c_{i_j}}, t_{fv_{i_j}}\}$. These transitions model the following actions:

- $t_{vs_{i_j}}$: starting task execution at voltage $v_j$;
- $t_{g1_{i_j}}$: processor granting;
- $t_{c1_{i_j}}$: executing the task first part;
- $t_{fv1_{i_j}}$: finalizing the first part execution at voltage $v_{ideal_H}$;
- $t_{g_{i_j}}$: processor granting;
- $t_{c_{i_j}}$: executing the task second part;
- $t_{fv_{i_j}}$: finalizing the second part execution at voltage $v_{ideal_L}$.

♢ Pre and post-conditions of the transitions are (flow relation $F_{p2v}$):

- $\bullet t_{vs_{i_j}} = \{p_{v_{i_j}}\}$; $t_{v_{i_j}} = \{p_{wg1_{i_j}}\}$;
- $\bullet t_{g1_{i_j}} = \{p_{wg1_{i_j}}, p_{proc}\}$; $t_{g1_{i_j}}\bullet = \{p_{wc1_{i_j}}\}$;
- $\bullet t_{c1_{i_j}} = \{p_{wc1_{i_j}}\}$; $t_{c1_{i_j}}\bullet = \{p_{wfv1_{i_j}}, p_{proc}\}$;
- $\bullet t_{fv1_{i_j}} = \{p_{wfv1_{i_j}}\}$; $t_{fv1_{i_j}}\bullet = \{p_{wg_{i_j}}\}$;
- $\bullet t_{g_{i_j}} = \{p_{wg_{i_j}}, p_{proc}\}$; $t_{g_{i_j}}\bullet = \{p_{wc1_{i_j}}\}$;
- $\bullet t_{c_{i_j}} = \{p_{wc_{i_j}}\}$; $t_{c1_{i_j}}\bullet = \{p_{wf_{i_j}, p_{proc}}\}$;
- $\bullet t_{fv_{i_j}} = \{p_{wf_{i_j}}\}$; $t_{fv_{i_j}}\bullet = \{p_{fv_i}\}$.

♢ $\forall (x,y) \in F_{p2v}, W_{p2v}(x,y) = \begin{cases} C_{1_{i_j}} & if\ (x,y) = (t_{vs_{i_j}}, p_{wg1_{i_j}}) \vee (x,y) = (p_{wfv1_{i_j}}, t_{fv1_{i_j}}) \\ C_{2_{i_j}} & if\ (x,y) = (t_{fv1_{i_j}}, p_{wg_{i_j}}) \vee (x,y) = (p_{wf_{i_j}}, t_{fv_{i_j}}) \\ 1 & otherwise \end{cases}$

♢ $M_{0_{p2v}}(p_x) = 0, \forall p_x \in P, p_x \neq p_{proc}$; and $M_{0_{p2v}}(p_{proc}) = 1$;

♢ $I_{p2v}(t_x) = [0,0], \forall t_x \in T_{p2v}, t_x \notin \{t_{c1_{i_j}}, t_{c_{i_j}}\}$; and $I_{p2v}(t_{c1_{i_j}}) = I_{p2v}(t_{c_{i_j}}) = [1,1]$;

♢ $\mathcal{E}_{p2v}(t_x) = 0.0\text{J}, \forall t_x \in T_{p2v}, t_x \notin \{t_{c1_{i_j}}, t_{c_{i_j}}\}$; $\mathcal{E}_{p2v}(t_{c1_{i_j}}) = (vef(v_{ideal_H}) \times c_1)/C_{1_{i_j}}$; and $\mathcal{E}_{p2v}(t_{c_{i_j}}) = (vef(v_{ideal_L}) \times c_2)/C_{2_{i_j}}$. $vef$ is the voltage-energy function. See Chapter 5 - Section 5.2.4;

♢ $\mathcal{CS}_{p2v}(t_{c1_{i_j}}) = \mathcal{CS}_{p2v}(t_{c_{i_j}}) = code_i$, in which $code_i \in \mathcal{ST}$.

The preemptive task structure block with 2 voltages is covered by 2 basic P-invariants:

$$\diamondsuit\ \mathcal{I}_{(p2v)(1)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{0} & \overset{p_{wg1_{i_j}}}{0} & \overset{p_{wc1_{i_j}}}{proc} & \overset{p_{wf1_{i_j}}}{0} & \overset{p_{wg_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{proc} & \overset{p_{wf_{i_j}}}{0} & \overset{p_{fv_i}}{0} & \overset{p_{proc}}{proc} \end{bmatrix}^T;$$

$$\diamondsuit\ \mathcal{I}_{(p2v)(2)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(\frac{C_{1_{i_j}} C_{2_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{(\frac{C_{2_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{(\frac{C_{2_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} \\ \overset{p_{wf1_{i_j}}}{(\frac{C_{2_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} & \overset{p_{wg_{i_j}}}{(\frac{C_{1_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} & \overset{p_{wc_{i_j}}}{(\frac{C_{1_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} & \overset{p_{wf_{i_j}}}{(\frac{C_{1_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} \\ \overset{p_{fv_i}}{(\frac{C_{1_{i_j}} C_{2_{i_j}}}{gcf(C_{1_{i_j}}, C_{2_{i_j}})})wg1_{i_j}} & \overset{p_{proc}}{0} \end{bmatrix}^T.$$

where $wg1_{i_j}, proc \in \mathbb{N}^*$. In the previous basic P-invariants, $gcf$ is a function that computes the greatest common factor of two integer numbers. A P-invariant covering all places is presented below:

$$\mathcal{I}_{(p2v)} = \mathcal{I}_{(p2v)(1)} + gcf(C_{1_{i_j}}, C_{2_{i_j}}) \times \mathcal{I}_{(p2v)(2)}$$

$$\mathcal{I}_{(p2v)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j} + proc} & \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \\ \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} & \overset{p_{wc_{i_j}}}{(C_{1_{i_j}})wg1_{i_j} + proc} & \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} & \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

The preemptive task structure block with 2 voltages is structurally conservative and structurally bounded, because $\mathcal{I}_{(p2v)} > 0$ and $\mathcal{I}^T_{(p2v)} \times A_{p2v} = 0$, in which $A_{p2v}$ is the respective incidence matrix.

### 6.3.7 Deadline Checking Block

Deadline missing (Figure 6.10) is an undesirable situation when considering hard real-time systems. Therefore, the scheduling algorithm (Chapter 8) should not reach deadline missing states, since those states do not allow finding out feasible schedules. The proposed block checks the occurrence of a deadline missing through transition $t_{d_i}$, which is enabled at the moment a task instance is ready for execution. For each place $p_{wc_{i_j}}$ and $p_{wc1_{i_j}}$ in each task structure block related to task $\tau_i$, a transition $t_{pc_{i_j}}$ is connected as post-condition. Whenever a task instance is executing and a deadline missing occurs (e.g., $t_{d_i}$ fires), the token is removed from place $p_{wc_{i_j}}$( or $p_{wc1_{i_j}}$) such that it is not possible to fire any other computation transition (e.g., $t_{c_{i_j}}$) in the model. In other words, the model enters in a deadlock state, since the processor is not released. In this case, during schedule generation, the proposed scheduling algorithm backtracks and selects other voltage level (and the respective maximum frequency) or task for execution. Besides, the timing interval for each transition $t_{pc_{i_j}}$ is [0,0], and for transition $t_{d_i}$ is the deadline constraint $d_i$ ($[d_i,d_i]$) of task $\tau_i$.

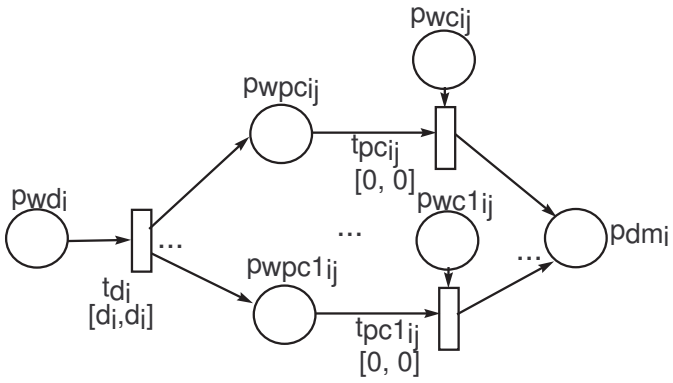


**Figure 6.10** Deadline checking block

The building block deadline-checking is a TPN $\mathcal{N}_d = (P_d, T_d, F_d, W_d, M_{0_d}, I_d, \mathcal{E}_d, \mathcal{CS}_d)$, such that:

♢ $P_d = \{p_{wd_i}, p_{dm_i}\} \cup P_{wpc} \cup P_{wpc1} \cup P_{wplc} \cup P_{wplc1} \cup P_{wc} \cup P_{wc1} \cup P_{wlc} \cup P_{wlc1}$. These places model the following situations:

- $p_{wd_i}$: waiting for deadline missing;
- $p_{dm_i}$: deadline missed;
- $p_{wpc_{i_j}} \in P_{wpc}$: waiting for computation removing (or waiting for computation removing of task second part when considering task structures with 2 voltages), such that $|P_{wpc}| = |P_{wc}| \wedge \exists p_{wc_{i_h}} \in P_{wc}, h = j$;
- $p_{wpc1_{i_j}} \in P_{wpc1}$: waiting for computation removing of task first part when considering task structures with 2 voltages, such that $|P_{wpc1}| = |P_{wc1}| \wedge \exists p_{wc1_{i_h}} \in P_{wc1}, h = j \wedge \exists p_{wpc_{i_h}} \in P_{wpc}, h = j$;
- $p_{wplc_{i_j}} \in P_{wplc}$: waiting for computation removing of the last time unit (or waiting for computation removing of the last time unit of task second part) when considering overheads, such that $|P_{wplc}| = |P_{wlc}| \wedge \exists p_{wlc_{i_h}} \in P_{wlc}, h = j$;
- $p_{wplc1_{i_j}} \in P_{wplc1}$: waiting for computation removing of the last time unit of task first part when considering overheads and 2 voltages, such that $|P_{wplc1}| = |P_{wlc1}| \wedge \exists p_{wlc1_{i_h}} \in P_{wlc1}, h = j \wedge \exists p_{wplc_{i_h}} \in P_{wplc}, h = j$;
- $p_{wc_{i_j}} \in P_{wc}$: waiting for task computation (or waiting for the computation of task second part when taking into account task structures with 2 voltages), such that $|P_{wc}| = |\mathcal{V}_{total_i}|$, $(\mathcal{V}_{total_i} = \mathcal{V}_{CPU} \cup \mathcal{V}_{ideal_i})$. $\mathcal{V}_{total_i}$ is the set of available voltage levels for executing a task $\tau_i$;
- $p_{wc1_{i_j}} \in P_{wc1}$: waiting for the computation of task first part when taking into account task structures with 2 voltages, such that $|P_{wc1}| = |\mathcal{V}_{ideal_i}|$. $\mathcal{V}_{ideal_i}$ is the set of ideal voltage levels for executing a task $\tau_i$;

- $p_{wlc_{i_j}} \in P_{wlc}$: waiting for the computation of the last time unit (or waiting for the computation of the last time unit of task second part) when taking into account overheads, such that $|P_{wlc}| = |\mathcal{V}_{total_i}|$, ($\mathcal{V}_{total_i} = \mathcal{V}_{CPU} \cup \mathcal{V}_{ideal_i}$). $\mathcal{V}_{total_i}$ is the set of available voltage levels for executing a task $\tau_i$;
- $p_{wlc1_{i_j}} \in P_{wlc1}$: waiting for the computation of the last time unit of task first part when taking into account overheads and 2 voltages, such that $|P_{wlc1}| = |\mathcal{V}_{ideal_i}|$. $\mathcal{V}_{ideal_i}$ is the set of ideal voltage levels for executing a task $\tau_i$.

♢ $T_d = \{t_{d_i}\} \cup T_{pc} \cup T_{pc1} \cup T_{plc} \cup T_{plc1}$. These transitions model the following actions:

- $t_{d_i}$: deadline missing;
- $t_{pc_{i_j}} \in T_{pc}$: computation removing (or computation removing of task second part when considering task structures with 2 voltages), such that $|T_{pc}| = |P_{wc}|$;
- $t_{pc1_{i_j}} \in T_{pc1}$: computation removing of task first part when considering task structures with 2 voltages, such that $|T_{pc1}| = |P_{wc1}|$;
- $t_{plc_{i_j}} \in T_{plc}$: computation removing of the last time unit (or computation removing of the last time unit of task second part) when considering overheads, such that $|T_{plc}| = |P_{wlc}|$;
- $t_{plc1_{i_j}} \in T_{plc1}$: computation removing of the last time unit of task first part when considering overheads and 2 voltages, such that $|T_{plc1}| = |P_{wlc1}|$.

♢ Pre and post-conditions of the transitions are (flow relation $F_d$):

- $\bullet t_{d_i} = \{p_{wd_i}\}$; $t_{d_i}\bullet = \{p_{wpc_{i_j}} | p_{wpc_{i_j}} \in (P_{wpc} \cup P_{wpc1} \cup P_{wplc} \cup P_{wplc1})\}$;
- $\forall t_{pc_{i_j}} \in T_{pc}, \bullet t_{pc_{i_j}} = \{p_{wc_{i_h}} \in P_{wc} | h = j\} \cup \{p_{wpc_{i_h}} \in P_{wpc} | h = j\}; t_{pc_{i_j}}\bullet = \{p_{dm_i}\}$;
- $\forall t_{pc1_{i_j}} \in T_{pc1}, \bullet t_{pc1_{i_j}} = \{p_{wc1_{i_h}} \in P_{wc1} | h = j\} \cup \{p_{wpc1_{i_h}} \in P_{wpc1} | h = j\}; t_{pc_{i_j}}\bullet = \{p_{dm_i}\}$;
- $\forall t_{plc_{i_j}} \in T_{plc}, \bullet t_{plc_{i_j}} = \{p_{wlc_{i_h}} \in P_{wlc} | h = j\} \cup \{p_{wplc_{i_h}} \in P_{wplc} | h = j\}$; $t_{plc_{i_j}}\bullet = \{p_{dm_i}\}$;
- $\forall t_{plc1_{i_j}} \in T_{plc1}, \bullet t_{plc1_{i_j}} = \{p_{wlc1_{i_h}} \in P_{wlc1} | h = j\} \cup \{p_{wplc1_{i_h}} \in P_{wplc1} | h = j\}; t_{plc_{i_j}}\bullet = \{p_{dm_i}\}$.

♢ $\forall (x, y) \in F_d, W_d(x, y) = 1$;

♢ $\forall p \in P_d, M_{0_d}(p) = 0$;

♢ $I_d(t_x) = [0, 0], \forall t_x \in T_d, t_x \neq t_{d_i}$; and $I_d(t_{d_i}) = [d_i, d_i]$;

♢ $\forall t \in T_d, \mathcal{E}_d(t) = 0.0$J;

♢ Representing $\mathcal{CS}_d$ as a set, $\mathcal{CS}_d = \emptyset$.

The deadline checking block is covered by $2^x$ basic P-invariants, such that x = $|P_{wc}|+|P_{wc1}|+|P_{wlc}|+|P_{wlc1}|$. These P-invariants represents every possible coverage of *waiting for task computation* places (e.g., $p_{wc_{i_j}}$) . Additionally, when a place $p_{wc_{i_j}} \in P_{wc} \cap {\bullet}t_{pc_{i_j}}$ (or $p_{wc1_{i_j}} \in P_{wc1} \cap {\bullet}t_{pc1_{i_j}}$) is covered by a basic P-invariant, a place $p_{wpc_{i_j}} \in P_{wpc} \cap {\bullet}t_{pc_{i_j}}$ (or $p_{wpc1_{i_j}} \in P_{wpc1} \cap {\bullet}t_{pc1_{i_j}}$) do not belong to the support of the **same basic** P-invariant. The converse is also true and the same situation occurs when considering preemptive task structure blocks with overheads.

As an example, assume $x = 2$ ($|P_{wc}| = 1$ and $|P_{wc1}|$=1). The respective basic P-invariants are presented as follows ($2^2 = 4$):

$$\diamondsuit\ \mathcal{I}_{(d)(0)} = \left[\begin{array}{cccccc} \overset{p_{wd_i}}{2d_0} & \overset{p_{wpc1_{i_j}}}{d_0} & \overset{p_{wc1_{i_j}}}{0} & \overset{p_{wpc_{i_j}}}{d_0} & \overset{p_{wc_{i_j}}}{0} & \overset{p_{dm_i}}{d_0} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1_j)} = \left[\begin{array}{cccccc} \overset{p_{wd_i}}{d_{1_j}} & \overset{p_{wpc1_{i_j}}}{0} & \overset{p_{wc1_{i_j}}}{d_{1_j}} & \overset{p_{wpc_{i_j}}}{d_{1_j}} & \overset{p_{wc_{i_j}}}{0} & \overset{p_{dm_i}}{d_{1_j}} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(j)} = \left[\begin{array}{cccccc} \overset{p_{wd_i}}{d_j} & \overset{p_{wpc1_{i_j}}}{d_j} & \overset{p_{wc1_{i_j}}}{0} & \overset{p_{wpc_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{d_j} & \overset{p_{dm_i}}{d_j} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1_j,j)} = \left[\begin{array}{cccccc} \overset{p_{wd_i}}{0} & \overset{p_{wpc1_{i_j}}}{0} & \overset{p_{wc_1 i_j}}{d_{1_j,j}} & \overset{p_{wpc_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{d_{1_j,j}} & \overset{p_{dm_i}}{d_{1_j,j}} \end{array}\right]^T.$$

in which $d_0, d_{1_j}, d_j, d_{1_j,j} \in \mathbb{N}^*$. For the previous example, a P-invariant covering all places can be obtained as follows:

$\mathcal{I}_{(d)} = \mathcal{I}_{(d)(0)} + \mathcal{I}_{(d)(1_j)} + \mathcal{I}_{(j)} + \mathcal{I}_{(1_j,j)}$

$$\mathcal{I}_{(d)} = \left[\begin{array}{ccccc} \overset{p_{wd_i}}{2d_0 + d_{1_j} + d_j} & \overset{p_{wpc1_{i_j}}}{d_0 + d_j} & \overset{p_{wc1_{i_j}}}{d_{1_j} + d_{1_j,j}} & \overset{p_{wpc_{i_j}}}{d_0 + d_{1_j}} & \overset{p_{wc_{i_j}}}{d_j + d_{1_j,j}} \end{array}\right.$$

$$\left.\begin{array}{c} \overset{p_{dm_i}}{d_0 + d_{1_j} + d_j + d_{1_j,j}} \end{array}\right]^T$$

Let us consider another example taking into account $x = 3$ ($|P_{wc}| = 2$ and $|P_{wc1}|$=1), the basic P-invariants are presented as follows ($2^x = 2^3 = 8$):

$$\diamondsuit\ \mathcal{I}_{(d)(0)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{3d_0} & \overset{p_{wpc_{i_1}}}{d_0} & \overset{p_{wc_{i_1}}}{0} & \overset{p_{wpc1_{i_j}}}{d_0} & \overset{p_{wc1_{i_j}}}{0} & \overset{p_{wpc_{i_j}}}{d_0} & \overset{p_{wc_{i_j}}}{0} & \overset{p_{dm_i}}{d_0} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{2d_1} & \overset{p_{wpc_{i_1}}}{0} & \overset{p_{wc_{i_1}}}{d_1} & \overset{p_{wpc1_{i_j}}}{d_1} & \overset{p_{wc1_{i_j}}}{0} & \overset{p_{wpc_{i_j}}}{d_1} & \overset{p_{wc_{i_j}}}{0} & \overset{p_{dm_i}}{d_1} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1_j)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{2d_{1_j}} & \overset{p_{wpc_{i_1}}}{d_{1_j}} & \overset{p_{wc_{i_1}}}{0} & \overset{p_{wpc1_{i_j}}}{0} & \overset{p_{wc1 i_1}}{d_{1_j}} & \overset{p_{wpc_{i_j}}}{d_{1_j}} & \overset{p_{wc_{i_j}}}{0} & \overset{p_{dm_i}}{d_{1_j}} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1,1_j)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{d_{1,1_j}} & \overset{p_{wpc_{i_1}}}{0} & \overset{p_{wc_{i_1}}}{d_{1,1_j}} & \overset{p_{wpc1_{i_j}}}{0} & \overset{p_{wc1_{i_j}}}{d_{1,1_j}} & \overset{p_{wpc_{i_j}}}{d_{1,1_j}} & \overset{p_{wc_{i_j}}}{0} & \overset{p_{dm_i}}{d_{1,1_j}} \end{array}\right]^T.$$

$$\diamondsuit\ \mathcal{I}_{(d)(j)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{2d_j} & \overset{p_{wpc_{i_1}}}{d_j} & \overset{p_{wc_{i_1}}}{0} & \overset{p_{wpc1_{i_j}}}{d_j} & \overset{p_{wc1_{i_j}}}{0} & \overset{p_{wpc_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{d_j} & \overset{p_{dm_i}}{d_j} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1,j)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{d_{1,j}} & \overset{p_{wpc_{i_1}}}{0} & \overset{p_{wc_{i_1}}}{d_{1,j}} & \overset{p_{wpc1_{i_j}}}{d_{1,j}} & \overset{p_{wc1_{i_j}}}{0} & \overset{p_{wpc_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{d_{1,j}} & \overset{p_{dm_i}}{d_{1,j}} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1_1,j)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{d_{1_j,j}} & \overset{p_{wpc_{i_1}}}{d_{1_j,j}} & \overset{p_{wc_{i_1}}}{0} & \overset{p_{wpc1_{i_j}}}{0} & \overset{p_{wc1_{i_j}}}{d_{1_j,j}} & \overset{p_{wpc_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{d_{1_j,j}} & \overset{p_{dm_i}}{d_{1_j,j}} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1,1_j,j)} = \left[\begin{array}{cccccccc} \overset{p_{wd_i}}{0} & \overset{p_{wpc_{i_1}}}{0} & \overset{p_{wc_{i_1}}}{d_{1,1_j,j}} & \overset{p_{wpc1_{i_j}}}{0} & \overset{p_{wc1_{i_j}}}{d_{1,1_j,j}} & \overset{p_{wpc_{i_j}}}{0} & \overset{p_{wc_{i_j}}}{d_{1,1_j,j}} & \overset{p_{dm_i}}{d_{1,1_j,j}} \end{array}\right]^T.$$

in which $d_0, d_1, d_{1_j}, d_{1,1_j}, d_j, d_{1,j}, d_{1_j,j}, d_{1,1_j,j} \in \mathbb{N}^*$. For this example, a P-invariant covering all places is presented as follows:

$$\mathcal{I}_{(d)} = \mathcal{I}_{(d)(0)} + \mathcal{I}_{(d)(1)} + \mathcal{I}_{(d)(1_j)} + \mathcal{I}_{(d)(1,1_j)} + \mathcal{I}_{(d)(j)} + \mathcal{I}_{(d)(1,j)} + \mathcal{I}_{(d)(1_1,j)} + \mathcal{I}_{(d)(1,1_j,j)}$$

$$\begin{array}{ll} \mathcal{I}_{(d)} = & \Big[\ 3d_0 + 2d_1 + 2d_{1_j} + \overset{p_{wd_i}}{d_{1,1_j}} + 2d_j + d_{1,j} + d_{1_j,j} \\ & d_0 + \overset{p_{wpc_{i_1}}}{d_{1_j}} + d_j + d_{1_j,j} \quad d_1 + \overset{p_{wc_{i_1}}}{d_{1,1_j}} + d_{1,j} + d_{1,1_j,j} \\ & d_0 + d_1 + \overset{p_{wpc1_{i_j}}}{d_j} + d_{1,j} \quad d_{1_j} + \overset{p_{wc1_{i_j}}}{d_{1,1_j}} + d_{1_j,j} + d_{1,1_j,j} \\ & d_0 + d_1 + d_{1_j} + \overset{p_{wpc_{i_j}}}{d_{1,1_j}} \quad d_j + \overset{p_{wc_{i_j}}}{d_{1,j}} + d_{1_j,j} + d_{1,1_j,j} \\ & d_0 + d_1 + d_{1_j} + d_{1,1_j} + \overset{p_{dm_i}}{d_j} + d_{1,j} + d_{1_j,j} + d_{1,1_j,j}\ \Big]^T \end{array}$$

Thus, for any $x$ value, the basic ($2^x$) P-invariants are presented below, assuming that $m = |\mathcal{V}_{total_i}|$ (i.e., the number of voltage levels available for task $\tau_i$):

$$\diamondsuit\ \mathcal{I}_{(d)(0)} = \left[\begin{array}{cccccccccccc} \overset{p_{wd_i}}{(x)d_0} & \overset{p_{wpc_{i_1}}}{d_0} & \overset{p_{wc_{i_1}}}{0} & \overset{p_{wpc_{i_2}}}{d_0} & \overset{p_{wc_{i_2}}}{0} & \overset{\cdots}{\cdots} & \overset{p_{wpc_{i_{m-1}}}}{d_0} & \overset{p_{wc_{i_{m-1}}}}{0} & \overset{p_{wpc_{i_m}}}{d_0} & \overset{p_{wc_{i_m}}}{0} & \overset{p_{dm_i}}{d_0} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1)} = \left[\begin{array}{cccccccccccc} \overset{p_{wd_i}}{(x-1)d_1} & \overset{p_{wpc_{i_1}}}{0} & \overset{p_{wc_{i_1}}}{d_1} & \overset{p_{wpc_{i_2}}}{d_1} & \overset{p_{wc_{i_2}}}{0} & \overset{\cdots}{\cdots} & \overset{p_{wpc_{i_{m-1}}}}{d_1} & \overset{p_{wc_{i_{m-1}}}}{0} & \overset{p_{wpc_{i_m}}}{d_1} & \overset{p_{wc_{i_m}}}{0} & \overset{p_{dm_i}}{d_1} \end{array}\right]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(2)} = \Big[\ \overset{p_{wd_i}}{(x-1)d_2}\ \ d_2\ \ 0\ \ 0\ \ \overset{p_{wpc_{i_1}}}{d_2}\ \ \overset{p_{wc_{i_1}}}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{d_2}\ \ \overset{p_{wc_{i_{m-1}}}}{0}\ \ \overset{p_{wpc_{i_m}}}{d_2}\ \ \overset{p_{wc_{i_m}}}{0}\ \ \overset{p_{dm_i}}{d_2}\ \Big]^T;$$

$$\vdots$$

$$\diamondsuit\ \mathcal{I}_{(d)(m-1)} = \Big[\ \overset{p_{wd_i}}{(x-1)d_{m-1}}\ \ d_{m-1}\ \ 0\ \ \overset{p_{wpc_{i_1}}}{d_{m-1}}\ \ \overset{p_{wc_{i_1}}}{0}\ \ \overset{p_{wpc_{i_2}}}{\cdots}\ \ \overset{p_{wc_{i_2}}}{0}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{0}\ \ \overset{p_{wc_{i_{m-1}}}}{d_{m-1}}\ \ \overset{p_{wpc_{i_m}}}{d_{m-1}}\ \ \overset{p_{wc_{i_m}}}{0}\ \ \overset{p_{dm_i}}{d_{m-1}}\ \Big]^T;$$

$$\vdots$$

$$\diamondsuit\ \mathcal{I}_{(d)(1,\ldots,m-1)} = \Big[\ \overset{p_{wd_i}}{(x-m-1)d_{1,\ldots,m-1}}\ \ \overset{p_{wpc_{i_1}}}{0}\ \ \overset{p_{wc_{i_1}}}{d_{1,\ldots,m-1}}\ \ \overset{p_{wpc_{i_2}}}{0}\ \ \overset{p_{wc_{i_2}}}{d_{1,\ldots,m-1}}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{0}\ \ \overset{p_{wc_{i_{m-1}}}}{d_{1,\ldots,m-1}}\ \ \overset{p_{wpc_{i_m}}}{d_{1,\ldots,m-1}}\ \ \overset{p_{wc_{i_m}}}{0}\ \ \overset{p_{dm_i}}{d_{1,\ldots,m-1}}\ \Big]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(m)} = \Big[\ \overset{p_{wd_i}}{(x-1)d_m}\ \ d_m\ \ 0\ \ \overset{p_{wpc_{i_1}}}{d_m}\ \ \overset{p_{wc_{i_1}}}{0}\ \ \overset{p_{wpc_{i_2}}}{\cdots}\ \ \overset{p_{wc_{i_2}}}{d_m}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{d_m}\ \ \overset{p_{wc_{i_{m-1}}}}{0}\ \ \overset{p_{wpc_{i_m}}}{0}\ \ \overset{p_{wc_{i_m}}}{d_m}\ \ \overset{p_{dm_i}}{d_m}\ \Big]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1,m)} = \Big[\ \overset{p_{wd_i}}{(x-2)d_{1,m}}\ \ \overset{p_{wpc_{i_1}}}{0}\ \ \overset{p_{wc_{i_1}}}{d_{1,m}}\ \ \overset{p_{wpc_{i_2}}}{d_{1,m}}\ \ \overset{p_{wc_{i_2}}}{0}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{d_{1,m}}\ \ \overset{p_{wc_{i_{m-1}}}}{0}\ \ \overset{p_{wpc_{i_m}}}{0}\ \ \overset{p_{wc_{i_m}}}{d_{1,m}}\ \ \overset{p_{dm_i}}{d_{1,m}}\ \Big]^T;$$

$$\vdots$$

$$\diamondsuit\ \mathcal{I}_{(d)(m-1,m)} = \Big[\ \overset{p_{wd_i}}{(x-2)d_{m-1,m}}\ \ d_{m-1,m}\ \ 0\ \ \overset{p_{wpc_{i_1}}}{d_{m-1,m}}\ \ \overset{p_{wc_{i_1}}}{0}\ \ \overset{p_{wpc_{i_2}}}{\cdots}\ \ \overset{p_{wc_{i_2}}}{0}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{0}\ \ \overset{p_{wc_{i_{m-1}}}}{d_{m-1,m}}\ \ \overset{p_{wpc_{i_m}}}{0}\ \ \overset{p_{wc_{i_m}}}{d_{m-1,m}}\ \ \overset{p_{dm_i}}{d_{m-1,m}}\ \Big]^T;$$

$$\vdots$$

$$\diamondsuit\ \mathcal{I}_{(d)(2,\ldots,m)} = \Big[\ \overset{p_{wd_i}}{d_{2,\ldots,m}}\ \ d_{2,\ldots,m}\ \ 0\ \ 0\ \ \overset{p_{wpc_{i_1}}}{d_{2,\ldots,m}}\ \ \overset{p_{wc_{i_1}}}{\cdots}\ \ \overset{p_{wpc_{i_2}}}{0}\ \ \overset{p_{wc_{i_2}}}{d_{2,\ldots,m}}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{0}\ \ \overset{p_{wc_{i_{m-1}}}}{d_{2,\ldots,m}}\ \ \overset{p_{wpc_{i_m}}}{0}\ \ \overset{p_{wc_{i_m}}}{d_{2,\ldots,m}}\ \ \overset{p_{dm_i}}{d_{2,\ldots,m}}\ \Big]^T;$$

$$\diamondsuit\ \mathcal{I}_{(d)(1,\ldots,m)} = \Big[\ \overset{p_{wd_i}}{0}\ \ \overset{p_{wpc_{i_1}}}{0}\ \ \overset{p_{wc_{i_1}}}{d_{1,\ldots,m}}\ \ \overset{p_{wpc_{i_2}}}{0}\ \ \overset{p_{wc_{i_2}}}{d_{1,\ldots,m}}\ \ \overset{\cdots}{\cdots}\ \ \overset{p_{wpc_{i_{m-1}}}}{0}\ \ \overset{p_{wc_{i_{m-1}}}}{d_{1,\ldots,m}}\ \ \overset{p_{wpc_{i_m}}}{0}$$

$$\overset{p_{wc_{i_m}}}{d_{1,\ldots,m}} \quad \overset{p_{dm_i}}{d_{1,\ldots,m}} \;\Big]^T.$$

in which $d_0, d_1, \cdots, d_{2,\ldots,m}, d_{1,\ldots,m} \in \mathbb{N}^*$.

For the sake of legibility, the P-invariant covering $x$ places will be represented by the following vector (note that $p_{wc_{i_m}}$ is replaced by $p_{wc_{i_j}}$)

$\mathcal{I}_{(d)} = \mathcal{I}_{(d)(0)} + \mathcal{I}_{(d)(1)} + \cdots + \mathcal{I}_{(d)(1,\ldots,j)}$

$$\mathcal{I}_{(d)} = \Big[ \overset{p_{wd_i}}{(x)d_0 + \ldots + d_{2,\ldots,j}} \quad \overset{p_{wpc_{i_1}}}{d_0 + \ldots + d_{2,\ldots,j}} \quad \overset{p_{wc_{i_1}}}{d_1 + \cdots + d_{1,\ldots,j}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{wpc_{i_j}}}{d_0 + \cdots + d_{1,j-1}}$$

$$\overset{p_{wc_{i_j}}}{d_j + \ldots + d_{1,\ldots,j}} \quad \overset{p_{dm_i}}{d_0 + \cdots + d_{1,\ldots,j}} \Big]^T$$

As $\mathcal{I}_{(d)}$ covers all places ($\mathcal{I}_{(d)} > 0$), and $\mathcal{I}_{(d)}^T \times A_d = 0$ ($A_d$ is the incidence matrix), the deadline checking block is structurally conservative as well as structurally bounded.

### 6.3.8 Task Instance Conclusion Block

This block (Figure 6.11) models the conclusion of a task instance. Transition $t_{f_i}$ represents this situation, such that, when it fires (e.g., task instance conclusion), a token is removed from place $p_{wd_i}$ (waiting for deadline missing).

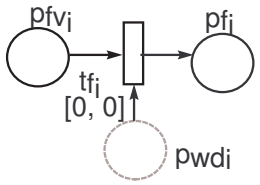


**Figure 6.11** Task instance conclusion block

The task instance conclusion block is modeled by a TPN $\mathcal{I}_c = (P_c, T_c, F_c, W_c, M_{0_c}, I_c, \mathcal{E}_c, \mathcal{CS}_c)$, such that:

♢ $P_c = \{p_{fv_i}, p_{f_i}, p_{wd_i}\}$. These places model the following situations:

- $p_{fv_i}$: task instance conclusion;
- $p_{f_i}$: end of task execution;
- $p_{wd_i}$: waiting for deadline missing.

♢ $T_c = \{t_{f_i}\}$. This transition models the following action:

- $t_{f_i}$: concluding task instance execution.

♢ Pre and post-conditions of the transition are (flow relation $F_c$):

- $\bullet t_{f_i} = \{p_{fv_i}, p_{wd_i}\}$; $t_{f_i}\bullet = \{p_{f_i}\}$.

♢ $\forall(x,y) \in F_c, W_c(x,y) = 1$;

♢ $\forall p \in P_c, M_{0_c}(p) = 0$;

♢ $I_c(t_{f_i}) = [0,0]$;

♢ $\mathcal{E}_c(t_{f_i}) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_c$ as a set, $\mathcal{CS}_c = \emptyset$.

The task instance conclusion block is covered by 2 basic P-invariants, which are depicted below:

♢ $\mathcal{I}_{(c)(1)} = \overset{\begin{matrix} p_{f_i} & p_{fv_i} & p_{wd_i} \end{matrix}}{\begin{bmatrix} fv_i & fv_i & 0 \end{bmatrix}^T}$;

♢ $\mathcal{I}_{(c)(2)} = \overset{\begin{matrix} p_{f_i} & p_{fv_i} & p_{wd_i} \end{matrix}}{\begin{bmatrix} wd_i & 0 & wd_i \end{bmatrix}^T}$.

such that$fv_i, wd_i \in \mathbb{N}^*$. Using these basic invariants, a P-invariant covering all places ($\mathcal{I}_{(c)}$) can be obtained:

$\mathcal{I}_{(c)} = \mathcal{I}_{(c)(1)} + \mathcal{I}_{(c)(2)}$

$\mathcal{I}_{(c)} = \overset{\begin{matrix} p_{f_i} & p_{fv_i} & p_{wd_i} \end{matrix}}{\begin{bmatrix} fv_i + wd_i & fv_i & wd_i \end{bmatrix}^T}$

As $\mathcal{I}_{(c)} > 0$ and $\mathcal{I}_{(c)}^T \times A_c = 0$ ($A_c$ is the incidence matrix), $\mathcal{N}_{(c)}$ is structurally conservative as well as structurally bounded.

### 6.3.9 Join Block

Usually, concurrent activities need to synchronize with each other. The join block execution states that all tasks in the system have concluded their execution in the schedule period. Figure 6.12 depicts the join block.

This block is modeled by a TPN $\mathcal{N}_j = (P_j, T_j, F_j, W_j, M_{0_j}, I_j, \mathcal{E}_j, \mathcal{CS}_j)$, in which:

♢ $P_j = \{p_{end_{spec}}\} \cup P_{final}$. These places model the following situations:

- $p_{end_{spec}}$: end of the system. A marking in this place ($M(p_{end_{spec}}) = 1$) represents the desired final marking ($M^F$). Additionally, *spec* is the specification name;
- $p_{f_i} \in P_{final}$: end of the $i^{th}$ task, such that $1 \leq i \leq n$ ($|P_f| = |\mathcal{T}| = n$). Note that $\mathcal{T}$ is the set of all tasks in the system.

♢ $T_j = \{t_{end_{spec}}\}$. This transition model the following action:

- $t_{end_{spec}}$: end of tasks in the system. *spec* is the specification name.

♢ Pre and post-conditions of the transition are (flow relation $F_j$):

- $\bullet t_{end_{spec}} = \{p_{f_i} | p_{f_i} \in P_f\}$;

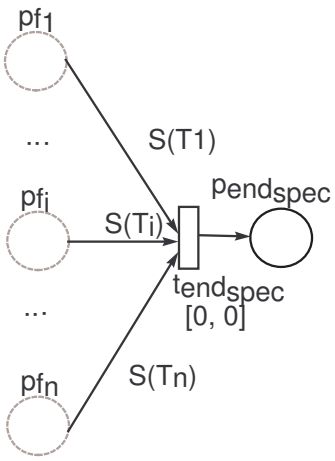


**Figure 6.12** Join block

- $t_{end_{spec}}\bullet = \{p_{end_{spec}}\}$.

♢ $W_j(p_{f_i}, t_{end_{spec}}) = \mathcal{S}(\tau_i), \forall p_{f_i} \in P_j, p_{f_i} \neq p_{end_{spec}}$; $W_j(t_{end_{spec}}, p_{end_{spec}}) = 1$;

♢ $M_{0_j}(p) = 0,\ \forall p \in P_j$;

♢ $I_j(t_{end_{spec}}) = [0, 0]$;

♢ $\forall t \in T_j, \mathcal{E}_j(t) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_j$ as a set, $\mathcal{CS}_j = \emptyset$.

In previous definition, $\mathcal{S}(\tau_i)$ defines the number of instances for a task $\tau_i$ in a schedule period (see Section 6.2).

The join block is covered by $n$ basic P-invariants ($n = |\mathcal{T}|$). See below:

♢ $\mathcal{I}_{(j)(1)} = \begin{bmatrix} \overset{p_{f_1}}{f_1} & \cdots & \overset{p_{f_i}}{0} & \cdots & \overset{p_{f_n}}{0} & \overset{p_{end_{spec}}}{f_1} \end{bmatrix}^T$;
⋮

♢ $\mathcal{I}_{(j)(i)} = \begin{bmatrix} \overset{p_{f_1}}{0} & \cdots & \overset{p_{f_i}}{f_i} & \cdots & \overset{p_{f_n}}{0} & \overset{p_{end_{spec}}}{f_i} \end{bmatrix}^T$;
⋮

♢ $\mathcal{I}_{(j)(n)} = \begin{bmatrix} \overset{p_{f_1}}{0} & \cdots & \overset{p_{f_i}}{0} & \cdots & \overset{p_{f_n}}{f_n} & \overset{p_{end_{spec}}}{f_n} \end{bmatrix}^T$.

in which $f_1, \cdots, f_i, \cdots f_n \in \mathbb{N}^*$ A P-invariant covering all places can be obtained in the following way:

$\mathcal{I}_{(j)} = \mathcal{I}_{(j)(1)} + \cdots + \mathcal{I}_{(j)(i)} + \cdots + \mathcal{I}_{(j)(n)}$

$$\mathcal{I}_{(j)} = \begin{bmatrix} \overset{p_{f_1}}{f_1} & \cdots & \overset{p_{f_i}}{f_i} & \cdots & \overset{p_{f_n}}{f_n} & \overset{p_{end_{spec}}}{f_1 + \cdots + f_i + \cdots + f_n} \end{bmatrix}^T$$

As $\mathcal{I}_{(j)}^T \times A_j = 0$, in which $A_j$ is the respective incidence matrix, and $\mathcal{I}_{(j)} > 0$, the join block is structurally conservative and structurally bounded.

### 6.3.10 Blocks for Modeling Overheads

Whenever designing hard real-time systems, overheads have to be taken into account for guaranteeing system predictability. Assuming a DVS-capable system, additional time and energy costs may occur and have to be considered in someway before runtime, such as: (i) preemptions; (ii) dispatcher or runtime scheduler overhead; and (iii) voltage/frequency switching. Several scheduling approaches disregard such overheads or include in tasks' computation time. Nevertheless, neglecting overheads may lead to timing or energy constraint violations, and the second alternative may be too pessimistic, since the total overhead is not known before schedule generation.

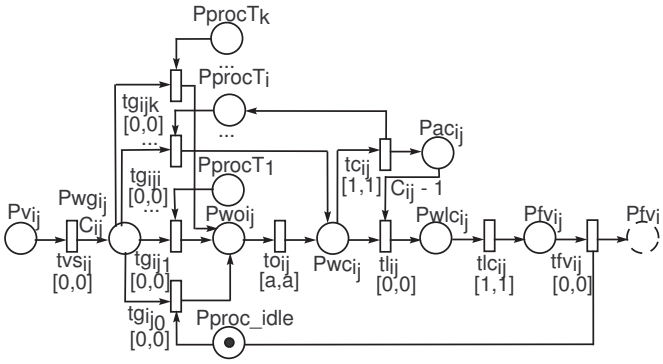


**Figure 6.13** Preemptive task structure with overhead block

This work explicitly models overheads and considers them during schedule generation, hence providing a more realistic system behavior. The first model (Figure 6.13) is a preemptive task structure with overhead block considering a single voltage. Assuming $k$ the number of tasks, places $p_{proc}T_x$, $1 \leq x \leq k$, represent *flags* that indicate the current task $T_x$ executing on processor $p_{proc}$. In a similar manner, place $p_{proc}\_idle$ represents the idle state of processor $p_{proc}$. Overheads are represented by transition $to_{ij}$, where its timing interval is equal to $[a, a]$, and an associated energy consumption value is assigned. In this work, the transition overhead contemplates: (i) dispatcher overhead, including context-switching; (ii) voltage/frequency related to the dispatcher execution; and (iii) voltage/frequency switching for executing the respective task. The proposed approach assumes the dispatcher execution at a fixed supply voltage, which is up to the designer to select the appropriate one. Transitions $tg_{ijx}$, $0 \leq x \leq k, i \neq j$, represent the processor granting and takes into account overheads that may occur in task start-up, context-saving or context-restoring. The overhead is not considered when the same task is executing without interruption, which is represented by transition $tg_{iji}$. As in preemptive task

structure block without overhead, the computation time is modeled using arc weights. Transition $tc_{ij}$ represents the execution of one task time unit related to the computation time ($C_{ij}$) and notifies the execution of task $\tau_i$ through place $p_{proc}T_i$. After the last computation time unit ($tlc_{ij}$), there is no need for indicating the task execution, and, next, the processor goes to idle state ($p_{proc}\_idle$). The processor $p_{proc}$ is not shown for the sake of readability. However, each processor-granting transition has an incoming arc from $p_{proc}$, and each computation transition has an outgoing arc to $p_{proc}$.

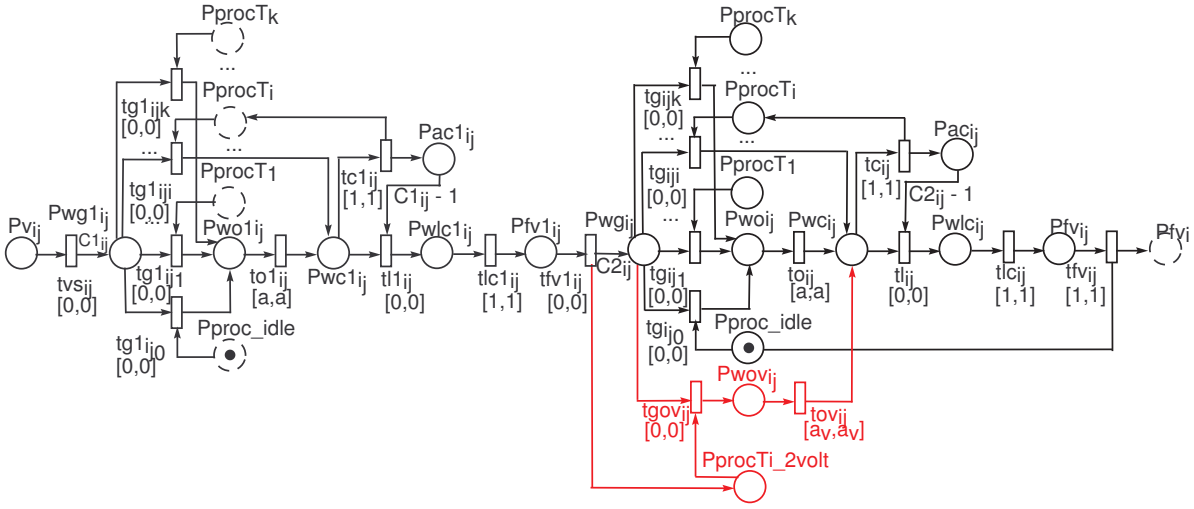


**Figure 6.14** Preemptive task structure with overhead and 2 voltages block

Figure 6.14 shows a preemptive task structure considering two voltage levels. As stated previously, this situation can occur when an ideal voltage is unavailable ($v_{ideal} \notin \mathcal{V}_{cpu} \wedge v_{ideal} \in \mathcal{V}_{ideal_i}$) and the two immediate neighbor voltages can be adopted ($v_{ideal_L}$, $v_{ideal_H} \in \mathcal{V}_{cpu}$). Without loss of generality, this block may be interpreted as two concatenated instances of the previous block, but with a slight difference. The difference lies on the overhead after executing the first part of the task at the immediate higher voltage ($v_{ideal_H}$). Since only a voltage/frequency switching is required to execute the second part of the task ($v_{ideal_L}$), an additional overhead transition ($tov_{ij}$) and place ($p_{proc}T_i\_2volt$) are considered. The timing interval $[a_v, a_v]$ and the energy consumption value associated with this overhead transition are smaller than those assigned to transition $to_{ij}$ due to the absence of unnecessary services. Additionally, as a new place ($p_{proc}T_i\_2volt$) is added, other tasks have to consider, in each preemptive structure block, a new processor-granting transition that has an incoming arc from $p_{proc}T_i\_2volt$. Again, $p_{proc}$ is not shown due to readability issues.

When considering non-preemptable tasks, the blocks presented in Figure 6.6 and Figure 6.8 may be adopted with only two modifications. The dispatcher overhead is associated with processor-granting transition ($t_{g_{i_j}}$) and, considering the block with two voltages, the additional overhead related to voltage/frequency switching is assigned to transition $tfv1_{i_j}$.

In the next paragraphs, the proposed overhead blocks are described algebraically.

The preemptive task structure block with overhead is a TPN $\mathcal{N}_o = (P_oT_o, F_o, W_o, M_{0_o}, I_o, \mathcal{E}_o, \mathcal{CS}_o)$, such that:

♢ $P_o = \{p_{v_{i_j}}, p_{wg_{i_j}}\} \cup P_{procT} \cup P_{procT_{ideal_i}} \cup \{p_{wo_{i_j}}, p_{wc_{i_j}}, p_{ac_{i_j}}, p_{wlc_{i_j}}, p_{fv_{i_j}}, p_{fv_i}, p_{proc}, p_{proc\_idle}\}$. These places model the following situations:

- $p_{v_{i_j}}$: voltage level $v_j$ selected;
- $p_{wg_{i_j}}$: waiting for processor granting;
- $p_{proc_{T_h}} \in P_{procT}$ : task $\tau_h$ executing, such that $1 \leq h \leq k$ and $k = |P_{procT}| = |\mathcal{T}|$. Note that $\mathcal{T}$ is the set of all tasks in the system;
- $p_{proc_{T_h}\_2volt} \in P_{procT_{ideal_i}}$ : task $\tau_h$ executing at a second voltage level, such that $(|P_{procT_{ideal_i}}| = |\{\tau_h \in \mathcal{T} - \{\tau_i\} | \mathcal{V}_{ideal_h} \neq \emptyset\}|) \wedge h \neq i \wedge \exists p_{proc_{T_l}} \in P_{procT}, l = h$. In other words, $P_{procT_{ideal_i}}$ is the set of places indicating a task, different from $\tau_i$, executing at a second voltage level;
- $p_{wo_{i_j}}$: waiting for overhead occurrence;
- $p_{wc_{i_j}}$: waiting for task computation;
- $p_{ac_{i_j}}$: accumulating computation time units;
- $p_{wlc_{i_j}}$: waiting for the last computation time unit;
- $p_{fv_{i_j}}$: waiting for the conclusion of task execution at voltage $v_j$ ;
- $p_{fv_i}$: conclusion of a task instance;
- $p_{proc}$: processor;
- $p_{proc\_idle}$: processor in idle state .

♢ $T_o = \{t_{vs_{i_j}}\} \cup T_{g_{i_j}} \cup T_{g_{ideal_{i_j}}} \cup \{t_{g_{i_{j_0}}}, t_{o_{i_j}}, t_{c_{i_j}}, t_{l_{i_j}}, t_{lc_{i_j}}, t_{fv_{i_j}}\}$. These transitions model the following actions:

- $t_{vs_{i_j}}$: starting task execution at voltage $v_j$;
- $t_{g_{i_{j_h}}} \in T_{g_{i_j}}$: processor granting, such that $1 \leq h \leq k$ and $k = |T_{g_{i_j}}| = |\mathcal{T}|$. Note that $\mathcal{T}$ is the set of all tasks in the system;
- $t_{g2v_{i_{j_h}}} \in T_{g_{ideal_{i_j}}}$: processor granting taking into account other task executing at a second voltage level, such that $|T_{g_{ideal_{i_j}}}| = |P_{procT_{ideal_i}}| = |\{\tau_h \in \mathcal{T} - \{\tau_i\} | \mathcal{V}_{ideal_h} \neq \emptyset\}| \wedge \exists t_{g_{i_{j_l}}} \in T_{g_{i_j}}, l = h \neq i$. In other words, $T_{g_{ideal_{i_j}}}$ is the set of transitions indicating a processor granting that will incur overheads due to a different task (than $\tau_i$) executing at a second voltage level;
- $t_{g_{i_{j_0}}}$: processor granting taking into account the CPU in the idle state;
- $t_{o_{i_j}}$: overhead occurrence;
- $t_{c_{i_j}}$: executing one TTU;

- $t_{l_{i_j}}$: preparing last computation time unit;
- $t_{lc_{i_j}}$: executing last TTU;
- $t_{fv_{i_j}}$: finalizing task execution at voltage $v_j$.

♢ Pre and post-conditions of the transitions are (flow relation $F_o$):

- $\bullet t_{vs_{i_j}} = \{p_{v_{i_j}}\}$; $t_{vs_{i_j}}\bullet = \{p_{wg_{i_j}}\}$;
- $\forall t_{g_{i_{j_h}}} \in T_{g_{i_j}}, h \neq i, \bullet t_{g_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}} \in P_{procT} | l = h\}$; $t_{g_{i_{j_h}}}\bullet = \{p_{wo_{i_j}}\}$;
- $\exists t_{g_{i_{j_h}}} \in T_{g_{i_j}}, h = i, \bullet t_{g_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}} \in P_{procT} | l = h = i\}; t_{g_{i_{j_h}}}\bullet = \{p_{wc_{i_j}}\}$
- $\forall t_{g2v_{i_{j_h}}} \in T_{g_{ideal_{i_j}}}, \bullet t_{g2v_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}\_2volt} \in P_{procT_{ideal_i}} | l = h \neq i\}; t_{g2v_{i_{j_h}}}\bullet = \{p_{wo_{i_j}}\}$;
- $\bullet t_{g_{i_{j_0}}} = \{p_{wg_{i_j}}, p_{proc}, p_{proc\_idle}\}$; $t_{g_{i_{j_0}}}\bullet = \{p_{wo_{i_j}}\}$;
- $\bullet t_{o_{i_j}} = \{p_{wo_{i_j}}\}; t_{o_{i_j}}\bullet = \{p_{wc_{i_j}}\}$;
- $\bullet t_{c_{i_j}} = \{p_{wc_{i_j}}\}$; $t_{c_{i_j}}\bullet = \{p_{proc}, p_{proc_{T_i}}, p_{ac_{i_j}}\}$;
- $\bullet t_{l_{i_j}} = \{p_{wc_{i_j}}, p_{ac_{i_j}}\}; t_{l_{i_j}}\bullet = \{p_{wlc_{i_j}}\}$;
- $\bullet t_{lc_{i_j}} = \{p_{wlc_{i_j}}\}; t_{lc_{i_j}}\bullet = \{p_{proc}, p_{fv_{i_j}}\}$;
- $\bullet t_{fv_{i_j}} = \{p_{fv_{i_j}}\}; t_{fv_{i_j}}\bullet = \{p_{proc\_idle}, p_{fv_i}\}$;

♢ $\forall (x,y) \in F_o, W_o(x,y) = \begin{cases} C_{i_j} & if\ (x,y) = (t_{vs_{i_j}}, p_{wg_{i_j}}) \\ C_{i_j} - 1 & if\ (x,y) = (p_{ac_{i_j}}, t_{l_{i_j}}) \\ 1 & otherwise \end{cases}$

♢ $M_{0_o}(p_x) = 0, \forall p_x \in P_o, p_x \notin \{p_{proc}, p_{proch}\_idle\}; M_{0_o}(p_{proc}) = M_{0_o}(p_{proch}\_idle) = 1$;

♢ $I_o(t_x) = [0,0], \forall t_x \in T_o, t_x \notin \{t_{o_{i_j}}, t_{c_{i_j}}, t_{lc_{i_j}}\}; I_o(t_{o_{i_j}}) = [o,o]$. o is the dispatcher WCET (see Chapter 5 Section 5.2.2); $I_o(t_{c_{i_j}}) = I_o(t_{lc_{i_j}}) = [1,1]$;

♢ $\mathcal{E}_o(t_x) = 0.0\text{J}, \forall t_x \in T_p, t_x \notin \{t_{o_{i_j}}, t_{c_{i_j}}, t_{lc_{i_j}}\}$; $\mathcal{E}_o(t_{o_{i_j}}) = o_{\mathcal{E}}$. $o_{\mathcal{E}}$ is the dispatcher energy consumption (Section 5.2.2); and $\mathcal{E}_o(t_{c_{i_j}}) = \mathcal{E}_o(t_{lc_{i_j}}) = (vef(v_j) \times c_i)/C_{i_j}$. $vef$ is the voltage-energy function. See Chapter 5 - Section 5.2.4;

♢ $\mathcal{CS}_o(t_{c_{i_j}}) = \mathcal{CS}_o(t_{lc_{i_j}}) = code_i$, in which $code_i \in \mathcal{ST}$.

The preemptive task structure with overhead block is covered by 3 basic P-invariants, which are described as follows:

$$\diamondsuit\ \mathcal{I}_{(o)(1)} = \Big[ \overset{p_{v_{i_j}}}{0} \quad \overset{p_{wg_{i_j}}}{0} \quad \overset{p_{wo_{i_j}}}{proc} \quad \overset{p_{wc_{i_j}}}{proc} \quad \overset{p_{ac_{i_j}}}{0} \quad \overset{p_{wlc_{i_j}}}{proc} \quad \overset{p_{fv_{i_j}}}{0} \quad \overset{p_{fv_i}}{0} \quad \overset{p_{proc_{T_1}}}{0} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{0} \quad \overset{\cdots}{\cdots}$$
$$\overset{p_{proc_{T_k}}}{0} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{0} \Big]^T;$$

$$\diamondsuit\ \mathcal{I}_{(o)(2)} = \Big[ \overset{p_{v_{i_j}}}{0} \quad \overset{p_{wg_{i_j}}}{0} \quad \overset{p_{wo_{i_j}}}{proc\_idle} \quad \overset{p_{wc_{i_j}}}{proc\_idle} \quad \overset{p_{ac_{i_j}}}{0} \quad \overset{p_{wlc_{i_j}}}{proc\_idle} \quad \overset{p_{fv_{i_j}}}{proc\_idle} \quad \overset{p_{fv_i}}{0}$$
$$\overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{0} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T;$$

$$\diamondsuit\ \mathcal{I}_{(o)(3)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}}$$
$$\overset{p_{proc_{T_1}}}{0} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{0} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{0} \quad \overset{p_{proc}}{0} \quad \overset{p_{proc\_idle}}{0} \Big]^T.$$

in which $proc, proc\_idle, wg_{i_j} \in \mathbb{N}^*$. Using these basic invariants, a P-invariant covering all places ($\mathcal{I}_{(o)}$) can be obtained in the following way:

$\mathcal{I}_{(o)} = \mathcal{I}_{(o)(1)} + \mathcal{I}_{(o)(2)} + \mathcal{I}_{(o)(3)}$

$$\mathcal{I}_{(o)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$
$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$
$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T$$

Since $\mathcal{I}_{(o)} > 0$ and $\mathcal{I}_{(o)}^T \times A_o = 0$ ($A_o$ is the incidence matrix), the preemptive task structure block with overhead is structurally conservative as well as structurally bounded.

The preemptive task structure block with overhead and 2 voltages is a TPN $\mathcal{N}_{o2v} = (P_{o2v}, T_{o2v}, F_{o2v}, W_{o2v}, M_{0_{o2v}}, I_{o2v}, \mathcal{E}_{o2v}, \mathcal{CS}_{o2v})$, such that:

- $\diamondsuit\ P_{o2v} = \{p_{v_{i_j}}, p_{wg1_{i_j}}\} \cup P_{procT} \cup P_{procT_{ideal_i}} \cup \{p_{wo1_{i_j}}, p_{wc1_{i_j}}, p_{ac1_{i_j}}, p_{wlc1_{i_j}}, p_{fv1_{i_j}}, p_{proc}, p_{proc\_idle}, p_{wg_{i_j}}, p_{wov_{i_j}}, p_{procT_i\_2volt}, p_{wo_{i_j}}, p_{wc_{i_j}}, p_{ac_{i_j}}, p_{wlc_{i_j}}, p_{fv_{i_j}}, p_{fv_i}\}$. These places model the following:
  - $p_{v_{i_j}}$: voltage level $v_j$ selected;
  - $p_{wg1_{i_j}}$: waiting for processor granting considering the execution of the first part;
  - $p_{proc_{T_h}} \in P_{procT}$ : task $\tau_h$ executing, such that $1 \leq h \leq k$ and $k = |P_{procT}| = |\mathcal{T}|$. Note that $\mathcal{T}$ is the set of all tasks in the system;

- $p_{proc_{T_h}\_2volt} \in P_{procT_{ideal_i}}$: task $\tau_h$ executing at a second voltage level, such that $(|P_{procT_{ideal_i}}| = |\{\tau_h \in \mathcal{T} - \{\tau_i\} | \mathcal{V}_{ideal_h} \neq \emptyset\}|) \wedge (\exists p_{proc_{T_l}} \in P_{procT}, l = h \neq i)$. In other words, $P_{procT_{ideal_i}}$ is the set of places indicating a task, different from $\tau_i$, executing at a second voltage level;
- $p_{wo1_{i_j}}$: waiting for overhead occurrence considering the execution of the first part;
- $p_{wc1_{i_j}}$: waiting for task computation considering the execution of the first part;
- $p_{ac1_{i_j}}$: accumulating computation time units considering the execution of the first part;
- $p_{wlc1_{i_j}}$: waiting for the last computation time unit considering the execution of the first part;
- $p_{fv1_{i_j}}$: waiting for the conclusion of the first part ;
- $p_{proc}$: processor;
- $p_{proc\_idle}$: processor in idle state;
- $p_{wg_{i_j}}$: waiting for processor granting considering the execution of the second part;
- $p_{wov_{i_j}}$: waiting for the voltage transition for executing the second part;
- $p_{proc_{T_i}\_2volt}$: task $\tau_i$ executing at a second voltage level;
- $p_{wo_{i_j}}$: waiting for overhead occurrence considering the execution of the second part;
- $p_{wc_{i_j}}$: waiting for task computation considering the execution of the second part;
- $p_{ac_{i_j}}$: accumulating computation time units considering the execution of the second part;
- $p_{wlc_{i_j}}$: waiting for the last computation time unit considering the execution of the second part;
- $p_{fv_{i_j}}$: waiting for the conclusion of task execution at voltage $v_j$ ;
- $p_{fv_i}$: conclusion of a task instance.

♢ $T_{o2v} = \{t_{vs1_{i_j}}\} \cup T_{g1_{i_j}} \cup T_{g1_{ideal_{i_j}}} \cup \{t_{g1_{i_{j_0}}}, t_{o1_{i_j}}, t_{c1_{i_j}}, t_{l1_{i_j}}, t_{lc1_{i_j}}, t_{fv1_{i_j}}\} \cup T_{g_{i_j}} \cup T_{g_{ideal_{i_j}}} \cup \{t_{g_{i_{j_0}}}, t_{go_{i_j}}, t_{ov_{i_j}}, t_{o_{i_j}}, t_{c_{i_j}}, t_{l_{i_j}}, t_{lc_{i_j}}, t_{fv_{i_j}}\}$. These transitions model the following actions:

- $t_{vs_{i_j}}$: starting task execution at voltage $v_j$;
- $t_{g1_{i_{j_h}}} \in T_{g1_{i_j}}$: processor granting considering the execution of the first part, such that $1 \leq h \leq k$ and $k = |T_{g1_{i_j}}| = |\mathcal{T}|$. Note that $\mathcal{T}$ is the set of all tasks in the system;

- $t_{g2v1_{i_{j_h}}} \in T_{g1_{ideal_{i_j}}}$: processor granting considering the execution of the first part and taking into account other task execution at a second voltage level, such that $|T_{g1_{ideal_{i_j}}}| = |P_{procT_{ideal_i}}| = |\{\tau_h \in \mathcal{T} - \{\tau_i\} | \mathcal{V}_{ideal_h} \neq \emptyset\}|) \wedge h \neq i \wedge (\exists t_{g_{i_{j_l}}} \in T_{g1_{i_j}}, l = h)$. In other words, $T_{g1_{ideal_{i_j}}}$ is the set of transitions indicating a processor granting that will incur overheads due to a different task (than $\tau_i$) executing at a second voltage level;
- $t_{g1_{i_{j_0}}}$: processor granting considering the execution of the first part taking into account the CPU in the idle state;
- $t_{o1_{i_j}}$: overhead occurrence considering the execution of the first part;
- $t_{c1_{i_j}}$: executing one TTU considering the execution of the first part;
- $t_{l1_{i_j}}$: preparing last computation time unit considering the execution of the first part;
- $t_{lc1_{i_j}}$: executing last TTU considering the execution of the first part;
- $t_{fv1_{i_j}}$: finalizing the first part execution at voltage $v_{ideal_H}$;
- $t_{g_{i_{j_h}}} \in T_{g_{i_j}}$: processor granting considering the execution of the second part, such that $1 \leq h \leq k$ and $k = |T_{g_{i_j}}| = |\mathcal{T}|$. Note that $\mathcal{T}$ is the set of all tasks in the system;
- $t_{g2v_{i_{j_h}}} \in T_{g_{ideal_{i_j}}}$: processor granting taking into account other task execution at a second voltage level, such that $(|T_{g_{ideal_{i_j}}}| = |P_{procT_{ideal_i}}| = |\{\tau_h \in \mathcal{T} - \{\tau_i\} | \mathcal{V}_{ideal_h} \neq \emptyset\}|) \wedge h \neq i \wedge (\exists t_{g_{i_{j_l}}} \in T_{g_{i_j}}, l = h)$. In other words, $T_{g_{ideal_{i_j}}}$ is the set of transitions indicating a processor granting that will incur overheads due to a different task (than $\tau_i$) executing at a second voltage level. Note that $T_{g_{ideal_{i_j}}}$ is related to the execution of the second part of the task;
- $t_{g_{i_{j_0}}}$: processor granting considering the execution of the second part and taking into account the CPU in the idle state;
- $t_{go_{i_{j_0}}}$: processor granting considering the execution of the second part and taking into account the voltage switching overhead;
- $t_{o_{i_j}}$: overhead occurrence considering the execution of the second part;
- $t_{ov_{i_j}}$: voltage switching overhead considering the execution of the second part;
- $t_{c_{i_j}}$: executing one TTU considering the execution of the second part;
- $t_{l_{i_j}}$: preparing last computation time unit considering the execution of the second part;
- $t_{lc_{i_j}}$: executing last TTU considering the execution of the second part;
- $t_{fv_{i_j}}$: finalizing the second part execution at voltage $v_{ideal_L}$.

$\diamondsuit$ Pre and post-conditions of the transitions are (flow relation $F_{o2v}$):

- $\bullet t_{vs_{i_j}} = \{p_{v_{i_j}}\}; \quad t_{vs_{i_j}}\bullet = \{p_{wg1_{i_j}}\};$
- $\forall t_{g1_{i_{j_h}}} \in T_{g1_{i_j}}, h \neq i, \bullet t_{g1_{i_{j_h}}} = \{p_{wg1_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}} \in P_{procT} | l = h\}; \; t_{g1_{i_{j_h}}}\bullet = \{p_{wo1_{i_j}}\};$
- $\exists t_{g1_{i_{j_h}}} \in T_{g1_{i_j}}, h = i, \bullet t_{g_{i_{j_h}}} = \{p_{wg1_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}} \in P_{procT} | l = h = i\}; t_{g1_{i_{j_h}}}\bullet = \{p_{wc1_{i_j}}\};$
- $\forall t_{g2v1_{i_{j_h}}} \in T_{g1_{ideal_{i_j}}}, \bullet t_{g2v1_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}\_2volt} \in P_{procT_{ideal_i}} | l = h\}; t_{g2v1_{i_{j_h}}}\bullet = \{p_{wo1_{i_j}}\};$
- $\bullet t_{g1_{i_{j_0}}} = \{p_{wg1_{i_j}}, p_{proc}, p_{proc\_idle}\}; \quad t_{g1_{i_{j_0}}}\bullet = \{p_{wo1_{i_j}}\};$
- $\bullet t_{o1_{i_j}} = \{p_{wo1_{i_j}}\}; t_{o1_{i_j}}\bullet = \{p_{wc1_{i_j}}\};$
- $\bullet t_{c1_{i_j}} = \{p_{wc1_{i_j}}\}; \quad t_{c1_{i_j}}\bullet = \{p_{proc}, p_{proc_{T_i}}, p_{ac1_{i_j}}\};$
- $\bullet t_{l1_{i_j}} = \{p_{wc1_{i_j}}, p_{ac1_{i_j}}\}; t_{l1_{i_j}}\bullet = \{p_{wlc1_{i_j}}\};$
- $\bullet t_{lc1_{i_j}} = \{p_{wlc1_{i_j}}\}; t_{lc1_{i_j}}\bullet = \{p_{proc}, p_{fv1_{i_j}}\};$
- $\bullet t_{fv1_{i_j}} = \{p_{fv1_{i_j}}\}; t_{fv1_{i_j}}\bullet = \{p_{proc_{T_h}\_2volt}, p_{wg_{i_j}}\};$
- $\forall t_{g_{i_{j_h}}} \in T_{g_{i_j}}, h \neq i, \bullet t_{g_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}} \in P_{procT} | l = h\}; \quad t_{g_{i_{j_h}}}\bullet = \{p_{wo_{i_j}}\};$
- $\exists t_{g_{i_{j_h}}} \in T_{g_{i_j}}, h = i, \bullet t_{g_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}} \in P_{procT} | l = h = i\}; t_{g_{i_{j_h}}}\bullet = \{p_{wc_{i_j}}\};$
- $\forall t_{g2v_{i_{j_h}}} \in T_{g_{ideal_{i_j}}}, \bullet t_{g2v_{i_{j_h}}} = \{p_{wg_{i_j}}, p_{proc}\} \cup \{p_{proc_{T_l}\_2volt} \in P_{procT_{ideal_i}} | l = h\}; t_{g2v_{i_{j_h}}}\bullet = \{p_{wo_{i_j}}\};$
- $\bullet t_{g_{i_{j_0}}} = \{p_{wg_{i_j}}, p_{proc}, p_{proc\_idle}\}; \quad t_{g_{i_{j_0}}}\bullet = \{p_{wo_{i_j}}\};$
- $\bullet t_{go_{i_j}} = \{p_{wg_{i_j}}, p_{proc}, p_{proc_{T_i}\_2volt}\}; \quad t_{g_{i_{j_0}}}\bullet = \{p_{wov_{i_j}}\};$
- $\bullet t_{o_{i_j}} = \{p_{wo_{i_j}}\}; t_{o_{i_j}}\bullet = \{p_{wc_{i_j}}\};$
- $\bullet t_{ov_{i_j}} = \{p_{wov_{i_j}}\}; t_{o_{i_j}}\bullet = \{p_{wc_{i_j}}\};$
- $\bullet t_{c_{i_j}} = \{p_{wc_{i_j}}\}; \quad t_{c_{i_j}}\bullet = \{p_{proc}, p_{proc_{T_i}}, p_{ac_{i_j}}\};$
- $\bullet t_{l_{i_j}} = \{p_{wc_{i_j}}, p_{ac_{i_j}}\}; t_{l_{i_j}}\bullet = \{p_{wlc_{i_j}}\};$
- $\bullet t_{lc_{i_j}} = \{p_{wlc_{i_j}}\}; t_{lc_{i_j}}\bullet = \{p_{proc}, p_{fv_{i_j}}\};$
- $\bullet t_{fv_{i_j}} = \{p_{fv_{i_j}}\}; t_{fv_{i_j}}\bullet = \{p_{proc\_idle}, p_{fv_i}\};$

$$\diamondsuit \; \forall (x,y) \in F_{o2v}, W_{o2v}(x,y) = \begin{cases} C1_{i_j} & if \; (x,y) = (t_{vs_{i_j}}, p_{wg1_{i_j}}) \\ C1_{i_j} - 1 & if \; (x,y) = (p_{ac1_{i_j}}, t_{l1_{i_j}}) \\ C2_{i_j} & if \; (x,y) = (t_{fv1_{i_j}}, p_{wg_{i_j}}) \\ C2_{i_j} - 1 & if \; (x,y) = (p_{ac_{i_j}}, t_{l_{i_j}}) \\ 1 & otherwise \end{cases}$$

♢ $M_{0_o}(p_x) = 0, \forall p_x \in P_{o2v}, p_x \notin \{p_{proc}, p_{proch\_idle}\}; M_{0_o}(p_{proc}) = M_{0_o}(p_{proch\_idle}) = 1;$

♢ $I_{o2v}(t_x) = [0,0], \forall t_x \in T_{o2v}, t_x \notin \{t_{o1_{i_j}}, t_{c1_{i_j}}, t_{lc1_{i_j}}, t_{o_{i_j}}, t_{ov_{i_j}}, t_{c_{i_j}}, t_{lc_{i_j}}\}$;
$I_{o2v}(t_{o1_{i_j}}) = I_{o2v}(t_{o_{i_j}}) = [o,o]$. $o$ is the dispatcher WCET; $I_{o2v}(t_{ov_{i_j}}) = [a_v, a_v]$. $a_v$ is the dispatcher execution time to only adjust the voltage level and the respective maximum CPU frequency (Section 5.2.2);
$I_{o2v}(t_{c1_{i_j}}) = I_{o2v}(t_{lc1_{i_j}}) = I_{o2v}(t_{c_{i_j}}) = I_{o2v}(t_{lc_{i_j}}) = [1,1]$;

♢ $\mathcal{E}_{o2v}(t_x) = 0.0\text{J}, \forall t_x \in T_p, t_x \notin \{t_{o1_{i_j}}, t_{c1_{i_j}}, t_{lc1_{i_j}}, t_{o_{i_j}}, t_{ov_{i_j}}, t_{c_{i_j}}, t_{lc_{i_j}}\}$;
$\mathcal{E}_{o2v}(t_{o1_{i_j}}) = \mathcal{E}_{o2v}(t_{o_{i_j}}) = o_{\mathcal{E}}$. $o_{\mathcal{E}}$ is the dispatcher energy consumption (Section 5.2.2);
$\mathcal{E}_{o2v}(t_{gov_{i_j}}) = a_{v_{\mathcal{E}}}$. $a_{v_{\mathcal{E}}}$ is the dispatcher energy consumption to just perform a voltage/frequency switching;
$\mathcal{E}_{o2v}(t_{c1_{i_j}}) = \mathcal{E}_{o2v}(t_{lc1_{i_j}}) = (vef(v_{ideal_H}) \times c_1)/C1_{i_j}$;
$\mathcal{E}_{o2v}(t_{c_{i_j}}) = \mathcal{E}_{o2v}(t_{lc_{i_j}}) = (vef(v_{ideal_L}) \times c_2)/C2_{i_j}$. $vef$ is the voltage-energy function. See Section 5.2.4;

♢ $\mathcal{CS}_{o2v}(t_{c1_{i_j}}) = \mathcal{CS}_{o2v}(t_{c_{i_j}}) = \mathcal{CS}_o(t_{lc1_{i_j}}) = \mathcal{CS}_o(t_{lc_{i_j}}) = code_i$, in which $code_i \in \mathcal{ST}$.

Additionally, the preemptive task structure with overhead and 2 voltages block is covered by 3 basic P-invariants, which are depicted as follows:

♢ $\mathcal{I}_{(o2v)(1)} =$

| $p_{v_{i_j}}$ | $p_{wg1_{i_j}}$ | $p_{wo1_{i_j}}$ | $p_{wc1_{i_j}}$ | $p_{ac1_{i_j}}$ | $p_{wlc1_{i_j}}$ | $p_{fv1_{i_j}}$ | $p_{proc_{T_1}}$ | $\cdots$ | $p_{proc_{T_i}}$ | $\cdots$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | $proc$ | $proc$ | 0 | $proc$ | 0 | 0 | 0 | 0 | 0 |

| $p_{proc_{T_k}}$ | $p_{proc}$ | $p_{proc\_idle}$ | $p_{wg_{i_j}}$ | $p_{wo_{i_j}}$ | $p_{wc_{i_j}}$ | $p_{ac_{i_j}}$ | $p_{wlc_{i_j}}$ | $p_{fv_{i_j}}$ | $p_{fv_i}$ | $p_{wov_{i_j}}$ | $p_{proc_{T_i\_2volt}}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $proc$ | 0 | 0 | $proc$ | $proc$ | 0 | $proc$ | 0 | 0 | $proc$ | 0 |

$]^T$;

♢ $\mathcal{I}_{(o2v)(2)} =$

| $p_{v_{i_j}}$ | $p_{wg1_{i_j}}$ | $p_{wo1_{i_j}}$ | $p_{wc1_{i_j}}$ | $p_{ac1_{i_j}}$ | $p_{wlc1_{i_j}}$ | $p_{fv1_{i_j}}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | $proc\_idle$ | $proc\_idle$ | 0 | $proc\_idle$ | $proc\_idle$ |

| $p_{proc_{T1}}$ | $\cdots$ | $p_{proc_{T_i}}$ | $\cdots$ | $p_{proc_{T_k}}$ | $p_{proc}$ | $p_{proc\_idle}$ | $p_{wg_{i_j}}$ | $p_{wo_{i_j}}$ |
|---|---|---|---|---|---|---|---|---|
| $proc\_idle$ | $\cdots$ | $proc\_idle$ | $\cdots$ | $proc\_idle$ | 0 | $proc\_idle$ | 0 | $proc\_idle$ |

| $p_{wc_{i_j}}$ | $p_{ac_{i_j}}$ | $p_{wlc_{i_j}}$ | $p_{fv_{i_j}}$ | $p_{fv_i}$ | $p_{wov_{i_j}}$ | $p_{proc_{T_i\_2volt}}$ |
|---|---|---|---|---|---|---|
| $proc\_idle$ | 0 | $proc\_idle$ | $proc\_idle$ | 0 | $proc\_idle$ | $proc\_idle$ |

$]^T$;

♢ $\mathcal{I}_{(o2v)(3)} =$

| $p_{v_{i_j}}$ | $p_{wg1_{i_j}}$ | $p_{wo1_{i_j}}$ | $p_{wc1_{i_j}}$ |
|---|---|---|---|
| $(C_{1_{i_j}} C_{2_{i_j}}) wg1_{i_j}$ | $(C_{2_{i_j}}) wg1_{i_j}$ | $(C_{2_{i_j}}) wg1_{i_j}$ | $(C_{2_{i_j}}) wg1_{i_j}$ |

| $p_{ac1_{i_j}}$ | $p_{wlc1_{i_j}}$ | $p_{fv1_{i_j}}$ | $p_{proc_{T_1}}$ | $\cdots$ | $p_{proc_{T_i}}$ | $\cdots$ |
|---|---|---|---|---|---|---|
| $(C_{2_{i_j}}) wg1_{i_j}$ | $(C_{1_{i_j}} C_{2_{i_j}}) wg1_{i_j}$ | $(C_{1_{i_j}} C_{2_{i_j}}) wg1_{i_j}$ | 0 | $\cdots$ | 0 | $\cdots$ |

| $p_{proc_{T_k}}$ | $p_{proc}$ | $p_{proc\_idle}$ | $p_{wg_{i_j}}$ | $p_{wo_{i_j}}$ | $p_{wc_{i_j}}$ | $p_{ac_{i_j}}$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | $(C_{1_{i_j}}) wg1_{i_j}$ | $(C_{1_{i_j}}) wg1_{i_j}$ | $(C_{1_{i_j}}) wg1_{i_j}$ | $(C_{1_{i_j}}) wg1_{i_j}$ |

$$\overset{p_{wlc_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i\_2volt}}}{0} \Big]^T.$$

in which $proc, proc\_idle, wg1_{i_j} \in \mathbb{N}^*$. For the sake of readability, all places in $P_{procT_{ideal_i}}$ are hidden in the basic P-invariants presented. Nevertheless, assume they are contained in the set $P_{procT}$, since all places in $P_{procT_{ideal_i}}$ are covered by the same basic P-invariant as the places in $P_{procT}$. A P-invariant covering all places ($\mathcal{I}_{(f)}$) is presented as follows:

$\mathcal{I}_{(o2v)} = \mathcal{I}_{(o2v)(1)} + \mathcal{I}_{(o2v)(2)} + \mathcal{I}_{(o2v)(3)}$

$$\mathcal{I}_{(o2v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg_{i_j}}$$

$$\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$

$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i\_2volt}}}{proc\_idle} \Big]^T$$

As $\mathcal{I}_{(o2v)} > 0$ and $\mathcal{I}^T_{(o2v)} \times A_{o2v} = 0$ ($A_{o2v}$ is the incidence matrix), $\mathcal{N}_{(o2v)}$ is structurally conservative as well as structurally bounded.

### 6.3.11 Blocks for Modeling Intertask Relations

The proposed modeling approach adopts three building blocks for representing intertask relations: (i) precedence pre-condition, (ii) exclusion pre-condition, and (iii) task instance conclusion with intertask relations. The first two blocks model, respectively, the pre-conditions of precedence and exclusion relations, whereas the new task instance conclusion block represents the post-conditions. These blocks are presented as follows.

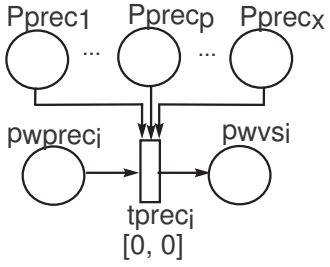


**Figure 6.15** Precedence pre-condition block

Precedence pre-condition block (Figure 6.15) is a TPN $\mathcal{N}_{iprep} = (P_{iprep}, T_{iprep}, F_{iprep}, W_{iprep}, M_{0_{iprep}}, I_{iprep}, \mathcal{E}_{iprep}, \mathcal{CS}_{iprep})$, such that:

♢ $P_{iprep} = \{p_{wprec_i}, p_{wvs_i}\} \cup P_{pre}$. These places model the following situations:

- $p_{wprec_i}$: waiting precedence pre-condition;
- $p_{wvs_i}$: waiting for voltage selection;
- $p_{pre_p} \in P_{pre}$: precedence pre-condition, in which $|P_{pre}| = x$ ($x$ is the number of tasks that precedes task $\tau_i$).

♢ $T_{iprep} = \{t_{prec_i}\}$. This transition models the following action:

- $t_{prec_i}$: acknowledging precedence pre-condition.

♢ Pre and post-conditions of the transition are (flow relation $F_{iprep}$):

- $\bullet t_{prec_i} = \{p_{wprec_i}\} \cup P_{pre}$; $t_{prec_i}\bullet = \{p_{wvs_i}\}$.

♢ $\forall (x, y) \in F_{iprep}, W_{iprep}(x, y) = 1$;

♢ $\forall p \in P_{iprep}, M_{0_{iprep}}(p) = 0$;

♢ $I_{iprep}(t_{prec_i}) = [0, 0]$;

♢ $\mathcal{E}_{iprep}(t_{prec_i}) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_{iprep}$ as a set, $\mathcal{CS}_{iprep} = \emptyset$.

Precedence pre-condition block is covered by $x + 1$ basic P-invariants ($x = |P_{pre}|$). See below:

♢ $\mathcal{I}_{(iprep)(1)} = \overset{\begin{matrix} p_{wprec_i} & p_{wvs_i} & p_{prec_1} & \cdots & p_{prec_p} & \cdots & p_{prec_x} \end{matrix}}{\left[\begin{matrix} prec_i & prec_i & 0 & \cdots & 0 & \cdots & 0 \end{matrix}\right]^T}$;

♢ $\mathcal{I}_{(iprep)(2)} = \overset{\begin{matrix} p_{wprec_i} & p_{wvs_i} & p_{prec_1} & \cdots & p_{prec_p} & \cdots & p_{prec_x} \end{matrix}}{\left[\begin{matrix} 0 & prec_{1_i} & prec_{1_i} & \cdots & 0 & \cdots & 0 \end{matrix}\right]^T}$;
⋮

♢ $\mathcal{I}_{(iprep)(p+1)} = \overset{\begin{matrix} p_{wprec_i} & p_{wvs_i} & p_{prec_1} & \cdots & p_{prec_p} & \cdots & p_{prec_x} \end{matrix}}{\left[\begin{matrix} 0 & prec_{p_i} & 0 & \cdots & prec_{p_i} & \cdots & 0 \end{matrix}\right]^T}$;
⋮

♢ $\mathcal{I}_{(iprep)(x+1)} = \overset{\begin{matrix} p_{wprec_i} & p_{wvs_i} & p_{prec_1} & \cdots & p_{prec_p} & \cdots & p_{prec_x} \end{matrix}}{\left[\begin{matrix} 0 & prec_{x_i} & 0 & \cdots & 0 & \cdots & prec_{x_i} \end{matrix}\right]^T}$.

in which $prec_i, prec_{1_i}, \cdots, prec_{p_i}, \cdots, prec_{x_i} \in \mathbb{N}^*$ A P-invariant covering all places can be obtained in the following way:

$$\mathcal{I}_{(iprep)} = \mathcal{I}_{(iprep)(1)} + \mathcal{I}_{(iprep)(2)} + \cdots + \mathcal{I}_{(iprep)(p+1)} + \cdots + \mathcal{I}_{(iprep)(x+1)}$$

$$\mathcal{I}_{(iprep)} = \begin{matrix} p_{wprec_i} & & p_{wvs_i} & & p_{prec_1} \\ \left[ \ wprec_i & wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i} & prec_{1_i} \right. \\ \cdots & p_{prec_p} \quad \cdots \quad p_{prec_x} & \\ \cdots & prec_{p_i} \quad \cdots \quad prec_{x_i} \ \left. \right]^T & \end{matrix}$$

As $\mathcal{I}^T_{(iprep)} \times A_{iprep} = 0$, in which $A_{iprep}$ is the respective incidence matrix, and $\mathcal{I}_{(iprep)} > 0$, the precedence pre-condition block is structurally conservative and structurally bounded.

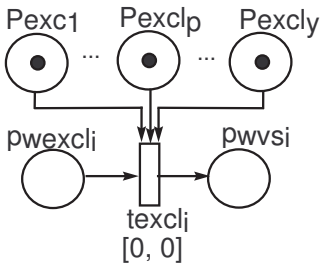


**Figure 6.16** Exclusion pre-condition block

Exclusion pre-condition block (Figure 6.16) is a TPN $\mathcal{N}_{ipree} = (P_{ipree}, T_{ipree}, F_{ipree}, W_{ipree}, M_{0_{ipree}}, I_{ipree}, \mathcal{E}_{ipree}, \mathcal{CS}_{ipree})$, such that:

♢ $P_{ipree} = \{p_{wexcl_i}, p_{wvs_i}\} \cup P_{excl}$. These places model the following situations:

- $p_{wexcl_i}$: waiting exclusion pre-condition;
- $p_{wvs_i}$: waiting for voltage selection;
- $p_{excl_p} \in P_{excl}$: exclusion pre-condition, in which $|P_{excl}| = y$ ($y$ is the number of tasks that excludes or excluded by task $\tau_i$).

♢ $T_{ipree} = \{t_{excl_i}\}$. This transition models the following action:

- $t_{prec_i}$: acknowledging exclusion pre-condition.

♢ Pre and post-conditions of the transition are (flow relation $F_{ipree}$):

- $\bullet t_{excl_i} = \{p_{wexcl_i}\} \cup P_{excl}$; $t_{excl_i} \bullet = \{p_{wvs_i}\}$;

♢ $\forall (x, y) \in F_{ipree}, W_{ipree}(x, y) = 1$;

♢ $M_{0_{ipree}}(p) = 0, \forall p \in P_{ipree}, p \notin P_{excl},; \forall p \in P_{excl}, M_{0_{ipree}}(p) = 1$;

♢ $I_{ipree}(t_{excl_i}) = [0, 0]$;

♢ $\mathcal{E}_{ipree}(t_{excl_i}) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_{ipree}$ as a set, $\mathcal{CS}_{ipree} = \emptyset$.

Exclusion pre-condition block is covered by $y+1$ basic P-invariants ($y = |P_{excl}|$):

$$\diamondsuit\ \mathcal{I}_{(ipree)(1)} = \overset{\begin{matrix} p_{wexcl_i} & p_{wvs_i} & p_{excl_1} & \cdots & p_{excl_p} & \cdots & p_{excl_y} \end{matrix}}{\left[\begin{matrix} wexcl_i & wexcl_i & 0 & \cdots & 0 & \cdots & 0 \end{matrix}\right]^T};$$

$$\diamondsuit\ \mathcal{I}_{(ipree)(2)} = \overset{\begin{matrix} p_{wexcl_i} & p_{wvs_i} & p_{excl_1} & \cdots & p_{excl_p} & \cdots & p_{excl_y} \end{matrix}}{\left[\begin{matrix} 0 & excl_{1_i} & excl_{1_i} & \cdots & 0 & \cdots & 0 \end{matrix}\right]^T};$$

$$\vdots$$

$$\diamondsuit\ \mathcal{I}_{(ipree)(p+1)} = \overset{\begin{matrix} p_{wexcl_i} & p_{wvs_i} & p_{excl_1} & \cdots & p_{excl_p} & \cdots & p_{excl_y} \end{matrix}}{\left[\begin{matrix} 0 & excl_{p_i} & 0 & \cdots & excl_{p_i} & \cdots & 0 \end{matrix}\right]^T};$$

$$\vdots$$

$$\diamondsuit\ \mathcal{I}_{(ipree)(y+1)} = \overset{\begin{matrix} p_{wexcl_i} & p_{wvs_i} & p_{excl_1} & \cdots & p_{excl_p} & \cdots & p_{excl_y} \end{matrix}}{\left[\begin{matrix} 0 & excl_{y_i} & 0 & \cdots & 0 & \cdots & excl_{y_i} \end{matrix}\right]^T}.$$

in which $excl_i, excl_{1_i}, \cdots, excl_{p_i}, \cdots, excl_{y_i} \in \mathbb{N}^*$ A P-invariant covering all places can be obtained in the following way:

$\mathcal{I}_{(ipree)} = \mathcal{I}_{(ipree)(1)} + \mathcal{I}_{(ipree)(2)} + \cdots + \mathcal{I}_{(ipree)(p+1)} + \cdots + \mathcal{I}_{(ipree)(y+1)}$

$$\mathcal{I}_{(ipree)} = \overset{\begin{matrix} p_{wexcl_i} & & p_{wvs_i} & & p_{excl_1} \end{matrix}}{\left[\begin{matrix} wexcl_i & wexcl_i + excl_{1_i} + \cdots + excl_{p_i} + \cdots + excl_{y_i} & excl_{1_i} \end{matrix}\right.}$$
$$\overset{\begin{matrix} \cdots & p_{excl_p} & \cdots & p_{excl_y} \end{matrix}}{\left.\begin{matrix} \cdots & excl_{p_i} & \cdots & excl_{y_i} \end{matrix}\right]^T}$$

As $\mathcal{I}_{(ipree)} > 0$ and $\mathcal{I}^T_{(ipree)} \times A_{ipree} = 0$, in which $A_{ipree}$ is the respective incidence matrix, the exclusion pre-condition is structurally conservative and structurally bounded.

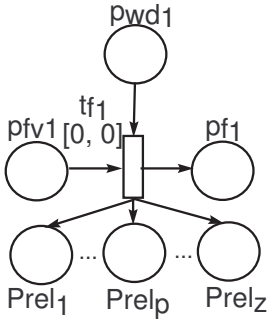


**Figure 6.17** Task instance conclusion with intertask relations block

Task instance conclusion with intertask relations block (Figure 6.17) is a TPN $\mathcal{N}_{cinter} = (P_{cinter}, T_{cinter}, F_{cinter}, W_{cinter}, M_{0_{cinter}}, I_{cinter}, \mathcal{E}_{cinter}, \mathcal{CS}_{cinter})$, such that:

♢ $P_{cinter} = \{p_{fv_i}, p_{f_i}, p_{wd_i}\} \cup P_{rel}$. These places model the following situations:

  - $p_{fv_i}$: task instance conclusion;
  - $p_{f_i}$: end of task execution;
  - $p_{wd_i}$: waiting for deadline missing;
  - $p_{rel_p} \in P_{rel}$: intertask relation (exclusion or precedence) post-condition, in which $|P_{rel}| = z$ ($z$ is the number of tasks preceded and/or excluded by task $\tau_i$).

♢ $T_{cinter} = \{t_{f_i}\}$. This transition models the following action:

  - $t_{f_i}$: concluding task instance execution.

♢ Pre and post-conditions of the transition are (flow relation $F_{cinter}$):

  - $\bullet t_{f_i} = \{p_{fv_i}, p_{wd_i}\}$; $t_{f_i}\bullet = \{p_{f_i}\} \cup P_{rel}$;

♢ $\forall (x, y) \in F_{cinter}, W_{cinter}(x, y) = 1$;

♢ $\forall p \in P_{cinter}, M_{0_{cinter}}(p) = 0$;

♢ $I_{cinter}(t_{f_i}) = [0, 0]$;

♢ $\mathcal{E}_{cinter}(t_{f_i}) = 0.0\text{J}$;

♢ Representing $\mathcal{CS}_{cinter}$ as a set, $\mathcal{CS}_{cinter} = \emptyset$.

The task instance conclusion with intertask relations block is covered by $2 \times (z + 1)$ basic P-invariants ($z = |P_{rel}|$), which are described as follows:

♢ $\mathcal{I}_{(cinter)(pf1)} = \overset{\begin{matrix} p_{fv_i} & p_{f_i} & p_{pwd_i} & p_{rel_1} & \cdots & p_{rel_p} & \cdots & p_{rel_z} \end{matrix}}{\begin{bmatrix} pf1_i & pf1_i & 0 & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}^T}$;

♢ $\mathcal{I}_{(cinter)(pf)} = \overset{\begin{matrix} p_{fv_i} & p_{f_i} & p_{pwd_i} & p_{rel_1} & \cdots & p_{rel_p} & \cdots & p_{rel_z} \end{matrix}}{\begin{bmatrix} 0 & pf_i & pf_i & 0 & \cdots & 0 & \cdots & 0 \end{bmatrix}^T}$;

♢ $\mathcal{I}_{(cinter)(rel1_1)} = \overset{\begin{matrix} p_{fv_i} & p_{f_i} & p_{pwd_i} & p_{rel_1} & \cdots & p_{rel_p} & \cdots & p_{rel_z} \end{matrix}}{\begin{bmatrix} rel1_1 & 0 & 0 & rel1_1 & \cdots & 0 & \cdots & 0 \end{bmatrix}^T}$;

♢ $\mathcal{I}_{(cinter)(rel_1)} = \overset{\begin{matrix} p_{fv_i} & p_{f_i} & p_{pwd_i} & p_{rel_1} & \cdots & p_{rel_p} & \cdots & p_{rel_z} \end{matrix}}{\begin{bmatrix} 0 & 0 & rel_1 & rel_1 & \cdots & 0 & \cdots & 0 \end{bmatrix}^T}$;

⋮

♢ $\mathcal{I}_{(cinter)(rel1_p)} = \overset{\begin{matrix} p_{fv_i} & p_{f_i} & p_{pwd_i} & p_{rel_1} & \cdots & p_{rel_p} & \cdots & p_{rel_z} \end{matrix}}{\begin{bmatrix} rel1_p & 0 & 0 & 0 & \cdots & rel1_p & \cdots & 0 \end{bmatrix}^T}$;

♢ $\mathcal{I}_{(cinter)(rel_p)} = \overset{\begin{matrix} p_{fv_i} & p_{f_i} & p_{pwd_i} & p_{rel_1} & \cdots & p_{rel_p} & \cdots & p_{rel_z} \end{matrix}}{\begin{bmatrix} 0 & 0 & rel_p & 0 & \cdots & rel_p & \cdots & 0 \end{bmatrix}^T}$;

⋮

$$\diamondsuit\ \mathcal{I}_{(cinter)(rel1_z)} = \overset{\quad p_{fv_i} \quad p_{f_i} \quad p_{pwd_i} \quad p_{rel_1} \quad \cdots \quad p_{rel_p} \quad \cdots \quad p_{rel_z}}{\left[\ rel1_z \quad 0 \quad 0 \quad 0 \quad \cdots \quad 0 \quad \cdots \quad rel1_z\ \right]^T};$$

$$\diamondsuit\ \mathcal{I}_{(cinter)(rel_z)} = \overset{\quad p_{fv_i} \quad p_{f_i} \quad p_{pwd_i} \quad p_{rel_1} \quad \cdots \quad p_{rel_p} \quad \cdots \quad p_{rel_z}}{\left[\ 0 \quad 0 \quad rel_z \quad 0 \quad \cdots \quad 0 \quad \cdots \quad rel_z\ \right]^T}.$$

in which $pf1_i, pf_i, \cdots, rel1_p, rel_p, \cdots, rel1_z, rel_z \in \mathbb{N}^*$. Using these basic invariants, a P-invariant covering all places ($\mathcal{I}_{(cinter)}$) can be obtained in the following way:

$\mathcal{I}_{(cinter)} = \mathcal{I}_{(cinter)(pf1)} + \mathcal{I}_{(cinter)(pf)} + \mathcal{I}_{(cinter)(rel1_1)} + \mathcal{I}_{(cinter)(rel_1)} + \cdots + \mathcal{I}_{(cinter)(rel1_p)} + \mathcal{I}_{(cinter)(rel_p)} + \cdots + \mathcal{I}_{(cinter)(rel1_z)} + \mathcal{I}_{(cinter)(rel_z)}$

$$\mathcal{I}_{(cinter)} = \Big[\ \overset{\quad p_{fv_i}}{pf1_i + rel1_i + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{\quad p_{f_i}}{pf1_i + pf_i}$$
$$\overset{\quad p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{\quad p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{\quad p_{rel_p}}{rel1_p + rel_p}$$
$$\overset{\cdots \qquad p_{rel_z}}{\cdots \quad rel1_z + rel_z\ \Big]^T}$$

Since $\mathcal{I}_{(cinter)} > 0$ and $\mathcal{I}^T_{(cinter)} \times A_{cinter} = 0$ ($A_{cinter}$ is the incidence matrix), the task instance conclusion with intertask relations block is structurally conservative as well as structurally bounded.

## 6.4 SUMMARY

This chapter initiated the presentation of the adopted formal approach for modeling hard real-time systems with energy constraints. The proposed approach utilizes a formal model based on a time Petri net extension, which allows the proper representation of intertask relations as well as timing and energy constraints. Next chapter describes the adopted composition rules for combining building block models and some modeling examples.

CHAPTER 7

# MODELING - COMPOSITION RULES AND EXAMPLES

This chapter describes the formal composition rules adopted in the modeling activity as well as some modeling examples. In the presentation of the composition rules, the respective properties are shown and proved. After that, modeling examples are presented and, finally, this chapter discusses about the analysis and verification of properties related to the generated models. Similar to previous chapter, the bullet is replaced by an open diamond in the itemized lists.

## 7.1 COMPOSITION RULES

The proposed modeling approach adopts two composition operators: net union and place renaming. Net union operator takes two TPNEs as input and generates a new one by merging the common places (if exist) of the operand nets. Place renaming is an auxiliary operator that takes a TPNE and renames a subset of its respective places in order to allow, for specific cases, net unions. Firstly, place renaming is presented, and, next, net union is detailed.

### 7.1.1 Place Renaming Operator

This section defines the place renaming operator (which is based on [112]) as well as depicts an example.

**Definition 7.1** (Place Renaming)**.** *Place renaming is a function denoted by* $\rho : P \rightarrow P'$*, which renames the places of a TPNE* $\mathcal{N}$ *= (P, T, F, W,* $m_0$*, I,* $\mathcal{E}$*,* $\mathcal{CS}$*), such that a new isomorphic TPNE* $\mathcal{N}$*' = (P', T', F', W',* $m_0$*', I',* $\mathcal{E}$*',* $\mathcal{CS}$*') is obtained:*

- $P' = \{\rho(p) | \forall p \in P\}$;
- $T' = T$;
- $|F'| = |F|$, *such that*
  - $\forall (p,t) \in F, (\rho(p),t) \in F'$, *in which* $p \in P \wedge t \in T \wedge t \in T' \wedge \rho(p) \in P'$;
  - $\forall (t,p) \in F, (t,\rho(p)) \in F'$, *in which* $p \in P \wedge t \in T \wedge t \in T' \wedge \rho(p) \in P'$;
- $|W'| = |W|$, *such that*
  - $\forall (p,t) \in F, W'(\rho(p),t) = W(p,t)$, *in which* $p \in P \wedge t \in T \wedge t \in T' \wedge \rho(p) \in P'$;
  - $\forall (t,p) \in F, W'(t,\rho(p)) = W(t,p)$, *in which* $p \in P \wedge t \in T \wedge t \in T' \wedge \rho(p) \in P'$;

$\diamondsuit$ $\forall p \in P, m'_0(\rho(p)) = m_0(p)$;

$\diamondsuit$ $I' = I$

$\diamondsuit$ $\mathcal{E}' = \mathcal{E}$

$\diamondsuit$ $\mathcal{CS}' = \mathcal{CS}$

*Notation $\mathcal{N}' = \mathcal{N}/\rho$ is adopted to represent the application of function $\rho$ in $\mathcal{N}$, obtaining as result $\mathcal{N}'$.*

In this work, $\rho$ is a bijective function, allowing the construction of the respective inverse operation. Besides, if function $\rho$ is defined as $\rho(p) = p$, $\forall p \in P$, the result is $\mathcal{N}' = \mathcal{N}$.

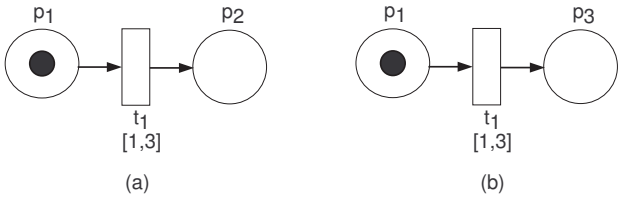


**Figure 7.1** Place renaming example

Take as an example the TPNE $\mathcal{N}_1$ presented in Figure 7.1(a). Additionally, consider:

$$\rho(p) = \begin{cases} p_3, & if\ (p = p_2) \\ p, & otherwise \end{cases}$$

The result of $\mathcal{N}_2 = \mathcal{N}_1/\rho$ is depicted in Figure 7.1(b).

### 7.1.2 Net Union Operator

This section describes the net union operator, which performs the fusion of common places (if exist) of the operand nets. The proposed operator is based on [113] and it has been adjusted to consider the characteristics of the adopted building blocks. Besides, properties related to the operator are presented as well as mathematically proved.

Initially, two sets are presented, which are adopted to define the domain and codomain of net union operator.

**Definition 7.2** (Set of all TPNEs - TPNESet)**.** *Let $TPNESet = \{\mathcal{N}_i | i \in \mathbb{N}^*\}$, in which $\mathcal{N}_i = (P_i, T_i, F_i, W_i, m_{0_i}, I_i, \mathcal{E}_i, \mathcal{CS}_i)$ is a TPN with energy consumption values and code annotations. TPNESet is the set of all TPNs with energy consumption values and code annotations (TPNE).*

**Definition 7.3** (Restricted TPNESet - RTPNESet)**.** *Let $RTPNESet \subset TPNESet$ be a restricted set of TPNEs, such that $\forall \mathcal{N}_i, \mathcal{N}_n \in RTPNESet, (P_i \cap P_n = \emptyset \wedge T_i \cap T_n = \emptyset) \vee (P_i \cap P_n \neq \emptyset \wedge T_i \cap T_n = \emptyset) \vee (P_i \cap P_n \neq \emptyset \wedge T_i \cap T_n \neq \emptyset \leftrightarrow \forall t \in T_i \cap T_n, I_i(t) =$*

*$I_n(t) \wedge \mathcal{E}_i(t) = \mathcal{E}_n(t) \wedge \mathcal{CS}_i(t) = \mathcal{CS}_n(t)$), in which $\mathcal{N}_i$= $(P_i, T_i, F_i, W_i, m_{0_i}, I_i, \mathcal{E}_i, \mathcal{CS}_i)$ and $\mathcal{N}_n$= $(P_n, T_n, F_n, W_n, m_{0_n}, I_n, \mathcal{E}_n, \mathcal{CS}_n)$ are two TPNEs. RTPNESet is a subset of TPNESet considering (i) disjoint nets, (2) nets with disjoint sets of transitions and (3) nets with common places and common transitions with the same timing constraints, energy consumption values and source codes.*

In the following definition, functions $W : A \to \mathbb{N}$, $I : T \to \mathbb{N} \times \mathbb{N}$, $\mathcal{E}$:$T \to \mathbb{R}_+ \cup \{0\}$ and $\mathcal{CS}$:$T \to \mathcal{ST}$ are represented as sets, more specifically, $W \subseteq A \times \mathbb{N}$, $I \subseteq T \times \mathbb{N} \times \mathbb{N}$, $\mathcal{E} \subseteq T \times \mathbb{R}_+ \cup \{0\}$ and $\mathcal{CS} \subseteq T \times \mathcal{ST}$.

**Definition 7.4** (Net Union)**.** *Net union is a function represented by $\sqcup : RTPNESet \times RTPNESet \to RTPNESet$ that merges two TPNEs. Let $\mathcal{N}_1$= $(P_1, T_1, F_1, W_1, m_{0_1}, I_1, \mathcal{E}_1, \mathcal{CS}_1)$ and $\mathcal{N}_2$= $(P_2, T_2, F_2, W_2, m_{0_2}, I_2, \mathcal{E}_2, \mathcal{CS}_2)$ be two TPNEs . $\mathcal{N}_3$= $(P_3, T_3, F_3, W_3, m_{0_3}, I_3, \mathcal{E}_3, \mathcal{CS}_3)$ is obtained by $\mathcal{N}_3 = \mathcal{N}_1 \sqcup \mathcal{N}_2$, such that:*

- $\diamondsuit$ $P_3 = P_1 \cup P_2$;
- $\diamondsuit$ $T_3 = T_1 \cup T_2$;
- $\diamondsuit$ $F_3 = F_1 \cup F_2$;
- $\diamondsuit$ $W_3 = W_1 \cup W_2$;
- $\diamondsuit$ $m_{0_3}(p) = \begin{cases} m_{0_1}(p), & if\ (p \in P_1 - P_2) \\ m_{0_2}(p), & if\ (p \in P_2 - P_1) \\ max(m_{0_1}(p), m_{0_2}(p)), & if\ (p \in P_1 \cap P_2) \end{cases}$
- $\diamondsuit$ $I_3 = I_1 \cup I_2$;
- $\diamondsuit$ $\mathcal{E}_3 = \mathcal{E}_1 \cup \mathcal{E}_2$;
- $\diamondsuit$ $\mathcal{CS}_3 = \mathcal{CS}_1 \cup \mathcal{CS}_2$.

*As presented, $\sqcup$ is adopted using infix notation for the sake of readability.*

As an example, consider the TPNEs $\mathcal{N}_1$ and $\mathcal{N}_2$ presented in Figure 7.2(a). Net $\mathcal{N}_3$ (Figure 7.2(b)) is generated by fusing the common places of $\mathcal{N}_1$ and $\mathcal{N}_2$ using net union operator: $\mathcal{N}_3 = \mathcal{N}_1 \sqcup \mathcal{N}_2$.

**Properties** Net union operator is a *commutative monoid*, since the operator is associative, commutative and contains an identity element. The reader may envisage this property, since all internal operations are also *commutative monoids*. Besides, considering the proposed building blocks, all resultant nets are *bounded* (see Section 3.5). Previous assertion is valid, since all building blocks are structurally bounded and conservative, and the net union operator preserves the place invariants (P-invariants). The proofs are presented as follows.

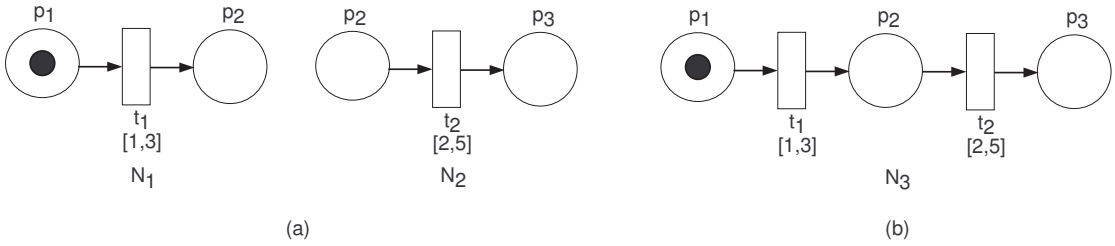


**Figure 7.2** Net union example

**Axiom 7.1.1** (Commutative Monoid)**.** *Let X be a binary operation defined for a domain D ($X : D \times D \to D$). $< D, X >$ is a commutative monoid if X is associative ($(aXb)Xc = aX(bXc), \forall a, b, c \in D$), commutative ( $aXb = bXa, \forall a, b \in D$) and contains an identity element $\emptyset \in D$ ($\emptyset Xa = aX\emptyset = a, \forall a \in D$).*

**Theorem 7.1.1** (Net Union - Associative Property)**.** *Net union operation is associative, since its internal operations are associative.*

*Proof.* Let $\mathcal{N}_1$,$\mathcal{N}_2$ and $\mathcal{N}_3 \in RTPNESet$ be three TPNs with energy consumption values and code annotations, such that $\mathcal{N}_1 = (P_1, T_1, F_1, W_1, m_{0_1}, I_1, \mathcal{E}_1)$, $\mathcal{N}_2 = (P_2, T_2, F_2, W_2, m_{0_2}, I_2, \mathcal{E}_2)$ and $\mathcal{N}_3 = (P_3, T_3, F_3, W_3, m_{0_3}, I_3, \mathcal{E}_3)$. If $\mathcal{N}_a = (P_a, T_a, F_a, W_a, m_{0_a}, I_a, \mathcal{E}_a)$ is a TPNE obtained by $\mathcal{N}_a = \mathcal{N}_1 \sqcup \mathcal{N}_2$, then:

$\diamondsuit$ $P_a = P_1 \cup P_2$;

$\diamondsuit$ $T_a = T_1 \cup T_2$;

$\diamondsuit$ $F_a = F_1 \cup F_2$;

$\diamondsuit$ $W_a = W_1 \cup W_2$;

$\diamondsuit$ $m_{0_a}(p) = \begin{cases} m_{0_1}(p), & if\ (p \in P_1 - P_2) \\ m_{0_2}(p), & if\ (p \in P_2 - P_1) \\ max(m_{0_1}(p), m_{0_2}(p)), & if\ (p \in P_1 \cap P_2) \end{cases}$

$\diamondsuit$ $I_a = I_1 \cup I_2$;

$\diamondsuit$ $\mathcal{E}_a = \mathcal{E}_1 \cup \mathcal{E}_2$;

$\diamondsuit$ $\mathcal{CS}_a = \mathcal{CS}_1 \cup \mathcal{CS}_2$.

If $\mathcal{N}_b = (P_b, T_b, F_b, W_b, m_{0_b}, I_b, \mathcal{E}_b)$ is a TPNE obtained by $\mathcal{N}_b = \mathcal{N}_a \sqcup \mathcal{N}_3$, then:

$\diamondsuit$ $P_b = P_a \cup P_3$;

$\diamondsuit$ $T_b = T_a \cup T_3$;

$\diamondsuit$ $F_b = F_a \cup F_3$;

$\diamondsuit$ $W_b = W_a \cup W_3$;

$\diamondsuit$ $m_{0_b}(p) = \begin{cases} m_{0_a}(p), & if\ (p \in P_a - P_3) \\ m_{0_3}(p), & if\ (p \in P_3 - P_a) \\ max(m_{0_a}(p), m_{0_3}(p)), & if\ (p \in P_a \cap P_3) \end{cases}$

$\diamondsuit$ $I_b = I_a \cup I_3$;

$\diamondsuit$ $\mathcal{E}_b = \mathcal{E}_a \cup \mathcal{E}_3$;

$\diamondsuit$ $\mathcal{CS}_b = \mathcal{CS}_a \cup \mathcal{CS}_3$.

If $\mathcal{N}_c = (P_c, T_c, F_c, W_c, m_{0_c}, I_c, \mathcal{E}_c)$ is a TPNE obtained by $\mathcal{N}_c = \mathcal{N}_2 \sqcup \mathcal{N}_3$, then:

$\diamondsuit$ $P_c = P_2 \cup P_3$;

$\diamondsuit$ $T_c = T_2 \cup T_3$;

$\diamondsuit$ $F_c = F_2 \cup F_3$;

$\diamondsuit$ $W_c = W_2 \cup W_3$;

$\diamondsuit$ $m_{0_c}(p) = \begin{cases} m_{0_2}(p), & if\ (p \in P_2 - P_3) \\ m_{0_3}(p), & if\ (p \in P_3 - P_2) \\ max(m_{0_2}(p), m_{0_3}(p)), & if\ (p \in P_2 \cap P_3) \end{cases}$

$\diamondsuit$ $I_c = I_2 \cup I_3$;

$\diamondsuit$ $\mathcal{E}_c = \mathcal{E}_2 \cup \mathcal{E}_3$;

$\diamondsuit$ $\mathcal{CS}_c = \mathcal{CS}_2 \cup \mathcal{CS}_3$.

If $\mathcal{N}_d = (P_d, T_d, F_d, W_d, m_{0_d}, I_d, \mathcal{E}_d)$ is a TPNE obtained by $\mathcal{N}_d = \mathcal{N}_1 \sqcup \mathcal{N}_c$, then:

$\diamondsuit$ $P_d = P_1 \cup P_c$;

$\diamondsuit$ $T_d = T_1 \cup T_c$;

$\diamondsuit$ $F_d = F_1 \cup F_c$;

$\diamondsuit$ $W_d = W_1 \cup W_c$;

$\diamondsuit$ $m_{0_d}(p) = \begin{cases} m_{0_1}(p), & if\ (p \in P_1 - P_c) \\ m_{0_c}(p), & if\ (p \in P_c - P_1) \\ max(m_{0_1}(p), m_{0_c}(p)), & if\ (p \in P_1 \cap P_c) \end{cases}$

$\diamondsuit$ $I_d = I_1 \cup I_c$;

$\diamondsuit$ $\mathcal{E}_d = \mathcal{E}_1 \cup \mathcal{E}_c$;

$\diamondsuit$ $\mathcal{CS}_d = \mathcal{CS}_1 \cup \mathcal{CS}_c$.

Therefore, $\mathcal{N}_b = \mathcal{N}_a \sqcup \mathcal{N}_3 = (\mathcal{N}_1 \sqcup \mathcal{N}_2) \sqcup \mathcal{N}_3$ and $\mathcal{N}_d = \mathcal{N}_1 \sqcup \mathcal{N}_c = \mathcal{N}_1 \sqcup (\mathcal{N}_2 \sqcup \mathcal{N}_3)$. Net union operation is associative if $\mathcal{N}_b = \mathcal{N}_d$. Thus, replacing each occurrence of $\mathcal{N}_a$ in $\mathcal{N}_b$:

$\diamondsuit$ $P_b = (P_1 \cup P_2) \cup P_3$;

$\diamondsuit$ $T_b = (T_1 \cup T_2) \cup T_3$;

$\diamondsuit$ $F_b = (F_1 \cup F_2) \cup F_3$;

$\diamondsuit$ $W_b = (W_1 \cup W_2) \cup W_3$;

$$\diamondsuit\ m_{0_b}(p) = \begin{cases} m_{0_1}(p), & if\ (p \in (P_1 - P_2) - P_3) \\ m_{0_2}(p), & if\ (p \in (P_2 - P_1) - P_3) \\ max(m_{0_1}(p), m_{0_2}(p)), & if\ (p \in (P_1 \cap P_2) - P_3) \\ m_{0_3}(p), & if\ (p \in P_3 - (P_1 \cup P_2)) \\ max(m_{0_1}(p), m_{0_3}(p)), & if\ (p \in (P_1 - P_2) \cap P_3) \\ max(m_{0_2}(p), m_{0_3}(p)), & if\ (p \in (P_2 - P_1) \cap P_3) \\ max(max(m_{0_1}(p), m_{0_2}(p)), m_{0_3}(p)), & if\ (p \in (P_1 \cap P_2) \cap P_3) \end{cases}$$

$\diamondsuit$ $I_b = (I_1 \cup I_2) \cup I_3$;

$\diamondsuit$ $\mathcal{E}_b = (\mathcal{E}_1 \cup \mathcal{E}_2) \cup \mathcal{E}_3$;

$\diamondsuit$ $\mathcal{CS}_b = (\mathcal{CS}_1 \cup \mathcal{CS}_2) \cup \mathcal{CS}_3$.

Replacing each occurrence of $\mathcal{N}_c$ in $\mathcal{N}_d$:

$\diamondsuit$ $P_d = P_1 \cup (P_2 \cup P_3)$;

$\diamondsuit$ $T_d = T_1 \cup (T_2 \cup T_3)$;

$\diamondsuit$ $F_d = F_1 \cup (F_2 \cup F_3)$;

$\diamondsuit$ $W_d = W_1 \cup (W_2 \cup W_3)$;

$$\diamondsuit\ m_{0_d}(p) = \begin{cases} m_{0_1}(p), & if\ (p \in P_1 - (P_2 \cup P_3)) \\ m_{0_2}(p), & if\ (p \in (P_2 - P_3) - P_1) \\ m_{0_3}(p), & if\ (p \in (P_3 - P_2) - P_1)) \\ max(m_{0_2}(p), m_{0_3}(p)), & if\ (p \in (P_2 \cap P_3) - P_1) \\ max(m_{0_1}(p), m_{0_2}(p)), & if\ (p \in P_1 \cap (P_2 - P_3)) \\ max(m_{0_1}(p), m_{0_3}(p)), & if\ (p \in P_1 \cap (P_3 - P_2)) \\ max(m_{0_1}(p), max(m_{0_2}(p), m_{0_3}(p))), & if\ (p \in P_1 \cap (P_2 \cap P_3)) \end{cases}$$

$\diamondsuit$ $I_d = I_1 \cup (I_2 \cup I_3)$;

$\diamondsuit$ $\mathcal{E}_d = \mathcal{E}_1 \cup (\mathcal{E}_2 \cup \mathcal{E}_3)$;

$\diamondsuit$ $\mathcal{CS}_d = \mathcal{CS}_1 \cup (\mathcal{CS}_2 \cup \mathcal{CS}_3)$.

Thus, net union operation is associative, since $\mathcal{N}_b = (\mathcal{N}_1 \sqcup \mathcal{N}_2) \sqcup \mathcal{N}_3$ and $\mathcal{N}_d = \mathcal{N}_1 \sqcup (\mathcal{N}_2 \sqcup \mathcal{N}_3)$, and $\mathcal{N}_b = \mathcal{N}_d$, in which:

$\diamondsuit$ $P_b = P_d = (P_1 \cup P_2) \cup P_3 = P_1 \cup (P_2 \cup P_3)$;

$\diamondsuit$ $T_b = T_d = (T_1 \cup T_2) \cup T_3 = T_1 \cup (T_2 \cup T_3)$;

$\diamondsuit$ $F_b = F_d = (F_1 \cup F_2) \cup F_3 = F_1 \cup (F_2 \cup F_3)$;

$\diamondsuit$ $W_b = W_d = (W_1 \cup W_2) \cup W_3 = W_1 \cup (W_2 \cup W_3)$;

$\diamondsuit$ $m_{0_b}(p) = m_{0_d}(p), \forall p \in (P_1 \cup P_2 \cup P_3)$;

$\diamondsuit$ $I_b = I_d = (I_1 \cup I_2) \cup I_3 = I_1 \cup (I_2 \cup I_3)$;

$\diamondsuit$ $\mathcal{E}_b = \mathcal{E}_d = (\mathcal{E}_1 \cup \mathcal{E}_2) \cup \mathcal{E}_3 = \mathcal{E}_1 \cup (\mathcal{E}_2 \cup \mathcal{E}_3)$;

$\diamondsuit$ $\mathcal{CS}_b = \mathcal{CS}_d = (\mathcal{CS}_1 \cup \mathcal{CS}_2) \cup \mathcal{CS}_3 = \mathcal{CS}_1 \cup (\mathcal{CS}_2 \cup \mathcal{CS}_3)$.

**Theorem 7.1.2** (Net Union - Commutative Property)**.** *Net union operation is commutative, since its internal operations are commutative.*

*Proof.* Let $\mathcal{N}_1$ and $\mathcal{N}_2 \in RTPNESet$ be two TPNEs, such that $\mathcal{N}_1 = (P_1, T_1, F_1, W_1, m_{0_1}, I_1, \mathcal{E}_1, \mathcal{CS}_1)$ and $\mathcal{N}_2 = (P_2, T_2, F_2, W_2, m_{0_2}, I_2, \mathcal{E}_2, \mathcal{CS}_2)$. If $\mathcal{N}_a = (P_a, T_a, F_a, W_a, , m_{0_a}, I_a, \mathcal{E}_a, \mathcal{CS}_a)$ is a TPNE obtained by $\mathcal{N}_a = \mathcal{N}_1 \sqcup \mathcal{N}_2$, then:

$\diamondsuit$ $P_a = P_1 \cup P_2$;

$\diamondsuit$ $T_a = T_1 \cup T_2$;

$\diamondsuit$ $F_a = F_1 \cup F_2$;

$\diamondsuit$ $W_a = W_1 \cup W_2$;

$\diamondsuit$ $m_{0_a}(p) = \begin{cases} m_{0_1}(p), & if\ (p \in P_1 - P_2) \\ m_{0_2}(p), & if\ (p \in P_2 - P_1) \\ max(m_{0_1}(p), m_{0_2}(p)), & if\ (p \in P_1 \cap P_2) \end{cases}$

$\diamondsuit$ $I_a = I_1 \cup I_2$;

$\diamondsuit$ $\mathcal{E}_a = \mathcal{E}_1 \cup \mathcal{E}_2$;

$\diamondsuit$ $\mathcal{CS}_a = \mathcal{CS}_1 \cup \mathcal{CS}_2$.

If $\mathcal{N}_b = (P_b, T_b, F_b, W_b, m_{0_b}, I_b, \mathcal{E}_b, \mathcal{CS}_b)$ is a TPNE obtained by $\mathcal{N}_b = \mathcal{N}_2 \sqcup \mathcal{N}_1$, then:

- $P_b = P_2 \cup P_1$;
- $T_b = T_2 \cup T_1$;
- $F_b = F_2 \cup F_1$;
- $W_b = W_2 \cup W_1$;
- $m_{0_b}(p) = \begin{cases} m_{0_2}(p), & if\ (p \in P_2 - P_1) \\ m_{0_1}(p), & if\ (p \in P_1 - P_2) \\ max(m_{0_2}(p), m_{0_1}(p)), & if\ (p \in P_2 \cap P_1) \end{cases}$
- $I_b = I_2 \cup I_1$;
- $\mathcal{E}_b = \mathcal{E}_2 \cup \mathcal{E}_1$;
- $\mathcal{CS}_b = \mathcal{CS}_2 \cup \mathcal{CS}_1$.

Thus, net union operation is commutative, since $\mathcal{N}_a = \mathcal{N}_1 \sqcup \mathcal{N}_2$,$\mathcal{N}_b = \mathcal{N}_2 \sqcup \mathcal{N}_1$, and $\mathcal{N}_a = \mathcal{N}_b$, in which:

- $P_a = P_b = P_1 \cup P_2 = P_2 \cup P_1$;
- $T_a = T_b = T_1 \cup T_2 = T_2 \cup T_1$;
- $F_a = F_b = F_1 \cup F_2 = F_2 \cup F_1$;
- $W_a = W_b = W_1 \cup W_2 = W_2 \cup W_1$;
- $m_{0_a}(p) = m_{0_b}(p), \forall p \in (P_1 \cup P_2)$;
- $I_a(f) = I_b(t) = I_1 \cup I_2 = I_2 \cup I_1$;
- $\mathcal{E}_a(t) = \mathcal{E}_b(t) = \mathcal{E}_1 \cup \mathcal{E}_2 = \mathcal{E}_2 \cup \mathcal{E}_1$;
- $\mathcal{CS}_a(t) = \mathcal{CS}_b(t) = \mathcal{CS}_1 \cup \mathcal{CS}_2 = \mathcal{CS}_2 \cup \mathcal{CS}_1$.

**Theorem 7.1.3** (Net Union - Identity Element)**.** *Let $\mathcal{N}_\emptyset \in RTPNESet$ be a time Petri net with energy consumption values and code annotations (TPNE), such that $\mathcal{N}_\emptyset = (P_\emptyset = \emptyset, T_\emptyset = \emptyset, F_\emptyset = \emptyset, W_\emptyset = \emptyset, m_{0_\emptyset} = \emptyset, I_\emptyset = \emptyset, \mathcal{E}_\emptyset = \emptyset)$. $\mathcal{N}_\emptyset$ is the identity element of net union operation, since the application of such operation between any net $\mathcal{N} \in RTPNESet$ and $\mathcal{N}_\emptyset$ results in $\mathcal{N}$.*

*Proof.* Let $\mathcal{N}_1$ and $\mathcal{N}_\emptyset \in RTPNESet$ be two TPNEs, such that $\mathcal{N}_1 = (P_1, T_1, F_1, W_1, m_{0_1}, I_1, \mathcal{E}_1, \mathcal{CS}_1)$ and $\mathcal{N}_\emptyset$ is the identity element. If $\mathcal{N}_a = (P_a, T_a, F_a, W_a, m_{0_a}, I_a, \mathcal{E}_a, \mathcal{CS}_a)$ is a TPNE obtained by $\mathcal{N}_a = \mathcal{N}_1 \sqcup \mathcal{N}_\emptyset$ or $\mathcal{N}_a = \mathcal{N}_\emptyset \sqcup \mathcal{N}_1$, then $\mathcal{N}_a = \mathcal{N}_1$:

$\diamondsuit$ $P_a = P_1 \cup \emptyset = \emptyset \cup P_1 = P_1$;

$\diamondsuit$ $T_a = T_1 \cup \emptyset = \emptyset \cup T_1 = T_1$;

$\diamondsuit$ $F_a = F_1 \cup \emptyset = \emptyset \cup F_1 = F_1$;

$\diamondsuit$ $W_a = W_1 \cup \emptyset = \emptyset \cup W_1 = W_1$;

$\diamondsuit$ $m_{0_a}(p) = m_{0_1}(p), \forall p \in P_1$;

$\diamondsuit$ $I_a = I_1 \cup \emptyset = \emptyset \cup I_1 = I_1$;

$\diamondsuit$ $\mathcal{E}_a = \mathcal{E}_1 \cup \emptyset = \emptyset \cup \mathcal{E}_1 = \mathcal{E}_1$.

$\diamondsuit$ $\mathcal{CS}_a = \mathcal{CS}_1 \cup \emptyset = \emptyset \cup \mathcal{CS}_1 = \mathcal{CS}_1$.

**Collorary 7.1.1** (Net Union - Commutative Monoid)**.** *Let RTPNESet be the set of restricted time Petri nets with energy consumption values and code annotations (TPNE) and $\sqcup$ is the net union operation. $< RTPNESet, \sqcup >$ is a commutative monoid and $\mathcal{N}_\emptyset$ is the respective identity element.*

The following lines demonstrate that all generated models, using net union operator and the proposed building block models, are structurally bounded.

**Definition 7.5** (Set of All TPNEs Generated from Building Block Models and Net Union Operator - $\mathcal{B}^*$)**.** *Let $\mathcal{B} \subseteq RTPNESet$ be the set of the proposed basic building block models, in which $\forall \mathcal{N}_i, \mathcal{N}_n \in \mathcal{B}, T_i \cap T_n \neq \emptyset \leftrightarrow \mathcal{N}_i = \mathcal{N}_n$ (note that $T_i$ and $T_n$ are the set of transitions of the respective nets), and $\mathcal{N}_\emptyset$ be the identity element. $\mathcal{B}^*$ is the set of all time Petri nets with energy consumption values and code annotations generated using net union operator and the proposed building blocks, such that: $\mathcal{B}^* = \bigcup_{i \in \mathbb{N}} \mathcal{B}_i = \mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3 \cup ...$, in which $\mathcal{B}_0 = \{\mathcal{N}_\emptyset\}$ and $\mathcal{B}_{i+1} = \{a \sqcup b | a \in \mathcal{B}_i \wedge b \in \mathcal{B}\}, i \geq 0$. Besides, $\mathcal{B} \subseteq \mathcal{B}^* \subseteq RTPNESet$.*

In previous definition, although there are other ways to compose the nets contained in each subset $\mathcal{B}_{i+1}$ using net union operator $\sqcup$, the adopted approach is only to define the set $\mathcal{B}^*$ in a clear manner. Besides, the constraint assumed in the set $\mathcal{B}$ is assured by the suffix *spec*, which is described in Section 6.3.

**Theorem 7.1.4** (Net Union and Basic Building Blocks - Structurally Conservative and Structurally Bounded Models)**.** *If $\mathcal{N} \in \mathcal{B}^*$, then $\mathcal{N}$ is structurally conservative as well as structurally bounded.*

*Proof.* Base Cases:

1. $\mathcal{N} \in \mathcal{B}_0$. As $\mathcal{B}_0 = \{\mathcal{N}_\emptyset\}$, $\mathcal{N}$ must be the identity element. Since the state space is empty for $\mathcal{N}_\emptyset$, assume $\mathcal{N}_a$ is structurally conservative and structurally bounded;

2. $\mathcal{N} \in \mathcal{B}_1$. $\mathcal{B}_1 = \{\mathcal{N}_a \sqcup \mathcal{N}_b | \mathcal{N}_a \in \mathcal{B}_0 \wedge \mathcal{N}_b \in \mathcal{B}\}$ is the set of the proposed building block models ($\mathcal{B}_1 = \mathcal{B}$), since the net union of the identity element ($\mathcal{B}_0 = \{\mathcal{N}_\emptyset\}$) and a building block results in the former building block model. Thus, $\mathcal{N}_b$ is structurally conservative and structurally bounded, as all building blocks do contain such properties (see Section 6.3);

3. $\mathcal{N}_c \in \mathcal{B}_2$, and $\mathcal{B}_2 = \{\mathcal{N}_d \sqcup \mathcal{N}_e | \mathcal{N}_d \in \mathcal{B}_1 \wedge \mathcal{N}_e \in \mathcal{B}\}$

   (a) $\mathcal{N}_c = \mathcal{N}_d \sqcup \mathcal{N}_e$, in which $d = e$. $\mathcal{N}_c$ is structurally bounded as well as structurally conservative , since the net union of a building block with itself results in the same building block. As presented previously, all building blocks are structurally bounded and conservative.

   (b) $\mathcal{N}_c = \mathcal{N}_d \sqcup \mathcal{N}_e$, in which $d \neq e \wedge P_d \cap P_e \neq \emptyset \wedge T_d \cap T_e = \emptyset$. $\mathcal{N}_c$ is structurally bounded and conservative, since all possible mergings of two building blocks result in structurally bounded and conservative nets (see Appendix A). In addition to the conservative components, the resultant net also preserves the places that allow the fusion with other building blocks.

   (c) $\mathcal{N}_c = \mathcal{N}_d \sqcup \mathcal{N}_e$, such that $d \neq e \wedge P_d \cap P_e = \emptyset \wedge T_d \cap T_e = \emptyset$. $N_c$ is structurally bounded and conservative, since the net union of disjoint nets preserves the P-invariants of each subnet in the generated model.

   (d) $\mathcal{N}_c = \mathcal{N}_d \sqcup \mathcal{N}_e$, such that $T_d \cap T_e \neq \emptyset$. Since $T_d \cap T_e \neq \emptyset$, then $\mathcal{N}_d = \mathcal{N}_e$, as demonstrated in the definition of $\mathcal{B}$ (remind $\mathcal{B}_1 = \mathcal{B}$). $\mathcal{N}_c$ is structurally conservative and structurally bounded, as the net union of a building block with itself results in the same building block.

Inductive Step: $\mathcal{N}_x \in \mathcal{B}_{i+1}$ is structurally conservative and structurally bounded (${\mathcal{I}_x}^T \times A_x = 0$, in which $A_x$ is the respective incidence matrix and ${\mathcal{I}_x}^T$ is a vector of positive integers).

The recursive definition of $\mathcal{B}^*$ indicates that each $\mathcal{N} \in \mathcal{B}_i$ is composed using one or more base cases presented previously: $\mathcal{B}^* = \bigcup_{i \in \mathbb{N}} \mathcal{B}_i = \mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2 \cup \mathcal{B}_3 \cup ...$, such that

$\diamondsuit$ $\mathcal{B}_1 = \{a \sqcup b | a \in \mathcal{B}_0 \wedge b \in \mathcal{B}\}$

$\diamondsuit$ $\mathcal{B}_2 = \{a \sqcup b | a \in \mathcal{B}_1 \wedge b \in \mathcal{B}\} = \{(b_0 \sqcup b_1) \sqcup b | b_0 \in \mathcal{B}_0 \wedge b_1, b \in \mathcal{B}\}$
...

$\diamondsuit$ $\mathcal{B}_i = \{a \sqcup b | a \in \mathcal{B}_i \wedge b \in \mathcal{B}\} = \{(((b_0 \sqcup b_1)...) \sqcup b_{i-1}) \sqcup b | b_0 \in \mathcal{B}_0 \wedge b_1, ..., b_{i-1}, b \in \mathcal{B}\}$

As all base cases do generate structurally conservative as well as structurally bounded nets (the conservative components are preserved and, also, the places that allow the fusion with other building blocks); and (ii) net union operator is a commutative monoid (the composition order does not matter), $\mathcal{N}$ is a structurally bounded net. Moreover, since $\mathcal{B}_{i+1} = \{((((b_0) \sqcup b_1)...) \sqcup b_{i-1}) \sqcup b_i) \sqcup b | b_0 \in \mathcal{B}_0 \wedge b_1, ..., b_{i-1}, b_i, b \in \mathcal{B}\})$ also utilizes the base cases and net union operator, $\mathcal{N}_x \in \mathcal{B}_{i+1}$ is structurally bounded as well as conservative. Thus, for any $\mathcal{N}_y \in \mathcal{B}^*$, $\mathcal{N}_y$ is structurally bounded and conservative. This concludes the proof $\square$.

## 7.2 MODELING REAL-TIME TASKS

In order to enlighten the modeling process of real-time tasks, consider the following specification consisting of two preemptive tasks: $\tau_1 = (0,\ 0,\ 240 \times 10^6,\ 20,\ 20)$ and $\tau_2 = (0,\ 5,\ 60 \times 10^6,\ 15,\ 20)$. In this specification, the task time unit is one second and the LCM is equal to $20s$, which points out the existence of two task instances ($\mathcal{S}(\tau_1) + \mathcal{S}(\tau_2) = 2$). In addition, assume the following supply voltages and the respective maximum frequencies $\mathit{vff} = \{(1\text{V},10\text{MHz}),(2\text{V},20\text{MHz})\}$. Moreover, an unavailable voltage/frequency of 1.5V/15MHz is considered, which can be "simulated" using the 2 immediate neighboring voltage levels (and the respective maximum CPU frequencies).

The TPNE model is composed by modeling each individual task and, next, making the appropriate connections to represent the CPU sharing, the initial condition (fork block) and the desired final marking (join block). In general, each task is modeled considering a combination of arrival, voltage selection, (non-)preemptive task structure, deadline, and conclusion blocks. Indeed, the combined blocks model the constraints as well as the energy consumption of a hard real-time task. For a better understanding, the modeling process is detailed into 2 subsections, which are presented as follows.

### 7.2.1 Modeling a Single Task

This section details the process for modeling a single task, more specifically, task $\tau_1$ (from previous specification). Although the proposed example considers preemptive tasks, the same approach is adopted for non-preemptive tasks. For task $\tau_1$, the following blocks needs to be instantiated:

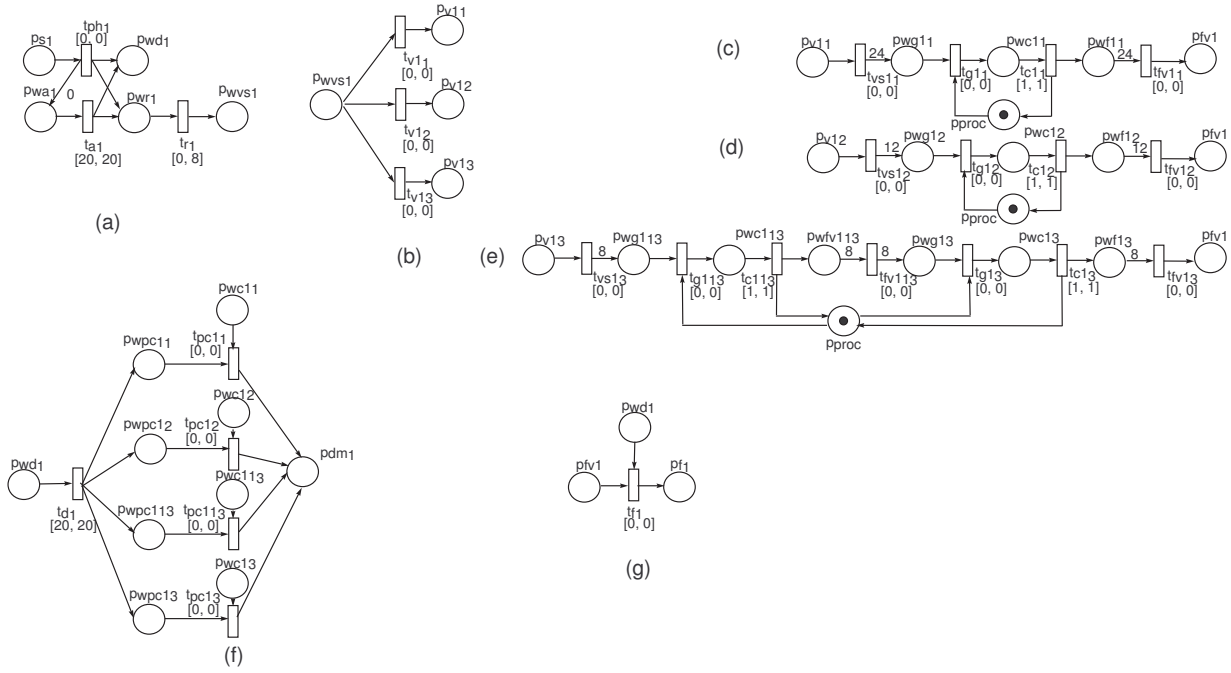


**Figure 7.3** Basic building blocks for task $\tau_1$

- ♢ One periodic task arrival block - $\mathcal{N}_{p_1}$(Figure 7.3(a));
- ♢ One voltage selection block, considering three voltage/frequency levels - $\mathcal{N}_{v_1}$(Figure 7.3(b));
- ♢ Three preemptive task structure blocks (one for each voltage/frequency level): $\mathcal{N}_{p_{1_3}}$ (Figure 7.3(c)) - 1V/10MHz, $\mathcal{N}_{p_{1_2}}$ (Figure 7.3(d)) - 2V/20MHz, and $\mathcal{N}_{p_{1_3}}$(Figure 7.3(e)) - 1.5V/15MHz;
- ♢ One deadline block considering four *waiting for task computation* places (Figure 7.3(f))- $\mathcal{N}_{d_1}$.
- ♢ One task instance conclusion block (Figure 7.3(g))- $\mathcal{N}_{d_1}$.

As presented, the proposed modeling approach guarantees that the generated models are *bounded*. Thus, for this example, the juxtaposition of the P-invariants (see Section 3.5) are presented in order to demonstrate the preservation of *boundness* property during the fusion of building blocks. For each basic Petri net presented previously, the P-invariants are:

♢ $\mathcal{N}_{a_1}$: $\mathcal{I}_{(a_1)} = \overset{p_{st_1}}{[\ wvs_1 + wd_1} \quad \overset{p_{wa_1}}{wvs_1 + wd_1} \quad \overset{p_{wd_1}}{wd_1} \quad \overset{p_{wr_1}}{wvs_1} \quad \overset{p_{wvs_1}}{wvs_1}\ ]^T$;

♢ $\mathcal{N}_{v_1}$: $\mathcal{I}_{(v_1)} = \overset{p_{wvs_1}}{[\ wvs'_1} \quad \overset{p_{v_{1_1}}}{wvs'_1} \quad \overset{p_{v_{1_2}}}{wvs'_1} \quad \overset{p_{v_{1_3}}}{wvs'_1}\ ]^T$;

♢ $\mathcal{N}_{p_{1_1}}$: $\mathcal{I}_{(p_{1_1})} = \overset{p_{v_{1_1}}}{[\ 24wg_{1_1}} \quad \overset{p_{wg_{1_1}}}{wg_{1_1}} \quad \overset{p_{wc_{1_1}}}{wg_{1_1} + proc} \quad \overset{p_{wf_{1_1}}}{wg_{1_1}} \quad \overset{p_{fv_1}}{24wg_{1_1}} \quad \overset{p_{proc}}{proc}\ ]^T$;

♢ $\mathcal{N}_{p_{1_2}}$: $\mathcal{I}_{(p_{1_1})} = \overset{p_{v_{1_2}}}{[\ 12wg_{1_2}} \quad \overset{p_{wg_{1_2}}}{wg_{1_2}} \quad \overset{p_{wc_{1_2}}}{wg_{1_2} + proc'} \quad \overset{p_{wf_{1_2}}}{wg_{1_2}} \quad \overset{p_{fv_1}}{12wg_{1_2}} \quad \overset{p_{proc}}{proc'}\ ]^T$;

♢ $\mathcal{N}_{p2v_{1_3}}$ : $\mathcal{I}_{(p2v_{1_3})} = \overset{p_{v_{1_3}}}{[\ 64wg1_{1_3}} \quad \overset{p_{wg1_{1_3}}}{8wg1_{1_3}} \quad \overset{p_{wc1_{1_3}}}{8wg1_{1_3} + proc''} \quad \overset{p_{wf1_{1_3}}}{8wg1_{1_3}} \quad \overset{p_{wg_{1_3}}}{8wg1_{1_3}}$

$\overset{p_{wc_{1_3}}}{8wg1_{1_3} + proc''} \quad \overset{p_{wf_{1_3}}}{8wg1_{1_3}} \quad \overset{p_{fv_1}}{64wg1_{1_3}} \quad \overset{p_{proc}}{proc''}\ ]^T$;

♢ $\mathcal{N}_{d_1}$: $\mathcal{I}_{(d_1)} = [\ 4d_0 + 3d_1 + 3d_2 + 2d_{1,2} + \overset{p_{wd_1}}{3d_{1_3}} + 2d_{1,1_3} + 2d_{2,1_3} + d_{1,2,1_3}$

$+3d_3 + 2d_{1,3} + 2d_{2,3} + d_{1,2,3} + 2d_{1_3,3} + d_{1,1_3,3} + d_{2,1_3,3}$

$d_0 + d_2 + d_{1_3} + d_{2,1_3} + \overset{p_{wpc_{1_1}}}{d_3} + d_{2,3} + d_{1_3,3} + d_{2,1_3,3}$

$d_1 + d_{1,2} + d_{1,1_3} + d_{1,2,1_3} + \overset{p_{wc_{1_1}}}{d_{1,3}} + d_{1,2,3} + d_{1,1_3,3} + d_{1,2,1_3,3}$

$$d_0 + d_1 + d_{1,3} + d_{1,1_3} + \overset{p_{wpc_{1_2}}}{d_3} + d_{1_3} + d_{1_3,3} + d_{1,1_3,3}$$

$$d_2 + d_{1,2} + d_{2,1_3} + d_{1,2,1_3} + \overset{p_{wc_{1_2}}}{d_{2,3}} + d_{1,2,3} + d_{2,1_3,3} + d_{1,2,1_3,3}$$

$$d_0 + d_1 + d_2 + d_{1,2} + \overset{p_{wpc1_{1_3}}}{d_3} + d_{1,3} + d_{2,3} + d_{1,2,3}$$

$$d_{1_3} + d_{1,1_3} + d_{2,1_3} + d_{1,2,1_3} + \overset{p_{wc1_{1_3}}}{d_{1_3,3}} + d_{1,1_3,3} + d_{2,1_3,3} + d_{1,2,1_3,3}$$

$$d_0 + d_1 + d_{1,2} + d_2 + \overset{p_{wpc_{1_3}}}{d_{1_3}} + d_{1,1_3} + d_{2,1_3} + d_{1,2,1_3}$$

$$d_3 + d_{1,3} + d_{2,3} + d_{1,2,3} + \overset{p_{wc_{1_3}}}{d_{1_3,3}} + d_{1,1_3,3} + d_{2,1_3,3} + d_{1,2,1_3,3}$$

$$d_0 + d_1 + d_{1,2} + d_2 + d_{1_3} + d_{1,1_3} + \overset{p_{dm_1}}{d_{2,1_3}} + d_{1,2,1_3} + d_3 + d_{1,3} + d_{2,3}$$

$$+d_{1,2,3} d_{1_3,3} + d_{1,1_3,3} + d_{2,1_3,3} + d_{1,2,1_3,3} \ \Big]^T;$$

$$\diamondsuit \ \mathcal{N}_{c_1}\colon \mathcal{I}_{(c_1)} = \ \Big[ \ \overset{p_{f_1}}{fv_1 + wd_1} \quad \overset{p_{fv_1}}{fv_1} \quad \overset{p_{wd_1}}{wd_1} \ \Big]^T.$$

As follows, the composition is performed by adopting the net union operator ($\sqcup$):

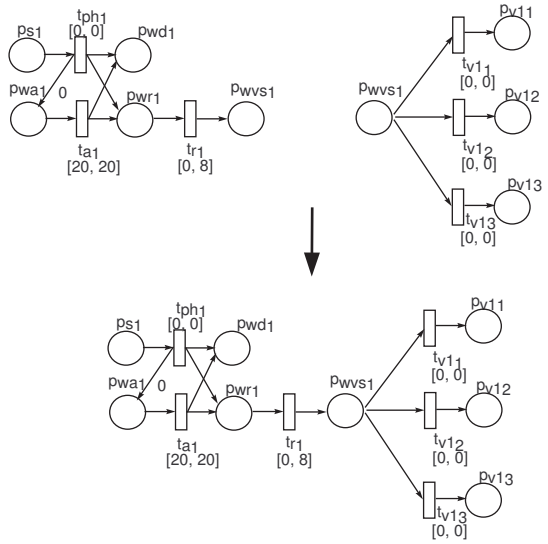


**Figure 7.4** Step 1

1. $\mathcal{N}_{(a1 \sqcup v1)} = \mathcal{N}_{p_1} \sqcup \mathcal{N}_{v_1}$(Figure 7.4). The following P-invariant is obtained by juxtaposition $\mathcal{J}(\mathcal{I}_{(a_1)}, \mathcal{I}_{(v_1)}))$, in which $wvs'_1 = wvs_1$:

$$\mathcal{I}_{(a_1 \sqcup v_1)} = \begin{matrix} p_{st_1} & p_{wa_1} & p_{wd_1} & p_{wr_1} & p_{wvs_1} & p_{v_{1_1}} & p_{v_{1_2}} & p_{v_{1_3}} \\ [\ wvs_1 + wd_1 & wvs_1 + wd_1 & wd_1 & wvs_1 & wvs_1 & wvs_1 & wvs_1 & wvs_1\ ]^T; \end{matrix}$$

2. $\mathcal{N}_{(a_1 \sqcup v_1 \sqcup p_{1_1})} = \mathcal{N}_{(a1 \sqcup v1)} \sqcup \mathcal{N}_{p_{1_1}}$(Figure 7.5). By juxtaposition ($\mathcal{J}(\mathcal{I}_{(a1 \sqcup v1)}, \mathcal{I}_{(p_{1_1})})$ and $wvs_1 = 24wg_{1_1}$), the following P-invariant is obtained:

$$\mathcal{I}_{(a_1 \sqcup v_1 \sqcup p_{1_1})} = \begin{matrix} p_{st_1} & p_{wa_1} & p_{wd_1} & p_{wr_1} & p_{wvs_1} & p_{v_{1_1}} \\ [\ 24wg_{1_1} + wd_1 & 24wg_{1_1} + wd_1 & wd_1 & 24wg_{1_1} & 24wg_{1_1} & 24wg_{1_1} \end{matrix}$$

$$\begin{matrix} p_{v_{1_2}} & p_{v_{1_3}} & p_{wg_{1_1}} & p_{wc_{1_1}} & p_{wf_{1_1}} & p_{fv_1} & p_{proc} \\ 24wg_{1_1} & 24wg_{1_1} & wg_{1_1} & wg_{1_1} + p_{proc} & wg_{1_1} & 24wg_{1_1} & proc\ ]^T; \end{matrix}$$

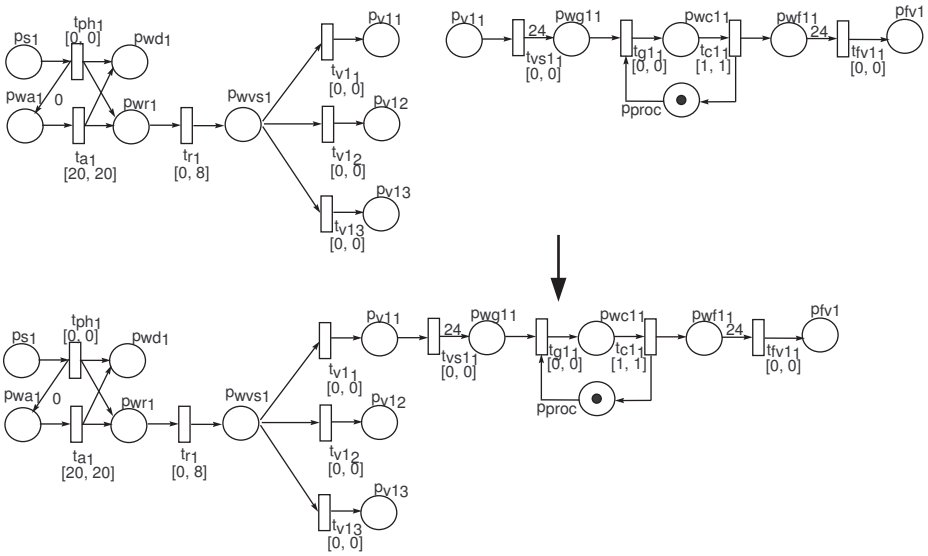


**Figure 7.5** Step 2

3. $\mathcal{N}_{(a_1 \sqcup v_1 \sqcup p_{1_1} \sqcup p_{1_2})} = \mathcal{N}_{(a1 \sqcup v1 \sqcup p_{1_1})} \sqcup \mathcal{N}_{p_{1_2}}$ (Figure 7.6). By $\mathcal{J}(\mathcal{I}_{(a1 \sqcup v1 \sqcup p_{1_1})}, \mathcal{I}_{(p_{1_2})})$ (assuming $wg_{1_1} = 12wg_1$, $wg_{1_2} = 24wg_1$ and $proc' = proc$), the P-invariant is:

$$\mathcal{I}_{(a_1 \sqcup v_1 \sqcup p_{1_1} \sqcup p_{1_2})} = \begin{matrix} p_{st_1} & p_{wa_1} & p_{wd_1} & p_{wr_1} & p_{wvs_1} \\ [\ 288wg_1 + wd_1 & 288wg_1 + wd_1 & wd_1 & 288wg_1 & 288wg_1 \end{matrix}$$

$$\begin{matrix} p_{v_{1_1}} & p_{v_{1_2}} & p_{v_{1_3}} & p_{wg_{1_1}} & p_{wc_{1_1}} & p_{wf_{1_1}} & p_{fv_1} & p_{proc} \\ 288wg_1 & 288wg_1 & 288wg_1 & 12wg_1 & 12wg_1 + proc & 12wg_1 & 288wg_1 & proc \end{matrix}$$

$$\begin{matrix} p_{wg_{1_2}} & p_{wc_{1_2}} & p_{wf_{1_2}} \\ 24wg_1 & 24wg_1 + proc & 24wg_1\ ]^T; \end{matrix}$$

4. $\mathcal{N}_{(a1 \sqcup v1 \sqcup p_{1_x})} = \mathcal{N}_{(a1 \sqcup v1 \sqcup p_{1_1} \sqcup p_{1_2})} \sqcup \mathcal{N}_{p_{1_3}}$ (Figure 7.7). The following P-invariant is obtained by juxtaposition ($wg_1 = 64wg'_1$, $wg_{1_3} = 288wg'_1$ and proc" = proc):

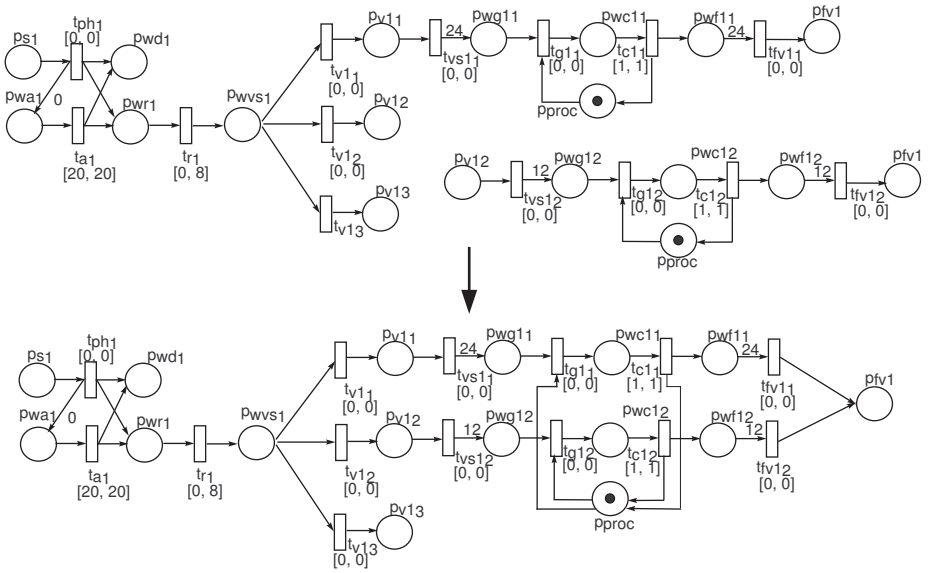


**Figure 7.6** Step 3

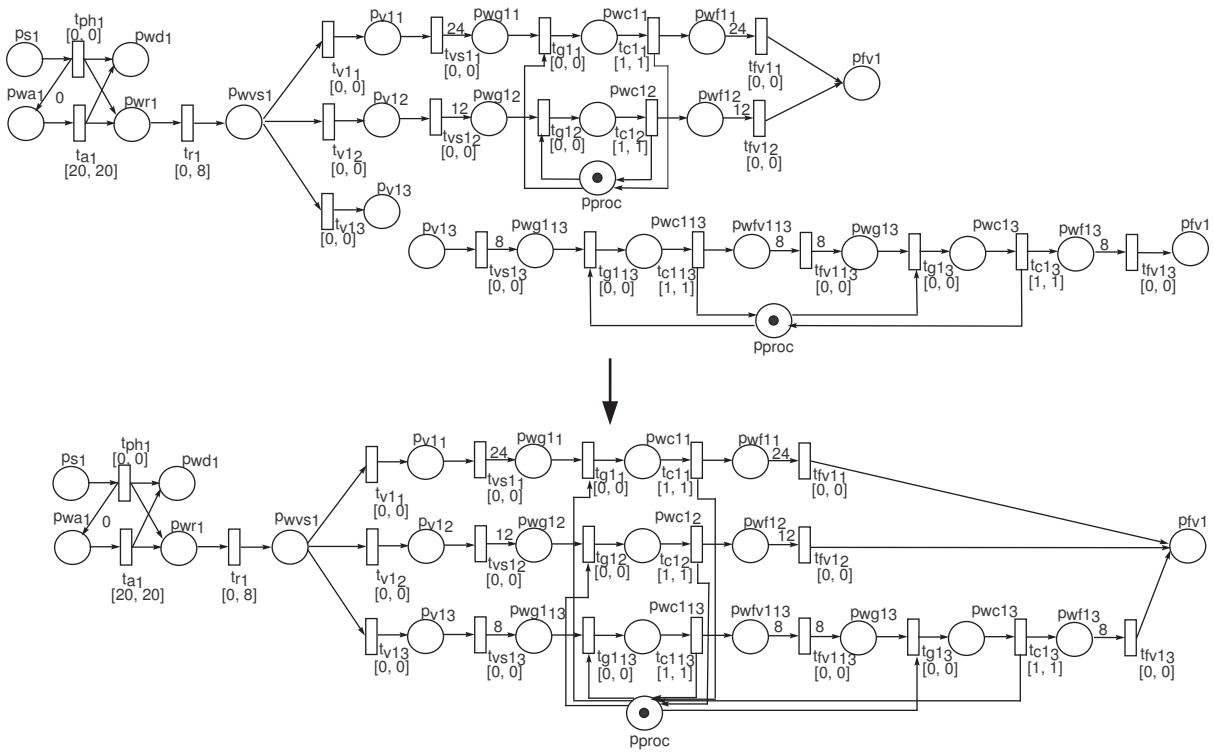


**Figure 7.7** Step 4

$$\mathcal{I}_{(a1 \sqcup v1 \sqcup p_{1_x})} = \overset{p_{st_1}}{\big[} 18432wg'_1 + wd_1 \quad \overset{p_{wa_1}}{18432wg'_1 + wd_1} \quad \overset{p_{wd_1}}{wd_1} \quad \overset{p_{wr_1}}{18432wg'_1} \quad \overset{p_{wvs_1}}{18432wg'_1}$$

$$\overset{p_{v_{1_1}}}{18432wg'_1} \quad \overset{p_{v_{1_2}}}{18432wg'_1} \quad \overset{p_{v_{1_3}}}{18432wg'_1} \quad \overset{p_{wg_{1_1}}}{768wg'_1} \quad \overset{p_{wc_{1_1}}}{768wg'_1 + proc} \quad \overset{p_{wf_{1_1}}}{768wg'_1} \quad \overset{p_{fv_1}}{18432wg_1}$$

$$\overset{p_{proc}}{proc} \quad \overset{p_{wg_{1_2}}}{1536wg'_1} \quad \overset{p_{wc_{1_2}}}{1536wg'_1 + proc} \quad \overset{p_{wf_{1_2}}}{1536wg'_1} \quad \overset{p_{wg1_{1_3}}}{2304wg'_1} \quad \overset{p_{wc1_{1_3}}}{2304wg'_1 + proc} \quad \overset{p_{wf1_{1_3}}}{2304wg'_1}$$

$$\overset{p_{wg_{1_3}}}{2304wg'_1} \quad \overset{p_{wc_{1_3}}}{2304wg'_1 + proc} \quad \overset{p_{wf_{1_3}}}{2304wg'_1} \ \big]^T;$$

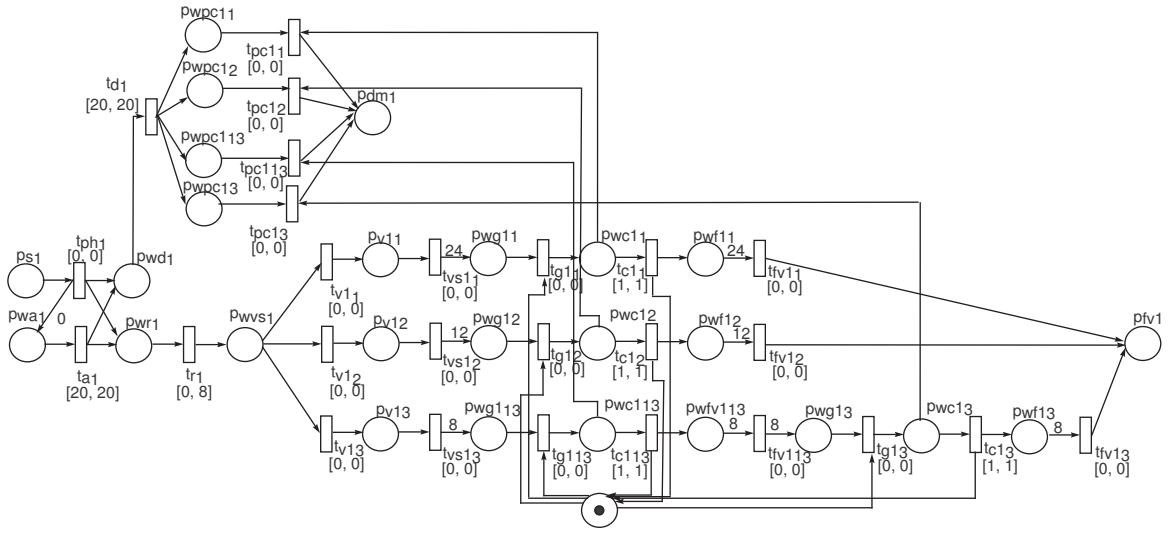


**Figure 7.8** Step 5

5. $\mathcal{N}_{(a1 \sqcup v1 \sqcup p_{1_x} \sqcup d_1)} = \mathcal{N}_{(a1 \sqcup v1 \sqcup p_{1_x})} \sqcup \mathcal{N}_{d_1}$ (Figure 7.8). The following P-invariant is obtained by juxtaposition ($d_{1,2} = d_{1,1_3} = d_{1,3} = d_{1,2,1_3} = d_{1,2,3} = d_{1,1_3,3} = d_{1,2,1_3,3} = d_{2,1_3} = d_{2,3} = d_{2,1_3,3} = d_{1_3,3} = d_0$, $d_1 = 768wg'_1$, $d_2 = 1536wg'_1$, $d_{1_3} = 2304wg'_1$, $d_3 = 2304wg'_1$, $wd_1 = 20d_0 + 2304wg'_1 + 4608wg'_1 + 6912wg'_1 + 6912wg'_1 = 20736wg'_1 + 20d_0$ and $proc = 7d_0$):

$$\mathcal{I}_{(a1 \sqcup v1 \sqcup p_{1_x} \sqcup d_1)} = \quad \big[ \ \overset{p_{st_1}}{39168wg'_1 + 20d_0} \quad \overset{p_{a_1}}{39168wg'_1 + 20d_0} \quad \overset{p_{wd_1}}{20d_0 + 20736wg'_1}$$

$$\overset{p_{wr_1}}{18432wg'_1} \quad \overset{p_{wvs_1}}{18432wg'_1} \quad \overset{p_{v_{1_1}}}{18432wg'_1} \quad \overset{p_{v_{1_2}}}{18432wg'_1} \quad \overset{p_{v_{1_3}}}{18432wg'_1} \quad \overset{p_{wg_{1_1}}}{768wg'_1} \quad \overset{p_{wc_{1_1}}}{768wg'_1 + 7d_0}$$

$$\overset{p_{wf_{1_1}}}{768wg'_1} \quad \overset{p_{fv_1}}{18432wg_1} \quad \overset{p_{proc}}{7d_0} \quad \overset{p_{wg_{1_2}}}{1536wg'_1} \quad \overset{p_{wc_{1_2}}}{1536wg'_1 + 7d_0} \quad \overset{p_{wf_{1_2}}}{1536wg'_1} \quad \overset{p_{wg1_{1_3}}}{2304wg'_1}$$

$$\overset{p_{wc1_{1_3}}}{2304wg'_1 + 7d_0} \quad \overset{p_{wf1_{1_3}}}{2304wg'_1} \quad \overset{p_{wg_{1_3}}}{2304wg'_1} \quad \overset{p_{wc_{1_3}}}{2304wg'_1 + 7d_0} \quad \overset{p_{wf_{1_3}}}{2304wg'_1} \quad \overset{p_{wpc_{1_1}}}{5d_0 + 6144wg'_1}$$

$$\overset{p_{wpc_{1_2}}}{5d_0 + 5376wg'_1} \quad \overset{p_{wpc1_{1_3}}}{5d_0 + 4608wg'_1} \quad \overset{p_{wpc_{1_3}}}{5d_0 + 4608wg'_1} \quad \overset{p_{dm_1}}{12d_0 + 6912wg'_1} \ \big]^T;$$

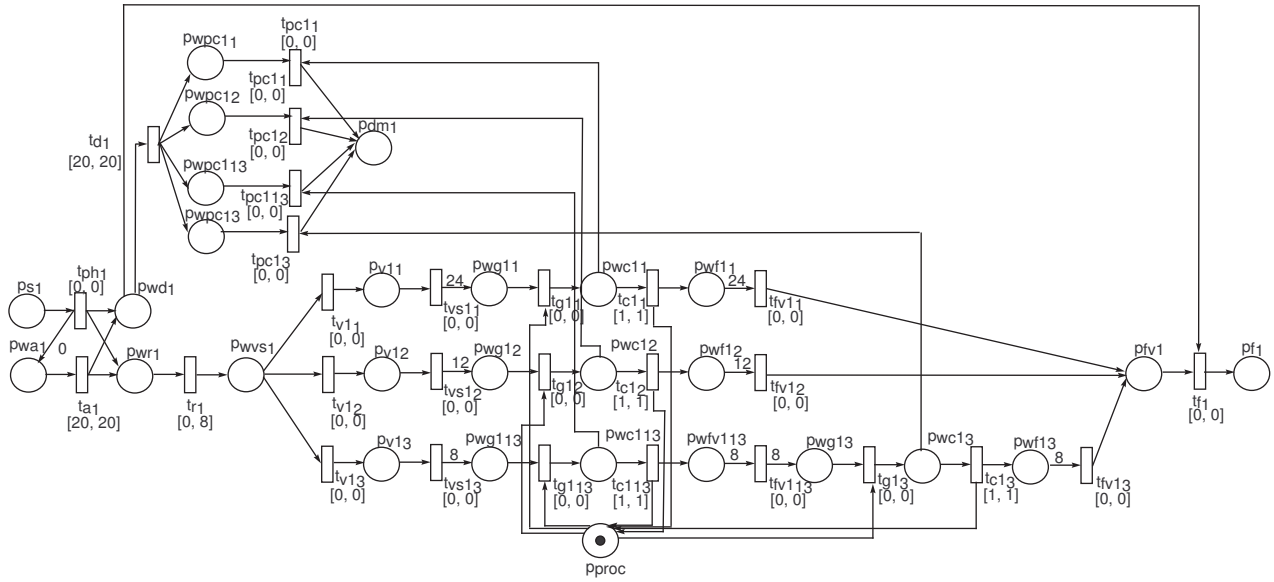


**Figure 7.9** Task $\tau_1$

6. $N_{\tau_1} = N_{(a1 \sqcup v1 \sqcup p_{1_x} \sqcup d_1)} \sqcup N_{c_1}$ (Figure 7.9). By juxtaposition, the P-invariant is ($fv_1 = 18432wg'_1$ and $wd_1 = 20d_0 + 20736wg_1$):

$$\begin{aligned}
\mathcal{I}_{\tau_1} = \Big[ & \overset{p_{st_1}}{39168wg'_1 + 20d_0} \quad \overset{p_{a_1}}{39168wg'_1 + 20d_0} \quad \overset{p_{wd_1}}{20d_0 + 20736wg'_1} \quad \overset{p_{wr_1}}{18432wg'_1} \\
& \overset{p_{wvs_1}}{18432wg'_1} \quad \overset{p_{v_{1_1}}}{18432wg'_1} \quad \overset{p_{v_{1_2}}}{18432wg'_1} \quad \overset{p_{v_{1_3}}}{18432wg'_1} \quad \overset{p_{wg_{1_1}}}{768wg'_1} \quad \overset{p_{wc_{1_1}}}{768wg'_1 + 7d_0} \quad \overset{p_{wf_{1_1}}}{768wg'_1} \\
& \overset{p_{fv_1}}{18432wg_1} \quad \overset{p_{proc}}{7d_0} \quad \overset{p_{wg_{1_2}}}{1536wg'_1} \quad \overset{p_{wc_{1_2}}}{1536wg'_1 + 7d_0} \quad \overset{p_{wf_{1_2}}}{1536wg'_1} \quad \overset{p_{wg1_{1_3}}}{2304wg'_1} \quad \overset{p_{wc1_{1_3}}}{2304wg'_1 + 7d_0} \\
& \overset{p_{wf1_{1_3}}}{2304wg'_1} \quad \overset{p_{wg_{1_3}}}{2304wg'_1} \quad \overset{p_{wc_{1_3}}}{2304wg'_1 + 7d_0} \quad \overset{p_{wf_{1_3}}}{2304wg'_1} \quad \overset{p_{wpc_{1_1}}}{5d_0 + 6144wg'_1} \quad \overset{p_{wpc_{1_2}}}{5d_0 + 5376wg'_1} \\
& \overset{p_{wpc1_{1_3}}}{5d_0 + 4608wg'_1} \quad \overset{p_{wpc_{1_3}}}{5d_0 + 4608wg'_1} \quad \overset{p_{dm_1}}{12d_0 + 6912wg'_1} \quad \overset{p_{f_1}}{20d_0 + 39168wg_1} \Big]^T;
\end{aligned}$$

Task $\tau_2$ ($\mathcal{N}_{\tau_2}$) (Figure 7.10) is composed similarly and the respective P-invariant is likewise obtained:

$$\begin{aligned}
\diamondsuit\ \mathcal{I}_{\tau_2} = \Big[ & \overset{p_{st_2}}{396wg'_2 + 20d'_0} \quad \overset{p_{a_2}}{396wg'_2 + 20d'_0} \quad \overset{p_{wd_2}}{20d'_0 + 324wg'_2} \quad \overset{p_{wr_2}}{72wg'_2} \quad \overset{p_{wvs_2}}{72wg'_2} \quad \overset{p_{v_{2_1}}}{72wg'_2} \\
& \overset{p_{wg_{2_1}}}{72wg'_2} \quad \overset{p_{wc_{2_1}}}{72_w g2'} \quad \overset{p_{v_{2_2}}}{12wg'_2} \quad \overset{p_{v_{2_3}}}{12wg'_2 + 7d'_0} \quad \overset{p_{wf_{2_1}}}{12wg'_2} \quad \overset{p_{fv_2}}{72wg'_2} \quad \overset{p_{proc}}{7d'_0} \quad \overset{p_{wg_{2_2}}}{24wg'_2} \quad \overset{p_{wc_{2_2}}}{24wg'_2 + 7d'_0} \\
& \overset{p_{wf_{2_2}}}{24wg'_2} \quad \overset{p_{wg1_{2_3}}}{36wg'_1} \quad \overset{p_{wc1_{2_3}}}{36wg'_2 + 7d'_0} \quad \overset{p_{wf1_{2_3}}}{36wg'_2} \quad \overset{p_{wg_{2_3}}}{36wg'_2} \quad \overset{p_{wc_{2_3}}}{36wg'_2 + 7d'_0} \quad \overset{p_{wf_{2_3}}}{36wg'_2} \quad \overset{p_{wpc_{2_1}}}{5d'_0 + 96wg'_2}
\end{aligned}$$

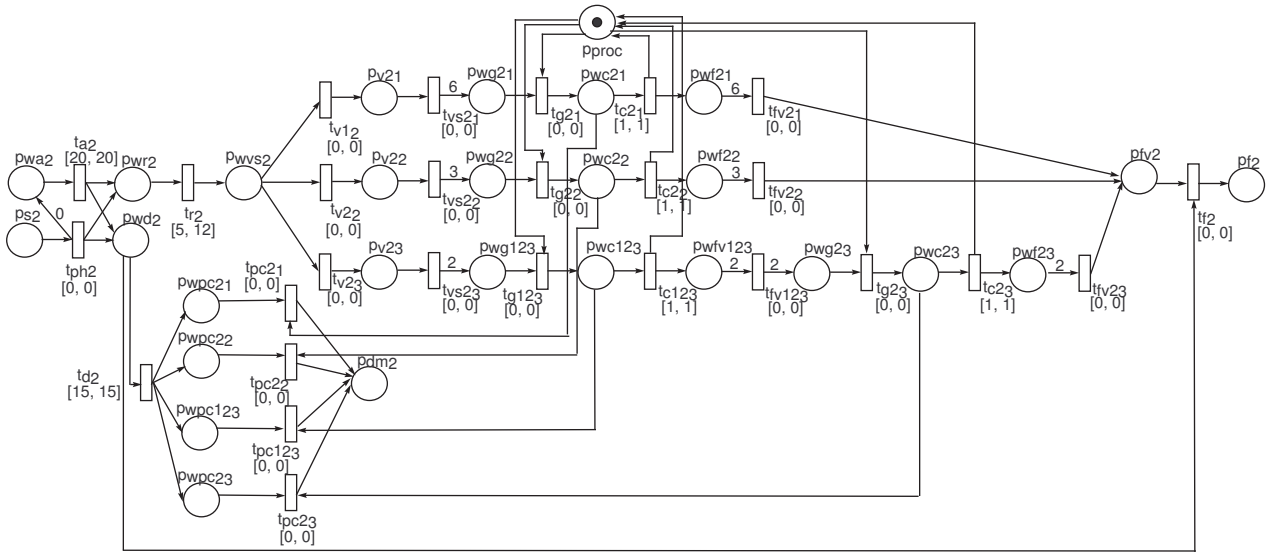


**Figure 7.10** Task $\tau_2$

$$\begin{matrix} p_{wpc_{2_2}} & p_{wpc1_{2_3}} & p_{wpc_{2_3}} & p_{dm_2} & p_{f_2} \\ 5d'_0 + 84wg'_2 & 5d'_0 + 72wg'_1 & 5d'_0 + 72wg'_1 & 12d'_0 + 108wg'_2 & 20d'_0 + 396wg'_2 \end{matrix} \Big]^T.$$

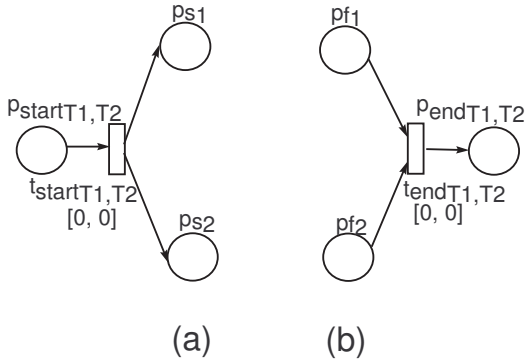


**Figure 7.11** Fork and join blocks

### 7.2.2 Merging the Modeled Tasks

After the composition of each individual task, the task models are merged and fused to the fork and join blocks, which are instantiated considering the number of tasks defined in the specification. Taking into account the adopted example, the fork and join blocks are instantiated considering two real-time tasks (see Figure 7.11(a) and Figure 7.11(b)). For these Petri net models, the respective P-invariants are:

$$\diamondsuit\ \mathcal{N}_f\colon \mathcal{I}_f = \begin{matrix} p_{start_{T_1,T_2}} & p_{st_1} & p_{st_2} \\ \Big[\ st_1 + st_2 & st_1 & st_2 \end{matrix} \Big]^T$$

$$\diamondsuit\ \mathcal{N}_j\text{: } \mathcal{I}_j = \overset{\begin{matrix} p_{f_1} & p_{f_2} & p_{end_{T_1,T_2}} \end{matrix}}{\begin{bmatrix} f_1 & f_2 & f_1+f_2 \end{bmatrix}^T}$$

As follows, the last steps are presented, including, also, the juxtaposition of the P-invariants:

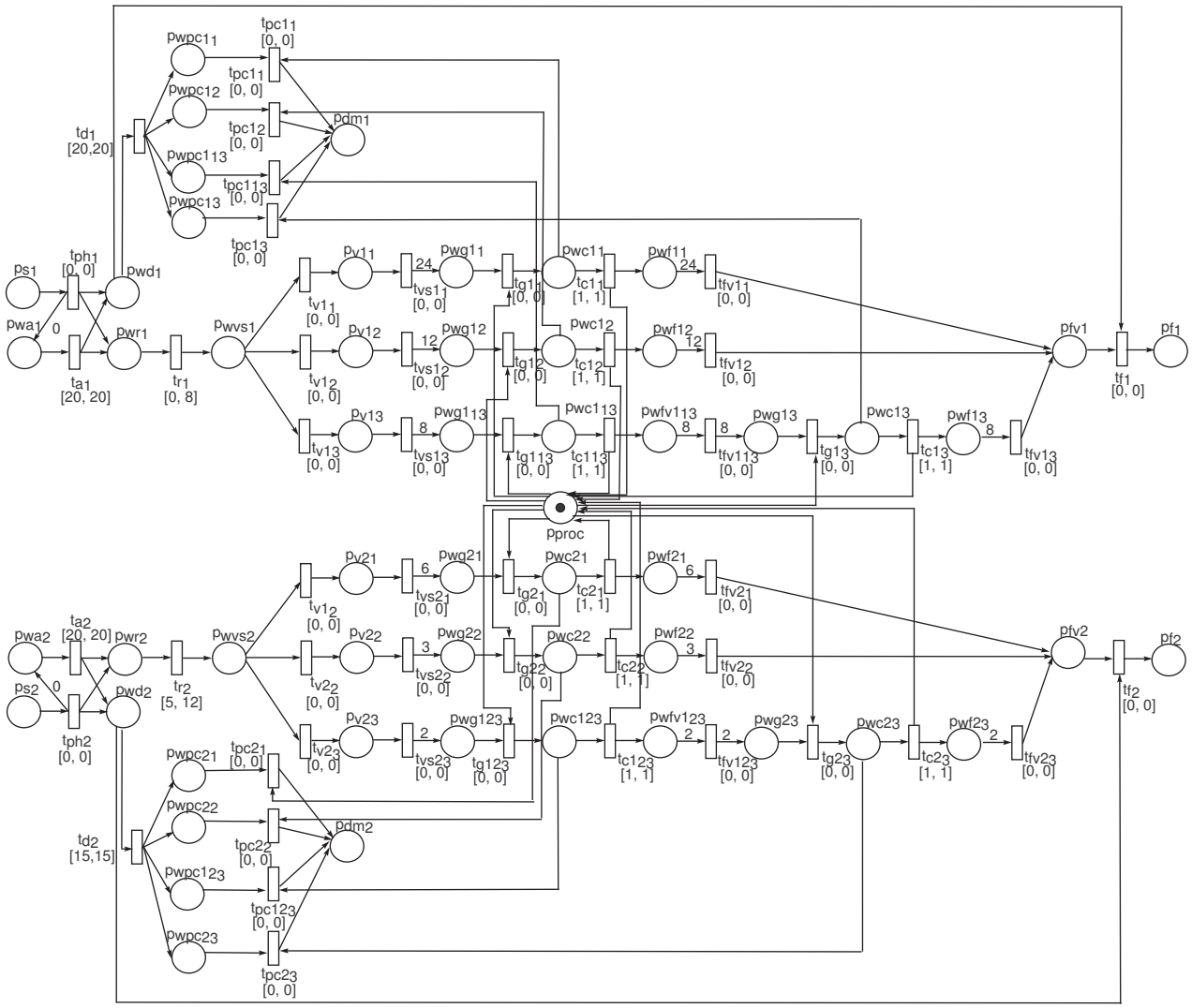


**Figure 7.12** Joining two tasks

1. $\mathcal{N}_{(\tau_1 \sqcup \tau_2)} = \mathcal{N}_{\tau_1} \sqcup \mathcal{N}_{\tau_2}$(Figure 7.12). By juxtaposition ($\mathcal{J}(\mathcal{I}_{\tau_1}, \mathcal{I}_{\tau_2}) = \mathcal{I}_{(\tau_1 \sqcup \tau_2)}$ and $d'_0 = d_0$):

$$\mathcal{I}_{(\tau_1 \sqcup \tau_2)} = \overset{\begin{matrix} p_{st_1} & & p_{a_1} & & p_{wd_1} & & p_{wr_1} \end{matrix}}{\begin{bmatrix} 39168wg'_1 + 20d_0 & 39168wg'_1 + 20d_0 & 20d_0 + 20736wg'_1 & 18432wg'_1 \end{bmatrix}}$$

$$\overset{\begin{matrix} p_{wvs_1} & p_{v_{1_1}} & p_{v_{1_2}} & p_{v_{1_3}} & p_{wg_{1_1}} & p_{wc_{1_1}} & p_{wf_{1_1}} \end{matrix}}{\begin{bmatrix} 18432wg'_1 & 18432wg'_1 & 18432wg'_1 & 18432wg'_1 & 768wg'_1 & 768wg'_1 + 7d_0 & 768wg'_1 \end{bmatrix}}$$

$$
\begin{array}{cccccccc}
p_{fv_1} & p_{proc} & p_{wg_{1_2}} & p_{wc_{1_2}} & p_{wf_{1_2}} & p_{wg1_{1_3}} & p_{wc1_{1_3}} \\
18432wg_1 & 7d_0 & 1536wg_1' & 1536wg_1'+7d_0 & 1536wg_1' & 2304wg_1' & 2304wg_1'+7d_0 \\
\\
p_{wf1_{1_3}} & p_{wg_{1_3}} & p_{wc_{1_3}} & p_{wf_{1_3}} & p_{wpc_{1_1}} & p_{wpc_{1_2}} \\
2304wg_1' & 2304wg_1' & 2304wg_1'+7d_0 & 2304wg_1' & 5d_0+6144wg_1' & 5d_0+5376wg_1' \\
\\
p_{wpc1_{1_3}} & p_{wpc_{1_3}} & p_{dm_1} & p_{f_1} \\
5d_0+4608wg_1' & 5d_0+4608wg_1' & 12d_0+6912wg_1' & 20d_0+39168wg_1 \\
\\
p_{st_2} & p_{a_2} & p_{wd_2} & p_{wr_2} & p_{wvs_2} & p_{v_{2_1}} & p_{v_{2_2}} \\
396wg_2'+20d_0 & 396wg_2'+20d_0 & 20d_0+324wg_2' & 72wg_2' & 72wg_2' & 72wg_2' & 72wg_2' \\
\\
p_{v_{2_3}} & p_{wg_{2_1}} & p_{wc_{2_1}} & p_{wf_{2_1}} & p_{fv_2} & p_{wg_{2_2}} & p_{wc_{2_2}} & p_{wf_{2_2}} \\
72_wg2' & 12wg_2' & 12wg_2'+7d_0 & 12wg_2' & 72wg_2' & 24wg_2' & 24wg_2'+7d_0 & 24wg_2' \\
\\
p_{wg1_{2_3}} & p_{wc1_{2_3}} & p_{wf1_{2_3}} & p_{wg_{2_3}} & p_{wc_{2_3}} & p_{wf_{2_3}} & p_{wpc_{2_1}} \\
36wg_1' & 36wg_2'+7d_0 & 36wg_2' & 36wg_2' & 36wg_2'+7d_0 & 36wg_2' & 5d_0+96wg_2' \\
\\
p_{wpc_{2_2}} & p_{wpc1_{2_3}} & p_{wpc_{2_3}} & p_{dm_2} & p_{f_2} \\
5d_0+84wg_2' & 5d_0+72wg_1' & 5d_0+72wg_1' & 12d_0+108wg_2' & 20d_0+396wg_2' \ \big]^T
\end{array}
$$

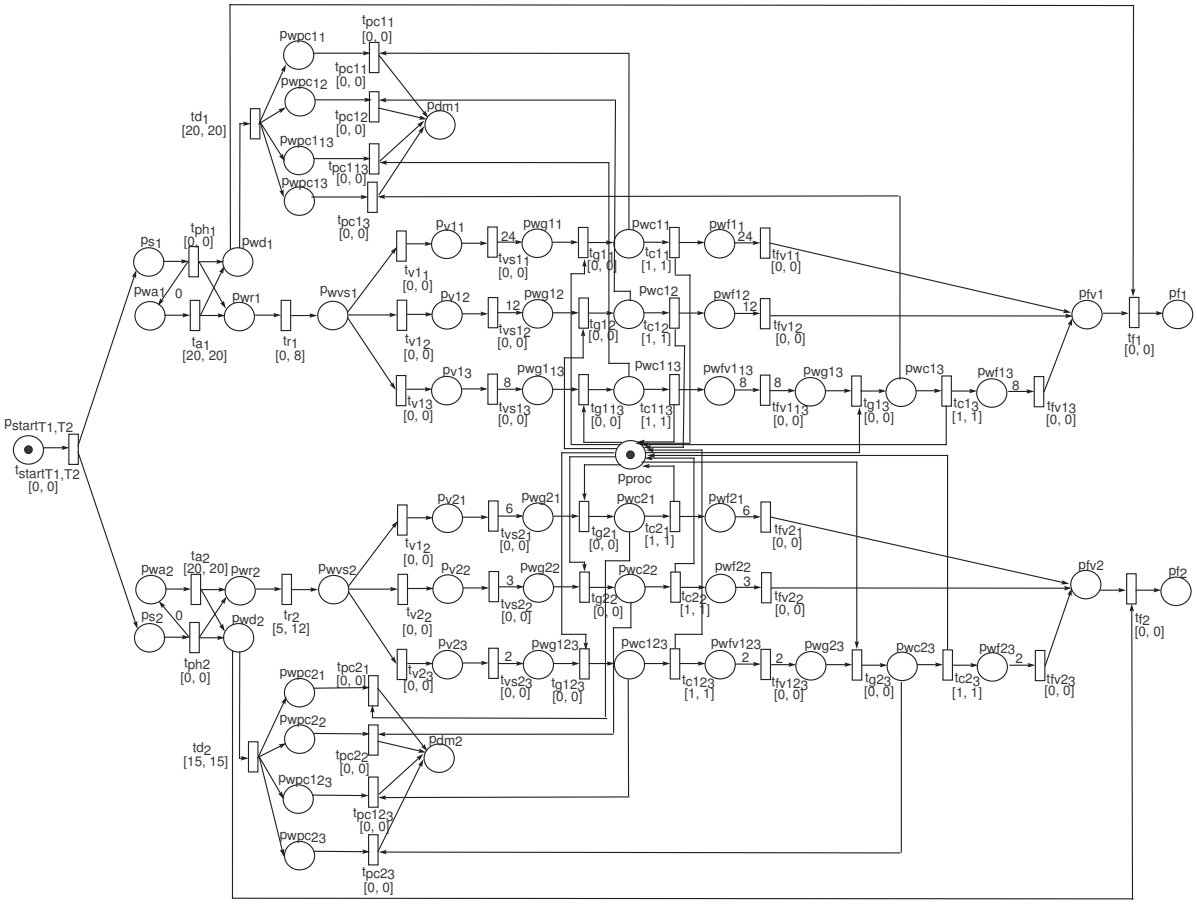


**Figure 7.13** Merging fork block

2. $\mathcal{N}_{(\tau_1 \sqcup \tau_2 \sqcup f)} = \mathcal{N}_{(\tau_1 \sqcup \tau_2)} \sqcup \mathcal{N}_f$(Figure 7.13). Adjusting $st_1 = 39168wg'_1 + 20d_0$ and $st_2 = 396wg'_2 + 20d_0$, the following P-invariant is obtained by juxtaposition:

$$\mathcal{I}_{(\tau_1 \sqcup \tau_2 \sqcup f)} = \begin{bmatrix} \overset{p_{st_1}}{39168wg'_1 + 20d_0} & \overset{p_{a_1}}{39168wg'_1 + 20d_0} & \overset{p_{wd_1}}{20d_0 + 20736wg'_1} & \overset{p_{wr_1}}{18432wg'_1} \\ \overset{p_{wvs_1}}{18432wg'_1} & \overset{p_{v_{1_1}}}{18432wg'_1} & \overset{p_{v_{1_2}}}{18432wg'_1} & \overset{p_{v_{1_3}}}{18432wg'_1} & \overset{p_{wg_{1_1}}}{768wg'_1} & \overset{p_{wc_{1_1}}}{768wg'_1 + 7d_0} & \overset{p_{wf_{1_1}}}{768wg'_1} \\ \overset{p_{fv_1}}{18432wg_1} & \overset{p_{proc}}{7d_0} & \overset{p_{wg_{1_2}}}{1536wg'_1} & \overset{p_{wc_{1_2}}}{1536wg'_1 + 7d_0} & \overset{p_{wf_{1_2}}}{1536wg'_1} & \overset{p_{wg1_{1_3}}}{2304wg'_1} & \overset{p_{wc1_{1_3}}}{2304wg'_1 + 7d_0} \\ \overset{p_{wf1_{1_3}}}{2304wg'_1} & \overset{p_{wg_{1_3}}}{2304wg'_1} & \overset{p_{wc_{1_3}}}{2304wg'_1 + 7d_0} & \overset{p_{wf_{1_3}}}{2304wg'_1} & \overset{p_{wpc_{1_1}}}{5d_0 + 6144wg'_1} & \overset{p_{wpc_{1_2}}}{5d_0 + 5376wg'_1} \\ \overset{p_{wpc1_{1_3}}}{5d_0 + 4608wg'_1} & \overset{p_{wpc_{1_3}}}{5d_0 + 4608wg'_1} & \overset{p_{dm_1}}{12d_0 + 6912wg'_1} & \overset{p_{f_1}}{20d_0 + 39168wg_1} & \overset{p_{st_2}}{396wg'_2 + 20d_0} \\ \overset{p_{a_2}}{396wg'_2 + 20d_0} & \overset{p_{wd_2}}{20d_0 + 324wg'_2} & \overset{p_{wr_2}}{72wg'_2} & \overset{p_{wvs_2}}{72wg'_2} & \overset{p_{v_{2_1}}}{72wg'_2} & \overset{p_{v_{2_2}}}{72wg'_2} & \overset{p_{v_{2_3}}}{72_w g2'} & \overset{p_{wg_{2_1}}}{12wg'_2} \\ \overset{p_{wc_{2_1}}}{12wg'_2 + 7d_0} & \overset{p_{wf_{2_1}}}{12wg'_2} & \overset{p_{fv_2}}{72wg'_2} & \overset{p_{wg_{2_2}}}{24wg'_2} & \overset{p_{wc_{2_2}}}{24wg'_2 + 7d_0} & \overset{p_{wf_{2_2}}}{24wg'_2} & \overset{p_{wg1_{2_3}}}{36wg'_1} & \overset{p_{wc1_{2_3}}}{36wg'_2 + 7d_0} \\ \overset{p_{wf1_{2_3}}}{36wg'_2} & \overset{p_{wg_{2_3}}}{36wg'_2} & \overset{p_{wc_{2_3}}}{36wg'_2 + 7d_0} & \overset{p_{wf_{2_3}}}{36wg'_2} & \overset{p_{wpc_{2_1}}}{5d_0 + 96wg'_2} & \overset{p_{wpc_{2_2}}}{5d_0 + 84wg'_2} & \overset{p_{wpc1_{2_3}}}{5d_0 + 72wg'_1} \\ \overset{p_{wpc_{2_3}}}{5d_0 + 72wg'_1} & \overset{p_{dm_2}}{12d_0 + 108wg'_2} & \overset{p_{f_2}}{20d_0 + 396wg'_2} & \overset{p_{start_{T_1,T_2}}}{39168wg'_1 + 396wg'_2 + 40d_0} \end{bmatrix}^T$$

3. $\mathcal{N}_{T_1,T_2} = \mathcal{N}_{(\tau_1 \sqcup \tau_2 \sqcup f)} \sqcup \mathcal{N}_j$(Figure 7.14). By juxtaposition ($f_1 = 20d_0 + 39168wg'_1$ and $f_2 = 20d_0 + 396wg'_2$), the following P-invariant is obtained:

$$\mathcal{I}_{(T_1,T_2)} = \begin{bmatrix} \overset{p_{st_1}}{39168wg'_1 + 20d_0} & \overset{p_{a_1}}{39168wg'_1 + 20d_0} & \overset{p_{wd_1}}{20d_0 + 20736wg'_1} & \overset{p_{wr_1}}{18432wg'_1} \\ \overset{p_{wvs_1}}{18432wg'_1} & \overset{p_{v_{1_1}}}{18432wg'_1} & \overset{p_{v_{1_2}}}{18432wg'_1} & \overset{p_{v_{1_3}}}{18432wg'_1} & \overset{p_{wg_{1_1}}}{768wg'_1} & \overset{p_{wc_{1_1}}}{768wg'_1 + 7d_0} \\ \overset{p_{wf_{1_1}}}{768wg'_1} & \overset{p_{fv_1}}{18432wg_1} & \overset{p_{proc}}{7d_0} & \overset{p_{wg_{1_2}}}{1536wg'_1} & \overset{p_{wc_{1_2}}}{1536wg'_1 + 7d_0} & \overset{p_{wf_{1_2}}}{1536wg'_1} & \overset{p_{wg1_{1_3}}}{2304wg'_1} \\ \overset{p_{wc1_{1_3}}}{2304wg'_1 + 7d_0} & \overset{p_{wf1_{1_3}}}{2304wg'_1} & \overset{p_{wg_{1_3}}}{2304wg'_1} & \overset{p_{wc_{1_3}}}{2304wg'_1 + 7d_0} & \overset{p_{wf_{1_3}}}{2304wg'_1} & \overset{p_{wpc_{1_1}}}{5d_0 + 6144wg'_1} \\ \overset{p_{wpc_{1_2}}}{5d_0 + 5376wg'_1} & \overset{p_{wpc1_{1_3}}}{5d_0 + 4608wg'_1} & \overset{p_{wpc_{1_3}}}{5d_0 + 4608wg'_1} & \overset{p_{dm_1}}{12d_0 + 6912wg'_1} & \overset{p_{f_1}}{20d_0 + 39168wg_1} \\ \overset{p_{st_2}}{396wg'_2 + 20d_0} & \overset{p_{a_2}}{396wg'_2 + 20d_0} & \overset{p_{wd_2}}{20d_0 + 324wg'_2} & \overset{p_{wr_2}}{72wg'_2} & \overset{p_{wvs_2}}{72wg'_2} & \overset{p_{v_{2_1}}}{72wg'_2} & \overset{p_{v_{2_2}}}{72wg'_2} \\ \overset{p_{v_{2_3}}}{72_w g2'} & \overset{p_{wg_{2_1}}}{12wg'_2} & \overset{p_{wc_{2_1}}}{12wg'_2 + 7d_0} & \overset{p_{wf_{2_1}}}{12wg'_2} & \overset{p_{fv_2}}{72wg'_2} & \overset{p_{wg_{2_2}}}{24wg'_2} & \overset{p_{wc_{2_2}}}{24wg'_2 + 7d_0} & \overset{p_{wf_{2_2}}}{24wg'_2} \end{bmatrix}$$

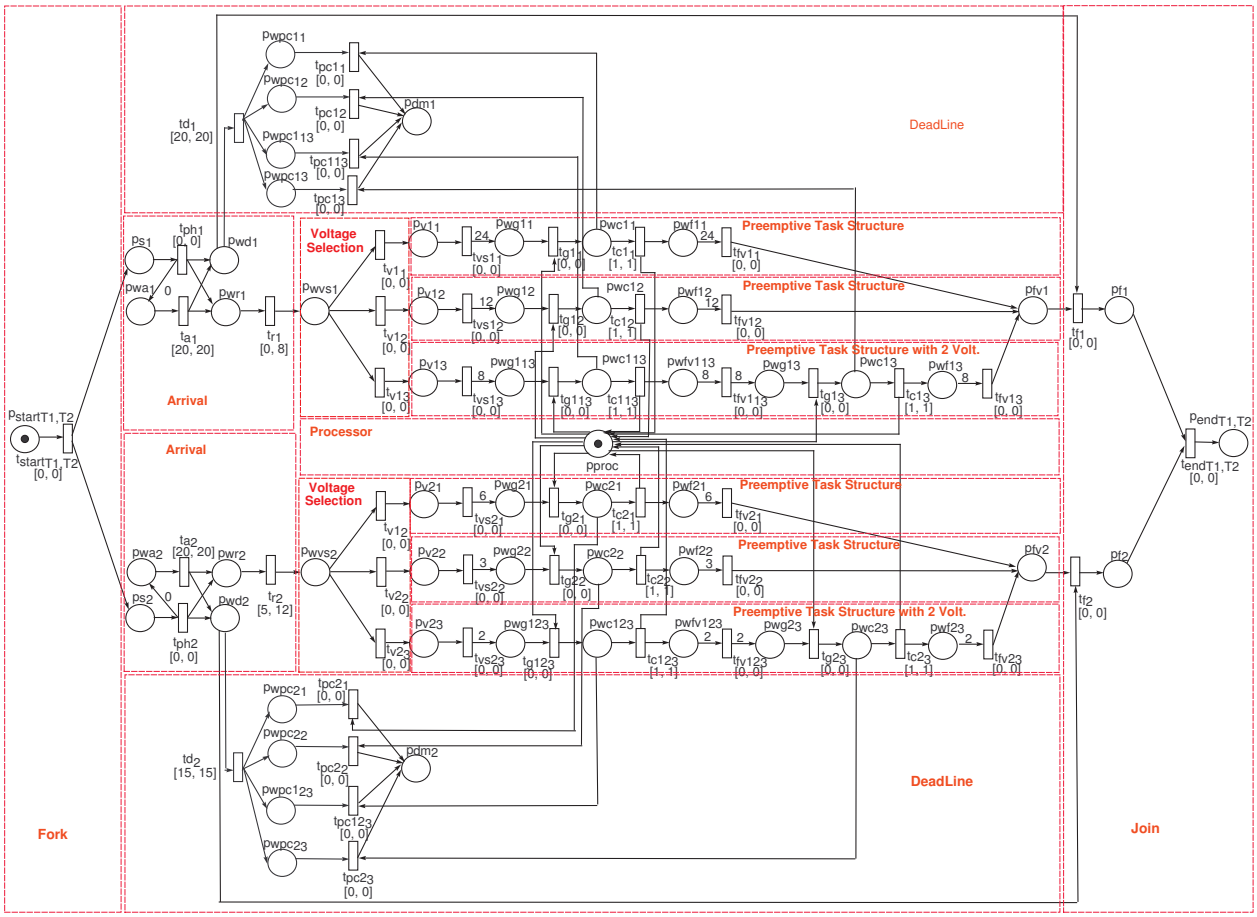


**Figure 7.14** Generated model

$$\begin{array}{ccccccc} p_{wg1_{2_3}} & p_{wc1_{2_3}} & p_{wf1_{2_3}} & p_{wg_{2_3}} & p_{wc_{2_3}} & p_{wf_{2_3}} & p_{wpc_{2_1}} \\ 36wg_1' & 36wg_2' + 7d_0 & 36wg_2' & 36wg_2' & 36wg_2' + 7d_0 & 36wg_2' & 5d_0 + 96wg_2' \end{array}$$

$$\begin{array}{ccccc} p_{wpc_{2_2}} & p_{wpc1_{2_3}} & p_{wpc_{2_3}} & p_{dm_2} & p_{f_2} \\ 5d_0 + 84wg_2' & 5d_0 + 72wg_1' & 5d_0 + 72wg_1' & 12d_0 + 108wg_2' & 20d_0 + 396wg_2' \end{array}$$

$$\begin{array}{cc} p_{start_{T_1,T_2}} & p_{end_{T_1,T_2}} \\ 39168wg_1' + 396wg_2' + 40d_0 & 39168wg_1 + 396wg_2' + 40d_0 \ ]^T \end{array}$$

As $\mathcal{I}^T_{(T_1,T_2)} \times A_{T_1,T_2} = 0$, in which $A_{T_1,T_2}$ is the respective incidence matrix and $\mathcal{I}^T_{(T_1,T_2)} > 0$, the model representing the specification is structurally conservative as well as structurally bounded. For a better understanding, the following lines explain the model generated.

In Figure 7.14, the fork block is responsible for starting tasks $\tau_1$ and $\tau_2$, such that, after the firing of transition $t_{start_{spec}}$, both tasks become eligible for execution. For a better visualization, the upper blocks model task $\tau_1$ and the lower blocks model task $\tau_2$, in such a way that both tasks are assigned to the same processor (place $Pproc$). Additionally, note that each voltage selection block takes into account 3 voltage/frequency

levels for executing tasks $\tau_1$ and $\tau_2$. More specifically, transitions $t_{v_{1_1}}$ and $t_{v_{2_1}}$ represent the tasks executing at 1V/10MHz, transitions $t_{v_{1_2}}$ and $t_{v_{2_2}}$ depict the execution at 2V/20MHz, and, similarly, transitions $t_{v_{1_3}}$ and $t_{v_{2_3}}$ represent the tasks' execution at 1.5V/15MHz. For instance, the computation time of task $\tau_1$ at 1V/10MHz is $24s$, since $C = \lceil 240 \times 10^6, /10 \times 10^6 \rceil = 24s$. Furthermore, each task structure block (including the block with 2 voltages) is connected to the deadline block of the respective task. These connections are required, since, if a deadline occurs during a task computation, the execution is no longer possible (see Section 6.3.7). Finally, after evaluating each task, transition $t_{end_{spec}}$ is fired, so that a token is stored in place $p_{end_{spec}}$, reporting that a feasible schedule has been found (desired final marking).

## 7.3 MODELING INTERTASK RELATIONS

This section provides the steps for modeling tasks with precedence and exclusion relations using the building blocks presented in Section 6.3.

### 7.3.1 Modeling Precedence Relations

As presented in Chapter 5, precedence relations are defined between pairs of tasks. Suppose that $\tau_i$ *PRECEDES* $\tau_j$ is specified, that is, one task can only start its execution after finishing the other task's execution. The adopted modeling approach adds a single place (initially unmarked) shared by the two tasks' models, such that this place is pre-condition for task $\tau_j$ and post-condition for task $\tau_i$. Only after the conclusion of task $\tau_i$, a token is made available in such place in order to allow the execution of task $\tau_j$.

As an example, consider the same specification presented in Section 7.2 : $\tau_1$ = (0, 0, $240 \times 10^6$, 20, 20) and $\tau_2$ = (0, 5, $60 \times 10^6$, 15, 20). Besides, assume the following CPU voltage levels $vff$ = {(1V,10MHz),(2V,20MHz)}, and also the unavailable level of 1.5V/15MHz (which can be "simulated" using the 2 immediate neighboring voltage levels and the respective maximum CPU frequencies). This specification is augmented with the following intertask relation: $\tau_1$ *PRECEDES* $\tau_2$.

In addition to the conventional blocks, task $\tau_1$ requires the utilization of one task instance conclusion with intertask relations block and task $\tau_2$ necessitates one precedence pre-condition block. These blocks are instantiated considering one intertask relation and one precedence relation, respectively. Figure 7.15 depicts the building blocks for modeling task $\tau_1$ and Figure 7.16 for task $\tau_2$. For the sake of conciseness, these figures show each task partially composed ($\mathcal{N}_{(v_1 \sqcup p_{1_x} \sqcup d_1)}$ and $\mathcal{N}_{(v_2 \sqcup p_{2_x} \sqcup d_2)}$). In this particular specification, place renaming operator must be applied in some building blocks before net union, so that place merging can be properly performed. The place renaming functions adopted are:

$$\rho_{cinter_1}(p) = \begin{cases} p_{prec_1}, & if\ (p = p_{rel_1}) \\ p, & else \end{cases}$$

$$\rho_{a_2}(p) = \begin{cases} p_{wprec_2}, & if\ (p = p_{wvs_2}) \\ p, & else \end{cases}$$

The composition steps are presented below:

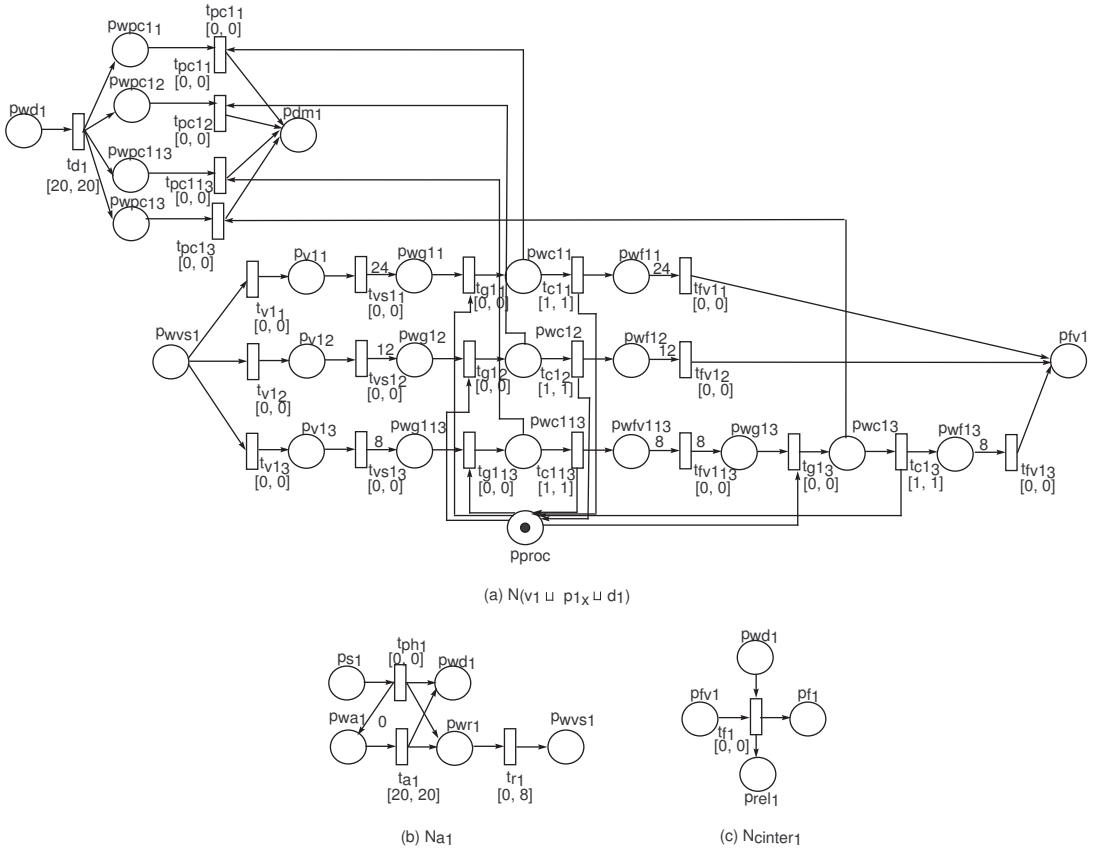


**Figure 7.15** Blocks for modeling task $\tau_1$ with precedence relation

1. $\mathcal{N}'_{cinter_1} = \mathcal{N}_{cinter_1}/\rho_{cinter_1}(p)$;
2. $\mathcal{N}'_{a2} = \mathcal{N}_{a2}/\rho_{a2}(p)$;
3. Considering the instantiation of fork ($\mathcal{N}_f$) and join blocks ($\mathcal{N}_j$) with 2 tasks, $\mathcal{N}_{T_1,T_{2pre}} = \mathcal{N}_f \sqcup \mathcal{N}_j \sqcup \mathcal{N}_{a_1} \sqcup \mathcal{N}'_{cinter_1} \sqcup \mathcal{N}_{(v_1 \sqcup p_{1_x} \sqcup d_1)} \sqcup \mathcal{N}'_{a2} \sqcup \mathcal{N}_{c_2} \sqcup \mathcal{N}_{(v_2 \sqcup p_{2_x} \sqcup d_2)}$.

Figure 7.17 shows the TPNE model for tasks $\tau_1$ and $\tau_2$, in which $\tau_1$ *PRECEDES* $\tau_2$ and considering that both tasks are preemptive.

### 7.3.2 Modeling Exclusion Relations

Exclusion relations are also defined between pairs of tasks. Suppose that $\tau_i$ *EXCLUDES* $\tau_j$ is specified. This relation models a situation that two tasks cannot be concurrently executed. In other words, if task $\tau_i$ starts executing, task $\tau_j$ has to wait up to task $\tau_i$ finishes its execution and vice-versa. The proposed modeling method adds a single place shared by the two tasks' models. This place must have one token (and only one) as pre-condition for the exclusive execution of any of these two tasks. Therefore, just one of them could be executing at a time.

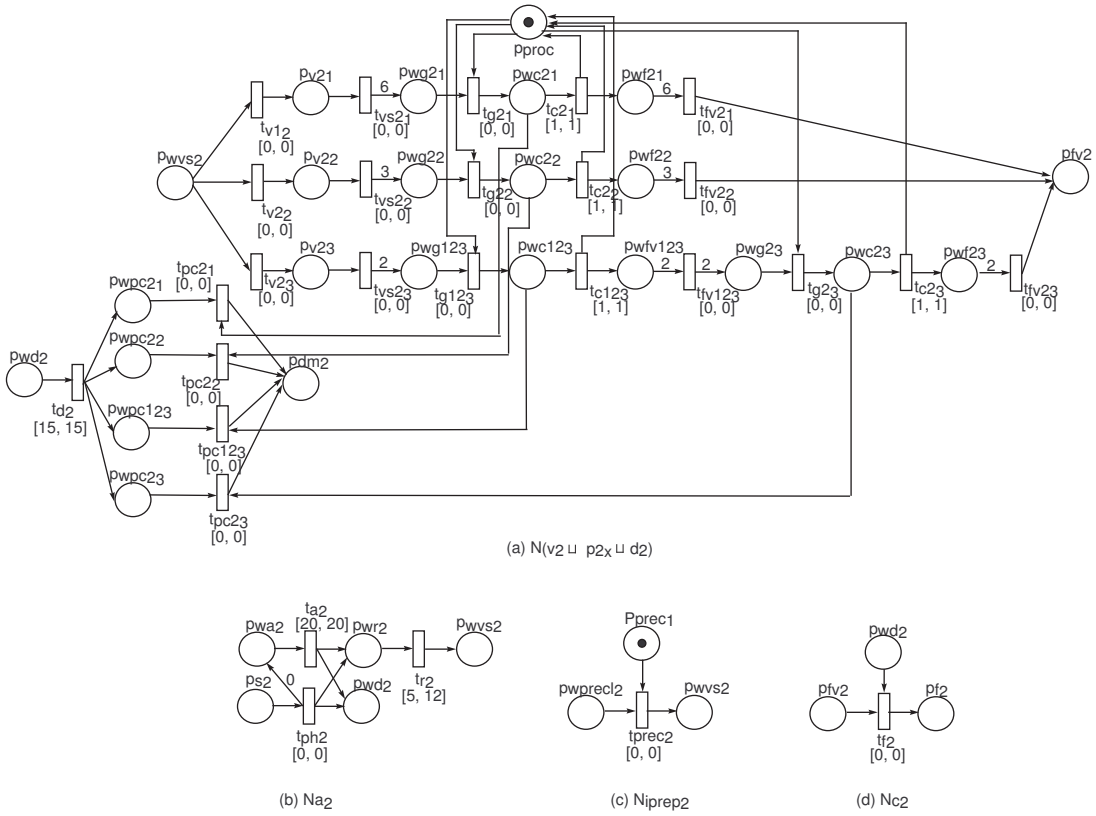


**Figure 7.16** Blocks for modeling task $\tau_2$ with precedence relation

In order to demonstrate an example, consider the specification that has been adopted in previous sections: $\tau_1 = (0,\ 0,\ 240 \times 10^6,\ 20,\ 20)$ and $\tau_2 = (0,\ 5,\ 60 \times 10^6,\ 15,\ 20)$. Additionally, assume the same CPU voltage levels $\mathit{vff} = \{(1\text{V},10\text{MHz}),(2\text{V},20\text{MHz})\}$, and also the unavailable level of 1.5V/15MHz (which can be "simulated" using the 2 immediate neighboring voltage levels and the respective maximum CPU frequencies). In this section, the specification is augmented with the following intertask relation: $\tau_1$ *EXCLUDES* $\tau_2$.

In addition to the conventional blocks, each individual task requires the utilization of one task instance conclusion with intertask relations block and one exclusion precondition block, in such a way that these blocks are instantiated considering one intertask relation and one exclusion relation, respectively. Figure 7.18 depicts the building blocks for modeling task $\tau_1$ and Figure 7.19 for task $\tau_2$. For the sake of conciseness, these figures show each task partially composed ($\mathcal{N}_{(v_1 \sqcup p_{1_x} \sqcup d_1)}$ and $\mathcal{N}_{(v_2 \sqcup p_{2_x} \sqcup d_2)}$).

Similar to the modeling of precedence relations, place renaming operator must be applied in some building blocks before net union, so that place merging is properly performed. Below, the place renaming functions that are going to be adopted in the composition:

$$\rho_{a_1}(p) = \begin{cases} p_{wexcl_1}, & if\ (p = p_{wvs_1}) \\ p, & else \end{cases}$$

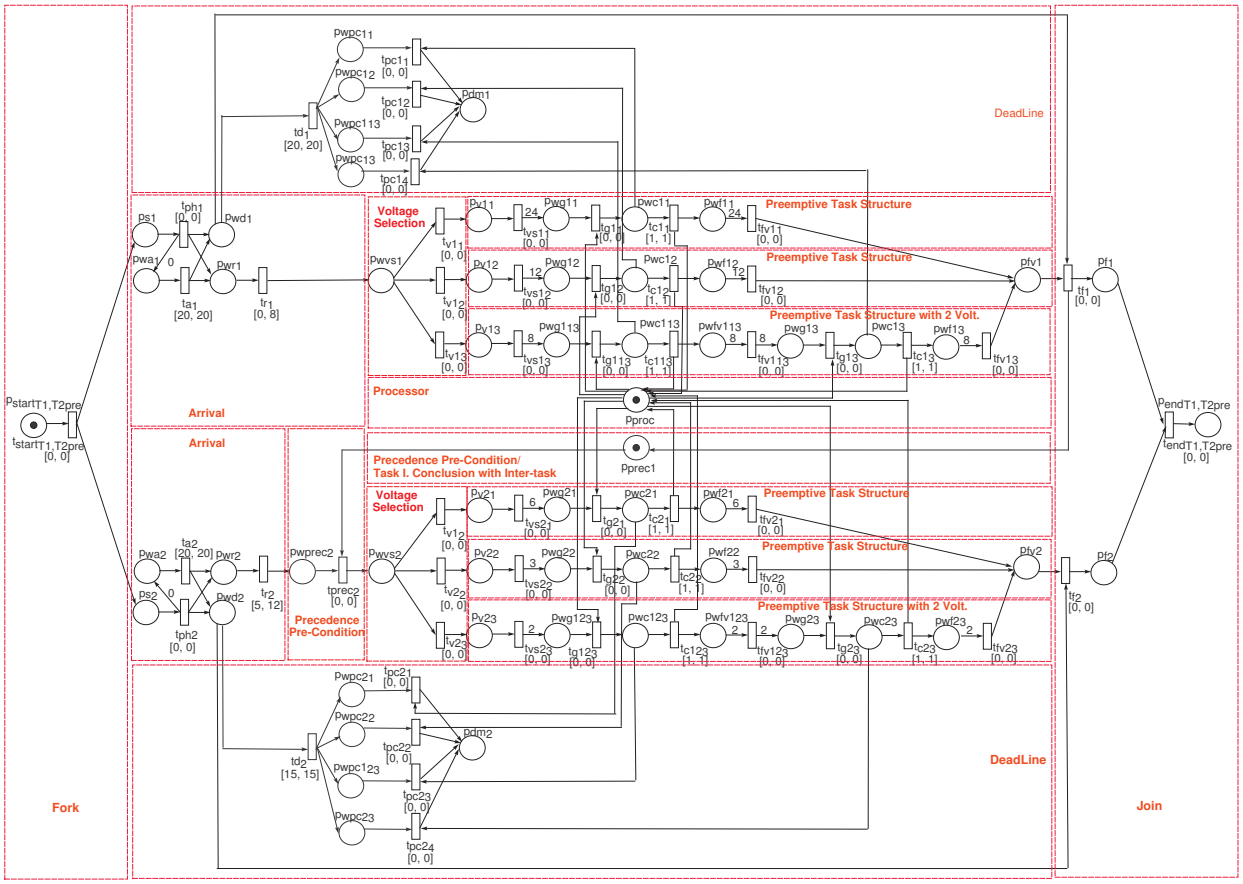


**Figure 7.17** Final model representing the precedence relation

$$\rho_{cinter_1}(p) = \begin{cases} p_{excl_1}, & if\ (p = p_{rel_1}) \\ p, & else \end{cases}$$

$$\rho_{a_2}(p) = \begin{cases} p_{wexcl_2}, & if\ (p = p_{wvs_2}) \\ p, & else \end{cases}$$

$$\rho_{cinter_2}(p) = \begin{cases} p_{excl_1}, & if\ (p = p_{rel_1}) \\ p, & else \end{cases}$$

The composition steps are presented as follows:

1. $\mathcal{N}'_{a_1} = \mathcal{N}_{a_1}/\rho_{a_1}(p)$;
2. $\mathcal{N}'_{cinter_1} = \mathcal{N}_{cinter_1}/\rho_{cinter_1}(p)$;
3. $\mathcal{N}'_{a_2} = \mathcal{N}_{a_2}/\rho_{a_2}(p)$;
4. $\mathcal{N}'_{cinter_2} = \mathcal{N}_{cinter_2}/\rho_{cinter_2}(p)$;
5. Considering the instantiation of fork ($\mathcal{N}_f$) and join blocks ($\mathcal{N}_j$) with 2 tasks, $\mathcal{N}_{T_1,T_{2exc}} = \mathcal{N}_f \sqcup \mathcal{N}_j \sqcup \mathcal{N}'_{a_1} \sqcup \mathcal{N}'_{cinter_1} \sqcup \mathcal{N}_{(v_1 \sqcup p_{1_x} \sqcup d_1)} \sqcup \mathcal{N}'_{a_2} \sqcup \mathcal{N}'_{cinter_2} \sqcup \mathcal{N}_{(v_2 \sqcup p_{2_x} \sqcup d_2)}$.

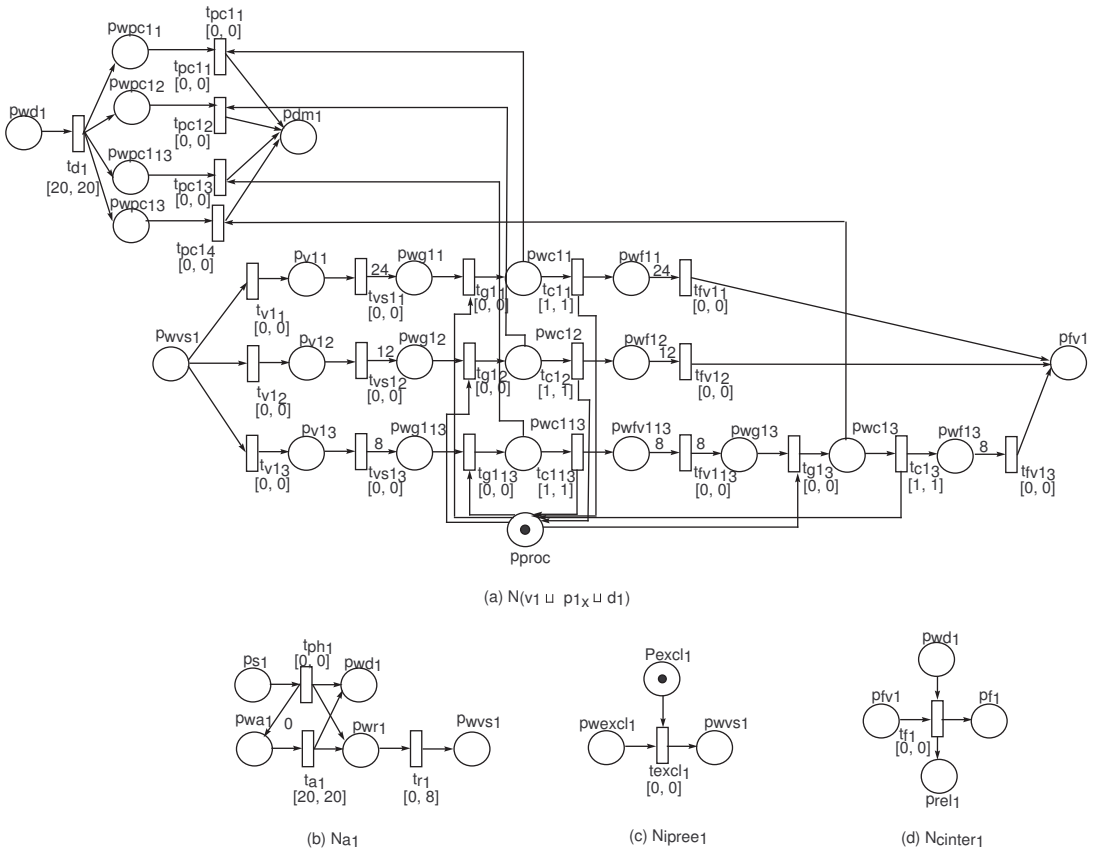


**Figure 7.18** Blocks for modeling Task $\tau_1$ with Exclusion Relation

Figure 7.20 shows the TPNE model for both tasks $\tau_1$ and $\tau_2$, in which the execution of one excludes the possibility of concurrent execution of the other. This model considers that both tasks are preemptive. When considering a task that is preceded by another task as well as has an exclusion relation with other one, there is a slight difference in the composition. The task's periodic arrival block is merged with the respective precedence pre-condition block, which is then merged with the exclusion pre-condition block.

## 7.4 MODELING OVERHEADS

As stated previously, runtime overheads may affect the timing and energy constraints of a hard real-time system. Therefore, overheads must be taken into account during the modeling and scheduling process in order to guarantee the predictability during system runtime. As a modeling example, consider the following task specification, which has been adopted as a standard example in this chapter: $\tau_1 = (0,\ 0,\ 240 \times 10^6,\ 20,\ 20)$ and $\tau_2 = (0,\ 5,\ 60 \times 10^6,\ 15,\ 20)$. Additionally, assume the following supply voltages (and the respective maximum CPU frequencies): $\mathit{vff} = \{(1\text{V},10\text{MHz}),(2\text{V},20\text{MHz})\}$. An unavailable level of 1.5V/15MHz is also considered, which can be "simulated" using the 2 immediate neighboring levels. For this example, the dispatcher WCET is 1 second ($o = 1$) as well as the time to only adjust the voltage/frequency level ($a_v = 1$). As in

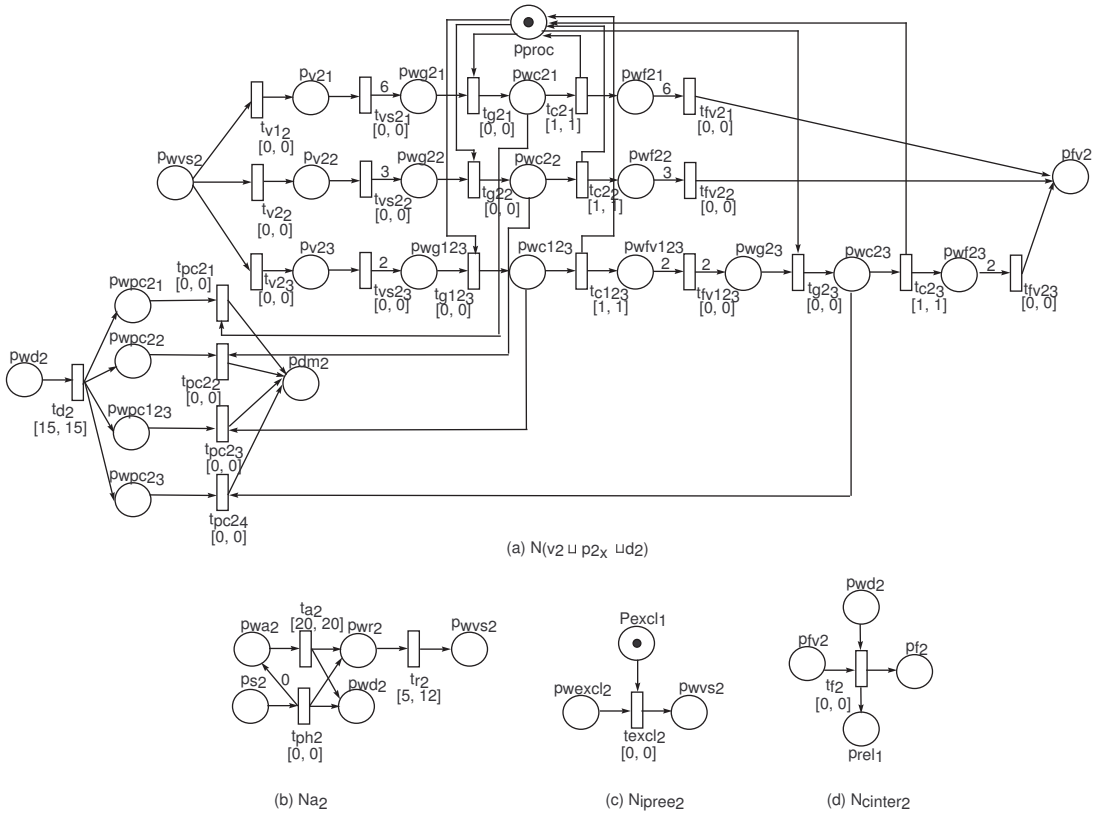


**Figure 7.19** Blocks for modeling Task $\tau_2$ with Exclusion Relation

previous sections, energy consumption values are not taken into account for the sake of simplicity.

For the purpose of demonstration as well as to reduce the dimensions of the final model, task $\tau_1$ is assumed to only execute at levels 1V/10MHz and 2V/20MHz, and $\tau_2$ at level 1.5V/15MHz. Figure 7.21 depicts the required building blocks for composing task $\tau_1$ and Figure 7.22 the basic blocks for task $\tau_2$. The composition steps are presented as follows:

1. $\mathcal{N}_{\tau_1} = \mathcal{N}_{a_1} \sqcup \mathcal{N}_{v_1} \sqcup \mathcal{N}_{d_1} \sqcup \mathcal{N}_{o1_1} \sqcup \mathcal{N}_{o1_2}$;
2. $\mathcal{N}_{\tau_2} = \mathcal{N}_{a_2} \sqcup \mathcal{N}_{v_2} \sqcup \mathcal{N}_{d_2} \sqcup \mathcal{N}_{o2v_2}$;
3. Considering the instantiation of fork ($\mathcal{N}_f$) and join blocks ($\mathcal{N}_j$) with 2 tasks, $\mathcal{N}_{T_1,T_{2o}} = \mathcal{N}_f \sqcup \mathcal{N}_j \sqcup \mathcal{N}_{\tau_1} \sqcup \mathcal{N}_{\tau_2}$.

Figure 7.23 shows the TPNE model for tasks $\tau_1$ and $\tau_2$, such that overheads are explicitly modeled.

## 7.5 ANALYSIS AND VERIFICATION OF PROPERTIES

An important goal of the proposed modeling approach is to allow the analysis and verification of properties. Depending on the context, these terms may have distinct meanings,

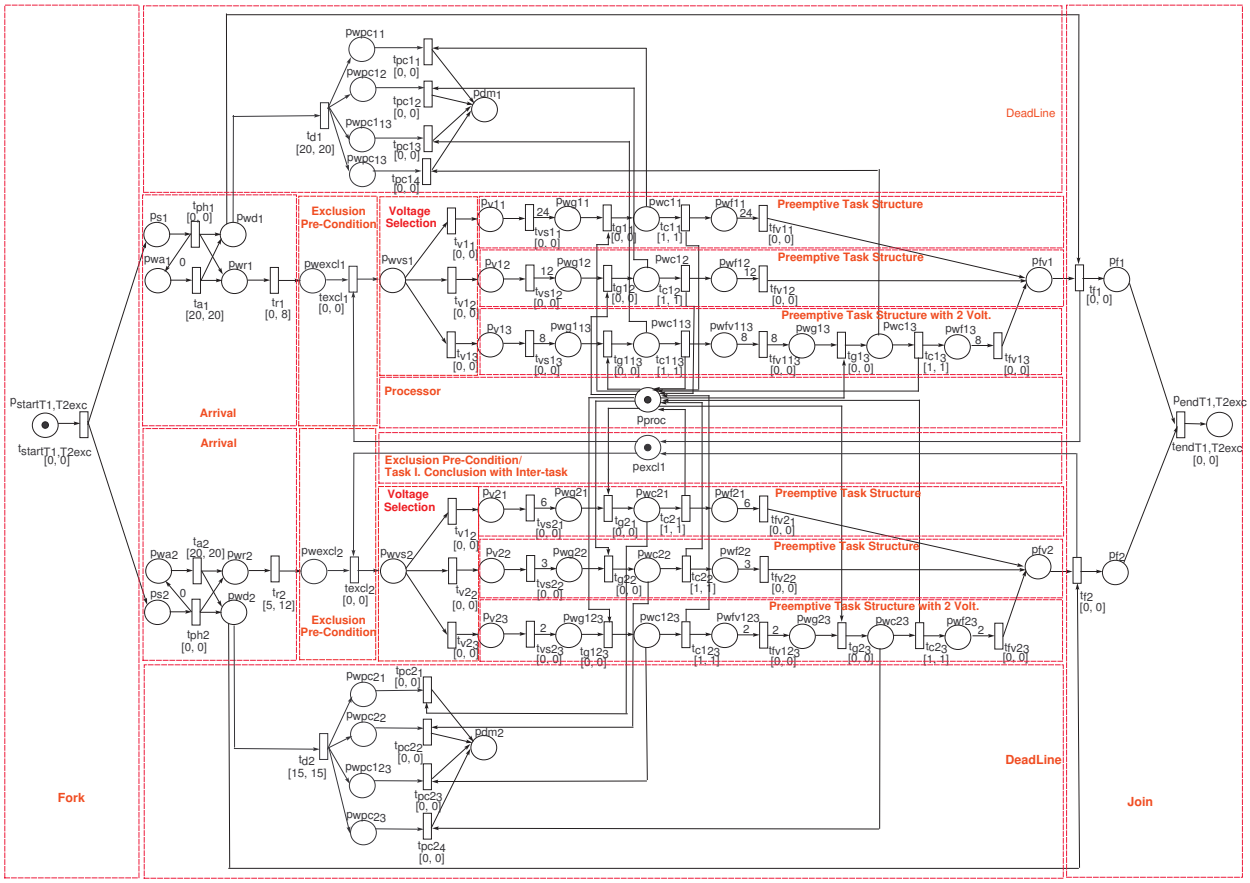


**Figure 7.20** Final model representing the Exclusion Relation between Tasks $\tau_1$ and $\tau_2$

but this work adopts the definition presented in [114]:

♢ "Verification is the act of proving or checking that a formal system has a formally stated property . In the strictest interpretation of the word, the goal is just to find rigorous evidence to the claim that the system is correct in the sense of having the property.";

♢ "Analysis means finding answers to formal questions about the behaviour of a system".

In some sense, the difference between both terms is a matter of point of view. Nevertheless, as described in [114], in analysis, most of the thinking is performed *after* running a tool, which may provide some properties associated with a model as a basis for reasoning about the system. Whereas, in verification, most of the thinking is performed *before* executing the tool, for instance, for the construction of a formulae in order to perform some kind of model-checking. Taking into account these definitions, the analysis and verification of properties is presented as follows.

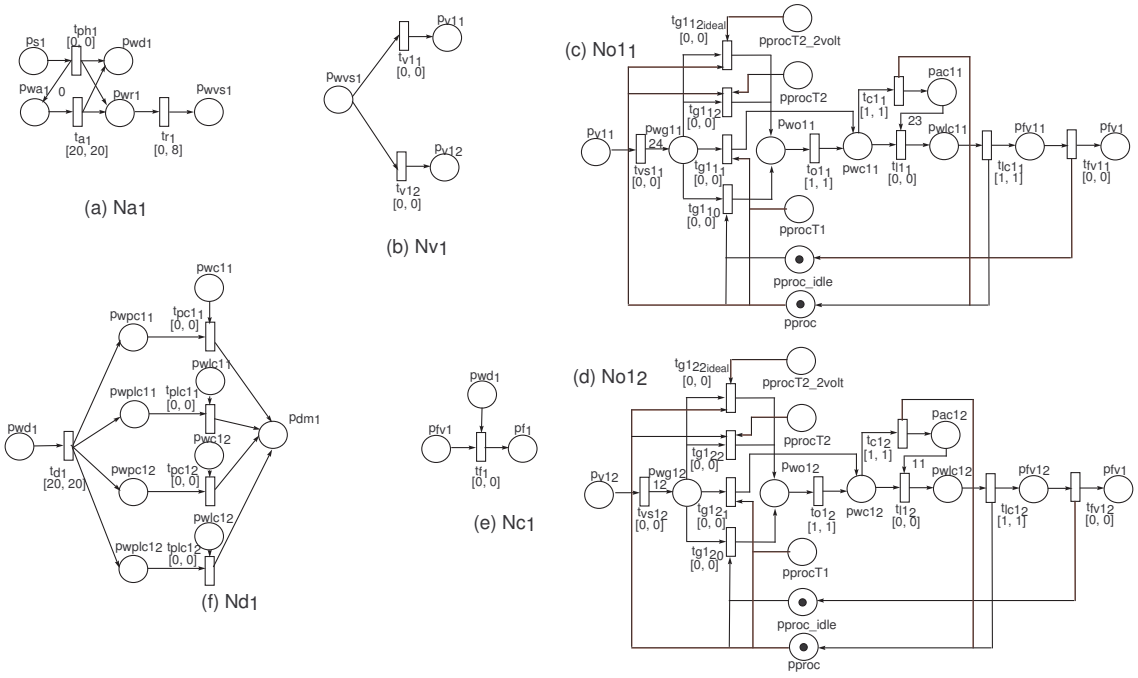


**Figure 7.21** Building blocks for composing task $\tau_1$ considering overheads

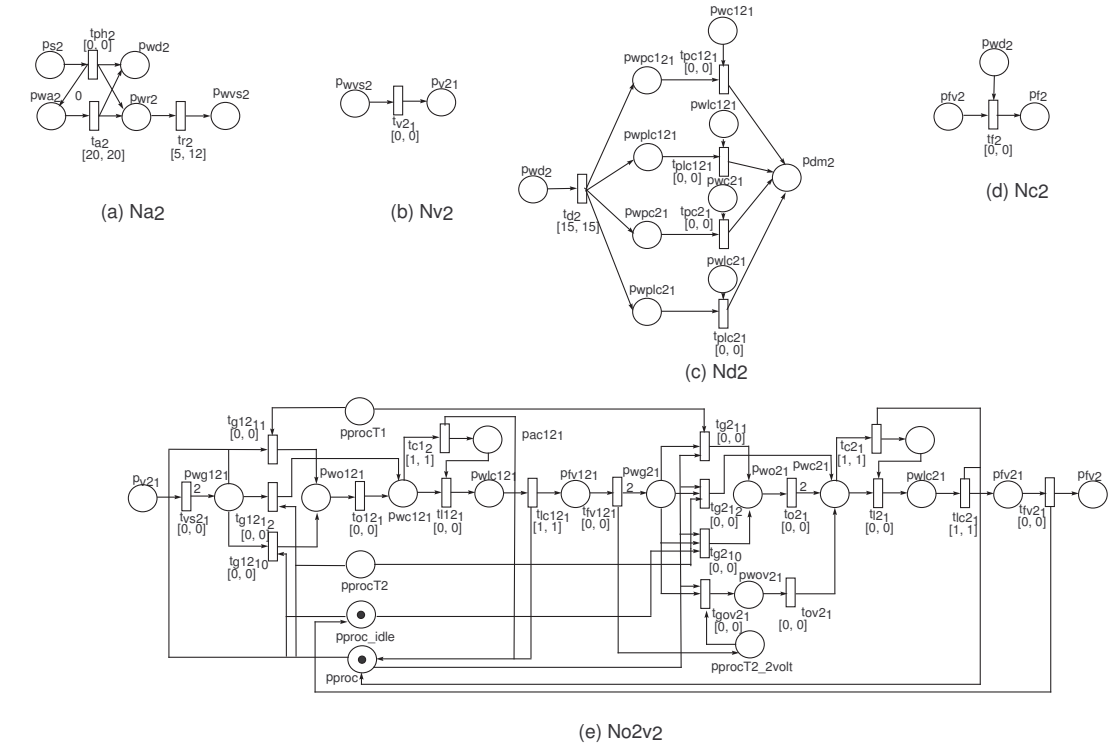


**Figure 7.22** Building blocks for composing task $\tau_2$ considering overheads

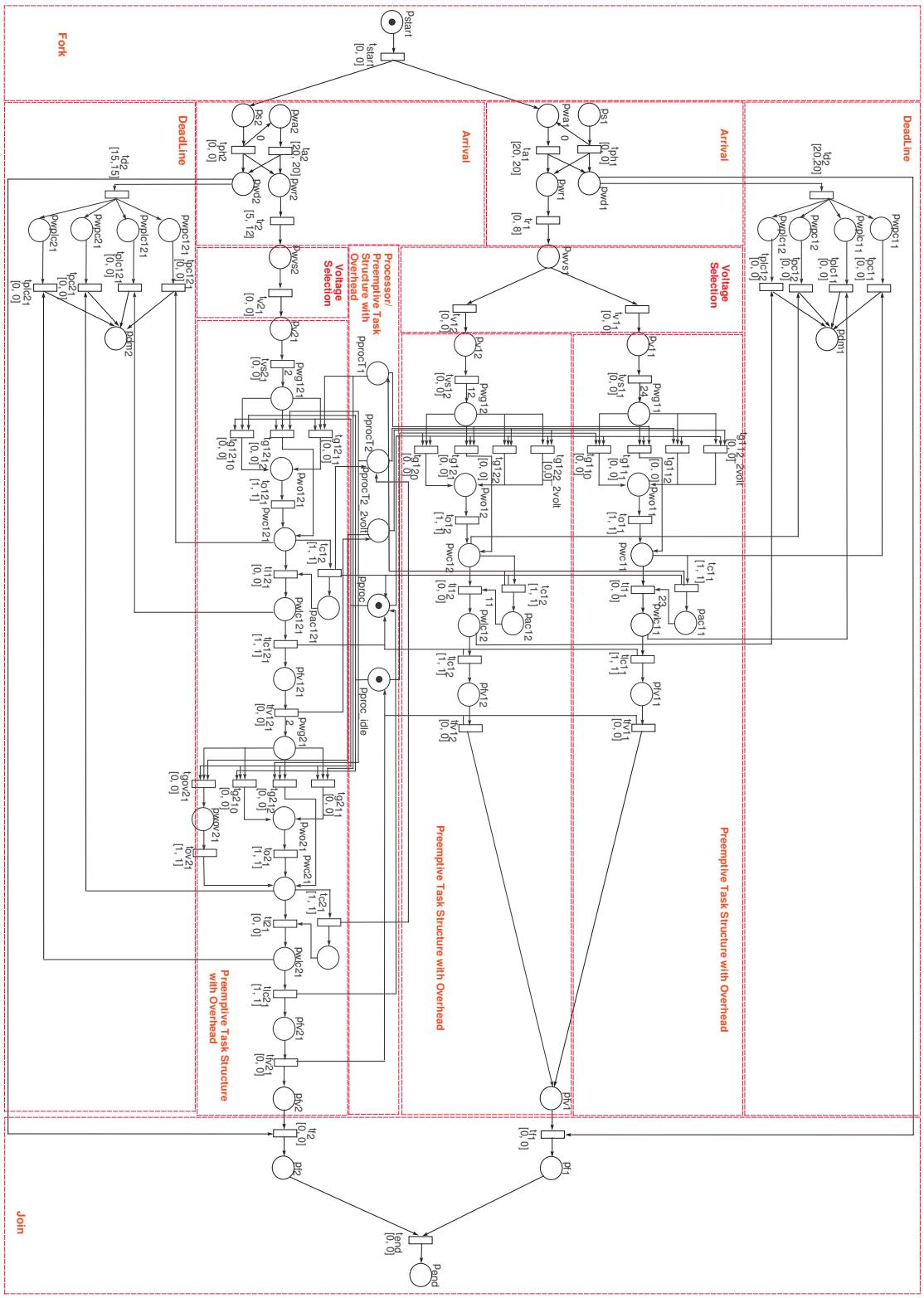


**Figure 7.23** Final model representing tasks $\tau_1$ and $\tau_2$ with overheads

### 7.5.1 Analysis

The Petri nets generated using the proposed modeling approach have some quantitative and qualitative properties, which are of the most importance when design hard real-time systems in the context of the proposed software synthesis method. Below, the properties are listed and reasoned:

♢ *Conservativeness and Boundness.* Since conservativeness is a particular case of boundness, attention is devoted to the latter. In short, boundness indicates that the state space size of a Petri net model is finite. This property is essential in the scheduling activity (see Chapter 8), as the pre-runtime scheduling algorithm is guaranteed to always finish, providing as result either a feasible schedule or none, in case no one exists. In Section 7.1, a proof is presented indicating that all generated models are structurally bounded. Note that structural boundness (qualitative property) is stronger than just boundness (quantitative property), since the former assures that the state space size is finite for any initial marking;

♢ *Deadlock.* Another property intrinsic to the models is the absence of *liveness* (see Section 3.5). In other words, the generated models contain deadlock states, which do not allow any transition to be fired. Indeed, deadlock states are important and necessary, since they represent the desired final marking (indication that a feasible schedule has been found) or a deadline missing. Obviously, the latter is avoided by the scheduling algorithm. Besides, deadlocks may occur due to bad specification of intertask relations. For instance, a cyclic dependence on resources can lead to a deadlock state, such that the execution of any task is no longer possible. In this situation, the algorithm backtracks and tries to reach other states. Nevertheless, it is possible not to find any feasible schedule due to the problems on the specification provided by the designer.

### 7.5.2 Verification

This section provides the verification of some important properties in the generated models using model checking. To assist this activity, INA tool [99] has been adopted, since it provides an infra-structure for performing model checking in time Petri nets using CTL (Computation Tree Logic) [115]. As the reachability graphs of the generated models do contain dead states (e.g., end vertexes), a loop is inserted in these states (e.g., an edge to itself), such that the evaluation of formulas about infinite paths is allowed.

For information about model checking and CTL, the interested reader should refer to [115] for a gentle introduction or may read Appendix B for an overview.

#### Processor Utilization

One important verification to be performed is to assure that only one task utilizes the processor at a time. However, before showing the CTL formula for this verification, it is important to first verify if a task running on the CPU is being executed at one and

only one voltage/frequency level. Obviously, there is no sense for a task running at two or more voltage/frequency levels simultaneously on a hardware platform with a single processor. The formula for this verification is presented as follows, taking into account task $\tau_1$ from the model depicted in Figure 7.14:

$$AG((p_{wc1_1} \wedge p_{wc1_2} \wedge (p_{wc1_3} \vee p_{wc1{1}_3})) \vee (\neg p_{wc1_1} \wedge p_{wc1_2} \wedge (p_{wc1_3} \vee p_{wc1{1}_3})) \vee (p_{wc1_1} \wedge \neg p_{wc1_2} \wedge (p_{wc1_3} \vee p_{wc1{1}_3})) \vee (p_{wc1_1} \wedge p_{wc1_2} \wedge \neg(p_{wc1_3} \vee p_{wc1{1}_3})))$$

Previous formula expresses that for all paths, the specified property is satisfied in all states. More specifically, the property indicates that there are not two or more *waiting for computation* places marked simultaneously for task $\tau_1$. For the state space of the the model depicted in Figure 7.14, INA returned **false** considering the presented formula. Furthermore, this formula can be extended to deal with more voltage/frequency levels as well as to deal with other tasks. For instance, considering task $\tau_2$, the equivalent formula is:

$$AG((p_{wc2_1} \wedge p_{wc2_2} \wedge (p_{wc2_3} \vee p_{wc1{2}_3})) \vee (\neg p_{wc2_1} \wedge p_{wc2_2} \wedge (p_{wc2_3} \vee p_{wc1{2}_3})) \vee (p_{wc2_1} \wedge \neg p_{wc2_2} \wedge (p_{wc2_3} \vee p_{wc1{2}_3})) \vee (p_{wc2_1} \wedge p_{wc2_2} \wedge \neg(p_{wc2_3} \vee p_{wc1{2}_3})))$$

Regarding the processor utilization, the formula below verifies the possibility of two tasks using the CPU at the same time for the model depicted in Figure 7.14:

$$AG((p_{wc1_1} \vee p_{wc1_2} \vee p_{wc1_3} \vee p_{wc1{1}_3}) \wedge (p_{wc2_1} \vee p_{wc2_2} \vee p_{wc2_3} \vee p_{wc1{2}_3}))$$

In other words, this formula verifies the absence of states with two *waiting for computation* places marked for two different tasks. INA tool returned **false** for the previous formula confirming the impossibility of two tasks using the processor simultaneously. Similarly, this formula can be extended to consider Petri net models with more tasks. For the Petri net model representing overheads (Figure 7.23), the following formula can also verify the problem in question:

$$AG(p_{proc_{T_1}} \wedge (p_{proc_{T_2}} \vee p_{proc_{T_2}\_2volt}))$$

Places $p_{proc_{T_1}}$, $p_{proc_{T_2}}$ and $p_{proc_{T_2}\_2volt}$ represent, respectively, task $\tau_1$ and $\tau_2$ in execution as well as task $\tau_2$ executing on a second voltage/frequency level. Since it is not possible for both tasks execute at the same moment on the CPU, INA also returned **false** for previous formula and the respective state space.

### Precedence Relation

As presented, a precedence relation $\tau_i$ *PRECEDES* $\tau_j$ states that task $\tau_j$ must await task $\tau_i$ to finish before executing. To verify whether this relation is met, *waiting for processor granting* places of both tasks cannot be marked at the same moment, since the simultaneous marking indicates that both tasks can execute without respecting this intertask relation. For the model depicted in Figure 7.17, the formula for this verification is:

$$AG((p_{wg_{1_1}} \vee p_{wg_{1_2}} \vee p_{wg_{1_3}} \vee p_{wg1_{1_3}}) \wedge (p_{wg_{2_1}} \vee p_{wg_{2_2}} \vee p_{wg_{2_3}} \vee p_{wg1_{2_3}}))$$

Taking into account this formula and the respective state space, INA returned **false**, confirming that the proposed modeling approach satisfies the precedence relation.

### Exclusion Relation

An intertask relation $\tau_i$ *EXCLUDES* $\tau_j$ states that no execution of $\tau_j$ can start while task $\tau_i$ is executing (and vice-versa). Likewise, to verify whether this relation is met, *waiting for processor granting* places of both tasks cannot be marked at the same moment, since the simultaneous marking indicates that both tasks can preempt each other. For the model depicted in Figure 7.20, the formula for this verification is:

$$AG((p_{wg_{1_1}} \vee p_{wg_{1_2}} \vee p_{wg_{1_3}} \vee p_{wg1_{1_3}}) \wedge (p_{wg_{2_1}} \vee p_{wg_{2_2}} \vee p_{wg_{2_3}} \vee p_{wg1_{2_3}}))$$

Taking into account this formula and the respective state space, INA returned **false**, confirming that the proposed modeling approach satisfies the exclusion relation.

## 7.6 SUMMARY

This chapter detailed the adopted compositions rules for combining basic building block models and it also presented some modeling examples. As demonstrated, the system models are constructed in such a way that some quantitative/qualitative properties are assured to be contained in all generated models. Besides, the generated Petri net models provide a basis for carrying out other activities in the proposed method (e.g., scheduling).

CHAPTER 8

# SCHEDULING AND CODE GENERATION

This chapter describes the scheduling and code generation activities of the proposed software synthesis method. From the Petri net model, the pre-runtime scheduling is performed and, next, the code generation is carried out. Scheduling has an prominent role in the design of hard real-time systems, since this activity must guarantee that all tasks meet the respective timing constraints, all intertask relations are satisfied, and the system energy constraint is not surpassed. Taking into account these requirements, the result of the scheduling activity is a feasible schedule, which defines the order of each task execution during system runtime, including the voltage/frequency levels associated to each task. From the feasible schedule, the code generation activity provides a predictable C-code, which includes not only the code of each task, but, also, a customized runtime support and a schedule table with the information about each task execution (e.g., the start time).

Besides, a technique for dealing with dynamic slack times is proposed in order to take advantage of new opportunities to further reduce energy consumption during system execution. This technique in conjunction with the proposed pre-runtime scheduling approach can be seen as a hybrid scheduling, since it mixes pre-runtime and runtime scheduling methods.

## 8.1 SCHEDULING

The proposed scheduling activity adopts a pre-runtime scheduling approach, in which a feasible schedule is generated at design-time. As described in Chapter 3, pre-runtime scheduling approaches provide several benefits in comparison with runtime counterparts, mainly, in the context of runtime predictability.

In this work, the pre-runtime scheduling is performed by a depth-first search algorithm that generates a TLTS (see Definition 6.5, Chapter 6) from a Petri net model. More specifically, the proposed algorithm performs a state space exploration [116], in the sense that, starting from an initial state, the algorithm recursively explores all successor states, by firing all firable transitions in each reached state. However, the state space exploration (e.g., reachability graph exploration) suffers from the *state space size explosion* [114] (e.g., the size of the reachability graph).

In general, the state space size grows exponentially in the number of represented objects. Assuming $n$ non-interacting tasks, each one with $k$ local states, the respective state space size is $O(k^n)$ [114]. The explosion is mainly related to the representation of concurrency using interleaving semantics. For instance, the analysis of $n$ concurrent actions has to tackle all $n!$ action interleaving possibilities (e.g., the order of transition firings), unless dependencies between these actions are considered. Nevertheless, not all states need to be explored in order to find a feasible schedule.

Next section presents the techniques adopted to tackle the state space size. After that, the proposed scheduling algorithm is presented, followed by an example and comments about the complexity.

### 8.1.1 Tackling State Space Size

As the state space grows exponentially in the number of tasks, the proposed scheduling activity adopts some techniques to tackle the state space size, more specifically: (i) the modeling itself; (ii) a preprocessing; (iii) partial order reduction; and (iv) the removing of undesirable states. As follows, each technique is described.

**8.1.1.1 Modeling** The proposed modeling approach (Chapter 6) explicitly represents the dependencies between actions, for instance, resource granting/releasing, precedence and exclusion relations between tasks. Therefore, the modeling itself may help in minimizing the state space size, since the amount of concurrent actions is reduced providing less interleaving possibilities.

**8.1.1.2 Preprocessing** The proposed modeling approach represents all voltage/frequency levels that a task may adopt to its execution. These levels are related to the DVS technology (Chapter 3), which provides a trade-off between energy consumption and performance. During schedule generation, the proposed algorithm selects a task for execution as well as one voltage/frequency level, which dictates the task execution time and energy consumption. One simple criteria for selecting a voltage/frequency level is the adoption of a *greedy* approach, in the sense that the minimum level available is chosen. Although this criteria may considerably reduce the energy consumption of the current task, it may affect the next task deadline, making the algorithm to perform several backtracks and, consequently, exploring more states. Conversely, selecting the maximum level available improves the current task execution time and minimally affects the next task. However, the energy saving considerably diminishes, as the consumption is proportional to the supply voltage squared. Another alternative is the selection of an unique constant speed $S$ [30], which maximizes the CPU utilization using the following equation: $\sum_{i=1}^{|\mathcal{T}|} \frac{c_i}{S.p_i} = 1$. For a better understanding, $\mathcal{T}$ is the set of system tasks, $c_i$ is the worst-case execution cycles of a task $\tau_i$, and $p_i$ is the period. Nevertheless, such equation generates interesting results when deadlines are equal to the periods, the release times are 0, and intertask relations are not taken into account.

In this work, before applying the proposed scheduling and modeling activities, the system specification is preprocessed considering an extension of Yao's algorithm [25] (LPEDF - Low-Power Earliest Deadline First), in which a set of discrete voltage/frequency levels is taken into account [26]. Yao's algorithm is adopted as a basis for resembling CPU's unavailable voltage/frequency levels by the nearest accessible levels as well as a guide for selecting an initial voltage/frequency for each task instance during scheduling generation. This algorithm finds an optimal DVS schedule for a set of real-time tasks using EDF scheduling policy, without constraining the release and deadline times of each task. However, LPEDF does not take into account intertask relations and overheads, but the latter

```
1 LPEDF(𝒥)
2 {
3   ℛ={};
4   i=0;
5   while (𝒥 is not empty) {
6     Find the critical interval I_i = [R_i, D_i] with the highest intensity
      (CPU speed), f(I_i) = (∑ c_n)/(D_i − R_i), in which the sum is taken over all
      task instances j_n in J_i with [r_n, d_n] ⊆ [R_i, D_i];
7     ℛ=ℛ ∪ {(I_i, J_i, f(I_i))};
8     𝒥 = 𝒥 - J_i;
9     for all j_k in 𝒥 {
10      if (d_k ∈ [R_i, D_i]) {
11        d_k = R_i;
12      } else if (d_k ≥ D_i) {
13        d_k = d_k - (D_i - R_i);
14      }
15      Adjust release times r_k similarly;
16    }//end - for
17    i++;
18  }//end - while
19  return ℛ;
20 }
```

**Figure 8.1** LPEDF - low-power earliest-deadline first

is considered after carrying out this first phase. Figure 8.1 depicts the Yao's algorithm. For a better understanding, the following paragraph presents an explanation.

Assume a set of task instances $\mathcal{J}$ requires to be executed in a given time interval. A critical interval $I_i$ for $\mathcal{J}$ is an interval in which a subset of task instances must be scheduled at maximum constant speed (or frequency) in any optimal schedule for $\mathcal{J}$. The algorithm schedules those task instances in that speed (lines 6-7), and constructs a subproblem for the remaining instances and solves it recursively (lines 8-17). In the code, $c_i$, $r_i$ and $d_i$ are the worst-case execution cycles, release time and deadline of a task instance, respectively. $R_i$ is equal to the smallest release time of a task instance contained in the interval, and $D_i$ is equal to the largest deadline. The algorithm result is a set of intervals ($\mathcal{R}$), in which each interval $I_i$ has a set of task instances and an associated voltage/frequency level for executing those tasks. Note that all calculated speeds (frequencies) not available on the CPU are replaced by the neighboring voltage/frequency levels using the technique described in [12] (see also Section 6.3.6). Obviously, previous technique does not assume the existence of a calculated speed that surpasses the maximum frequency available on the processor.

After executing LPEDF algorithm, the overheads related to dispatcher callings, voltage/frequency switching, and preemptions are included in the interval $I_i$. As a consequence, each task instance contained in the interval $I_i$ has a new voltage/frequency level, which is higher than the original one, in order to meet timing constraints. The new level is obtained by associating a variable $o_i$ to each interval $I_i$ (which represents the amount of overheads in $I_i$) and redefining the intensity function $f$ to consider $o_i$: $f'(I_i) = \frac{\sum_{jn \in \mathcal{J}_i} c_n}{D_i - R_i - o_i}$. Depending on the result, the new level needs to be replaced by the neighboring volt-

age/frequency levels available on the CPU using the technique described in [12], if it does not exist on the processor. As the final result, each task $\tau_k$ has a set of ideal voltages and frequencies available for the respective instances during the schedule generation (in addition to those available on the cpu), and they are represented by $\mathcal{V}_{ideal_k}$ and $\mathcal{F}_{ideal_k}$, respectively. The mapping is performed by function $vff_{ideal_k} : \mathcal{V}_{ideal_k} \rightarrow \mathcal{F}_{ideal_k}$. Next, the TPNE model is generated also taking into account these ideal levels, which are modeled using the building blocks described in Chapter 6.

Depending on the task specification, previous technique for including overheads in the intervals may lead to higher speeds not available on the CPU [27]. Instead of assuming the unavailability of a feasible schedule previously, this work leaves to the pre-runtime scheduling algorithm the possibility to find other execution order (as well as voltage/frequency level) for task instances in intervals violating CPU maximum frequency.

As an example of Yao's algorithm execution, assume the following task set: $\mathcal{T} = \{\tau_1 = (0, 0, 240 \times 10^6, 20, 20), \tau_2 = (0, 5, 60 \times 10^6, 15, 20)\}$. As presented in Chapter 5, each task is represented by a tuple $\tau_i = (ph_i, r_i, c_i, d_i, p_i)$, in which $ph_i$ is the initial phase; $r_i$ is the release time; $c_i$ is the worst-case execution cycles (WCEC) required for executing task $\tau_i$; $d_i$ is the deadline; and $p_i$ is the period. For this specification, the LCM is equal to 20, which points out the existence of 2 task instances ($\mathcal{S}(\tau_1) = 1$ and $\mathcal{S}(\tau_2) = 1$). In addition, assume the following supply voltages and the respective maximum frequencies $vff = \{$(1V,10MHz),(2V,20MHz)$\}$. In the first (and only) iteration of the algorithm execution, two intervals are obtained: $I'_1 = [0, 20]$ with $J'_1 = \{\tau^1_1, \tau^1_2\}$ and $I''_1 = [5, 15]$ with $J''_1 = \{\tau^1_2\}$. As $I'_1$ has the highest intensity ($f(I'_1) = 300 \times 10^6/20$ =15MHz), $I'_1$ is selected as the critical interval ($I_1 = I'_1$). Since there are no more task instances to be scheduled, the algorithm stops. Although the CPU specification does not contain any voltage/frequency level with 15MHz, this level can be "simulated" using the two neighboring levels available on the processor. If overheads are taken into account and assuming $o_1 = 1$ (one second), the final level is ($f'(I_1) = 300 \times 10^6/(20 - 1) \approx$15.789MHz).

In a suitable implementation, the time complexity of Yao's algorithm [25] is $O(Nlog^2N)$ - in which $N$ is the number of tasks' instances - whereas a scheduling problem with inter-task relations is NP-hard [117]. Experiments have shown that the preprocessing greatly improves scheduling generation processing time as well as the state space size by avoiding inappropriate voltage/frequency levels. For a better understanding, when choosing a voltage/frequency level for a task instance, the proposed scheduling algorithm first selects a level taking into account the result obtained by the preprocessing phase, and prunes transitions that represent lower levels. If the selected voltage/frequency always leads to an undesirable state, such as deadline missing, the transition that represents this level is disregarded and the immediate higher voltage/frequency level available on CPU, which leads to a computation time less than or equal to the deadline, is selected.

**8.1.1.3 Partial-Order Reduction** One feasible approach to tackle the state space size is minimizing the impact of interleaving on the state space exploration by exploiting the independence of actions. More specifically, if actions can be executed in any order, such that the model always reaches the same state, these actions are denominated *independent*. In other words, it does not matter in which order these actions

are executed [116, 118]. Previous characteristic is denominated *diamond property* and techniques, that exploit the independence of actions, are usually named *partial-order reduction methods*.

Partial-order reduction methods throw some interleaving possibilities away by executing just a subset of the firable actions, called *persistent set*, in each reached state. The pruned states (resulted from the reduction of interleaving possibilities) are assumed to not affect the property(ies) of interest due to the diamond property.

The partial-order reduction adopted in this work defines some classes of transitions, in which the highest-choice priority levels are given to classes that represent independent actions and the lowest ones to dependent actions. Independent actions are related to transitions that do not disable other actions, such as arrival, precedence,and dependent actions are related to processor granting and exclusion relations, for instance. When changing from one state to another, the highest choice-priority class of transitions is analyzed (resulting in the persistent set) whereas the other classes are pruned. As a consequence, this technique decreases the state space size as well as allows checking unavailability of feasible schedules more quickly.

Table 8.1 depicts the choice-priority levels of each transition class, where the lowest number indicates the highest priority. In this table, transition $tr_i$ (release) does not have a specific choice-priority, since it is a special type of transition that, once it is firable, may fire in any transition class. Considering the class that transition $tr_i$ is firable, $tr_i$ is selected as the first option. Such mechanism allows tasks to be released for execution at any moment according to the timing interval provided by $tr_i$.

**Table 8.1** Choice-priority levels

| Choice-Priority | Type | Transition |
|---|---|---|
| - | Release | $tr_i$ |
| 1 | Final | $tf_i, tfv_{i_n}, tfv1_{i_n}$ |
| 2 | Arrival | $ta_i, tph_i$ |
| 3 | Voltage | $tv_{i_n}$ |
| 4 | Others | $tvs_{i_n}, tl_{i_n}$ |
| 5 | Precedence | $tprec_{ij}$ |
| 6 | Overhead | $to_{i_n}, to1_{i_n}$ |
| 7 | Computation | $tc_{i_n}, tc1_{i_n}, tlc_{i_n}$ |
| 8 | Exclusion | $texc_{ij}$ |
| 9 | ProcessorGranting | $tg_{i_n}, tg1_{i_n}$ |

**8.1.1.4 Removing Undesirable States** Chapter 6 presents a building block able to find out deadline missing, which is an undesirable reachable state. Since the proposed method deals with hard real-time systems with energy constraints, only schedules, which do not reach any of these undesirable states, are of interest. During the TLTS generation, transitions leading to undesirable states are discarded by the scheduling algorithm, for instance, deadline-checking transitions.

```
1 scheduling-synthesis(S,M^F,TPNE, e_max)
2 {
3   if (S.M = M^F) return TRUE;
4   tag(S);
5   PT = pruning(firable(S));
6   if (|PT| = 0) return FALSE;
7   for each (⟨t,θ⟩ ∈ PT) {
8     S'= fire(S, t, θ, TPNE);
9     if (untagged(S') ∧
10      scheduling-synthesis (S',M^F,TPNE, e_max)){
11      add-in-trans-system (S,S',t,θ);
12      return TRUE;
13    }
14  }
15  return FALSE;
16 }
```

**Figure 8.2** Scheduling algorithm

### 8.1.2 Pre-Runtime Scheduling Algorithm

The proposed algorithm (Figure 8.2) is a depth-first search method for TLTS generation that aims achieving the *stop criterion* (final marking $M^F$ reachability - see Section 6.3.7) without generating the whole state space. Whenever the stop criterion is achieved, a feasible schedule is generated. This algorithm is an extension of Barreto's approach [16] taking into account DVS as well as new techniques for dealing with the state space size explosion. An explanation is presented as follows.

Considering that (i) the Petri net model is surely bounded, (ii) the timing constraints are enclosed by finite intervals, and (iii) the algorithm strictly implements the transition firing rule of the time Petri nets, the TLTS is finite and thus the proposed algorithm always finishes, providing as result either a feasible schedule or none.

In this algorithm, `S` is the current state, $\mathtt{M}^F$ is the desired final marking, `TPNE` is the extended time Petri net model, and $\mathtt{e}_{max}$ is the system energy constraint (Section 5.2). The only way the algorithm returns TRUE is when it reaches the desired final marking ($M^F$, *stop criterion*), implying that a feasible schedule has been found (line 3). The *tagging scheme* (lines 4 and 9) ensures that no state is visited more than once. The state space generation algorithm incorporates the state space pruning (line 5), in which, for the set of firable transitions (function `firable`), function `pruning` is executed according to the rules described in Section 8.1.1 (including the preprocessing phase). `PT` is a set of ordered pairs $\langle t, \theta \rangle$ representing, for each firable transition (post-pruning), all possible firing times in the firing domain. The function `fire` (line 8) returns a new generated state ($S'$) due to the transition $t$ firing at time $\theta$. The feasible schedule is represented by a timed labeled transition system that is generated by the function `add-in-trans-system` (line 11). Only when the system does not have a feasible schedule, the whole state space is analyzed.

**8.1.2.1 Tagging Scheme** As explained previously, the tagging scheme is adopted to avoid a state to be visited more than once. This process can be time consuming, since

each new reached state needs to be verified against all previously visited states. In order to tackle such a problem, this work generated several experiments' state space, which led to the following solution. When a new state is reached, only visited states that occurred at the same global time instant (assuming a global clock) need to be verified. Indeed, besides the marking, a state also takes into account the time $\theta$ when the transition $t$, in previous state, was fired. The proposed solution adopts a hash table for storing the visited states, where the key for a state $s_i$ is the global time when such a state was reached: $KEY(s_i) = \sum_{j=0}^{i-1} \theta_j$. Because each hash table entry may have several stored states, each entry points to a binary tree in order to optimize even more the verification. Experiments have demonstrated that this solution improves in 50 % the algorithm's execution time.

### 8.1.3 Algorithm Execution Example

To demonstrate the execution of the scheduling algorithm, consider the following task specification, which has been adopted as a standard example in thesis: $\mathcal{T} = \{\tau_1 = (0,\ 0,\ 240 \times 10^6,\ 20,\ 20),\ \tau_2 = (0,\ 5,\ 60 \times 10^6,\ 15,\ 20)\}$. The reader should remind that each task is represented by a tuple $\tau_i = (ph_i, r_i, c_i, d_i, p_i)$, in which $ph_i$ is the initial phase; $r_i$ is the release time; $c_i$ is the worst-case execution cycles (WCEC) required for executing task $\tau_i$; $d_i$ is the deadline; and $p_i$ is the period. For this specification, the time unit is equal to one second and the LCM is equal to 20. In addition, assume the following supply voltages and the respective maximum frequencies: $\mathit{vff} = \{(1\text{V},10\text{MHz}),(2\text{V},20\text{MHz})\}$. Furthermore, for the sake of this example, the energy consumption is 1nJ/cycle at 1V/10MHz, 2nJ/cycle at 2V/20MHz, and overheads are not taken into account.

Firstly, the preprocessing is performed, which assigns 1.5V/15MHz as the initial voltage/frequency level for each task instance (Section 8.1.1.2). Next, the TPNE model is generated (Figure 8.3) considering the voltage/frequency levels available on the CPU (1V/10MHz,2V/20MHz) and the unavailable level obtained with the preprocessing (1.5V/15MHz). After obtaining the TPNE model, the scheduling algorithm is executed (see Table 8.2). In this table, *visited* represents the amount of visited states up to that point; *state* shows the reached state identification; $ET$ is the set of enabled transitions in that state; $C$ is the clock vector. For the sake of readability, the symbol # is not presented in that vector. $FT$ is the set of firable transitions; $PT$ is the post-pruned set of firable transitions; *selected* represents the fired transition in that state and the respective elapsed time; and *energy* depicts the accumulated energy consumption (in joules) from the initial state up to that point.

Initially, at state 0, only transition $t_{start_{T_1,T_2}}$ is enabled, since the initial marking is $m(p_{start_{T_1,T_2}}) = m(p_{proc}) = 1$. After the firing of $t_{start_{T_1,T_2}}$, both tasks $T_1$ and $T_2$ become eligible for execution, but task $T_1$ is released first (state 2) due to its timing constraints ($r_i = 0$). The reader should note the partial-order reduction at state 3, which has 4 firable transitions. Only transition $tph2$ is not pruned, since it has the highest choice-priority level (see Table 8.1). Similar circumstance occurs in further states (e.g., state 17 and 28). In relation to the preprocessing, state 4 depicts an interesting situation. Despite the availability of 3 voltage/frequency levels for executing $T_1$, transition $tv_{1_1}$ is pruned, since the respective voltage/frequency level of 1V/10MHz is below the level obtained by

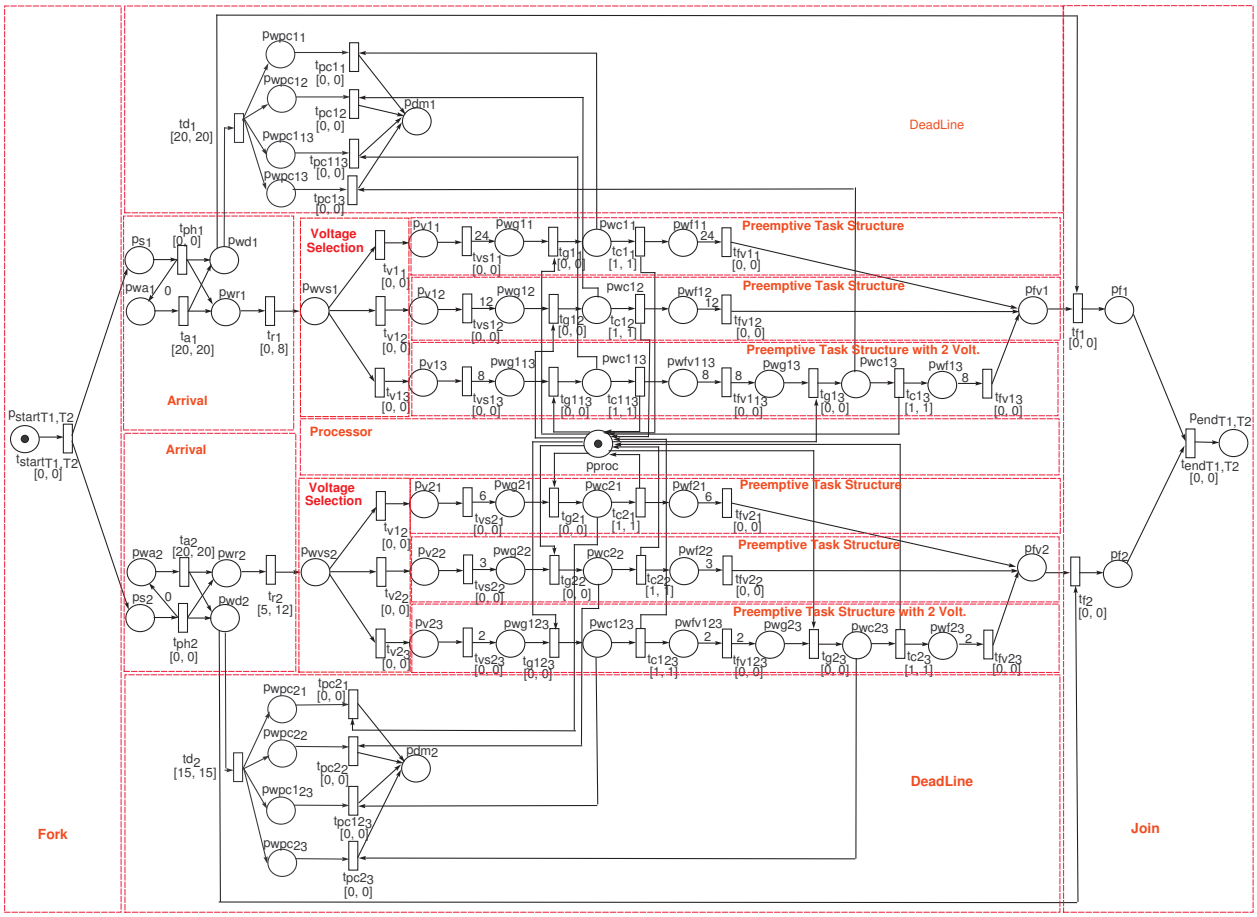


**Figure 8.3** Generated model for tasks $\tau_1$ and $\tau_2$

Yao's algorithm (1.5V/15MHz - transition $tv_{1_3}$). Thus, transition $tv_{1_3}$ is selected and $T_1$ acquires the processor granting in order to start its execution (state 6).

Observe that, after each time unit related to the computation time of task $T_1$ (e.g., state 7), the processor is released, and task $T_1$ needs to take the processor granting again (e.g., state 8) for continuing its computation. Such approach is required to allow other concurrent tasks to be assigned to the CPU, since the preemptive scheduling method is adopted (Section 6.3.5). At state 16, task $T_2$ is released for execution ($r_2 = 5$), and the voltage/frequency level of 1.5V/15MHz (transition $tv_{2_3}$) is selected (state 17). As $T_2$ has an earlier deadline, task $T_1$ is preempted at state 19. Additionally, the reader should notice that the voltage/frequency level is reduced from 2V/20MHz to 1V/10MHz at state 23, because 1.5V/15MHz is "simulated" using both voltage/frequency levels. At state 29, task $T_2$ finishes its execution, and, consequently, $T_1$ returns from preemption at state 30. From state 52 to 54, transition $td1$ is fireable. Nevertheless, since such transition represents the occurrence of an undesirable state (deadline missing), the pruning is applied (Section 8.1.1): $td1 \notin PT$. Finally, after concluding all tasks' execution, a token is stored in place $p_{end_{T_1,T_2}}$, indicating that a feasible schedule has been found. In this example, no backtrack occurs, in other words, a schedule has been found in the first attempt, and the

**Table 8.2** Algorithm execution

| visited | state | ET | C | FT | PT | selected | energy |
|---|---|---|---|---|---|---|---|
| 1 | 0 | {tstart$_{T1,T2}$} | [0] | {tstart} | {tstart} | (tstart,0) | 0.00 |
| 2 | 1 | {tph1,tph2} | [0,0] | {tph1,tph2} | {tph1,tph2} | (tph1,0) | 0.00 |
| 3 | 2 | {td1,tr1,tph2} | [0,0,0] | {tr1,tph2} | {tr1,tph2} | (tr1,0) | 0.00 |
| 4 | 3 | {td1,tv11,tv12,tv13,tph2} | [0,0,0,0,0] | {tv11,tv12,tv13,tph2} | {tph2} | (tph2,0) | 0.00 |
| 5 | 4 | {td1,tv11,tv12,tv13,td2,tr2} | [0,0,0,0,0,0] | {tv11,tv12,tv13} | {tv12,tv13} | (tv13,0) | 0.00 |
| 6 | 5 | {td1,tvs13,td2,tr2} | [0,0,0,0] | {tvs13} | {tvs13} | (tvs13,0) | 0.00 |
| 7 | 6 | {td1,tg113,td2,tr2} | [0,0,0,0] | {tg113} | {tg113} | (tg113,0) | 0.00 |
| 8 | 7 | {td1,tc113,td2,tr2} | [0,0,0,0] | {tc113} | {tc113} | (tc113,1) | 0.00 |
| 9 | 8 | {td1,tg113,td2,tr2} | [1,0,1,1] | {tg113} | {tg113} | (tg113,0) | 0.04 |
| 10 | 9 | {td1,tc113,td2,tr2} | [1,0,1,1] | {tc113} | {tc113} | (tc113,1) | 0.04 |
| 11 | 10 | {td1,tg113,td2,tr2} | [2,0,2,2] | {tg113} | {tg113} | (tg113,0) | 0.08 |
| 12 | 11 | {td1,tc113,td2,tr2} | [2,0,2,2] | {tc113} | {tc113} | (tc113,1) | 0.08 |
| 12 | 12 | {td1,tg113,td2,tr2} | [3,0,3,3] | {tg113} | {tg113} | (tg113,0) | 0.12 |
| 14 | 13 | {td1,tc113,td2,tr2} | [3,0,3,3] | {tc113} | {tc113} | (tc113,1) | 0.12 |
| 15 | 14 | {td1,tg113,td2,tr2} | [4,0,4,4] | {tg113} | {tg113} | (tg113,0) | 0.16 |
| 16 | 15 | {td1,tc113,td2,tr2} | [4,0,4,4] | {tc113} | {tc113} | (tc113,1) | 0.16 |
| 17 | 16 | {td1,tg113,td2,tr2} | [5,0,5,5] | {tg113,tr2} | {tr2,tg113} | (tr2,0) | 0.20 |
| 18 | 17 | {td1,tg113,td2,tv21,tv22,tv23} | [5,0,5,0,0,0] | {tg113,tv21,tv22,tv23} | {tv22,tv23} | (tv23,0) | 0.20 |
| 19 | 18 | {td1,tg113,td2,tvs23} | [5,0,5,0] | {tg113,tvs23} | {tvs23} | (tvs23,0) | 0.20 |
| 20 | 19 | {td1,tg113,td2,tg123} | [5,0,5,0] | {tg113,tg123} | {tg123,tg113} | (tg123,0) | 0.20 |
| 21 | 20 | {td1,td2,tc123} | [5,5,0] | {tc123} | {tc123} | (tc123,1) | 0.20 |
| 22 | 21 | {td1,tg113,td2,tg123} | [6,0,6,0] | {tg113,tg123} | {tg123,tg113} | (tg123,0) | 0.24 |
| 23 | 22 | {td1,td2,tc123} | [6,6,0] | {tc123} | {tc123} | (tc123,1) | 0.24 |
| 24 | 23 | {td1,tg113,td2,tfv123} | [7,0,7,0] | {tg113,tfv123} | {tfv123} | (tfv123,0) | 0.28 |
| 25 | 24 | {td1,tg113,td2,tg23} | [7,0,7,0] | {tg113,tg23} | {tg23,tg113} | (tg23,0) | 0.28 |
| 26 | 25 | {td1,td2,tc23} | [7,7,0] | {tc23} | {tc23} | (tc23,1) | 0.28 |
| 27 | 26 | {td1,tg113,td2,tg23} | [8,0,8,0] | {tg113,tg23} | {tg23,tg113} | (tg23,0) | 0.29 |
| 28 | 27 | {td1,td2,tc23} | [8,8,0] | {tc23} | {tc23} | (tc23,1) | 0.29 |
| 29 | 28 | {td1,tg113,td2,tfv23} | [9,0,9,0] | {tg113,tfv23} | {tfv23} | (tfv23,0) | 0.30 |
| 30 | 29 | {td1,tg113,td2,tf2} | [9,0,9,0] | {tg113,tf2} | {tf2} | (tf2,0) | 0.30 |
| 31 | 30 | {td1,tg113} | [9,0] | {tg113} | {tg113} | (tg113,0) | 0.30 |
| 32 | 31 | {td1,tc113} | [9,0] | {tc113} | {tc113} | (tc113,1) | 0.30 |
| 33 | 32 | {td1,tg113} | [10,0] | {tg113} | {tg113} | (tg113,0) | 0.34 |
| 34 | 33 | {td1,tc113} | [10,0] | {tc113} | {tc113} | (tc113,1) | 0.34 |
| 35 | 34 | {td1,tg113} | [11,0] | {tg113} | {tg113} | (tg113,0) | 0.38 |
| 36 | 35 | {td1,tc113} | [11,0] | {tc113} | {tc113} | (tc113,1) | 0.38 |
| 37 | 36 | {td1,tfv113} | [12,0] | {tfv113} | {tfv113} | (tfv113,0) | 0.38 |
| 38 | 37 | {td1,tg13} | [12,0] | {tg13} | {tg13} | (tg13,0) | 0.42 |
| 39 | 38 | {td1,tc13} | [12,0] | {tc13} | {tc13} | (tc13,1) | 0.42 |
| 40 | 39 | {td1,tg13} | [13,0] | {tg13} | {tg13} | (tg13,0) | 0.43 |
| 41 | 40 | {td1,tc13} | [13,0] | {tc13} | {tc13} | (tc13,1) | 0.43 |
| 42 | 41 | {td1,tg13} | [14,0] | {tg13} | {tg13} | (tg13,0) | 0.44 |
| 43 | 42 | {td1,tc13} | [14,0] | {tc13} | {tc13} | (tc13,1) | 0.44 |
| 44 | 43 | {td1,tg13} | [15,0] | {tg13} | {tg13} | (tg13,0) | 0.45 |
| 45 | 44 | {td1,tc13} | [15,0] | {tc13} | {tc13} | (tc13,1) | 0.45 |
| 46 | 45 | {td1,tg13} | [16,0] | {tg13} | {tg13} | (tg13,0) | 0.46 |
| 47 | 46 | {td1,tc13} | [16,0] | {tc13} | {tc13} | (tc13,1) | 0.46 |
| 48 | 47 | {td1,tg13} | [17,0] | {tg13} | {tg13} | (tg13,0) | 0.47 |
| 49 | 48 | {td1,tc13} | [17,0] | {tc13} | {tc13} | (tc13,1) | 0.47 |
| 50 | 49 | {td1,tg13} | [18,0] | {tg13} | {tg13} | (tg13,0) | 0.48 |
| 51 | 50 | {td1,tc13} | [18,0] | {tc13} | {tc13} | (tc13,1) | 0.48 |
| 52 | 51 | {td1,tg13} | [19,0] | {tg13} | {tg13} | (tg13,0) | 0.49 |
| 53 | 52 | {td1,tc13} | [19,0] | {td1,tc13} | {tc13} | (tc13,1) | 0.49 |
| 54 | 53 | {td1,tfv13} | [20,0] | {td1,tfv13} | {tfv13} | (tfv13,0) | 0.50 |
| 55 | 54 | {td1,tf1} | [20,0] | {td1,tf1} | {tf1} | (tf1,0) | 0.50 |
| 56 | 55 | {tend$_{T1,T2}$} | [0] | {tend} | {tend} | (tend,0) | 0.50 |

respective TLTS (see Definition 6.5, Chapter 6) is:

$$s_0 \xrightarrow{(t_{start_{T_1,T_2}},0)} s_1 \xrightarrow{(t_{ph1},0)} s_2 \xrightarrow{(t_{r1},0)} s_3 \xrightarrow{(t_{ph2},0)} s_4 \xrightarrow{(t_{v13},0)} s_5 \xrightarrow{(t_{vs13},0)} s_6 \xrightarrow{(t_{g113},0)} s_7 \xrightarrow{(t_{c113},1)} s_8 \xrightarrow{(t_{g113},0)}$$
$$s_9 \xrightarrow{(t_{c113},1)} s_{10} \xrightarrow{(t_{g113},0)} s_{11} \xrightarrow{(t_{c113},1)} s_{12} \xrightarrow{(t_{g113},0)} s_{13} \xrightarrow{(t_{c113},1)} s_{14} \xrightarrow{(t_{g113},0)} s_{15} \xrightarrow{(t_{c113},1)} s_{16} \xrightarrow{(t_{r2},0)} s_{17} \xrightarrow{(t_{v23},0)}$$
$$s_{18} \xrightarrow{(t_{vs23},0)} s_{19} \xrightarrow{(t_{g123},0)} s_{20} \xrightarrow{(t_{c123},1)} s_{21} \xrightarrow{(t_{g123},0)} s_{22} \xrightarrow{(t_{c123},1)} s_{23} \xrightarrow{(t_{fv123},0)} s_{24} \xrightarrow{(t_{g23},0)} s_{25} \xrightarrow{(t_{c23},1)}$$
$$s_{26} \xrightarrow{(t_{g23},0)} s_{27} \xrightarrow{(t_{c23},1)} s_{28} \xrightarrow{(t_{fv23},0)} s_{29} \xrightarrow{(t_{f2},0)} s_{30} \xrightarrow{(t_{g113},0)} s_{31} \xrightarrow{(t_{c113},1)} s_{32} \xrightarrow{(t_{g113},0)} s_{33} \xrightarrow{(t_{c113},1)} s_{34} \xrightarrow{(t_{g113},0)}$$
$$s_{35} \xrightarrow{(t_{c113},1)} s_{36} \xrightarrow{(t_{fv113},0)} s_{37} \xrightarrow{(t_{g13},0)} s_{38} \xrightarrow{(t_{c13},1)} s_{39} \xrightarrow{(t_{g13},0)} s_{40} \xrightarrow{(t_{c13},1)} s_{41} \xrightarrow{(t_{g13},0)} s_{42} \xrightarrow{(t_{c13},1)} s_{43} \xrightarrow{(t_{g13},0)}$$

$s_{44} \xrightarrow{(t_{c13},1)} s_{45} \xrightarrow{(t_{g13},0)} s_{46} \xrightarrow{(t_{c13},1)} s_{47} \xrightarrow{(t_{g13},0)} s_{48} \xrightarrow{(t_{c13},1)} s_{49} \xrightarrow{(t_{g13},0)} s_{50} \xrightarrow{(t_{c13},1)} s_{51} \xrightarrow{(t_{g13},0)} s_{52} \xrightarrow{(t_{c13},1)} s_{53} \xrightarrow{(t_{fv13},0)} s_{54} \xrightarrow{(t_{f1},0)} s_{55} \xrightarrow{(t_{end_{T_1,T_2}},0)} s_{56}.$

### 8.1.4 Complexity

As presented, the state space grows exponentially in the number of tasks, more specifically, $O(k^n)$ [114], in which $n$ is the amount of non-interacting tasks, each one with $k$ local states. Considering a Petri net model generated using the proposed method, $O(k^n)$ provides an interesting upper bound estimation of the respective state space size.

In the adopted modeling approach, the number of local states ($k$) of each task is somewhat affected by the following attributes: (i) the number of task instances; (ii) the number of available voltage/frequency levels for the task; (iii) the respective release interval; and (iv) when regarding preemptive tasks, the arc weights that represents the computation time at a specific voltage (which is affected by the adopted task time unit). Intertask relations and timing constraints, excluding release, have not been taken into account in the state space size computation. Indeed, these are reasonable assumptions, since: (i) when intertask relations are explicitly modeled, the model may generate fewer reachable states; (ii) time elapsing states, which do not consider transition firing, are discarded; (iii) the set of possible reachable markings generated from a TPNE model is a subset of or equal to the marking reachability set of the respective untimed Petri net model [119]; and (iv) undesirable states (e.g, deadline missing) are avoided.

Assume that the amount of local states $k$ for a task $\tau_j$ is directly affected by the number of states $s$ in each voltage/frequeny level $i$: $k = s_1^j + s_2^j + ... + s_i^j$. Additionally, consider that each task has the same amount of voltage/frequency levels for its execution (CPU as well as ideal levels) and, for a specific level, the number of states are the same for all tasks, i.e. for two tasks $\tau_1$ and $\tau_2$, $s_1^1 = s_1^2 = s_1$, $s_2^1 = s_2^2 = s_2$, and so on. Thus, assuming $i$ voltage/frequency levels and $n$ tasks, $O(k^n) = O((s_1 + s_2 + ... + s_i)^n)$. Regarding a lower bound, each task (including the respective instances) needs to execute sequentially in the maximum voltage/frequency level, which provides the minimum number of states for a task execution ($s_{min}$). Taking into account $n$ tasks, the lower bound can be represented by $\Omega(s_{min}^1 + s_{min}^2 + ... + s_{min}^n) = \Omega(s_{min} + s_{min} + ... + s_{min}) = \Omega(ns_{min}) = \Omega(n)$.

The complexity of the state space is not only related to the adopted formalism, but, primarily, due to the scheduling problem in question. For instance, other formal methods, such as process algebras and automata, face similar complexity to tackle this scheduling problem. In the case of automata, $k^n$ states may be required to model all possible situations of a hard real-time system. Nevertheless, the proposed method only reaches the states of interest in consequence of state space reduction techniques.

Regarding time complexity, the execution of a depth-first search method is generally bounded by $O(e) = O(v^2)$ [120], in which is $e$ and $v$ are the number of edges and nodes of a graph, respectively. Since the state space of Petri nets is represented by a reachability graph [10] and the estimated size of the proposed model's state space is $O(k^n)$, the execution time related to the pre-runtime scheduling algorithm is bounded by $O((k^n)^2) = O(k^{2n})$. However, this execution time would only occur if there is no feasible schedule for a given specification, situation that would lead the algorithm to visit the

whole (reduced) state space. Additionally, the scheduling algorithm adopts some techniques that also considerably improve the average execution time. Concerning a lower bound, $\Omega(n)$ represents the minimal amount of time to find a feasible schedule. This situation may occur when the scheduling algorithm finds a feasible schedule in the first attempt due to the values obtained in the preprocessing.

Chapter 9 provides quantitative results regarding the execution time as well as the number of states actually reached by the adopted scheduling algorithm.

**8.1.4.1 Petri Net Components** For Petri net practitioners, it is interesting to provide a closer relation between the state space and the components of the conceived model (e.g., the number of places). This section provides an estimate of a state space size considering the number of states associated with the building blocks proposed. More specifically, this work has considered the following building blocks: (i) periodic task arrival; (ii) non-preemptive task structure; (iii) preemptive task structure; (iv) non-preemptive task structure with 2 voltages; and (v) preemptive task structure with 2 voltages.

Before presenting the state space for each block, remember that, for each task $\tau_i$: $r_i$ is the release time; $d_i$ is the deadline; $C_{min_i}$ is the computation time at the highest voltage level; $C_{i_j}$ is the computation time at a voltage level $v_j$; $C1_{i_j}$ and $C2_{i_j}$ are the computation times for simulating an unavailable voltage level $v_j$; and $\mathcal{S}(\tau_i)$ is the number of task instances. Moreover, consider that: $\mathcal{A}$ is the set of all system tasks; $\mathcal{V}_{cpu}$ is the set of CPU voltage levels; $\mathcal{V}_{ideal}$ is the set of ideal voltage levels, such that $\mathcal{V}_{ideal} \cap \mathcal{V}_{cpu} = \emptyset$; $\mathcal{V}_{ideal_i} \subseteq \mathcal{V}_{ideal}$ is the set of ideal voltage levels for a task $\tau_i$ (obtained with the preprocessing); and $rel\_int_i$ is the number of possibilities for releasing a task $\tau_i$ (i.e., the possible firing times of transition $t_{r_i}$ in periodic task arrival block), which is calculated in the following way: $rel\_int_i = d_i - C_{min_i} - r_i + 1$.

The number of states regarding each building block is detailed as follows:

(i) Periodic task arrival. ARRIVAL: $\mathcal{A} \rightarrow \mathbb{N}$, in which ARRIVAL$(\tau_i) = 3 + (rel_int_i \times \mathcal{I}(\tau_i))$;

(ii) Non-preemptive task structure. NON_PREEMP: $\mathcal{A} \rightarrow \mathbb{N}$, such that NON_PREEMP$(\tau_i) = 5 \times rel_int_i \times \mathcal{I}(\tau_i)$;

(iii) Preemptive task structure. PREEMP: $\mathcal{A} \times \mathcal{V}_{cpu} \rightarrow \mathbb{N}$, where PREEMP$(\tau_i, v_j) = rel_int_i \times \mathcal{I}(\tau_i) \times (3 + 2C_{i_j})$;

(iv) Non-preemptive task structure with 2 voltages. NON_PREEMP_2V: $\mathcal{A} \rightarrow \mathbb{N}$, in which NON_PREEMP_2V$(\tau_i) = 7 \times rel_int_i \times \mathcal{I}(\tau_i)$;

(v) Preemptive task structure with 2 voltages. PREEMP_2V: $\mathcal{A} \times \mathcal{V}_{ideal} \rightarrow \mathbb{N}$, such that PREEMP_2V$(\tau_i, v_j) = rel_int_i \times \mathcal{I}(\tau_i) \times (4 + 2C1_{i_j} + 2C2_{i_j})$,

As presented, the number of local states of each building block is represented by a function, which is further adopted to estimate the state space size of a specification. In addition, consider the following auxiliary functions:

- NON_PREEMPT_SEL: $\mathcal{A}\times\mathcal{V}_{cpu}\cup\mathcal{V}_{ideal} \rightarrow \mathbb{N}$, in which NON_PREEMPT_SEL$(\tau_i, v_j)$ $= \begin{cases} \text{NON\_PREEMP}(\tau_i), & if\ (v_j \in \mathcal{V}_{cpu}) \\ \text{NON\_PREEMP\_2V}(\tau_i), & if\ (v_j \in \mathcal{V}_{ideal_i}) \end{cases}$
- PREEMPT_SEL: $\mathcal{A} \times \mathcal{V}_{cpu} \cup \mathcal{V}_{ideal} \rightarrow \mathbb{N}$, such that

  PREEMPT_SEL$= \begin{cases} \text{PREEMP}(\tau_i, v_j), & if\ (v_j \in \mathcal{V}_{cpu}) \\ \text{PREEMP_2V}(\tau_i, v_j), & if\ (v_j \in \mathcal{V}_{ideal_i}) \end{cases}$

The functions above calculate the state space size for a task structure block based on whether a voltage level is available on the CPU ($\mathcal{V}_{cpu}$) or it is an ideal (unavailable) level obtained with the preprocessing ($\mathcal{V}_{ideal_i}$). In the latter situation, the unavailable voltage is simulated using the two neighboring voltage levels available on the processor. Taking into account these auxiliar functions, the following items describe two functions for estimating the number of states for a non-preemptable and a preemptable task, respectively:

- NON_PREEMP_TASK: $\mathcal{A} \rightarrow \mathbb{N}$, where NON_PREEMP_TASK$(\tau_i)$ = ARRIVAL$(\tau_i)$ + $\sum_{(v\in\mathcal{V}_{cpu}\cup\mathcal{V}_{ideal_i})}$ NON_PREEMPT_ SEL$(\tau_i, v)$
- PREEMP_TASK: $\mathcal{A} \rightarrow \mathbb{N}$, in which PREEMP_TASK$(\tau_i)$ = ARRIVAL$(\tau_i)$ + $\sum_{(v\in\mathcal{V}_{cpu}\cup\mathcal{V}_{ideal_i})}$NON_PREEMPT_SEL $(\tau_i, v)$

For a given task specification ($\mathcal{T} \subseteq \mathcal{A}$), the functions below estimate the state space size:

- Considering non-preemptable tasks, NON_PREEMP_SPEC: $\mathcal{P}(\mathcal{A}) \rightarrow \mathbb{N}$, in which NON_PREEMP_SPEC$(\mathcal{T})$ = $\prod_{(\tau_i\in\mathcal{T})}$NON_PREEMP_TASK$(\tau_i)$
- Assuming preemptable tasks, PREEMP_SPEC: $\mathcal{P}(\mathcal{A}) \rightarrow \mathbb{N}$, where PREEMP_SPEC$(\mathcal{T})$ = $\prod_{(\tau_i\in\mathcal{T})}$ PREEMP_TASK$(\tau_i)$

The presented functions can also utilize the asymptotic notation adopted in previous section. For instance. An upper bound for preemptable tasks can be represented by $O$(PREEMP_SPEC$(\mathcal{T})$). Regarding a lower bound, the sets of voltage levels in functions NON_PREEMP_TASK and PREEMP_TASK need to be replaced by a set containing only the highest CPU voltage level. Assuming PREEMP_SPEC' utilizes the adjusted function PREEMP_TASK' (which considers only the highest CPU level), the lower bound for preemptable tasks is $\Omega$(PREEMP_SPEC'$(\mathcal{T})$). The same procedure is applied to non-preemptable tasks,

### 8.1.5 DENTES Tool

The modeling and scheduling activites are automated by DENTES tool [121] - **D**evelopment **E**nvironment for **T**ime-Critical **E**mbedded **S**ystems - such that, from a system specification, the respective time Petri net model is automatically generated, and the pre-runtime

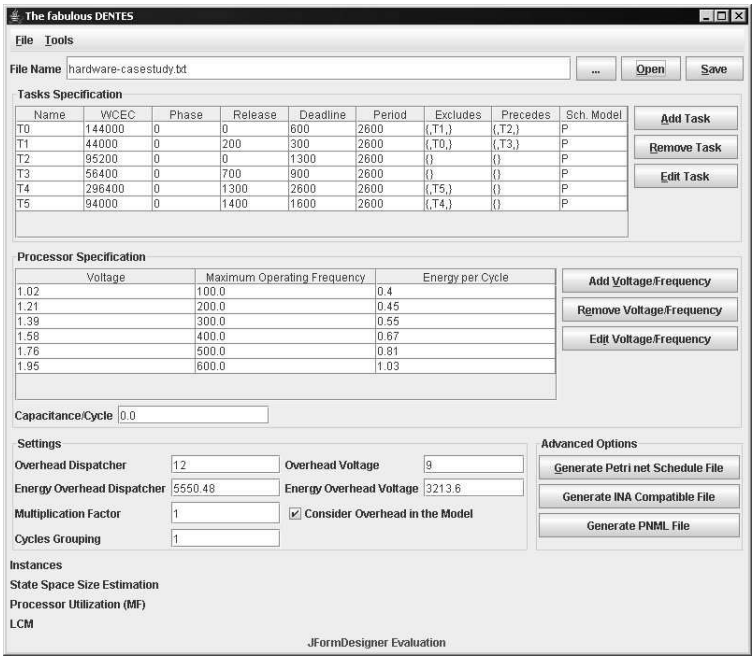


**Figure 8.4** DENTES tool - specification screen

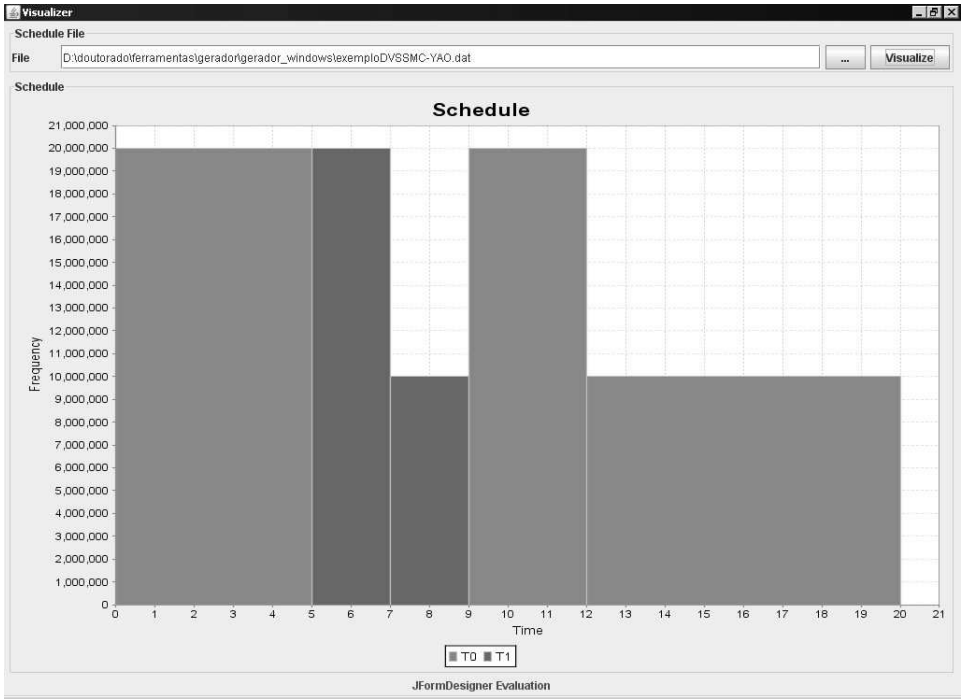


**Figure 8.5** DENTES tool - feasible schedule

scheduling is performed, hiding the details for non-specialized users. Besides, DENTES provides an interfacing mechanism that allow the analysis and verification of properties

using INA tool [99]. Figure 8.4 depicts a screenshot of DENTES tool regarding the specification screen and Figure 8.5 shows a screenshot of the feasible schedule presented in Section 8.1.3 generated using this tool. It is important to state that DENTES tool is another contribution of this thesis.

## 8.2 CODE GENERATION

This section aims at presenting the approach for C-code generation. In the proposed method, the code is generated by traversing the TLTS (see Chapter 6, Definition 6.5) (feasible firing schedule), and detecting the times when each task will execute, ensuring that the constraints are also met during system execution. The code generation activity includes not only the code of each task (implemented by C functions), but also a customized small dispatcher. Firstly, the approach for traversing the TLTS is presented, followed by an explanation of the dispatcher. Lastly, an example of code generation is provided.

### 8.2.1 Traversing TLTS

For the sake of readability, two algorithms are presented for traversing TLTSs: (i) one for non-preemptable tasks (Figure 8.6); and (ii) other assuming preemptable tasks (Figure 8.7). The basic idea is the same, but the algorithm for preemptable tasks has a small difference, which is going to be explained further. In addition, both algorithms assume overheads have been explicitly considered during schedule generation. As follows, the focus is on the generation of a schedule table, which contains all information regarding the task executions. More specifically, the schedule table contains for each task instance: (i) the start time; (ii) a flag indicating if it is a new instance, a preemption resuming, or a voltage/frequency switching; (iii) the task id; (iv) a voltage/frequency level; and (v) a function pointer. Moreover, such table lays down a basis for customizing the runtime support, namely, dispatcher.

Initially, consider the algorithm presented in Figure 8.6. The algorithm receives as an input a TLTS and executes until a state with the final marking $M^F$ is reached (line 7). The reader should note that, as the TLTS is finite and do contain a state with the desired final marking, the algorithm always stops assuming a valid input (i.e., a feasible schedule).

In each iteration, variable `globalClock` is updated to keep the current global time (line 9), which is calculated by summing the elapsed times in each previous state as well as the time elapsed in the current one. Next, the algorithm checks if the transition ($t$) fired in the current state represents a processor granting ($t_{g_{i_j}}$) or a voltage/frequency switching ($t_{fv1_{i_j}}$) (line 11). Both transitions are assumed to include the overheads for executing a task instance (see Section 6.3.10), and, therefore, they are adopted to indicate the start time or the time the voltage/frequency is switched for the same instance. Additionally, an entry is created in the schedule table ($\texttt{item}_i$) when one of these transitions is encountered.

The start time (or the time for switching the voltage/frequency level) is obtained (line 12) from the subtraction between the global time (`globalClock`) and the time spent on the current state (where the transition is fired). For a better understanding, consider a

```
1 traverseTLTSNonPreemptive(ℒ = (S, Σ, →, s0))
2 {
3   scheduleTable = ∅;
4   currentState = s0;
5   globalClock = 0;
6   i = 0;
7   while (currentState do not contain final marking M^F) {
8     t = getTransition(currentState,→);
9     globalClock += getElapsedTime(currentState,→);
10
11    if(isGrantTrans(t) || isVoltageSwitchingTrans(t)) {
12     start = globalClock - getElapsedTime(currentState,→);
13
14     if(start < 0 ∧ i = 0) {
15      item_i.startTime = 0;
16     } else {
17      item_i.startTime = start;
18     }
19
20     if(isGrantTrans(t)) {
21      item_i.flag = INSTANCE;
22     } else {
23      item_i.flag = VOLT_SWICTH;
24     }
25
26     item_i.taskID = getTask(t);
27     item_i.voltageFrequency = getVoltageFrequency(t);
28     item_i.functionPointer = getCodeFunction(t);
29     scheduleTable = scheduleTable ∪ item_i;
30     i++;
31    }// end - if
32    currentState = getNextState(currentState,→);
33  }// end - for
34
35  return scheduleTable;
36 }
```

**Figure 8.6** Algorithm for traversing the TLTS assuming non-preemptable tasks

TLTS representing only one task execution with release time at 5 and overhead of 1 time unit. Assuming that the processor-granting transition ($t_{g_{i_j}}$) fires in the current state, the `globalClock` value is 6 (5 + 1 = 6), but the start time of the task instance is 5 (6 - 1 = 5). Also, the algorithm needs to check if the time obtained is a negative integer (line 14). Such situation may occur when the first instance is being examined and the respective release time is 0 (e.g., assuming the previous example, 0 - 1 = -1). In this case, the start time is set to zero (line 15). Afterwards, variable $\texttt{item}_i$ is filled with information regarding the behaviour (line 20-24) (new instance or voltage/frequency switching), the task ID (line 26), the voltage/frequency level (line 27), and a pointer to the task function (line 28). Next, $\texttt{item}_i$ is added into the table (line 29). After reaching the final state (which contains the marking $M^F$), the algorithm returns the schedule table (line 35) with all entries $\texttt{item}_i$ ordered in ascending order in accord with $i$.

Figure 8.7 shows the algorithm for traversing a TLTS taking into account preemptable tasks. The difference from previous algorithm is due to some checks that must

```
1 traverseTLTSPreemptive(ℒ = (S, Σ, →, s0))
2 {
3    scheduleTable = ∅;
4    currentState = s0;
5    globalClock = 0;
6    i = 0;
7    while (currentState do not contain final marking M^F) {
8       t = getTransition(currentState,→);
9       globalClock += getElapsedTime(currentState,→);
10
11      if(isOverheadTrans(t) || isVoltageSwitchingTrans(t)) {
12       start = globalClock - getElapsedTime(currentState,→);
13
14       if(start < 0 ∧ i = 0) {
15        item_i.startTime = 0;
16       } else {
17        item_i.startTime = start;
18       }
19
20       if(isOverheadTrans(t) ∧ isTaskConcluded(t)) {
21        item_i.flag = INSTANCE;
22       } else if(isOverheadTrans(t) ∧ ¬isTaskConcluded(t)) {
23        item_i.flag = RETURN;
24       } else if(isVoltageSwitchingTrans(t) ∧ isTaskLastCompTrans(t)) {
25        item_i.flag = VOLT_SWITCH;
26       } else if(isVoltageSwitchingTrans(t) ∧ ¬isTaskLastCompTrans(t)) {
27        item_i.flag = RETURN;
28       }
29
30       item_i.taskID = getTask(t);
31       item_i.voltageFrequency = getVoltageFrequency(t);
32       item_i.functionPointer = getCodeFunction(t);
33       scheduleTable = scheduleTable ∪ item_i;
34       i++;
35      }// end - if
36      else if(isComputationTrans(t)) {
37       keepTrackTask(t);
38      }
39
40      currentState = getNextState(currentState,→);
41   }// end - for
42
43   return scheduleTable;
44 }
```

**Figure 8.7** Algorithm for traversing the TLTS assuming preemptable tasks

be performed in order to capture the preemption points. Similarly, this algorithm creates an entry in the schedule table, if the transition fired in the current state represents an overhead ($t_{o_{i_j}}$ or $t_{o1_{i_j}}$) or a voltage/frequency switching ($t_{ov_{i_j}}$) (line 11). To assist the verification of preemption points, 3 functions are adopted: (i) `isTaskConcluded`; (ii) `isTaskLastCompTrans`; and (iii) `keepTrackTask`. Function `isTaskConcluded(t)` returns a true value, when there is a finished task $\tau$ associated with transtion $t$. Function `isTaskLastCompTrans(t)` provides a true value only in situations where a task $\tau$, associated with transtion $t$, fired the last computation transition. In order to support previous functions, function `keepTrackTask` traces the execution of each task by verifying the occurrences of all computation transitions (e.g., $tc_{i_j}$ and $tlc_{i_j}$). From these transitions, the function obtains the task as well as the associated voltage/frequency level, and, therefore, it can keep track the number of firings required for concluding a task $\tau$. Previous checking is not difficult to visualize, since dividing the worst-case execution cycles by CPU frequency returns the amount of occurrences for the computation transitions in a voltage/frequency level. Thus, when the number of firings (regarding computation transitions) reaches the amount required for a task execution, function `keepTrackTask`

considers the task concluded (i.e., resets the respective counter).

Taking into account the new functions, lines 22 and 26 show the verifications for the preemption points. At line 22, if an overhead transition is fired and the task have not concluded yet, this situation indicates that the task was preempted. The overhead transition is mandatorily fired in the beginning of a task execution and function `keepTrackTask` only assumes a running task after the first firing of a computation transition. More specifically, when a task instance is starting the execution (i.e, the overhead transition is fired for the first time), function `isTaskConcluded` returns false. Thus, more than one occurrence of an overhead transition before the conclusion, indicates that a preemption has occurred. Line 26 checks if the transition fired represents a voltage/frequency switching and the last computation transition is not associated with the same task. This condition can happen when the task was preempted immediately before performing the voltage/frequency switching.

### 8.2.2 Dispatcher

The dispatcher is responsible for providing runtime services, in such a way that it is always activated when a timer interrupt occurs. In order to control the executions of each task, this component utilizes the information provided in the schedule table.

```
 1 void dispatcher()
 2 {
 3   struct SchItem item = sch[schIndex];
 4   globalClock = item.starttime;
 5
 6   if(currentTaskPreempted) {
 7     // context saving
 8   }
 9   if(item.flag == RETURN) {
10     // context restoring
11   }
12   else {
13     taskFunction = item.functionPointer;
14   }
15   schIndex = ((++schIndex)%SCHEDULE_SIZE);
16   progrTimer(sch[schIndex].starttime);
17   adjustVoltFreq(item.voltagefrequency);
18   restoreOrExecuteTask();
19 }
```

**Figure 8.8** Dispatcher

The dispatcher (Figure 8.8) adopts a set of data structures, which includes the schedule table that contains, as presented previously, the following information for each task instance: (i) the start time; (ii) a flag indicating if it is a new instance, a preemption resuming, or a voltage/frequency switching; (iii) the task id; (iv) a voltage/frequency level; and (v) a function pointer. Such table is stored in an array of type `SchItem`. Besides, there are some shared variables that stores information about the size of the schedule (`SCHEDULE_SIZE`), information of the new task to be executed (`struct SchItem item`), and a pointer to the task function (`taskFunction`). An explanation of the dispatcher is as follows:

1, During the system initialization, the timer is programmed using the first entry in the schedule table. Whenever a timer interrupt occurs, the control is transferred to the dispatcher;

2, The dispatcher accesses the schedule table (line 3) to retrieve the information regarding the next task execution. Additionally, variable `globalClock`, which keeps the current global time, is updated;

3, Before calling a task, the dispatcher checks if the current task has been preempted (line 6). In this case, the task context is saved (line 7);

4, If the next task was preempted and now it is being resumed (line 9), the dispatcher prepares the respective context to be restored (line 10);

5, If it is a new task instance (line 12), the dispatcher just stores the function pointer (line 13) in the variable `taskFunction` (utilized by the dispatcher at line 18);

6, Next, variable `schIndex` is updated (line 15), such that it points to the task that is going to be executed in the next dispatcher activation;

7, After that, the timer is programmed to generate an interrupt at the start time provide by the next table entry (which is pointed by `schIndex`);

8, Lastly, the CPU voltage/frequency level is adjusted (line 17), and the dispatcher restores the task context or creates a new instance (line 18).

It is worth stating that the proposed approach assumes the dispatcher execution at a fixed voltage/frequency level, which is up to the designer to select the appropriate one (see Section 6.3.10). Although not depicted in Figure 8.8, the CPU voltage/frequency level is adjusted at the beginning of dispatcher code to consider the level defined by the designer. Besides, since a TLTS provides information regarding the number of context-switchings as well as if preemptions indeed occur, a customized dispatcher is generated for each given specification.

### 8.2.3 Example

As an example, consider the specification presented in Section 8.1.3. Although the respective Petri net model (Figure 8.3) does not take into account overheads (a requirement for the algorithms presented in Section 8.2.1), an alternative TLTS considering such transitions is presented as follows (assuming the overhead cost is 0):

$$s_0 \xrightarrow{(t_{start},0)} s_1 \xrightarrow{(t_{ph1},0)} s_2 \xrightarrow{(t_{r1},0)} s_3 \xrightarrow{(t_{ph2},0)} s_4 \xrightarrow{(t_{v13},0)} s_5 \xrightarrow{(t_{vs13},0)} s_6 \xrightarrow{(t_{g1130},0)} s_7 \xrightarrow{(t_{o113},0)} s_8 \xrightarrow{(t_{c113},1)}$$
$$s_9 \xrightarrow{(t_{g1131},0)} s_{10} \xrightarrow{(t_{c113},1)} s_{11} \xrightarrow{(t_{g1131},0)} s_{12} \xrightarrow{(t_{c113},1)} s_{13} \xrightarrow{(t_{g1131},0)} s_{14} \xrightarrow{(t_{c113},1)} s_{15} \xrightarrow{(t_{g1131},0)} s_{16} \xrightarrow{(t_{c113},1)}$$
$$s_{17} \xrightarrow{(t_{r2},0)} s_{18} \xrightarrow{(t_{v23},0)} s_{19} \xrightarrow{(t_{vs23},0)} s_{20} \xrightarrow{(t_{g1231},0)} s_{21} \xrightarrow{(t_{o123},0)} s_{22} \xrightarrow{(t_{c123},1)} s_{23} \xrightarrow{(t_{g1232},0)} s_{24} \xrightarrow{(t_{l123},0)}$$
$$s_{25} \xrightarrow{(t_{lc123},1)} s_{26} \xrightarrow{(t_{fv123},0)} s_{27} \xrightarrow{(t_{go23},0)} s_{28} \xrightarrow{(t_{ov23},0)} s_{29} \xrightarrow{(t_{c23},1)} s_{30} \xrightarrow{(t_{g232},0)} s_{31} \xrightarrow{(t_{l23},0)} s_{32} \xrightarrow{(t_{lc23},1)}$$
$$s_{33} \xrightarrow{(t_{fv23},0)} s_{34} \xrightarrow{(t_{f2},0)} s_{35} \xrightarrow{(t_{g1130},0)} s_{36} \xrightarrow{(t_{o113},0)} s_{37} \xrightarrow{(t_{c113},1)} s_{38} \xrightarrow{(t_{g1131},0)} s_{39} \xrightarrow{(t_{c113},1)} s_{40} \xrightarrow{(t_{g1131},0)}$$

$s_{41} \xrightarrow{(t_{l113},0)} s_{42} \xrightarrow{(t_{lc113},1)} s_{43} \xrightarrow{(t_{fv113},0)} s_{44} \xrightarrow{(t_{go13},0)} s_{45} \xrightarrow{(t_{ov13},0)} s_{46} \xrightarrow{(t_{c13},1)} s_{47} \xrightarrow{(t_{g131},0)} s_{48} \xrightarrow{(t_{c13},1)}$
$s_{49} \xrightarrow{(t_{g131},0)} s_{50} \xrightarrow{(t_{c13},1)} s_{51} \xrightarrow{(t_{g131},0)} s_{52} \xrightarrow{(t_{c13},1)} s_{53} \xrightarrow{(t_{g131},0)} s_{54} \xrightarrow{(t_{c13},1)} s_{55} \xrightarrow{(t_{g131},0)} s_{56} \xrightarrow{(t_{c13},1)} s_{57} \xrightarrow{(t_{g131},0)}$
$s_{58} \xrightarrow{(t_{c13},1)} s_{59} \xrightarrow{(t_{g131},0)} s_{60} \xrightarrow{(t_{l13},0)} s_{61} \xrightarrow{(t_{lc13},1)} s_{62} \xrightarrow{(t_{fv13},0)} s_{63} \xrightarrow{(t_{f1},0)} s_{64} \xrightarrow{(t_{end},0)} s_{65}.$

Traversing the TLTS using the algorithm depicted in Figure 8.7, the code depicted in Figure 8.9 is obtained. The algorithm found out that tasks $\tau_1$ and $\tau_2$ need to be executed one time, $\tau_1$ for clock value equal to 0, and $\tau_2$ for clock value equal to 5. Since task $\tau_1$ is preempted at 5 by task $\tau_2$, $\tau_1$ is resumed from preemption at 9. Observe that, in this example, each task instance is executed at two different voltage/frequency levels in order to simulate the unavailable voltage/frequency of 1.5v/15MHz [12]. In this case, when the dispatcher is called at clock values equal to 7 and 12, only a voltage/frequency tuning is performed, incurring a minimal overhead during system execution. In other words, no context-saving or restoring needs to be carried out. For this example, the task time unit is equal to 1 second. Although this work considers C code as the behavioral description, the method is programming language independent.

```
void codeT1() {...}
void codeT2() {...}
#define SCHEDULE_SIZE 5
struct SchItem sch[SCHEDULE_SIZE] =
{
  {0, INSTANCE, 1, 2V/20MHz,(int *)codeT1},
  {5, INSTANCE, 2, 2V/20MHz,(int *)codeT2},
  {7, VOLT_SWITCH, 2, 1V/10MHz,(int *)codeT2},
  {9, RETURN, 1, 2V/20MHz,(int *)codeT1},
  {12, VOLT_SWITCH, 1, 1V/10MHz, (int *)codeT1},
};
```

**Figure 8.9** Generated code example

## 8.3 HANDLING DYNAMIC SLACK TIMES

During system runtime, slack times (CPU idle times) may appear due to tasks' early completion. In order to take advantage of such slacks for reducing even more energy consumption, a small runtime scheduler is proposed for adjusting the starting times as well as the voltage/frequency levels associated to each task instance.

Initially, during system runtime, a dispatcher is adopted to execute tasks according to the feasible schedule generated in design-time. If a task instance completes its execution earlier than the respective WCEC, the scheduler is executed to advance the next task instance and to adjust the voltage/frequency to a lower level than the one defined for such instance in the pre-runtime schedule. Nevertheless, only if the next task can start its execution earlier and the energy saving compensates the overhead incurred by the scheduling, the adjusting is performed. It is important to state that the scheduler does not violate inter-task relations, since it follows the tasks' order defined in the pre-runtime schedule. Additionally, the scheduler overhead related to the feasibility test is considered in the pre-runtime method, more specifically, in the overhead block presented in Section 6.3.10.

Before presenting the runtime scheduler algorithm, some concepts are required firstly. In the proposed pre-runtime scheduling, a set of concurrent tasks is scheduled considering a given time unit (e.g., milliseconds), namely, task time unit. In consequence, the pre-runtime schedule is partitioned in several *time slices* of the same size, in which each slice corresponds to one task time unit, and the total amount is equal to the LCM. For instance, Figure 8.11(a) depicts a feasible schedule, in which the total amount of time slices is equal to 10. These slices can be grouped into segments in such a way that represent task executions. Such segments are denominated *task segments*, and each one is represented by an interval ([start,end]). When a task is not completely executed within a segment, the task is preempted, in other words, it is carried out through more segments. Considering Figure 8.11(a), the segments are: (i) $\tau_1 = [1, 2]$; (ii) $\tau_2^1 = [2, 4]$; (iii) $\tau_3 = [4, 6]$; (iv) $\tau_2^2 = [6, 9]$; (v) $\tau_4 = [9, 10]$. These intervals resemble a pre-runtime schedule table, which stores information about the execution of each task instance. Moreover, a global clock (`globalclock`) is adopted for tracking the current time (e.g., the accumulated number of time slices). Taking into account the previous concepts, the runtime schedule algorithm is depicted in Fig.8.10 using a C syntax notation.

In order to check the early completion of a task instance, the runtime scheduler is executed at the end of each segment, in such a way that its execution does not conflict with the dispatcher execution. Firstly, the scheduler verifies which is the next segment (line 2) in the pre-runtime schedule, since it is the candidate for adjusting the respective voltage/frequency level as well as the start time. If there is no segment to be executed - the remaining segments are returns from preemption of finished instances or the last segment was already executed - the original pre-runtime schedule is kept (line 5). Also, the pre-runtime schedule is not changed whether the start time of the next segment is equal to the release time assigned to the respective task instance. Considering that there is an available segment, the respective start time is set so that the release time is not violated (line 8). If the next segment can be promptly started, the start time is tuned for taking into account the scheduler WCET (worst-case execution time). It is worth noting that the adjustment is only performed whenever the improvements compensate the scheduling overhead (line 12). Regarding Ishihara's theorem (line 13), the reader should also refer to Section 6.3.6 for details.

For a better understanding, consider the schedule depicted in Figure 8.11(a). A DVS platform based on [64] is adopted, considering the following voltage/frequency levels ($vff$) and the respective energy consumptions ($vef$ - Section 5.2.4): $vff = \{$(1.21V,20MHz), (1.39V,30MHz), (1.76V,50MHz), (1.95V,60MHz)$\}$ and $vef = \{$(1.21V, $0.41nJ$/cycle), (1.39V, $0.54nJ$/cycle), (1.76V,$0.87nJ$/cycle), (1.95V,$1.07nJ$/cycle)$\}$. In this example, if task $\tau_2$ completes its execution earlier at 7 (Figure 8.11(b)) , the proposed scheduler attempts to adjust the voltage/frequency level as well as the start time of the next segment ($\tau_4$). Considering that $\tau_4$ release time is equal to 6, $\tau_4$ can start its execution earlier and utilize a lower voltage/frequency level (Figure 8.11(c)). Assuming that WCEC of each task are $c_1 = 50 \times 10^6$, $c_2 = 150 \times 10^6$, $c_3 = 100 \times 10^6$, $c_4 = 60 \times 10^6$, the energy consumption is reduced from 0.2433J (Figure 8.11(b)) to 0.2037J (Figure 8.11(c)).

Although the runtime scheduler consumes a small amount of time and energy, the overhead may be unbearable for some applications due to very tight constraints. Nev-

```
1 scheduler() {
2   nextSegment = retriveNextSegment();
3   taskInstance = nextSegment.taskInstance;
4
5   if(nextSegment not exists || taskInstance.release == nextSegment.start)
6     {return; //keep the pre-runtime schedule}
7
8   if(taskInstance.release > globalClock)
9     {nextSegment.start = taskInstance.release}
10   else {nextSegment.start = globalClock + schedulerWCET }
11
12   if (energy saving compensates the overhead)
13      {adjust voltage according to Ishihara's theorem [12]
       and the new execution window (nextSegment.end - nextSegment.start);
14      Prepare dispatcher to execute at nextSegment.start;}
15   else {return; //keep the pre-runtime schedule}
16 }
```

**Figure 8.10** Runtime scheduler algorithm

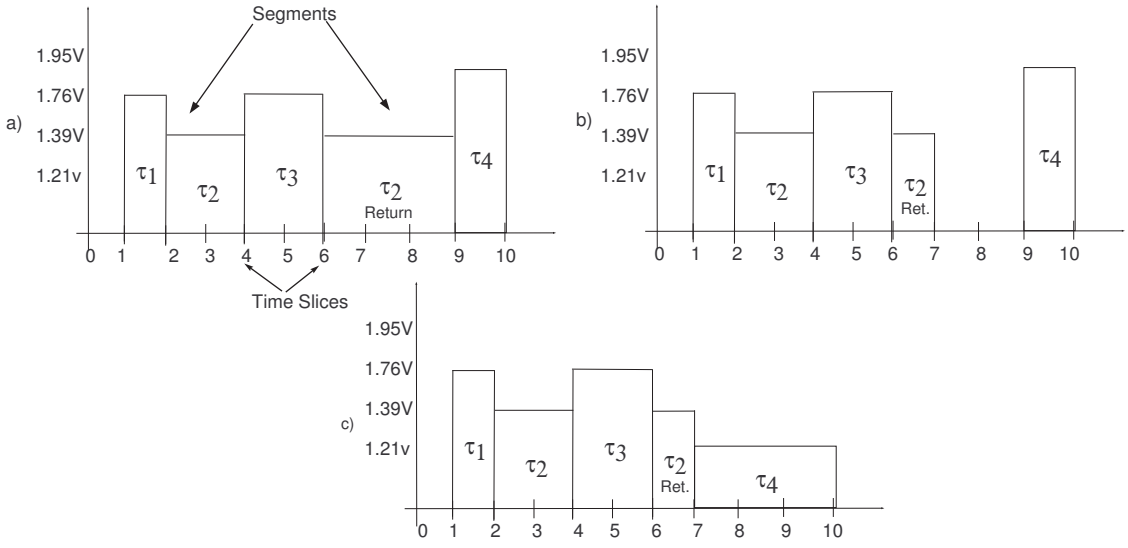


**Figure 8.11** Runtime scheduler example

ertheless, the scheduler WCEC can be considerably improved by utilizing a table with pre-calculated values [31] instead of computing the voltage/frequency levels at runtime (line 13). Since predictability is one of the most important properties of hard real-time systems, the designer can verify the situations in which the tasks terminate earlier and store pre-calculated levels for such situations. In the experiments adopted in this work (see next chapter), this approach has been adopted.

## 8.4 SUMMARY

This chapter described two important activities of the proposed software synthesis method: scheduling and code generation. The scheduling activity adopts a pre-runtime scheduling algorithm in order to find a feasible schedule that satisfies inter-task relations, timing and

energy constraints. Such algorithm performs a state space exploration, applying some efficient techniques to avoid the state space size explosion. From a feasible schedule, the code generation activity provides a predictable code with a customized runtime support, namely dispatcher. As the code is generated from a feasible schedule, all specified constraints are met during system runtime. Besides, this chapter presented a runtime scheduler for dealing with slack times that may appear due to the earlier completion of tasks. The slack times provide interesting opportunities to further reduce even more the energy consumption of a hard real-time system.

CHAPTER 9

# CASE STUDIES

To evaluate the proposed software synthesis method, this work has adopted some case studies, which are composed of real-world applications as well as custom-built examples (that simulates real-world situations). The case studies provide interesting scenarios for evaluating not only the software synthesis, but also the pre-runtime and runtime scheduling algorithms. Firstly, case studies related to the pre-runtime scheduling algorithm are presented, and, next, results concerning the runtime scheduler and the software synthesis are described.

Regarding the software synthesis and the runtime scheduler, the cases studies have been implemented in a real hardware platform, which was conceived to facilitate the measurement and validation activities. Therefore, before presenting the results, the hardware platform is described followed by the validation activity.

## 9.1 PRE-RUNTIME SCHEDULING

Table 9.1 summarizes the case studies adopted to evaluate the pre-runtime scheduling algorithm. For all case studies, the timing constraints were met. Additionally, in that table, column *task* represents the number of tasks; *inst.* represents the number of tasks' instances; *size* depicts a state space size estimation; *sch.* is the number of states of the feasible schedule; *found* counts the number of states actually verified for finding a feasible schedule; *w/DVS* is the estimated energy consumed (in joules) by the found feasible schedule using DVS; o/DVS is the estimated energy consumed in joules by an alternative schedule that disregards DVS; *lpedf* is the energy consumed (in joules) by a schedule generated using the optimal scheduling mechanism proposed by Yao et al., considering discrete voltage/frequency levels [26]; and *time* expresses the algorithm execution time (in seconds) for finding the feasible schedule. All experiments were performed on a Pentium D 3GHz, 4Gb RAM, Linux, and compiler GCC 3.3.2.

**Table 9.1** Experimental results summary

| Case Study | task | inst. | size | sch. | found | w/DVS(J) | o/DVS(J) | lpedf | time(s) |
|---|---|---|---|---|---|---|---|---|---|
| 1.Motiv. Example | 4 | 4 | $7 \times 10^7$ | 48 | 141 | 0.2474 | 0.3132 | 0.17090 | 0.001 |
| 2.Example 2 | 6 | 6 | $7 \times 10^{35}$ | 4377 | 518406 | 0.00069 | 0.00105 | 0.00048 | 35.200 |
| 3.Example 3 | 12 | 12 | $2 \times 10^{32}$ | 551 | 9906 | 267.84000 | 360.00000 | 254.11840 | 0.282 |
| 4.Kwon's Example | 4 | 4 | $5 \times 10^{14}$ | 246 | 246 | 279.0000 | 371.0000 | 279.0000 | 0.003 |
| 5.CNC Control | 8 | 289 | $9 \times 10^{70}$ | 235852 | 1884381 | 0.11900 | 0.34500 | 0.09440 | 291.221 |
| 6.Pulse Oximeter | 3 | 10 | $2 \times 10^8$ | 83 | 4268 | 0.00021 | 0.00023 | 0.00014 | 0.234 |
| 7.MP3 & GSM | 8 | 3604 | $3 \times 10^{68}$ | 381313 | 381313 | 3.86200 | 4.76600 | 3.85410 | 9.606 |
| 8. Thermal Printer | 5 | 10 | $9 \times 10^{18}$ | 320 | 85085 | 0.01607 | 0.01682 | 0.01223 | 3.949 |

For a better comprehension, the following sections detail each case study. The reader should remember that, as presented in Chapter 5, each task is represented by a tuple

$\tau_i = (ph_i, r_i, c_i, d_i, p_i, code_i)$, in which $ph_i$ is the initial phase; $r_i$ is the release time; $c_i$ is the worst-case execution cycles (WCEC) required for executing task $\tau_i$; $d_i$ is the deadline; and $p_i$ is the period; and $code_i$ is the task C code. Because not all case studies are adopted in the code generation activity, assume $code_i$ is optional. Additionally, function $vff$ represents the voltage/frequency levels and $vef$ the energy consumption per clock cycle in each level. Moreover, it is important to state that some original specifications do not provide the WCECs of each task, but the worst-case execution times (WCET). In order to obtain the number of cycles, each WCET was multiplied by the maximum frequency level available in the specification.

After presenting each case study, the results depicted in Table 9.1 are commented analytically. Next, this section presents some data regarding the algorithm scalability.

### 9.1.1 Motivational Example and Example 2

Case studies 1 and 2 are based on example 2 presented in [6], which demonstrates conditions in which pre-runtime approaches would be able to find feasible schedules, whereas runtime methods may fail due to the exclusion relation between tasks. For a complete explanation, Section 3.2.3 details the situation, which leads to occasions where the processor need to be left idle to find a valid schedule. In this work, Xu's example is extended with additional tasks and intertask relations. Motivational example incorporates 2 additional tasks (see Section 3.2.3 for details) and Example 2 takes into account 4 new tasks as well as overheads (voltage/frequency switching and dispatcher execution).

For Example 2, the task time unit is $1\mu s$ and the task set $\mathcal{T}$ is composed of 6 preemptable tasks: $\tau_1 = (0, 0, 147 \times 10^3, 6, 26)$, $\tau_2 = (0, 2, 47 \times 10^3, 3, 26)$, $\tau_3 = (0, 0, 976 \times 10^2, 13, 26)$, $\tau_4 = (0, 7, 582 \times 10^2, 9, 26)$, $\tau_5 = (0, 13, 2982 \times 10^2, 26, 26)$, and $\tau_6 = (0, 14, 97 \times 10^3, 16, 26)$. In addition to timing constraints, the specification contains the following intertask relations: $\tau_1$ *excludes* $\tau_2$; $\tau_1$ *precedes* $\tau_3$; $\tau_2$ *excludes* $\tau_1$; $\tau_2$ *precedes* $\tau_4$; $\tau_5$ *excludes* $\tau_6$; and $\tau_6$ *excludes* $\tau_5$. Moreover, the voltage/frequency levels are the same as the platform described in [64], more specifically, $vff = \{$(1.02V, 10MHz),(1.04V, 20MHz),(1.07V, 30MHz), (1.15V, 40MHz),(1.26V, 50MHz),(1.38V, 60MHz)$\}$. Assuming an average switching capacitance of 0.28nF/cycle [72], $vef = \{$(1.02V, $0.3nJ$/cycle),(1.04V, $0.31nJ$/cycle),(1.07V, $0.34nJ$/cycle), (1.15V, $0.38nJ$/cycle),(1.26V, $0.45nJ$/cycle),(1.38V, $0.54nJ$/cycle$\}$. Besides, the dispatcher overhead is $o = 60\mu s$ at 60MHz and the time overhead related to voltage/frequency switching is $a = 10\mu s$.

As depicted in Table 9.1, in addition to meet all timing constraints, the proposed scheduling algorithm reduced energy consumption by adopting DVS.

### 9.1.2 Example 3

As in the previous cases, Case study 3 is based on Figure 2 of [11], which also depicts a condition in which runtime scheduling methods may not work. Since tasks may finish their executions earlier than their respective worst-case execution times, a runtime scheduler may promptly start a task for execution due to the release time and postpone another task with an earlier deadline. The generated schedule may be infeasible, because, assuming an exclusion relation between these tasks, the task with the earlier deadline is forced to

wait for the other task's execution, and so, its timing constraints may be violated. The pre-runtime approach, however, avoids this undesirable situation, since the dispatcher always starts tasks' execution in accordance with the sequence defined in the feasible schedule. To adopt the example, the computation times were tuned for allowing voltage scaling.

The task set is presented as follows: $\tau_{A0}$ = (0, 0, 500 × $10^3$, 80, 120), $\tau_{A1}$ = (0, 10, 1000 × $10^3$, 100, 120), $\tau_{A2}$ = (0, 30, 300 × $10^3$, 120, 120), $\tau_B$ = (0, 20, 1000 × $10^3$, 120, 240), $\tau_C$ = (0, 30, 1000 × $10^3$, 50, 120), and $\tau_D$ = (0, 90, 1000 × $10^3$, 110, 240), $\tau_E$ = (0, 0, 400 × $10^3$, 240, 240), and $\tau_F$ = (0, 0, 1000 × $10^3$, 240, 240). The intertask relations are: $\tau_{A0}$ *excludes* $\tau_D$; $\tau_D$ *excludes* $\tau_{A0}$; $\tau_{A1}$ *excludes* $\tau_D$; $\tau_D$ *excludes* $\tau_{A1}$; $\tau_{A2}$ *excludes* $\tau_D$; $\tau_D$ *excludes* $\tau_{A2}$; $\tau_D$ *excludes* $\tau_E$; $\tau_E$ *excludes* $\tau_D$;$\tau_{A0}$ *excludes* $\tau_F$; $\tau_F$ *excludes* $\tau_{A0}$; $\tau_{A1}$ *excludes* $\tau_F$; $\tau_F$ *excludes* $\tau_{A1}$; $\tau_{A2}$ *excludes* $\tau_F$; $\tau_F$ *excludes* $\tau_{A2}$; $\tau_{A1}$ *excludes* $\tau_C$; $\tau_C$ *excludes* $\tau_{A1}$; $\tau_D$ *excludes* $\tau_F$; $\tau_F$ *excludes* $\tau_D$; $\tau_E$ *excludes* $\tau_F$; $\tau_F$ *excludes* $\tau_E$; $\tau_C$ *excludes* $\tau_F$; $\tau_F$ *excludes* $\tau_C$; $\tau_{A0}$ *precedes* $\tau_C$; $\tau_{A0}$ *precedes* $\tau_{A1}$ and $\tau_{A1}$ *precedes* $\tau_{A2}$. Additionally, all tasks are preemptable and the processor model adopted in this experiment is based on [12]. More specifically, the voltage/frequency levels are $vff$ = {(2V,20MHz),(3V,30MHz),(5V,50MHz)}, and the respective energy consumptions are $vff$ = {(2V,6.4$nJ$/cycle),(3V,14.4$nJ$/cycle),(5V,40$nJ$/cycle)}. Results are shown in Table 9.1.

### 9.1.3 Kwon's Example

Case study 4 is depicted on Table 1 in [26] and does not consider any intertask relation. In this case, the proposed approach finds a feasible schedule that consumes the same amount of energy as the approach described in [26] (see Table 9.1), which is the Yao's algorithm extended with discrete set of voltage levels. The task set $\mathcal{T}$ is composed of the following preemptable tasks: $\tau_1$ = (0, 0, 150 × $10^6$, 11, 11), $\tau_2$ = (0, 3, 120 × $10^6$, 8, 11), $\tau_3$ = (0, 5, 180×$10^6$, 8, 11), and $\tau_4$ = (0, 9, 80×$10^6$, 11, 11). The voltage/frequency levels and the respective energy consumption values are: $vff$ = {(3V,30MHz),(5V,50MHz),(7V,70MHz)} and $vef$ = {(3V, 300$nJ$/cycle),(5V, 500$nJ$/cycle),(7V, 700$nJ$/cycle)}. Moreover, the task time unit is 100 $ms$.

### 9.1.4 CNC Control

Case study 5 is the control software of a Computerized Numerical Control (CNC) machine [122], which is an automatic machining tool adopted for manufacturing user-designed workpieces. The CNC Controller specification is composed of several concurrent tasks with exclusion relations, which becomes an interesting case study for evaluating the proposed pre-runtime scheduling algorithm. The task set $\mathcal{T}$ is composed of the following preemptable tasks: $\tau_{sampl} = (0, 0, 2450, 2400, 2400)$, $\tau_{calv} = (0, 0, 2800, 2400, 2400)$, $\tau_{dist} = (0, 0, 12600, 4800, 4800)$, $\tau_{stts} = (0, 0, 50400, 4800, 4800)$, $\tau_{xref} = (0, 0, 11500, 2400, 2400)$, $\tau_{yref} = (0, 0, 50400, 4800, 4800)$, $\tau_{xctrl} = (0, 0, 49900, 4000, 9600)$, and $\tau_{yctrl} = (0, 0, 39900, 4000, 7800)$. Regarding intertask relations, Figure 9.1 depicts the interactions between the tasks in the specification. In that figure, ellipses represent the tasks and rectangles correspond to the shared resources. Any incoming or outgoing arrow from a resource

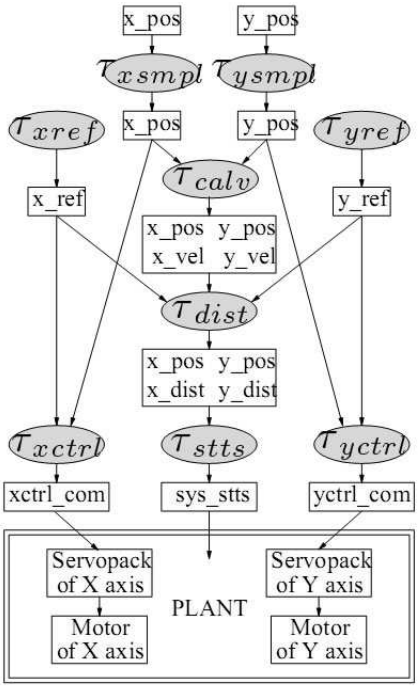


**Figure 9.1** CNC intertask relations

points out the utilization of such resource by a task. Therefore, tasks utilizing the same resource have exclusion relations between them.

For this case study, the processor model is based on [12], and the respective voltage/frequency levels are $vff = \{$(3V,30MHz),(4V,40MHz),(5V,50MHz),(6V,60MHz),(7V,70 MHz)$\}$. Concerning energy consumption, the values are $vef = \{$(3V, $14.4nJ$/cycle),(4V, $25.6nJ$/cycle),(5V, $40nJ$/cycle),(6V, $57.8nJ$/cycle),(7V, $78.4nJ$/cycle)$\}$. Moreover, the task time unit is $1\mu s$.

As depicted in Table 9.1, in addition to meet all timing constraints, the proposed scheduling algorithm reduced energy consumption by adopting DVS.

### 9.1.5 Pulse Oximeter

Case study 6 is a pulse oximeter [123], which is an electronic device responsible for measuring the blood oxygen saturation using a non-invasive method. Originally, the pulse oximeter specification is composed of several small tasks with precedence relations. Nevertheless, these small tasks have been merged into three larger tasks due to limitations (i.e., the dispatcher overhead) in the adopted hardware platform (see Section 9.2.1). As will be presented, this case study is also adopted to evaluate the software synthesis method.

The pulse-oximeter is represented by the following task set, which has a task time unit of $1ms$ and all tasks are non-preemptable: (i) excitation $T_{exc} = (0, 0, 2100, 2, 2,$ codeTExc{...}); (ii) acquisition $T_{acq} = (0, 0, 2928, 16, 16,$ codeTAcq{...}); and (iii) control $T_{ctr} = (0, 0, 102120, 16, 16,$ codeTCtr{...}). The specification also contains the intertask re-

lation $T_{acq}$ *precedes* $T_{ctr}$ and adopts the hardware platform described in Section 9.2.1. Additionally, the dispatcher worst-case execution time is $o = 100\mu s$ at 1.95V/60MHz, since there are no preemptions in such experiment. Table 9.1 presents the results.

### 9.1.6 MP3 & GSM

Case study 7 [124] is an application composed of a MP3 player and GSM decoder, in which the respective specification contains several tasks ($\mathcal{T} = MP3 \cup GSM$) with precedence relations. More specifically, regarding the MP3 player, the tasks are: MP3=$\{\tau_{ScaleFactor}$ = (0, 0, 407757, 200, 200), $\tau_{HuffmanDecode}$ = (0, 0, 752598, 200, 200), $\tau_{DequantizeSample}$ = (0, 0, 1823756, 200, 200), and $\tau_{SubbandSynthesis}$ = (0, 0, 1065185, 200, 200)$\}$. For the GSM decoder, the following tasks are taken into account: $\tau_{RPEDecoding}$ = (0, 0, 366742200, 180000, 180000), $\tau_{LTSynthesisFilter}$ = (0, 0, 827857800, 180000, 180000), $\tau_{STSynthesisFilter}$ = (0, 0, 274359800, 180000, 180000), and $\tau_{PostProcessing}$ = (0, 0, 123840200, 180000, 180000). Additionally, the intertask relations are: $\tau_{ScaleFactor}$ *precedes* $\tau_{HuffmanDecode}$; $\tau_{HuffmanDecode}$ *precedes* $\tau_{DequantizeSample}$; $\tau_{DequantizeSample}$ *precedes* $\tau_{SubbandSynthesis}$; $\tau_{RPEDecoding}$ *precedes* $\tau_{LTSynthesisFilter}$; $\tau_{LTSynthesisFilter}$ *precedes* $\tau_{STSynthesisFilter}$; and $\tau_{STSynthesisFilter}$ *precedes* $\tau_{PostProcessing}$.

For this case study, the task time unit is $100ms$ and the Intel XScale PXA250 is adopted as the processor model. Thus, the voltage/frequency levels are $vff = \{$(0.9357V, 132.7MHz),(1.1V, 199.1MHz),(1.21V, 298.7MHz),(1.43V, 398.2MHz)$\}$ and the respective energy consumptions are represented by $vef = \{$(0.9357V, $0.49nJ$/cycle), (1.1V, $0.66nJ$/cycle), (1.21V, $0.74nJ$/cycle),(1.43V, $0.91nJ$/cycle)$\}$. Table 9.1 shows the energy savings obtained using the proposed approach.

### 9.1.7 Thermal Printer

Case study 8 is a thermal printer, which generates a printed image by heating some areas of a special paper. The respective specification are composed of time-critical tasks, in the sense that if a constraint is violated, the equipment can be damaged (or the final document can be malformed).

The thermal printer takes into account the following preemptable tasks: (i) sendDataHead, $T_{SDH}$ = (0,0,17400,50,250, codeTSDH{...}), (ii) advanceMotor, $T_{AM}$ =(0,39,180,50,50, codeTAM{...}); (iii) setStrobeOn, $T_{SSO}$ =(0,50,250,70,250, codeTSSO{...}); (iv) setStrobeOff, $T_{SSF}$ =(0,219,250,250,250, codeTSSF{...}); and (v) dotlineGenerator, $T_D$ =(0,0,18800,125,125, codeTD{...}). In addition, the specification contains the following intertask relations: $T_D$ *excludes* $T_{SDH}$, $T_{SDH}$ *excludes* $T_D$, and $T_{SSO}$ *precedes* $T_{SSF}$. The hardware platform described in Section 9.2.1 is taken into account, and the dispatcher worst-case execution time is 100 $\mu s$ at 1.95V/60MHz (a customized version has been generated for this case study). Besides, the task time unit is 50 $\mu s$. Table 9.1 presents the results.

### 9.1.8 Analytical Comments

The presented case studies demonstrate that the pre-runtime scheduling algorithm has provided meaningful results (Table 9.1), since it has significantly reduced the number of

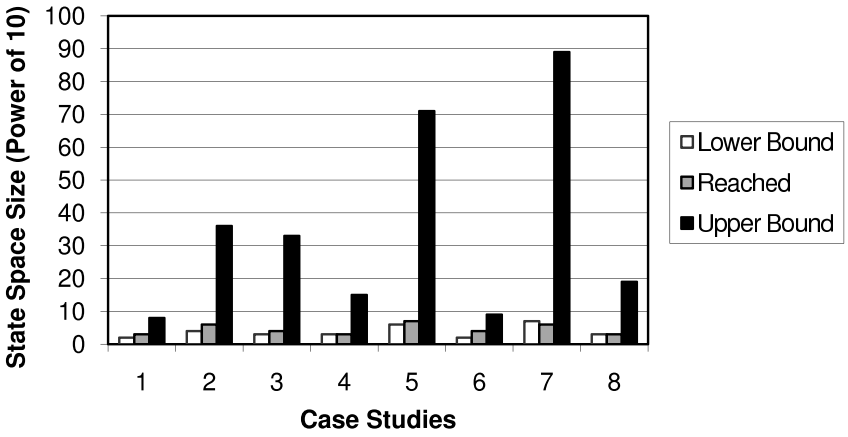


**Figure 9.2** Case studies: state space

visited states, found feasible schedules in which runtime counterparts may not, and, also, allowed energy saving by the adoption of DVS.

For a better visualization, Figure 9.2 depicts the lower and upper bounds for each case study, regarding the state space size (see Section 8.1.4). Due to the large difference of sizes, this figure represents all values as a power of ten (i.e., the figure depicts the exponents). As the reader should note, the number of states visited by the scheduling algorithm is several orders of magnitude less than the whole state space in most experiments. Besides, the algorithm tends to visit a minimal number of states, mainly, due to the techniques adopted to reduce the state space (e.g., preprocessing).

Taking account execution time, assume that, in a loose manner, $50\mu s$ is the average time to reach a single state. Such time has been obtained considering the execution time of each experiment as well as the number of visited states. Moreover, as explained in Section 8.1.4, the time to visit the whole state space is related to the square of the space size. Considering previous information, Figure 9.3 depicts the lower and upper bounds in seconds for the algorithm execution in each case study. Again, the values are presented as powers of ten. At first sight, the reader should observe that Figure 9.3 resembles Figure 9.2. Indeed, the execution time is associated with the number of visit states, and the time is also benefited by the techniques adopted to control the state space.

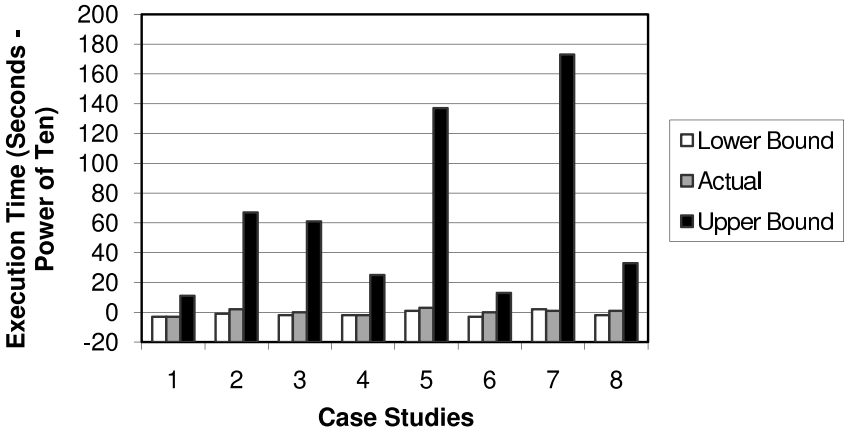


**Figure 9.3** Case studies: execution time

In the context of energy consumption, Figure 9.4 depicts a comparison between feasible schedules generated by the proposed approach (with DVS), alternative schedules without DVS, and optimal solutions provided by Yao's Algorithm (LPEDF). In this figure, the energy consumption is normalized considering the highest value in each case study. Analyzing the results, the proposed approach generated feasible schedules that consume only 30% more energy (in average) than Yao's optimal solution and, in some experiments, provided the same consumption. Besides, it is important to emphasize that Yao's method does not consider precedence and exclusion relations. In other words, the values provided in column *lpedf* (Table 9.1) assume a set of independent tasks, thus, the respective schedules are not feasible for most case studies. Nevertheless, those values still serve as an interesting parameter for comparison purposes.

### 9.1.9 Scalability

To provide a better visualization of the scheduling algorithm scalability, this work extended case study 2 to incrementally accommodate two new tasks with exclusion relation between them. Figure 9.5 and 9.6 depict the results related to the state space generation and execution time, respectively. Although the state spaces grow exponentially with the number of tasks (Section 8.1.4), the amount of reached states (dashed bar) is signifi-

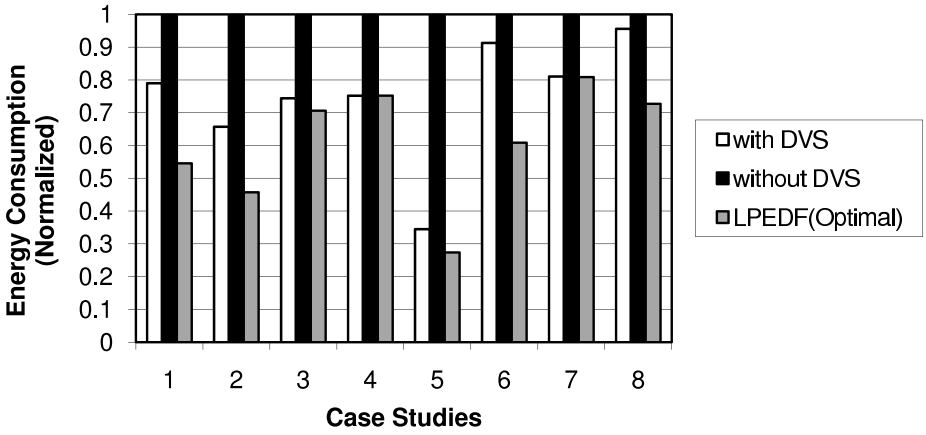


**Figure 9.4** Case studies: energy consumption

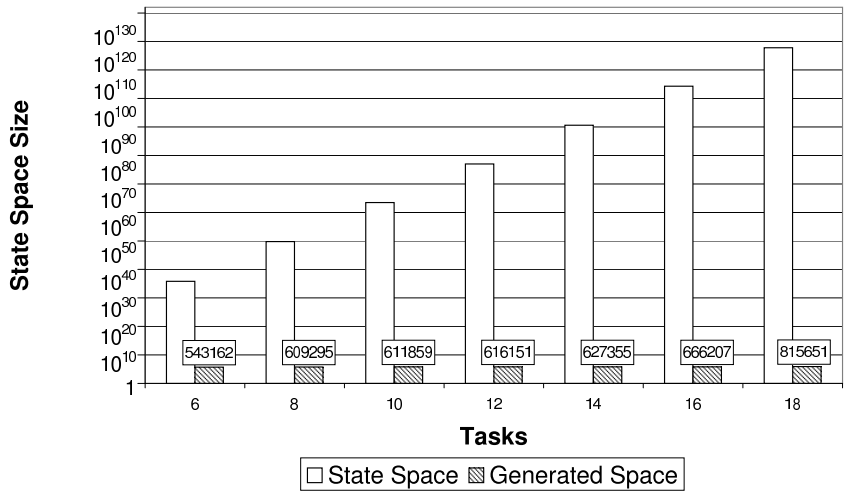


**Figure 9.5** Scalability: state Spaces

cantly reduced due to state space reduction techniques. Nevertheless, the execution time considerably increases, mainly, because of the tagging scheme adopted to keep track of the visited states. However, in the proposed work, the major issue is the state space explosion, which is substantially mitigated as described before. Besides, although the algorithm execution time increases (e.g., in consequence of the tagging scheme), observe that, for a system with a state space size of approximately $10^{120}$ states, the algorithm found a feasible schedule in less than one hour.

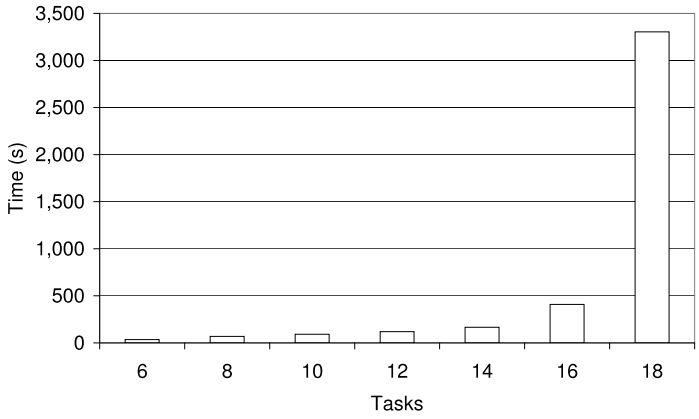


**Figure 9.6** Scalability: execution times

## 9.2 SOFTWARE SYNTHESIS AND RUNTIME SCHEDULER

This section presents results concerning the proposed software synthesis as well as the runtime scheduler. Firstly, the adopted hardware platform is described followed by an explanation about the validation activity.

### 9.2.1 Hardware Platform

This work adopts a hardware platform based on [64], which utilizes a Philips LPC2106 processor [72], an 32-bit microcontroller with ARM7 core (see Figure 9.7). This platform has been devised to facilitate the instrumentation of hardware components for the measurement and validation activities. In this work, the voltage levels have been adjusted, since the original levels provided in [64] caused some instability for the adopted experiments. More specifically, the voltage/frequency levels have been tuned considering the minimum level described in [64] and the maximum level detailed in the datasheet [72], leading to the following relation: $vff = \{$(1.02V, 10MHz), (1.21V, 20MHz), (1.39V,30MHz, (1.58V,40MHz), (1.76V, 50MHz),(1.95V, 60MHz)$\}$.

Although a specific DVS platform is considered, the proposed software synthesis method can be utilized for other hardware platforms, for instance, Intel XScale.

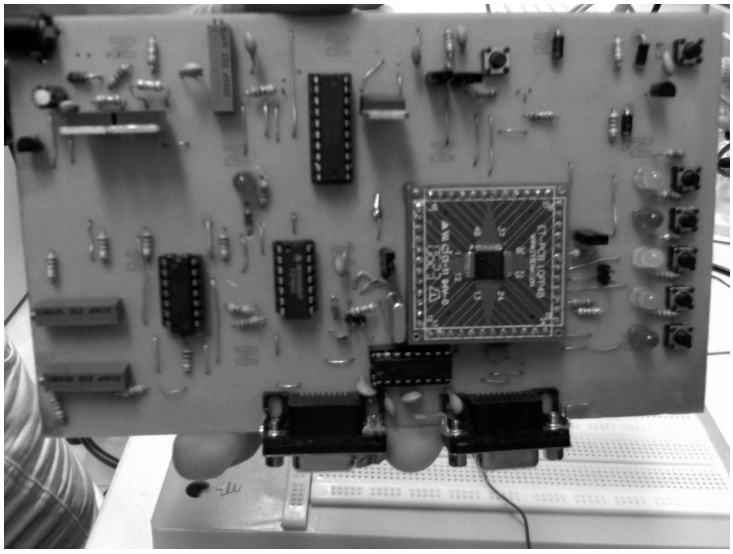

**Figure 9.7** DVS platform

### 9.2.2 Validation

In this work, the validation activity concerns checking the schedule energy consumption and tasks' timing constraints related with a generated code as well as with the execution of the runtime scheduler.

Initially, the generated code is adjusted, such that tasks' WCEC always occur and the dispatcher points out the schedule periodic executions. Such an approach is required to allow the comparison between the estimated value for the pre-runtime schedule (worst-case assumption) and the value measured in the real hardware. To provide a more accurate result, the energy consumption related to the idle times of the pre-runtime schedule is also taken into account in the estimated value. Regarding tasks' timing constraints, all dispatcher executions are captured in order to verify whether each task executes in accordance with the feasible schedule.

The validation scheme is presented in Figure 9.8. In the same way as the measuring approach, a PC is connected to an oscilloscope (Agilent DSO03202A) for capturing the CPU current draw. However, to catch the voltage scaling, one oscilloscope channel, more specifically, channel 1, is connected to the supply voltage pin (Vdd). The voltage scaling is not only adopted to calculate the energy consumption in consequence of DVS technology, but also to verify the schedule and dispatcher executions.

For indicating the schedule periodic execution, the dispatcher adjusts the CPU supply voltage to 2.55V before executing the first task instance and keep it for a small amount of time (sufficient to be detected by the oscilloscope). Using the interval between two pulses of 2.55V, it is possible to calculate the schedule energy consumption as well as the execution time (which must be equal to the LCM of the feasible schedule). Although 2.55V surpasses the adopted platform voltage/frequency range (1.02V-1.95V), a protection circuitry was developed to avoid damages. Basically, the protection circuitry detects a prohibited level, and maintains the CPU supply voltage in a secure one until

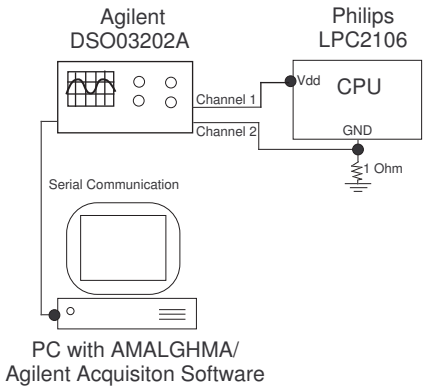


**Figure 9.8** Validation scheme

the unsupported level decreases to the allowed range.

To confirm that all tasks are meeting the respective timing constraints, the interval between two consecutive dispatcher executions is verified, such that the relative start time (or return from preemption) of each task can be calculated. Since this work assumes the dispatcher execution at maximum voltage/frequency level to impact timing constraints as minimum as possible, the oscilloscope timing diagram provides the required data to track each task execution.

Concerning the runtime scheduler, similar approach is adopted, except for the adjustments to force each task WCEC. Besides, since the runtime scheduler also executes in the maximum voltage/frequency available(1.95V/60MHz), the technique described for the dispatcher can also be adopted for the scheduler to track each task execution.

Moreover, AMALGHMA tool [121] also comprises functionalities that permit the measurement of schedules' time and energy consumption using statistical techniques. In relation to tasks' timing constraints, this work has adopted Agilent Scope Connect software [125] to verify the dispatcher executions. Although such tool does not provide a very fine-grained resolution, it gives interesting insights about a software timing behaviour. Detailed examples are presented in next section.

### 9.2.3 Software Synthesis

In order to demonstrate the software synthesis method in details, three experiments are adopted, namely, example 2, pulse oximeter, and thermal printer. These experiments were presented previously in order to provide quantitative results for the scheduling algorithm. Therefore, the values depicted in Table 9.2 do not consider the energy consumption in consequence of the idle times. However, for the following experiments, the idle times are taken into account, such that a closer comparison with the energy consumption values measured in the hardware platform can be performed. Besides, the pulse oximeter is adopted to demonstrate step-by-step the sequence of activities related to the proposed method.

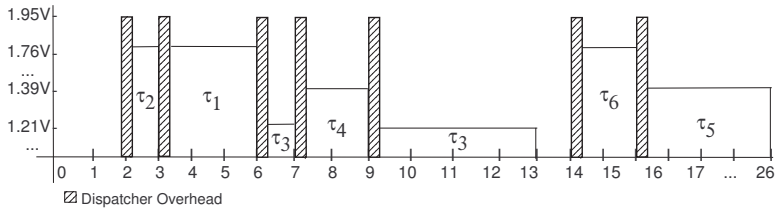


**Figure 9.9** Case study 2: feasible schedule

**Example 2** Examples 2 is adopted to validate the proposed method in the DVS platform described in Section 9.2.1. Taking into account the previous specification, 7 schedules (Figure 9.9) have been generated to compare the estimated energy consumptions with the respective values measured in the real hardware. More specifically, the difference lies on the energy consumption per clock cycle adopted for the voltage/frequency levels in each schedule. Although the tasks' timing constraints are the same, the code of each task has been adjusted to consider different functionalities, such as arithmetic division and logical operations, which incur distinct energy consumption values.

**Table 9.2** Energy consumption values

| Function | scheduling. hard. w/ DVS | hard. o/ DVS | |
|---|---|---|---|
| Loop | 737613 | 781361 | $2617 \times 10^3$ |
| Division and Multiplication | 830644 | 865065 | $2894 \times 10^3$ |
| Multiplication | 811771 | 831960 | $2884 \times 10^3$ |
| Division | 820540 | 856791 | $2823 \times 10^3$ |
| Sum | 820299 | 810230 | $2795 \times 10^3$ |
| Subtraction | 829049 | 820552 | $2757 \times 10^3$ |
| Logical Operations | 890922 | 900684 | $2771 \times 10^3$ |

```
void codeT1() {...} void codeT2() {...}
void codeT3() {...} void codeT4() {...}
void codeT5() {...} void codeT6() {...}
#define SCHEDULE_SIZE 7
struct SchItem sch[SCHEDULE_SIZE] =
{
  {2, INSTANCE, 2, 1.76V/50MHz,(int *)codeT2},
  {3, INSTANCE, 1, 1.76V/50MHz,(int *)codeT1},
  {6, INSTANCE, 3, 1.21V/20MHz,(int *)codeT3},
  {7, INSTANCE, 4, 1.39V/30MHz,(int *)codeT4},
  {9, RETURN, 3, 1.21V/20MHz,(int *)codeT3},
  {14, INSTANCE, 6, 1.76V/50MHz, (int *)codeT6},
  {16, INSTANCE, 5, 1.39V/30MHz, (int *)codeT5},
};
```

**Figure 9.10** Case study 2: generated code

Disregarding the tasks' internal code, Figure 9.10 depicts the code generated for the schedules. In relation to the estimated energy consumptions and the values measured in the real hardware, Table 9.2 depicts the results. In this table, *function* represents the tasks' functionality; *scheduling* depicts an estimation of the energy consumption using

the proposed scheduling mechanism (in nanojoules); *hard. w/ DVS* is the mean value of the code energy consumption measured in the hardware platform adopting DVS (in nanojoules); and *hard. o/ DVS* depicts the mean value of the code energy consumption in the hardware platform without DVS (in nanojoules). It is worth stating that tasks' WCEC always took place during the measuring process in the hardware platform. The t-paired test [110] was conducted on data depicted on Table 9.2, considering a confidence degree of 95%. Since $0 \in [-38051.85, 2107.561]$, there is no evidence to reject the hypothesis of equivalence between the model and the system.

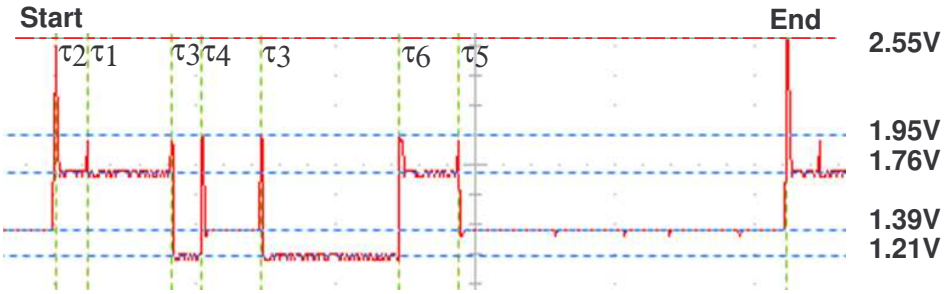


**Figure 9.11** Case study 2: oscilloscope timing diagram

Concerning tasks' timing constraints, Figure 9.11 depicts the code execution captured by the oscilloscope. In this figure, each task instance can be tracked using the first pulse of 2.55V as reference and taking into account the time instants when the dispatcher was running. More specifically, $\tau_2$ started at 0, $\tau_1$ at 1.02 and $\tau_3$ at 4.01. $\tau_4$ preempted $\tau_3$ at 5.01 and $\tau_3$ returned at 7.01. Lastly, $\tau_6$ started at 12 and $\tau_5$ at 14. The reader should note that the idle time (from 0 to 2) at beginning of the feasible schedule is located at the end of oscilloscope timing diagram, since the pulse of 2.55V is generated immediately before the first task instance. Thus, for each value, 2 time units need to be added in order to obtain the correct time instant (e.g., $\tau_5$ started at 16). Besides, some values are not integer numbers (e.g. 1.02) due to the oscilloscope software resolution (as stated in Chapter 5). According to the collected data, all tasks met the timing constraints and the energy consumption was reduced.

**Pulse Oximeter** In order to demonstrate the utilization of the proposed software synthesis method in a real-word application, the pulse oximeter case study is adopted (Section 9.1.5). For this case study, the activities of the proposed method are explained below:

- *Measurement.* Each task and the dispatcher are measured to obtain the respective WCECs as well as the hardware characterization is performed. Remember that the task codes are adjusted to force the worst-case situation in terms of execution time;
- *Specification.* The non-functional specification is composed of three non-preemptable tasks ($T_{exc}$, $T_{acq}$ and $T_{ctr}$) and the associated precedence relations (see Section 9.1.5).

Besides, the hardware platform is described considering the voltage/frequency levels and the respective energy consumption per clock cycle obtained in previous activity;

- *Modeling.* From the non-functional specification, the TPNE model is generated using the proposed building blocks and composition rules. Before performing the scheduling activity, the designer may choose to carry out property analysis/verification, such as model checking;

- *Scheduling.* Adopting the TPNE model, the scheduling activity generates an ordering of task executions at design-time, such that timing and energy constraints are met. Table 9.1 provides results related to the scheduling algorithm;

```
void codeExc() {...} void codeAcq() {...}
void codeCtr() {...}
#define SCHEDULE_SIZE 10
struct SchItem sch[SCHEDULE_SIZE] =
{
  {0, INSTANCE, 1, 1.02V/10MHz,(int *)codeTExc},
  {1, INSTANCE, 2, 1.02V/10MHz,(int *)codeTAcq},
  {2, INSTANCE, 1, 1.02V/10MHz,(int *)codeTExc},
  {4, INSTANCE, 1, 1.02V/10MHz,(int *)codeTExc},
  {6, RETURN, 1, 1.02V/10MHz,(int *)codeTExc},
  {8, INSTANCE, 1, 1.02V/10MHz, (int *)codeTExc},
  {10, INSTANCE, 1, 1.02V/10MHz, (int *)codeTExc},
  {12, INSTANCE, 1, 1.02V/10MHz, (int *)codeTExc},
  {13, INSTANCE, 3, 1.95V/60MHz, (int *)codeTCtr},
  {15, INSTANCE, 1, 1.02V/10MHz, (int *)codeTExc}
};
```

**Figure 9.12** Pulse-oximeter code

- *Code Generation.* Since a feasible schedule has been found, a customized predictable code is automatically generated. Figure 9.12 depicts such a code, in which timing and energy constraints are expected to be met;

- *Validation.* At last, the embedded software is validated. Figure 9.13 shows the respective behaviour captured by the oscilloscope, in which indicates that all timing constraints were satisfied. Taking into account the idle times of the feasible firing schedule, the energy consumption estimate (309541 $nJ$) is only 17% higher than the real measured value. The difference is mainly related to the unnecessity of switching the voltage/frequency level for executing task $T_{ctr}$ and the dispatcher before the last task instance, since the level of 1.95V/60MHz is already established (ellipse at Figure 9.13).

**Thermal Printer** Regarding another real-world application, the thermal printer is also taken into account in order to demonstrate the feasibility of the software synthesis method. Figure 9.14 shows the generated code for the thermal printer and Figure 9.15 depicts the respective behaviour captured by the oscilloscope. Concerning energy consumption, the estimated value (171486 nJ) is 27% higher than the value measured in the

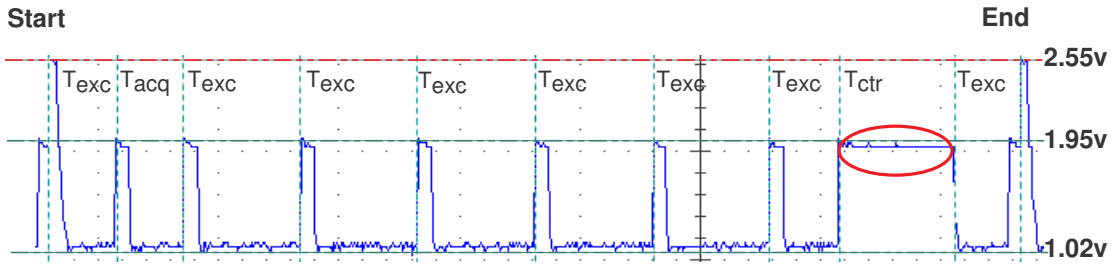


**Figure 9.13** Pulse oximeter: oscilloscope timing diagram

```
void codeTSDH() {...} void codeTAD() {...}
void codeTSSO() {...} void codeTD() {...}
void codeTSSF() {...}
#define SCHEDULE_SIZE 12
struct SchItem sch[SCHEDULE_SIZE] =
{
  {0, INSTANCE, 1, 1.95V/60MHz,(int *)codeTSDH},
  {39, INSTANCE, 2, 1.21V/20MHz,(int *)codeTAM},
  {50, INSTANCE, 1, 1.21V/20MHz,(int *)codeTSSO},
  {61, INSTANCE, 1, 1.95V/60MHz,(int *)codeTD},
  {89, RETURN, 1, 1.21V/20MHz,(int *)codeTAM},
  {100, RETURN, 1, 1.95V/60MHz, (int *)codeTD},
  {139, INSTANCE, 1, 1.39V/30MHz, (int *)codeTAM},
  {150, INSTANCE, 1, 1.58V/40MHz, (int *)codeTD},
  {189, INSTANCE, 3, 1.39V/30MHz, (int *)codeTAM},
  {200, RETURN, 1, 1.58V/40MHz, (int *)codeTD},
  {228, INSTANCE, 1, 1.39V/30MHz, (int *)codeTSSF},
  {239, INSTANCE, 1, 1.39V/30MHz, (int *)codeTAM}
};
```

**Figure 9.14** Thermal printer: code

hardware platform, mainly, due to the unnecessity of switching the voltage/frequency level for executing some tasks and the dispatcher (see ellipses at Figure 9.15). Nevertheless, the energy constraint (i.e, the upper bound) is not violated during system execution.

Previous experiments demonstrated the feasibility of the software synthesis method described in this work, in the sense that, from a system specification, customized code was generated with reduced energy consumption as well as satisfying all timing constraints.

### 9.2.4 Runtime Scheduler

An experiment based on case study 2 (Section 9.1.1) has been adopted to highlight some features related to the proposed runtime scheduler. In this experiment, 7 concurrent tasks with precedence and exclusion relations were taken into account. More specifically, considering the feasible schedule generated for the original specification (Figure 9.9), a new task ($\tau_7$) was added to fill the idle times (Figure 9.16). In order to evaluate the runtime scheduler, tasks $\tau_1$,$\tau_2$,$\tau_3$ and $\tau_4$ varied the execution cycles at the same time from 100% to 10% in relation to each task WCEC. Figure 9.17 depicts quantitative data, in which the energy consumption values are normalized.

Four techniques are compared. The first one adopts only the dispatcher (*only dispatcher*), in the sense that no action is performed when slack times occur. In other words,

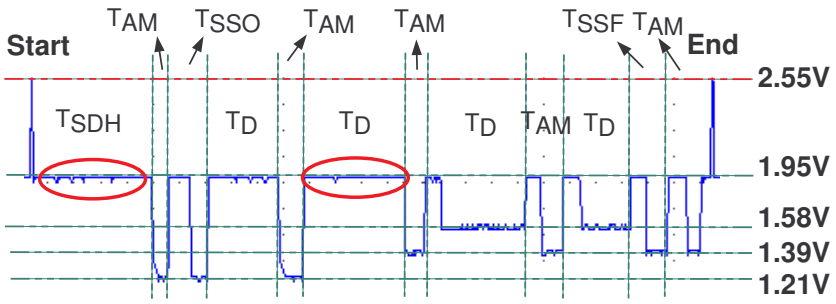


**Figure 9.15** Thermal printer: oscilloscope timing diagram

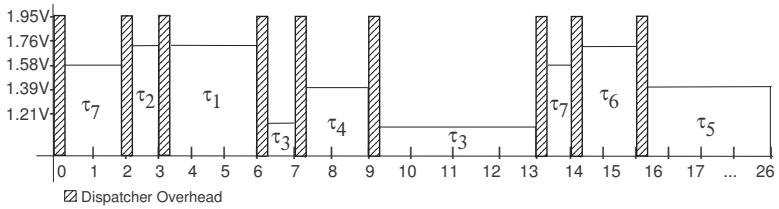


**Figure 9.16** Runtime scheduler: feasible schedule

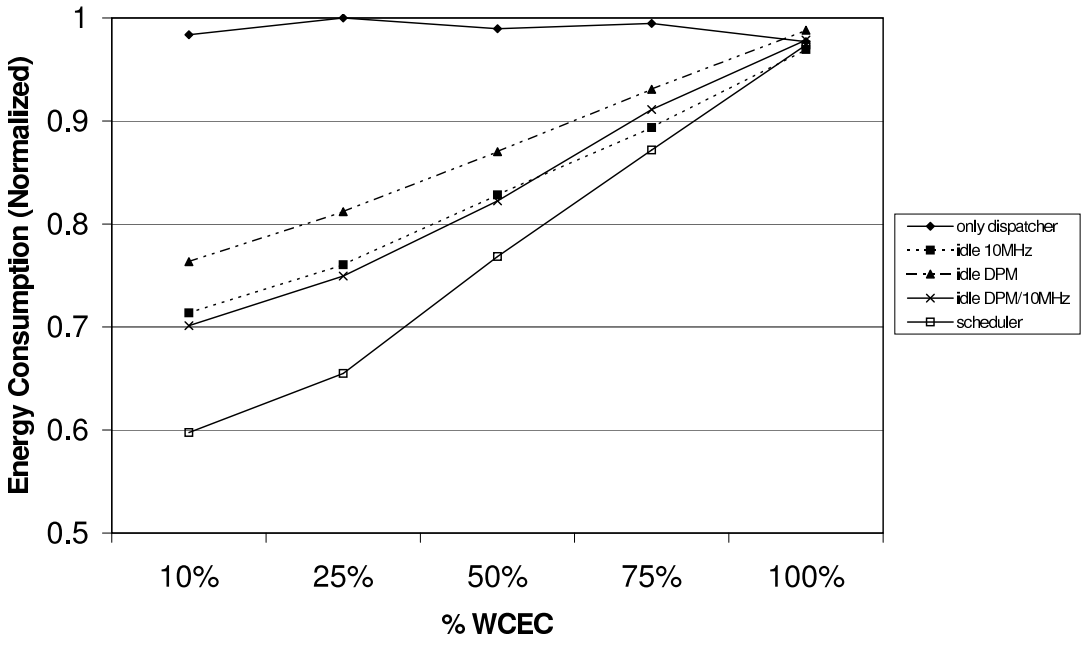


**Figure 9.17** Comparison runtime scheduler and other approaches

the pre-runtime schedule is followed without modification. *idle 10MHz* reduces the voltage/frequency level to the minimum level available during the slack times, whereas *idle DPM* turns off the microcontroller. The technique *idle DPM/10MHz* reduces the voltage/frequency level to the minimum level as well as it turns off the microcontroller after the voltage/frequency switching. *scheduler* is the adoption of the proposed lightweight scheduler during slack times (see Chapter 8).

Comparing the runtime scheduler to an approach based only on the dispatcher, the results show a significant reduction in the energy consumption with the former (more than 40% in one scenario). The dispatcher only follows the schedule table and performs no action to improve the consumption due to changes in the task executions. Additional savings can be obtained adjusting the dispatcher to reduce the voltage/frequency level to 1.02V/10MHz in the idle periods. Nevertheless, the runtime scheduler still provides better results. Instead of switching to the minimum voltage/frequency level, consider the dispatcher turns off the microcontroller in the idle periods (DPM). Reducing to the minimum level seems more efficient, but the runtime scheduler provides greater savings. Besides, mixing DPM and the minimum voltage/frequency level slightly improves the consumption in some situations, regarding the idle periods. Nevertheless, for one scenario, the runtime scheduler saved more than 10% in relation to previous approach. The reader should note that when the execution cycles of each task approximate the WCEC, the impact of each technique is diminished, since the amount of slack times is decreased.

## 9.3 SUMMARY

This chapter presented the case studies adopted to evaluate the software synthesis method as well as the pre-runtime and runtime scheduling algorithms. Regarding the pre-runtime scheduling, some case studies demonstrated the practicability of the proposed algorithm, in the sense that the feasible schedules were found in situations that runtime counterparts may fail, tiny subsets of the corresponding state spaces were visited, and energy consumption was reduced due to the adoption of DVS technology. Concerning the software synthesis, experiments corroborated the effectiveness of the proposed method, since predictable codes were generated satisfying intertask relations, timing and energy constraints. In the context of the runtime scheduler, some scenarios were considered in order to provide a comparison between the scheduler and other approaches. The runtime scheduler provided better results in all situations when the execution cycles of some tasks varied below the respective WCEC.

CHAPTER 10

# CONCLUSIONS

Over the last years, energy consumption has been a great concern in embedded systems' design due to many factors, for instance: (i) mobility issues; (ii) environmental problems and (iii) energy costs. As software account for the majority of embedded system functionalities, attention has been devoted to improve energy utilization at application and behavioral level. Most methods available in the literature adopt dynamic power management techniques (DPM), more specifically, dynamic voltage scaling (DVS), cooperatively with specialized operating systems to control energy consumption during system runtime. However, DVS technology must be adopted with caution in hard real-time systems in order not to violate stringent timing constraints. Although several methods have been developed to deal with DPM techniques in time-critical systems, many issues are still open as described below:

- Most works neglect intertask relations, for instance, precedence and exclusion relations, such that the respective methods may produce infeasible results when considering real-world applications;
- Overheads, such as dispatcher and frequency/voltage switchings, are generally disregarded. Indeed, if overheads are neglected, tasks' constraints may be affected and even the benefits acquired with such energy saving methods may be significantly reduced;
- In general, formal models are not taken into account by most energy reduction methods which may hinder the verification and analysis of quantitative as well as qualitative properties;
- Several techniques are implemented as operating system services, which can incur considerable overheads during system execution, affecting timing as well as energy constraints. Usually, operating systems provide several services for an application, but, in many situations, not all services are utilized.

In order to tackle the issues presented previously, this work described a software synthesis method for hard real-time embedded systems with energy constraints. The proposed method contemplates a set of activities as well as integrated tools, such that, from a system specification, a customized code is generated satisfying timing as well as energy constraints. Additionally, the method is based on a formal model, namely, time Petri net, which allows the verification and analysis of several properties. Among the techniques for reducing energy consumption, dynamic voltage scaling (DVS) has been adopted, which provides interesting results comparing to other DPM techniques.

The following sections describe the principal contributions of the proposed method as well as the future works.

## 10.1 CONTRIBUTIONS

This thesis presented a novel approach for synthesizing customized embedded software in the context of hard real-time systems with energy constraints. The method is an important part of MEMBROS methodology, which is also composed of other methods, such as requirement validation and performance evaluation. Nevertheless, the reader should note that the software synthesis method is the contribution of this thesis. For a better visualization, the contributions are detailed below according to each activity:

- **Measurement.** Statistical techniques have been adopted to estimate the energy consumption in each voltage/frequency level of a DVS platform as well as the WCEC of each software component (e.g., dispatcher). In the proposed work, two techniques are available to perform the measurement activity, more specifically, adopting bootstrap or a parametric method. The former assumes no previous information about the population, whereas the latter allows the designer to indicate a relative precision. Besides, the validation activity also takes into account these statistical techniques, which allow to quantify the system behavior in the context of energy consumption and execution time. Moreover, a tool, namely, AMALGHMA, has been implemented to support and automate the measuring process. The tool is also a contribution;

- **Modeling.** This work proposes a formal model based on time Petri nets in order to represent real-time systems with intertask relations, overheads and energy constraints. A bottom-up approach is adopted, in which composition rules are considered for combining building block models. By construction, the generated models are assured to be bounded and conservative, in the sense that the respective state spaces are finite. Besides, some verifications based on CTL (Computational Tree Logic) have been taken into account to check the availability of undesirable conditions in the models (which were not detected). Moreover, DENTES tool has been devised to automate the modeling and scheduling activities, in the sense that from a non-functional specification, the TPNE model is generated, and the scheduling activity can be carried out. DENTES tool is also another contribution;

- **Scheduling.** The adopted scheduling approach is a pre-runtime scheduling, which performs a depth-first search on the state space of a Petri net model in order to find a feasible schedule. Although the state space exploration suffers from the state space size explosion, several techniques are proposed to tackle the space size. Additionally, since the pre-runtime scheduling cannot benefit from the slack times that occur during system runtime, a lightweight runtime scheduler is also proposed to improve energy consumption in such situations. This mixture of pre-runtime and runtime scheduling can be visualized as a hybrid scheduling approach;

- **Code Generation.** From a feasible schedule, a predictable customized code is generated satisfying the specified constraints. More specifically, the TLTS is traversed to detect the times when each task will execute, obtaining as a result a schedule table. From such table, the dispatcher is customized in order to provide

only the required services at system runtime. Since the schedule is already defined, the generated code incurs minimal overheads during system execution.

As far as the authors are aware, there is no similar method for time-critical systems that, from a system specification, customized code is generated meeting intertask relations, timing and energy constraints as well as adopting dynamic voltage scaling.

## 10.2 FUTURE WORKS

Although this thesis tackles some issues regarding embedded software development for energy-constrained hard real-time systems, there are many possibilities to improve and extend the current work. The following items summarize some possibilities:

- This work has focused on single-processor architectures with DVS technology. Nevertheless, there is an increasing requirement for multiprocessor and distributed architectures in order to deal with more complex specifications that may be infeasible to implement in a single processor. Considering the Petri net model, an extension may consider a new building block for representing inter-processor communications and the renaming function may be adopted to indicate the CPU associated with each task structure block;

- The effectiveness of the runtime scheduler is somewhat limited by the arrangement of each task in the feasible schedule. In order to improve the arrangement, the pre-runtime scheduling algorithm may be modified to organize the tasks, which are supposed to execute less than the WCEC (most of the time), in such a way that the scheduler can better stretch the executions of other tasks;

- In the past few years, some approaches have come to light to deal with fault-tolerance in real-time systems that adopt DVS for reducing energy consumption. Indeed, some embedded systems are deployed in hostile environments, in which repair and maintenance are not easily performed. Perhaps, an extension of the proposed runtime scheduler may consider fault-tolerant techniques to cope with unexpected situations during system runtime;

- Reduction techniques [10] are available for Petri nets in order to facilitate property analysis in large models. Although the proposed modeling approach assures the generated models have important properties by construction, those reduction techniques may be an interesting approach for providing insights about a specification schedulability.

## 10.3 FINAL REMARKS

This work proposed a software synthesis method for hard real-time systems with energy constraints, taking into account some issues not considered in similar works (e.g., formal models). Despite the results presented in this thesis, the research area related to software construction for energy-constrained time-critical systems has other open issues, which lay down several possibilities for further development of new techniques.

# BIBLIOGRAPHY


[1] N. Velázquez. Impact of rising energy costs on small business. Technical report, House Small Business Committee. United States of House of Representatives, August 2006.

[2] G. Yeap. *Practical low power digital VLSI design.* Kluwer Academic Publishers, 1998.

[3] K. Mihic and et al. Reliability and power management of integrated systems. In *Euromicro Symposium on Digital System Design (DSD'04).* IEEE Society, 2004.

[4] T. Lee and P. Hsiung. Embedded software synthesis and prototyping. *IEEE Transactions on Consumer Electronics*, 50:386–392, 2004.

[5] M. Pedram. Power optimization and management in embedded systems. In *Asia and South Pacific Design Automation Conference (ASP-DAC'01)*, pages 239–244, 2001.

[6] J. Xu and D. Parnas. Priority scheduling versus pre-run-time scheduling. *Real-Time Systems*, 18:7–23, January 2000.

[7] X. Hu, G. Quan, and B. Mochocki. A realistic variable voltage scheduling model for real-time applications. *International Conference on Computer-Aided Design (ICCAD'02)*, 00:726–731, 2002.

[8] A. Singhal. Real time systems: A survey. Technical report, Computer Science Department. University of Rochester, December 1996.

[9] N. Min-allah and et al. Towards dynamic voltage scaling in real-time systems- a survey. *International Journal of Computer Sciences and Engineering Systems*, 1(2), 2007.

[10] T. Murata. Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580, April 1989.

[11] J. Xu. On inspection and verification of software with timing requirements. *IEEE Transactions on Software Engineering*, 29(8):705–720, August 2003.

[12] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low Power Electronics and Design (ISLPED'98)*, pages 197–202, 1998.

[13] P. Merlin and D. J. Faber. Recoverability of communication protocols: Implicatons of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, Sept. 1976.

[14] N. Jha. Low-power system scheduling, synthesis and displays. *IEE Proceedings Computers and Digital Techniques*, 152:344–352, 2005.

[15] M. Cornero, F. Thoen, G. Goossens, and F. Curatelli. Software synthesis for real-time information processing systems. *Code Generation for Embedded Processors*, pages 260–279, 1995.

[16] R. Barreto. *A Time Petri Net-Based Methodology for Embedded Hard Real-Time Systems Software Synthesis*. PhD Thesis, Centro de Informática. Universidade Federal de Pernambuco, April 2005.

[17] E. Tavares and et al. Hard real-time tasks' scheduling considering voltage scaling, precedence and exclusion relations. *Information Processing Letters*, 108(2):50–59, 2008.

[18] E. Tavares and et al. Modeling hard real-time systems considering inter-task relations, dynamic voltage scaling and overheads. *Microprocessor and Microsystems*, 32(8):460–473, 2008.

[19] E. Tavares and et al. A time petri net-based approach for hard real-time systems scheduling considering dynamic voltage scaling, overheads, precedence and exclusion relations. *Symposium on Integrated Circuits and Systems Design (SBCCI'07)*, 2007.

[20] E. Tavares, P. Maciel, B. Silva, M. Oliveira Jr., R. Rodrigues, and R. Marques. Dynamic voltage scaling in hard real-time systems considering precedence and exclusion relations. *International Conference on Systems, Man, and Cybernetics (SMC'07)*, 2007.

[21] E. Tavares, P. Maciel, B. Silva, M. Oliveira Jr., and R.Rodrigues. Modeling and scheduling hard real-time biomedical systems with timing and energy constraints. *Electronic Letters*, (19), September 2007.

[22] E. Tavares and et al. A hybrid dvs scheduling approach for hard real-time systems. *International Conference on Systems, Man, and Cybernetics (SMC'09)*, 2009.

[23] E. Tavares and et al. Software synthesis for hard real-time embedded systems with energy constraints. *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'08)*, 2008.

[24] E. Tavares and et al. An environment for measuring and scheduling time-critical embedded systems with energy constraints. *International Conferences on Software Engineering and Formal Methods (SEFM'08)*, 2008.

[25] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. *36th Annual Symposium on Foundations of Computer Science*, 00:374, 1995.

[26] W. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. *ACM Transactions in Embedded Computing Systems (TECS)*, 4(1):211–230, 2005.

[27] B. Mochocki, X. Sharon Hu, and G. Quan. A unified approach to variable voltage scheduling for nonideal dvs processors. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(9):1370–1377, 2004.

[28] G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proceedings of the 38th conference on Design automation*, pages 828–833, 2001.

[29] L. Leung, C. Tsui, and X. Hu. Exploiting dynamic workload variation in low energy preemptive task scheduling. In *Design, Automation and Test in Europe'05*, pages 634–639, 2005.

[30] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Comp.*, 53(5):584–600, 2004.

[31] L. Cortés, P. Eles, and Z. Peng. Quasi-static assignment of voltages and optional cycles for maximizing rewards in real-time systems with energy constraints. In *Design Automation Conference (DAC'05)*, pages 889–894, 2005.

[32] R. Jejurikar and R. Gupta. Energy aware non-preemptive scheduling for hard real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS'05)*.

[33] B. Mochocki and et al. Transition-overhead-aware voltage scheduling for fixed-priority real-time systems. *ACM Transactions on Design Automation of Electronic Systems*, 12:1084–4309, 2007.

[34] A. Andrei and et al. Overhead-conscious voltage selection for dynamic and leakage energy reduction of time-constrained systems. *IEE Proc. Comp. and Digital Tech.*, 152:28–35, 2005.

[35] W. Kim and et al. Preemption-aware dynamic voltage scaling in hard real-time systems. *International Symposium on Low Power Electronics and Design (ISLPED'04)*, pages 393–398, 2004.

[36] Y. Cai et al. Workload-ahead-driven online energy minimization techniques for battery-powered embedded systems with time-constraints. *ACM Trans. Des. Autom. Electron. Syst.*, 12(1):5, 2007.

[37] D. Shin and J. Kim. Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 408–413, 2003.

[38] F. Balarin and et al. Synthesis of software programs for embedded control applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):834–849, June 1999.

[39] F. Balarin and et al. *Hardware-software Co-design of Embedded Systems: the POLIS approach*. Kluwer Academic Publishers, 1997.

[40] L.J. Van Bokhoven, J.P.M. Voeten, and M.C.W. Geilen. Software synthesis for system level design using process execution trees. *25th Euromicro Conference (EUROMICRO '99)*, 1:1463, 1999.

[41] M. Sgroi, L. Lavagno, Y. Watanabe, and A. Sangiovanni-Vincentelli. Synthesis of embedded software using free-choice petri nets. *Design Automation Conference (DAC'99)*, 1999.

[42] P.-A. Hsiung. Formal synthesis and code generation of embedded real-time software. *9th Int. Symp. Hw/Sw Codesign (CODES'01)*, pages 208–213, April 2001.

[43] T. Amnell et al. Code synthesis for timed automata. *Nordic Journal of Computing*, 2003.

[44] A. Nácul and T.Givargis. Synthesis of time-constrained multitasking embedded software. *ACM Transactions on Design Automation of Electronic Systems*, 11(4):822–847, 2006.

[45] W. Wang, A. Raghunathan, G. Lakshminarayana, and N. Jha. Input space adaptive embedded software synthesis. In *ASP-DAC*, pages 711–718. ACM, 2002.

[46] S. Udayanarayanan and C. Chakrabarti. Energy-efficient code generation for dsp56000 family. *International Symposium on Low Power Electronics and Design (ISLPED '00)*, pages 247– 249, 2000.

[47] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. Corner, and E. Berger. Eon: a language and runtime system for perpetual systems. In *SenSys '07: Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 161–174. ACM, 2007.

[48] D. Gajski, F. Vahid, S. Narayan, and J. Gong. *Specification and Design of Embedded Systems*. Prentice-Hall, New Jersey, 1994.

[49] W.M.P. van der Aalst. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype. In *Business Process Trends (BPTrends'05)*, 2005.

[50] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *ACM Journal*, 20(1):46–61, January 1973.

[51] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[52] A. K. Mok. The design of real-time programming systems based on process models. *IEEE Real-Time Systems Symposium*, pages 5–17, 1984.

[53] J. Xu and D. Parnas. On satisfying timing constraints in hard real-time systems. *IEEE Trans. Soft. Engineering*, 19(1):70–84, January 1993.

[54] J. Xu and D. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Soft. Engineering*, 16(3):360–369, March 1990.

[55] G. Fohler. *Flexibility in Statically Scheduled Hard Real-Time Systems*. PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. Vienna, Austria, 1994.

[56] J. Xu and K. Lam. Integrating run-time scheduling and pre-run-time scheduling of real-time processes. In *23rd IFAC/IFIP Workshop on Real-Time Programming*. Shantou, China, june 1998.

[57] W. Wang, A. Mok, and G. Fohler. Pre-scheduling. In *IEEE Transactions on Computers*. IEEE, 2004.

[58] L. Mazzoni. Power-aware design for embedded systems. *Electronics Systems and Software*, 1:12–17, 2003.

[59] L. Benini, A. Bogliolo, and G. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems.*, 2000.

[60] Y. Zhang and K. Chakrabarty. A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2006.

[61] Intel Corporation. The intel xscale microarchitecture. Technical report, 2000.

[62] Advanced Micro Devices. Mobile amd athlon4 processor model 6 cpga data sheet rev:e. Technical report.

[63] M. Fleischmann. Longrun power management: Dynamic power management for crusoe processors. Technical report, Transmeta Corporation.

[64] T. Phatrapornnant and M. Pont. Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling. *IEEE Trans. on Comp.*, 55(2):113–124, 2006.

[65] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 140–148, 2004.

[66] J. M. Rabaey and M. Pedram. *Low Power Design Methodologies*. Kluwer Academic Publishers, 1996.

[67] A. Chandrakasan, S. Sheng, and R. Brodersen. Low-power cmos digital design. *IEEE Journal of Solid-State Circuits*, 27(4):473–484, 1992.

[68] R. Jejurikar and et al. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Design Automation Conference (DAC'04)*, pages 275–280, 2004.

[69] M. Liebelt et al. An energy efficient rate selection algorithm for voltage quantized dynamic voltage scaling. *International Symposium on Systems Synthesis (ISSS '01)*.

[70] D. Shin, J. Kim, and S. Lee. Intra-task voltage scheduling for low-energy, hard real-time applications. *IEEE Design and Test*, 18(2):20–30, 2001.

[71] J. Seo, T. Kim, and K. Chung. Profile-based optimal intra-task voltage scheduling for hard real-time applications. *Design Automation Conference*, 00:87–92, 2004.

[72] Philips Semiconductors. Lpc2104/2105/2106; single-chip 32-bit microcontrollers. data sheet. Technical report.

[73] E. A. Lee. Embedded software. In M. Zelkowitz, editor, *Advances in Computers*, volume 56. 2002.

[74] D. Lanneer, J. Van Praet, A. Kifli, K. Schoofs, W. Geurts, F. Thoen, and G. Goossens. Chess: retargetable code generation for embedded dsp processors. In *Code Generation for Embedded Processors*, pages 85–102, 1994.

[75] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[76] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, 1998.

[77] C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[78] M. Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1996.

[79] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, January 1985.

[80] J. C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, 2005.

[81] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1982.

[82] C. Hoare. *Communicating Sequential Process*. Prentice-Hall, 1985.

[83] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 322–335, 1990.

[84] R. Alur and D. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[85] J. Baeten and J. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[86] J. Baeten and J. Bergstra. Real time process algebra with infinitesimals. page 167, 1995.

[87] J. Baeten and J. Bergstra. Discrete time process algebra. *Formal Aspects of Computing*, 8(2):188–208, 1996.

[88] W. Zuberek. Timed petri nets - definitions, properties and applications. *Microelectronics and Reliability (Special Issue on Petri Nets and Related Graph Models)*, 31:627–644, 1991.

[89] C. A. Petri. *Kommunikation mit Automaten*. PhD Dissertation, Darmstadt University, Germany, 1962.

[90] J. Desel and W. Reisig. Place/transition nets. *Lectures on Petri Nets I: Basic Models, LNCS 1491*, pages 122–173, June 1998.

[91] E. W. Dijsktra. *Hierarchical ordering of sequential processes*, volume 1. Acta Informatzca, 1971.

[92] P. Maciel and et al. *Introdução às Redes de Petri e Aplicações*. X Escola de Computação, 1996.

[93] M. Silva and et al. Linear algebraic and linear programming techniques for the analysis of place or transition net systems. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, pages 309–373. Springer-Verlag, 1998.

[94] A. Perkusich and et al. Putting petri nets to work for controlling flexible manufacturing systems. In *Annual Conference of the IEEE Industrial Electronics Society (IECON'91)*, pages 1631–1636, 1991.

[95] T. Barros. *Uma Técnica de Modelagem por Redes de Petri Voltada a Automação da Manufatura*. Msc Thesis, Universidade Federal de Pernambuco, 1990.

[96] R. Valette and et al. Putting petri nets to work for controlling flexible manufacturing systems. In *IEEE International Symposium on Circuits and Systems*, pages 929–932, 1985.

[97] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzé. A unified high-level Petri net formalism for time-critical systems. *IEEE Trans Soft Engineering*, 17(2):160–172, Feb 1991.

[98] G. Balbo. Introduction to stochastic petri nets. In *European Educational Forum: School on Formal Methods and Performance Analysis*, pages 84–155, 2000.

[99] P. Starke and S. Roch. *INA - Integrated Net Analyzer - Version 2.2*. Humbolt Universität zu Berlin - Institut für Informatik, 1999.

[100] Object Management Group. Omg systems modeling language (omg sysml), v1.0". Technical report, 2007.

[101] Object Management Group. A uml profile for marte, beta 1. Technical report, 2007.

[102] K. Jensen. Coloured petri nets and the invariant method. *Theoretical Computer Science*, 1:317–336, 1981.

[103] E. Carneiro and et al. Mapping sysml state machine diagram to time petri net for analysis and verification of embedded real-time systems with energy constraints. In *ENICS*, 2008.

[104] G. Callou and et al. A coloured petri net based approach for estimating execution time and energy consumption in embedded systems. In *Symposium on Integrated Circuits and Systems Design (SBCCI'08)*, 2008.

[105] R. Wilhelm and et al. The worst-case execution-time problem-overview of methods and survey of tools. *ACM Transactions in Embedded Computing Systems (TECS)*, 7, 2008.

[106] M. Oliveira Jr. *Estimativa do Consumo de Energia devido ao Software: Uma Abordagem Baseada em Redes de Petri Coloridas (in portuguese)*. PhD Thesis, Centro de Informática, Universidade Federal de Pernambuco, Outubro 2006.

[107] B. Efron and R. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall, 1993.

[108] C. Chung. *Simulation Modeling Handbook: A Practical Approach*. CRC, 2003.

[109] G. Bolch and et al. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, 2006.

[110] M. Triola. *Elementary Statistics*. Addison Wesley, 9 edition, 2004.

[111] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD Thesis, Dept Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1983.

[112] K. van Hee and et al. Architecture of information systems using the theory of petri nets. *Lecture notes for Systeemmodelleren 1 (2M310)*, 2004.

[113] G. Albuquerque Jr. *Avaliação de Desempenho de Cadeias de Suprimentos Utilizando Componentes GSPN (in portuguese)*. MSc Thesis, Centro de Informática, Universidade Federal de Pernambuco, August 2007.

[114] A. Valmari. The state explosion problem. *LNCS: Lectures on Petri Nets I: Basic Models*, 1491:429–528, June 1998.

[115] M. Muller-Olm, D. Schmidt, and B. Steffen. Model checking: a tutorial introduction. In *SAS'99*, 1999.

[116] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*. PhD Thesis, University of Liege, Nov. 1994.

[117] M. Garey and D. Johnson. *Computer and Intractability: a Guide to the Theory of the NP-Completeness*. W. H. Freeman and Company, 1979.

[118] J. Lilius. Efficient state space search for time petri nets. In *Electronic Notes in Theoretical Computer Science*, volume 18. Elsevier Science, 1998.

[119] N. Leveson and J. Stolzy. Safety analysis using petri nets. *IEEE Trans. Softw. Eng.*, 13:386–397, 1987.

[120] K. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill, 2007.

[121] Amalghma and dentes tools. http://www.cin.ufpe.br/~eagt/tools. 2008.

[122] N. Kim and et al. Visual assessment of a real-time systems design: A case study on a CNC controller. *Real-Time Systems Symposium (RTSS'96)*, pages 300–310, 1996.

[123] M. Oliveira Jr. *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso (in portuguese)*. MSc Thesis, Departamento de Biofísica e Radiobiologia, Universidade Federal de Pernambuco, August 1998.

[124] R. Prathipati. *Energy Efficient Scheduling Techniques for Real-Time Embedded Systems*. MSc Thesis, Texas A&M University, USA, 2004.

[125] Agilent Technologies. Scope connect software. Technical report.

[126] G. Palshikar. An introduction to model-checking. *Embedded Systems Programming*, December 2004. http://www.embedded.com/columns/technicalinsights/17603352?_requestid=94440.

APPENDIX A

# BASIC BUILDING BLOCKS: JUXTAPOSITION OF P-INVARIANTS

The adopted modeling approach assures that the generated models are structurally *bounded* and structurally *conservative*. In order to prove previous assertion, an important step is to demonstrate that the allowed fusions of basic building block models generate structurally *conservative* as well as structurally *bounded* models (e.g., the conservative components are preserved). As follows, the preservation of *boundness* property is demonstrated using the juxtaposition of P-invariants. Assume that the variables in each vector belongs to the set of positive integers.

## A.1 Fork Block and Periodic Task Arrival Block

$$\mathcal{I}_{(f)} = \begin{bmatrix} \overset{p_{start_{spec}}}{st_1 + \cdots + st_i + \cdots + st_n} & \overset{p_{st_1}}{st_1} & \overset{\cdots}{\cdots} & \overset{p_{st_i}}{st_i} & \overset{\cdots}{\cdots} & \overset{p_{st_n}}{st_n} \end{bmatrix}^T$$

$$\mathcal{I}_{(a)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i+1)(wvs_i+wd_i)} & \overset{p_{wa_i}}{wvs_i+wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wvs_i}}{wvs_i} \end{bmatrix}^T$$

$\mathcal{N}_{(f \sqcup a)} = \mathcal{N}_{(f)} \sqcup \mathcal{N}_{(a)}$. By juxtaposition $\mathcal{I}_{(f \sqcup a)} = \mathcal{J}(\mathcal{I}_{(f)}, \mathcal{I}_{(a)})$ and $st_i = (\alpha_i+1)(wvs_i + wd_i)$:

$$\mathcal{I}_{(f \sqcup a)} = \begin{bmatrix} \overset{p_{start_{spec}}}{st_1 + \cdots + (\alpha_i+1)(wvs_i+wd_i) + \cdots + st_n} & \overset{p_{st_1}}{st_1} & \overset{\cdots}{\cdots} & \overset{p_{st_i}}{(\alpha_i+1)(wvs_i+wd_i)} \\ \overset{\cdots}{\cdots} & \overset{p_{st_n}}{st_n} & \overset{p_{wa_i}}{wvs_i+wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wvs_i}}{wvs_i} \end{bmatrix}^T$$

Since $\mathcal{I}_{(f \sqcup a)} > 0$ and $\mathcal{I}^T_{(f \sqcup a)} \times A_{(f \sqcup a)} = 0$, in which $A_{(f \sqcup a)}$ is the incidence matrix, $\mathcal{N}_{(f \sqcup a)}$ is structurally conservative as well as structurally bounded.

## A.2 Periodic Task Arrival Block and Deadline Checking Block

$$\mathcal{I}_{(a)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i+1)(wvs_i+wd_i)} & \overset{p_{wa_i}}{wvs_i+wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wvs_i}}{wvs_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(d)} = \begin{bmatrix} \overset{p_{wd_i}}{(x)d_0 + ... + d_{2,...,x}} & \overset{p_{wpc_{i_1}}}{d_0 + ... + d_{2,...,x}} & \overset{p_{wc_{i_1}}}{d_1 + \cdots + d_{1,...,x}} & \overset{\cdots}{\cdots} & \overset{p_{wpc_{i_j}}}{d_0 + \cdots + d_{1,l}} \\ \overset{p_{wc_{i_j}}}{d_x + ... + d_{1,...,x}} & \overset{p_{dm_i}}{d_0 + \cdots + d_{1,...,x}} \end{bmatrix}^T$$

$\mathcal{N}_{(a \sqcup d)} = \mathcal{N}_{(a)} \sqcup \mathcal{N}_{(d)}$. By juxtaposition $\mathcal{I}_{(a \sqcup d)} = \mathcal{J}(\mathcal{I}_{(a)}, \mathcal{I}_{(d)})$, such that $wd_i = (x)d_0 + ... + d_{2,...,x}$:

$$\mathcal{I}_{(a \sqcup d)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i+1)(wvs_i + ((x)d_0 + ... + d_{2,...,x})} & \overset{p_{wa_i}}{wvs_i + ((x)d_0 + ... + d_{2,...,x})} \\ \overset{p_{wd_i}}{(x)d_0 + ... + d_{2,...,x}} & \overset{p_{wr_i}}{wvs_i} \quad \overset{p_{wvs_i}}{wvs_i} \quad \overset{p_{wpc_{i_1}}}{d_0 + ... + d_{2,...,x}} \quad \overset{p_{wc_{i_1}}}{d_1 + \cdots + d_{1,...,x}} \quad \overset{\cdots}{\cdots} \\ \overset{p_{wpc_{i_j}}}{d_0 + \cdots + d_{1,l}} & \overset{p_{wc_{i_j}}}{d_x + ... + d_{1,...,x}} \quad \overset{p_{dm_i}}{d_0 + \cdots + d_{1,...,x}} \end{bmatrix}^T$$

$\mathcal{N}_{(a \sqcup d)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}^T_{(f \sqcup a)} \times A_{(a \sqcup d)} = 0$, in which $A_{(a \sqcup d)}$ is the respective incidence matrix, and $\mathcal{I}_{(f \sqcup a)} > 0$.

### A.3 Periodic Task Arrival Block and Voltage Selection Block

$$\mathcal{I}_{(a)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i+1)(wvs_i + wd_i)} & \overset{p_{wa_i}}{wvs_i + wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wvs_i}}{wvs_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wvs'_i} & \overset{p_{v_{i_1}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs'_i} \end{bmatrix}^T$$

$\mathcal{N}_{(a \sqcup v)} = \mathcal{N}_{(a)} \sqcup \mathcal{N}_{(v)}$. By juxtaposition $\mathcal{I}_{(a \sqcup v)} = \mathcal{J}(\mathcal{I}_{(a)}, \mathcal{I}_{(v)})$ and $wvs'_i = wvs_i$:

$$\mathcal{I}_{(a \sqcup v)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i+1)(wvs_i + wd_i)} & \overset{p_{wa_i}}{wvs_i + wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wvs_i}}{wvs_i} & \overset{p_{v_{i_1}}}{wvs_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs_i} \\ \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs_i} \end{bmatrix}^T$$

$\mathcal{N}_{(a \sqcup v)}$ is structurally conservative as well as structurally bounded, since $\mathcal{I}^T_{(f \sqcup a)} \times A_{(a \sqcup d)} = 0$ and $\mathcal{I}_{(f \sqcup a)} > 0$.

### A.4 Voltage Selection Block and Non-Preemptive Task Structure Block

$$\mathcal{I}_{(v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wvs'_i} & \overset{p_{v_{i_1}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs'_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(np)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$\mathcal{N}_{(v \sqcup np)} = \mathcal{N}_v \sqcup \mathcal{N}_{np}$. By juxtaposition $\mathcal{I}_{(v \sqcup np)} = \mathcal{J}(\mathcal{I}_{(v)}, \mathcal{I}_{(np)})$, such that $wvs'_i = wg_{i_j}$:

$$\mathcal{I}_{(v \sqcup np)} = \begin{bmatrix} \overset{p_{wvs_i}}{wg_{i_j}} & \overset{p_{v_{i_1}}}{wg_{i_j}} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wg_{i_j}} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

As $\mathcal{I}^T_{(v \sqcup np)} \times A_{(v \sqcup np)} = 0$, in which $A_{(v \sqcup np)}$ is the incidence matrix, and $\mathcal{I}_{(v \sqcup np)} > 0$, $\mathcal{N}_{(v \sqcup np)}$ is structurally conservative as well as structurally bounded.

## A.5 Voltage Selection Block and Preemptive Task Structure Block

$$\mathcal{I}_{(v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wvs'_i} & \overset{p_{v_{i_1}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs'_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(p)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C)wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{(C)wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$\mathcal{N}_{(v \sqcup p)} = \mathcal{N}_v \sqcup \mathcal{N}_p$ . By juxtaposition $\mathcal{I}_{(v \sqcup p)} = \mathcal{J}(\mathcal{I}_{(v)}, \mathcal{I}_{(p)})$, such that $wvs'_i = wg_{i_j}$:

$$\mathcal{I}_{(v \sqcup p)} = \begin{bmatrix} \overset{p_{wvs_i}}{(C)wg_{i_j}} & \overset{p_{v_{i_1}}}{(C)wg_{i_j}} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{(C)wg_{i_j}} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{(C)wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} \\ \overset{p_{fv_i}}{(C)wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$\mathcal{N}_{(v \sqcup p)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}^T_{(v \sqcup p)} \times A_{(v \sqcup p)} = 0$, in which $A_{(v \sqcup p)}$ is the respective incidence matrix, and $\mathcal{I}_{(v \sqcup p)} > 0$.

## A.6 Voltage Selection Block and Non-Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wvs'_i} & \overset{p_{v_{i_1}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs'_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(np2v)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wg_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wc_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg1_{i_j}} & \overset{p_{fv_i}}{wg1_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$\mathcal{N}_{(v \sqcup np2v)} = \mathcal{N}_v \sqcup \mathcal{N}_{np2v}$. By juxtaposition $\mathcal{I}_{(v \sqcup p)} = \mathcal{J}(\mathcal{I}_{(v)}, \mathcal{I}_{(np2v)})$, such that $wvs'_i = wg_{i_j}$:

$$\mathcal{I}_{(v \sqcup np2v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wg1_{i_j}} & \overset{p_{v_{i_1}}}{wg1_{i_j}} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wg1_{i_j}} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{wg1_{i_j}+proc} \\ \overset{p_{wg_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wc_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg1_{i_j}} & \overset{p_{fv_i}}{wg1_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

Since $\mathcal{I}_{(v \sqcup np2v)} > 0$ and $\mathcal{I}^T_{(v \sqcup np2v)} \times A_{(v \sqcup np2v)} = 0$, in which $A_{(v \sqcup np2v)}$ is the incidence matrix, $\mathcal{N}_{(v \sqcup np2v)}$ is structurally conservative as well as structurally bounded.

## A.7 Voltage Selection Block and Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wvs'_i} & \overset{p_{v_{i_1}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs'_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs'_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(p2v)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_1C_2)wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{(C_2)wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{(C_2)wg1_{i_j}+proc} & \overset{p_{wf1_{i_j}}}{(C_2)wg1_{i_j}} & \overset{p_{wg_{i_j}}}{(C_1)wg1_{i_j}} \end{bmatrix}$$

$$\overset{p_{wc_{i_j}}}{(C_1)wg1_{i_j}+proc} \quad \overset{p_{wf_{i_j}}}{(C_1)wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_1C_2)wg1_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$\mathcal{N}_{(v\sqcup p2v)} = \mathcal{N}_v \sqcup \mathcal{N}_{p2v}$. By juxtaposition $\mathcal{I}_{(v\sqcup p2p)} = \mathcal{J}(\mathcal{I}_{(v)}, \mathcal{I}_{(p2v)})$, such that $wvs'_i = wg_{i_j}$:

$$\mathcal{I}_{(v\sqcup p2v)} = \Big[ \overset{p_{wvs_i}}{(C_1C_2)wg1_{i_j}} \quad \overset{p_{v_{i_1}}}{(C_1C_2)wg1_{i_j}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{v_{i_j}}}{(C_1C_2)wg1_{i_j}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{v_{i_m}}}{(C_1C_2)wg1_{i_j}}$$
$$\overset{p_{wg1_{i_j}}}{(C_2)wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_2)wg1_{i_j}+proc} \quad \overset{p_{wf1_{i_j}}}{(C_2)wg1_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_1)wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{(C_1)wg1_{i_j}+proc} \quad \overset{p_{wf_{i_j}}}{(C_1)wg1_{i_j}}$$
$$\overset{p_{fv_i}}{(C_1C_2)wg1_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

As $\mathcal{I}_{(v\sqcup p2v)} > 0$ and $\mathcal{I}^T_{(v\sqcup p2v)} \times A_{(v\sqcup p2v)} = 0$, in which $A_{(v\sqcup p2v)}$ is the incidence matrix, $\mathcal{N}_{(v\sqcup p2v)}$ is structurally conservative as well as structurally bounded.

## A.8 Non-Preemptive Task Structure Block and Non-Preemptive Task Structure Block

$$\mathcal{I}_{(np_j)} = \Big[ \overset{p_{v_{i_j}}}{wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$
$$\mathcal{I}_{(np_s)} = \Big[ \overset{p_{v_{i_s}}}{wg_{i_s}} \quad \overset{p_{wg_{i_s}}}{wg_{i_s}} \quad \overset{p_{wc_{i_s}}}{wg_{i_s}+proc'} \quad \overset{p_{wf_{i_s}}}{wg_{i_s}} \quad \overset{p_{fv_i}}{wg_{i_s}} \quad \overset{p_{proc}}{proc'} \Big]^T$$

$\mathcal{N}_{(np_j\sqcup np_s} = \mathcal{N}_{np_j} \sqcup \mathcal{N}_{np_s}$. By juxtaposition $\mathcal{I}_{(np_j\sqcup np_s)} = \mathcal{J}(\mathcal{I}_{(np_j)}, \mathcal{I}_{(np_s)})$, such that $wg_{i_s} = wg_{i_j}$ and $proc' = proc$:

$$\mathcal{I}_{(np_j\sqcup np_s)} = \Big[ \overset{p_{v_{i_j}}}{wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{wg_{i_j}} \quad \overset{p_{proc}}{proc} \quad \overset{p_{v_{i_s}}}{wg_{i_j}} \quad \overset{p_{wg_{i_s}}}{wg_{i_j}} \quad \overset{p_{wc_{i_s}}}{wg_{i_j}+proc}$$
$$\overset{p_{wf_{i_s}}}{wg_{i_s}} \Big]^T$$

$\mathcal{N}_{(np_s\sqcup np_j)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(np_s\sqcup np_j)} > 0$ and $\mathcal{I}^T_{(np_s\sqcup np_j)} \times A_{(np_s\sqcup np_j)} = 0$, in which $A_{(np_s\sqcup np_j)}$ is the respective incidence matrix.

## A.9 Non-Preemptive Task Structure Block and Non-Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(np_j)} = \Big[ \overset{p_{v_{i_j}}}{wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$
$$\mathcal{I}_{(np2v_s)} = \Big[ \overset{p_{v_{i_s}}}{wg1_{i_s}} \quad \overset{p_{wg1_{i_s}}}{wg1_{i_s}} \quad \overset{p_{wc1_{i_s}}}{wg1_{i_s}+proc'} \quad \overset{p_{wg_{i_s}}}{wg1_{i_s}+proc} \quad \overset{p_{wc_{i_s}}}{wg1_{i_s}+proc'} \quad \overset{p_{wf_{i_s}}}{wg1_{i_s}} \quad \overset{p_{fv_i}}{wg1_{i_s}}$$
$$\overset{p_{proc}}{proc'} \Big]^T$$

$\mathcal{N}_{(np_j \sqcup np2v_s)} = \mathcal{N}_{np_j} \sqcup \mathcal{N}_{np2v_s}$. By juxtaposition $\mathcal{I}_{(np_j \sqcup np2v_s)} = \mathcal{J}(\mathcal{I}_{(np_j)}, \mathcal{I}_{(np2v_s)})$, such that $wg_{i_s} = wg_{i_j}$ and $proc' = proc$:

$$\mathcal{I}_{(np_j \sqcup np2v_s)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{wg_{i_j}} & \overset{proc}{proc} & \overset{p_{v_{i_s}}}{wg_{i_j}} & \overset{p_{wg1_{i_s}}}{wg_{i_j}} & \overset{p_{wc1_{i_s}}}{wg_{i_j}+proc} \\ \overset{p_{wg_{i_s}}}{wg_{i_s}+proc} & \overset{p_{wc_{i_s}}}{wg_{i_s}+proc} & \overset{p_{wf_{i_s}}}{wg_{i_s}} \end{bmatrix}^T$$

Since $\mathcal{I}_{(np_j \sqcup np2v_s)} > 0$ and $\mathcal{I}^T_{(np_j \sqcup np2v_s)} \times A_{(np_j \sqcup np2v_s)} = 0$, in which $A_{(np_j \sqcup np2v_s)}$ is the incidence matrix, $\mathcal{N}_{(np_j \sqcup np2v_s)}$ is structurally conservative as well as structurally bounded.

## A.10 Non-Preemptive Task Structure Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers three *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$\mathcal{I}_{(np)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$$\mathcal{I}_{(d)} = \begin{bmatrix} \overset{p_{wd_i}}{3d_0+2d_1+2d_{1j}+d_{1,1j}+2d_j+d_{1,j}+d_{1j,j}} \\ \overset{p_{wpc_{i_1}}}{d_0+d_{1j}+d_j+d_{1j,j}} & \overset{p_{wc_{i_1}}}{d_1+d_{1,1j}+d_{1,j}+d_{1,1j,j}} \\ \overset{p_{wpc1_{i_j}}}{d_0+d_1+d_3+d_{1,j}} & \overset{p_{wc1_{i_j}}}{d_{1j}+d_{1,1j}+d_{1j,j}+d_{1,1j,j}} \\ \overset{p_{wpc_{i_j}}}{d_0+d_1+d_{1j}+d_{1,1j}} & \overset{p_{wc_{i_j}}}{d_j+d_{1,j}+d_{1j,j}+d_{1,1j,j}} \\ \overset{p_{dm_i}}{d_0+d_1+d_{1j}+d_j+d_{1,1j}+d_{1,j}+d_{1j,j}+d_{1,1j,j}} \end{bmatrix}^T$$

In order to allow the juxtaposition, variable *proc* is replaced by $(2^x/2-1)d_0$, and all variables in $\mathcal{I}_{(d)}$ that cover more than one *waiting for task computation* place (e.g., $d_{1j,j}$) are substituted by $d_0$ . Additionally, variable $d_j$ is replaced by $wg_{i_j}$, since such variable covers only one *waiting for task computation* place in $\mathcal{I}_{(d)}$, more specifically, $p_{wc_{i_j}} \in P_{np} \cap P_d$.

The composition is demonstrated as follows. $\mathcal{N}_{(np \sqcup d)} = \mathcal{N}_{np} \sqcup \mathcal{N}_d$. Thus, by juxtaposition $\mathcal{I}_{(np \sqcup d)} = \mathcal{J}(\mathcal{I}_{(np)}, \mathcal{I}_{(d)})$, such that $proc = 3d_0$, $d_{1,1j} = d_{1,j} = d_{1j,j} = d_{1,1j,j} = d_0$ and $d_j = wg_{i_j}$:

$$\mathcal{I}_{(np \sqcup d)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+3d_0} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{wg_{i_j}} & \overset{p_{proc}}{3d_0} & \overset{p_{wd_i}}{6d_0+2d_1+2d_{1j}+2wg1_{i_j}} \\ \overset{p_{wpc_{i_1}}}{2d_0+d_{1j}+wg1_{i_j}} & \overset{p_{wc_{i_1}}}{d_1+3d_0} & \overset{p_{wpc1_{i_j}}}{2d_0+d_1+wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{d_{1j}+3d_0} & \overset{p_{wpc_{i_j}}}{2d_0+d_1+d_{1j}} \end{bmatrix}$$

$$\overset{p_{dm_i}}{5d_0 + d_1 + d_{1j} + wg1_{i_j}}\ \Big]^T$$

As $\mathcal{I}_{(np \sqcup d)} > 0$ and $\mathcal{I}^T_{(np \sqcup d)} \times A_{(np \sqcup d)} = 0$, in which $A_{(np \sqcup d)}$ is the incidence matrix, $\mathcal{N}_{(np \sqcup d)}$ is structurally conservative as well as structurally bounded.

## A.11 Non-Preemptive Task Structure Block and Task Instance Conclusion Block

$$\mathcal{I}_{(np)} = \Big[ \overset{p_{v_{i_j}}}{wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$$\mathcal{I}_{(c)} = \Big[ \overset{p_{f_i}}{fv_i + wd_i} \quad \overset{p_{fv_i}}{fv_i} \quad \overset{p_{wd_i}}{wd_i} \Big]^T$$

$\mathcal{N}_{(np \sqcup c)} = \mathcal{N}_{np} \sqcup \mathcal{N}_c$. By juxtaposition $\mathcal{I}_{(np \sqcup c)} = \mathcal{J}(\mathcal{I}_{(np)}, \mathcal{I}_{(c)})$, such that $fv_i = wg_{i_j}$:

$$\mathcal{I}_{(np \sqcup c)} = \Big[ \overset{p_{v_{i_j}}}{wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{wg_{i_j}} \quad \overset{proc}{proc} \quad \overset{p_{f_i}}{wg_{i_j} + wd_i} \quad \overset{p_{wd_i}}{wd_i} \Big]^T$$

$\mathcal{N}_{(np \sqcup c)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(np \sqcup d)} > 0$ and $\mathcal{I}^T_{(np \sqcup c)} \times A_{(np \sqcup c)} = 0$, in which $A_{(np \sqcup c)}$ is the respective incidence matrix.

## A.12 Preemptive Task Structure Block and Preemptive Task Structure Block

$$\mathcal{I}_{(p_j)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$$\mathcal{I}_{(p_s)} = \Big[ \overset{p_{v_{i_s}}}{(C_{i_s})wg_{i_s}} \quad \overset{p_{wg_{i_s}}}{wg_{i_s}} \quad \overset{p_{wc_{i_s}}}{wg_{i_s} + proc'} \quad \overset{p_{wf_{i_s}}}{wg_{i_s}} \quad \overset{p_{fv_i}}{(C_{i_s})wg_{i_s}} \quad \overset{p_{proc}}{proc'} \Big]^T$$

$\mathcal{N}_{(p_j \sqcup p_s)} = \mathcal{N}_{p_j} \sqcup \mathcal{N}_{p_s}$. By juxtaposition $\mathcal{I}_{(p_j \sqcup p_s)} = \mathcal{J}(\mathcal{I}_{(p_j)}, \mathcal{I}_{(p_s)})$, such that $wg_{i_j} = (C_{i_s})wg'_{i_j}$, $wg_{i_s} = (C_{i_j})wg'_{i_j}$ and $proc' = proc$:

$$\mathcal{I}_{(p_j \sqcup p_s)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j}C_{i_s})wg'_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{i_s})wg'_{i_j}} \quad \overset{p_{wc_{i_j}}}{(C_{i_s})wg'_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{i_s})wg'_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j}C_{i_s})wg'_{i_j}}$$
$$\overset{proc}{proc} \quad \overset{p_{v_{i_s}}}{(C_{i_s}C_{i_j})wg'_{i_j}} \quad \overset{p_{wg_{i_s}}}{(C_{i_j})wg'_{i_j}} \quad \overset{p_{wc_{i_s}}}{(C_{i_j})wg'_{i_j} + proc} \quad \overset{p_{wf_{i_s}}}{(C_{i_j})wg'_{i_j}} \Big]^T$$

Since $\mathcal{I}_{(p_j \sqcup p_s)} > 0$ and $\mathcal{I}^T_{(p_j \sqcup p_s)} \times A_{(p_j \sqcup p_s)} = 0$, in which $A_{(p_j \sqcup p_s)}$ is the incidence matrix, $\mathcal{N}_{(p_j \sqcup p_s)}$ is structurally conservative as well as structurally bounded.

## A.13 Preemptive Task Structure Block and Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(p_j)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$$\mathcal{I}_{(p2v_s)} = \begin{bmatrix} \overset{p_{v_{i_s}}}{(C_{1_{i_s}}C_{2_{i_s}})wg1_{i_s}} & \overset{p_{wg1_{i_s}}}{(C_{2_{i_s}})wg1_{i_s}} & \overset{p_{wc1_{i_s}}}{(C_{2_{i_s}})wg1_{i_s}+proc'} & \overset{p_{wf1_{i_s}}}{(C_{2_{i_s}})wg1_{i_s}} & \overset{p_{wg_{i_s}}}{(C_{1_{i_s}})wg1_{i_s}} \\ \overset{p_{wc_{i_s}}}{(C_{1_{i_s}})wg1_{i_s}+proc} & \overset{p_{wf_{i_s}}}{(C_{1_{i_s}})wg1_{i_s}} & \overset{p_{fv_i}}{(C_{1_{i_s}}C_{2_{i_s}})wg1_{i_s}} & \overset{p_{proc}}{proc'} \end{bmatrix}^T$$

$\mathcal{N}_{(p_j \sqcup p2v_s)} = \mathcal{N}_{p_j} \sqcup \mathcal{N}_{p2v_s}$. By juxtaposition $\mathcal{I}_{(p_j \sqcup p2v_s)} = \mathcal{J}(\mathcal{I}_{(p_j)}, \mathcal{I}_{(p2v_s)})$ , such that $wg_{i_j} = (C_{1_{i_s}}C_{2_{i_s}})wg'_{i_j}$, $wg1_{i_s} = (C_{i_j})wg'_{i_j}$ and $proc' = proc$:

$$\mathcal{I}_{(p_j \sqcup pv2_s)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{i_j}C_{1_{i_s}}C_{2_{i_s}})wg'_{i_j}} & \overset{p_{wg_{i_j}}}{(C_{1_{i_s}}C_{2_{i_s}})wg'_{i_j}} & \overset{p_{wc_{i_j}}}{(C_{1_{i_s}}C_{2_{i_s}})wg'_{i_j}+proc} & \overset{p_{wf_{i_j}}}{(C_{1_{i_s}}C_{2_{i_s}})wg'_{i_j}} \\ \overset{p_{fv_i}}{(C_{i_j}C_{1_{i_s}}C_{2_{i_s}})wg'_{i_j}} & \overset{p_{proc}}{proc} & \overset{p_{v_{i_s}}}{(C_{1_{i_s}}C_{2_{i_s}}C_{i_j})wg'_{i_j}} & \overset{p_{wg1_{i_s}}}{(C_{2_{i_s}}C_{i_j})wg'_{i_j}} & \overset{p_{wc1_{i_s}}}{(C_{2_{i_s}}C_{i_j})wg'_{i_j}+proc} \\ \overset{p_{wf1_{i_s}}}{(C_{2_{i_s}}C_{i_j})wg'_{i_j}} & \overset{p_{wg_{i_s}}}{(C_{1_{i_s}}C_{i_j})wg'_{i_j}} & \overset{p_{wc_{i_s}}}{(C_{1_{i_s}}C_{i_j})wg'_{i_j}+proc} & \overset{p_{wf_{i_s}}}{(C_{1_{i_s}}C_{i_j})wg'_{i_j}} \end{bmatrix}^T$$

As $\mathcal{I}_{(p_j \sqcup pv2_s)} > 0$ and $\mathcal{I}^T_{(p_j \sqcup pv2_s)} \times A_{(p_j \sqcup pv2_s)} = 0$, in which $A_{(p_j \sqcup pv2_s)}$ is the respective incidence matrix, $\mathcal{N}_{(p_j \sqcup pv2_s)}$ is structurally conservative as well as structurally bounded.

## A.14 Preemptive Task Structure Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers three *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$\mathcal{I}_{(p)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wc_{i_j}}}{wg_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg_{i_j}} & \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} & \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$$\mathcal{I}_{(d)} = \begin{bmatrix} \overset{p_{wd_i}}{3d_0+2d_1+2d_{1j}+d_{1,1j}+2d_j+d_{1,j}+d_{1j,j}} \\ \overset{p_{wpc_{i_1}}}{d_0+d_{1j}+d_j+d_{1j,j}} & \overset{p_{wc_{i_1}}}{d_1+d_{1,1j}+d_{1,j}+d_{1,1j,j}} \\ \overset{p_{wpc1_{i_j}}}{d_0+d_1+d_3+d_{1,j}} & \overset{p_{wc1_{i_j}}}{d_{1j}+d_{1,1j}+d_{1j,j}+d_{1,1j,j}} \\ \overset{p_{wpc_{i_j}}}{d_0+d_1+d_{1j}+d_{1,1j}} & \overset{p_{wc_{i_j}}}{d_j+d_{1,j}+d_{1j,j}+d_{1,1j,j}} \\ \overset{p_{dm_i}}{d_0+d_1+d_{1j}+d_j+d_{1,1j}+d_{1,j}+d_{1j,j}+d_{1,1j,j}} \end{bmatrix}^T$$

In order to allow the juxtaposition, variable *proc* is replaced by $(2^x/2-1)d_0$, and all variables in $\mathcal{I}_{(d)}$ that cover more than one *waiting for task computation* place (e.g., $d_{1j,j}$) are substituted by $d_0$ . Additionally, variable $d_j$ is replaced by $wg_{i_j}$, since such variable covers only one *waiting for task computation* place in $\mathcal{I}_{(d)}$, more specifically, $p_{wc_{i_j}} \in P_p \cap P_d$.

The composition is demonstrated as follows. $\mathcal{N}_{(p \sqcup d)} = \mathcal{N}_p \sqcup \mathcal{N}_d$. Thus, by juxtaposition $\mathcal{I}_{(p \sqcup d)} = \mathcal{J}(\mathcal{I}_{(p)}, \mathcal{I}_{(d)})$, such that $proc = 3d_0$, $d_{1,1j} = d_{1,j} = d_{1j,j} = d_{1,1j,j} = d_0$ and $d_j = wg_{i_j}$:

$$\mathcal{I}_{(p \sqcup d)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + 3d_0} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc}}{3d_0}$$

$$\overset{p_{wd_i}}{6d_0 + 2d_1 + 2d_{1j} + 2wg1_{i_j}} \quad \overset{p_{wpc_{i_1}}}{2d_0 + d_{1j} + wg1_{i_j}} \quad \overset{p_{wc_{i_1}}}{d_1 + 3d_0} \quad \overset{p_{wpc1_{i_j}}}{2d_0 + d_1 + wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{d_{1j} + 3d_0}$$

$$\overset{p_{wpc_{i_j}}}{2d_0 + d_1 + d_{1j}} \quad \overset{p_{dm_i}}{5d_0 + d_1 + d_{1j} + wg1_{i_j}} \quad \overset{p_{dm_i}}{5d_0 + d_1 + d_{1j} + wg1_{i_j}} \Big]^T$$

As $\mathcal{I}_{(p \sqcup d)} > 0$ and $\mathcal{I}^T_{(p \sqcup d)} \times A_{(p \sqcup d)} = 0$, in which $A_{(p \sqcup d)}$ is the incidence matrix, $\mathcal{N}_{(p \sqcup d)}$ is structurally conservative as well as structurally bounded.

### A.15 Preemptive Task Structure Block and Task Instance Conclusion Block

$$\mathcal{I}_{(p)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$$\mathcal{I}_{(c)} = \Big[ \overset{p_{f_i}}{fv_i + wd_i} \quad \overset{p_{fv_i}}{fv_i} \quad \overset{p_{wd_i}}{wd_i} \Big]^T$$

$\mathcal{N}_{(p \sqcup c)} = \mathcal{N}_p \sqcup \mathcal{N}_v$. By juxtaposition $\mathcal{I}_{(p \sqcup c)} = \mathcal{J}(\mathcal{I}_{(p)}, \mathcal{I}_{(c)})$, such that $fv_i = wg_{i_j}$:

$$\mathcal{I}_{(p \sqcup c)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc}}{proc} \quad \overset{p_{f_i}}{(C_{i_j})wg_{i_j} + wd_i}$$

$$\overset{p_{wd_i}}{wd_i} \Big]^T$$

$\mathcal{N}_{(p \sqcup c)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(np \sqcup c)} > 0$ and $\mathcal{I}^T_{(np \sqcup c)} \times A_{(p \sqcup c)} = 0$, in which $A_{(p \sqcup c)}$ is the respective incidence matrix.

### A.16 Non-Preemptive Task Structure with 2 Voltages Block and Non-Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(np2v_j)} = \Big[ \overset{p_{v_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wg_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wc_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg1_{i_j}} \quad \overset{p_{fv_i}}{wg1_{i_j}}$$

$$\overset{p_{proc}}{proc} \Big]^T$$

$$\mathcal{I}_{(np2v_s)} = \begin{bmatrix} \overset{p_{v_{i_s}}}{wg1_{i_s}} & \overset{p_{wg1_{i_s}}}{wg1_{i_s}} & \overset{p_{wc1_{i_s}}}{wg1_{i_s}+proc} & \overset{p_{wg_{i_s}}}{wg1_{i_s}+proc'} & \overset{p_{wc_{i_s}}}{wg1_{i_s}+proc} & \overset{p_{wf_{i_s}}}{wg1_{i_s}} & \overset{p_{fv_i}}{wg1_{i_s}} \\ \overset{p_{proc}}{proc'} \end{bmatrix}^T$$

$\mathcal{N}_{(np2v_j \sqcup np2v_s)} = \mathcal{N}_{np2v_j} \sqcup \mathcal{N}_{np2v_s}$. By juxtaposition $\mathcal{I}_{(np2v_j \sqcup np2v_s)} = \mathcal{J}(\mathcal{I}_{(np2v_j)}, \mathcal{I}_{(np2v_s)})$, such that $wg1_{i_s} = wg1_{i_j}$ and $proc' = proc$:

$$\mathcal{I}_{(np2v_j \sqcup np2v_s)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wg_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wc_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg1_{i_j}} \\ \overset{p_{fv_i}}{wg1_{i_j}} & \overset{p_{proc}}{proc} & \overset{p_{v_{i_s}}}{wg1_{i_j}} & \overset{p_{wg1_{i_s}}}{wg1_{i_j}} & \overset{p_{wc1_{i_s}}}{wg1_{i_j}+proc} & \overset{p_{wg_{i_s}}}{wg1_{i_j}+proc} & \overset{p_{wc_{i_s}}}{wg1_{i_j}+proc} & \overset{p_{wf_{i_s}}}{wg1_{i_j}} \end{bmatrix}^T$$

Since $\mathcal{I}_{(np \sqcup c)} > 0$ and $\mathcal{I}^T_{(np2v_j \sqcup np2v_s)} \times A_{(np2v_j \sqcup np2v_s)} = 0$, in which $A_{(np2v_j \sqcup np2v_s)}$ is the incidence matrix, $\mathcal{N}_{(np2v_j \sqcup np2v_s)}$ is structurally conservative as well as structurally bounded.

## A.17 Non-Preemptive Task Structure with 2 Voltages Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers four *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$\mathcal{I}_{(np2v)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{wg1_{i_j}} & \overset{p_{wg1_{i_j}}}{wg1_{i_j}} & \overset{p_{wc1_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wg_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wc_{i_j}}}{wg1_{i_j}+proc} & \overset{p_{wf_{i_j}}}{wg1_{i_j}} & \overset{p_{fv_i}}{wg1_{i_j}} \\ \overset{p_{proc}}{proc} \end{bmatrix}^T$$

$$\mathcal{I}_{(d)} = \begin{bmatrix} 3d_0+2d_1+2d_{1j}+\overset{p_{wd_i}}{d_{1,1j}}+2d_j+d_{1,j}+d_{1j,j} \\ \overset{p_{wpc_{i_1}}}{d_0+d_{1j}+d_j+d_{1j,j}} & \overset{p_{wc_{i_1}}}{d_1+d_{1,1j}+d_{1,j}+d_{1,1j,j}} \\ \overset{p_{wpc1_{i_j}}}{d_0+d_1+d_j+d_{1,j}} & \overset{p_{wc1_{i_j}}}{d_{1j}+d_{1,1j}+d_{1j,j}+d_{1,1j,j}} \\ \overset{p_{wpc_{i_j}}}{d_0+d_1+d_{1j}+d_{1,1j}} & \overset{p_{wc_{i_j}}}{d_j+d_{1,j}+d_{1j,j}+d_{1,1j,j}} \\ d_0+d_1+d_{1j}+d_j+\overset{p_{dm_i}}{d_{1,1j}}+d_{1,j}+d_{1j,j}+d_{1,1j,j} \end{bmatrix}^T$$

In order to allow the juxtaposition, variable *proc* is replaced by $(2^x/2-1)d_0$, and all variables in $\mathcal{I}_{(d)}$ that cover more than one *waiting for task computation* place (e.g., $d_{1j,j}$) are substituted by $d_0$ . Additionally, variable $d_j$ and $d_{1j}$, which cover only one *waiting for task computation* place in $\mathcal{I}_{(d)}$ ($p_{wc_{i_j}}$ and $p_{wc1_{i_j}}$, respectively) are replaced by $wg1_{i_j}$.

Besides, $\{p_{wc_{i_j}}, p_{wc1_{i_j}}\} \subseteq P_{np2v} \cap P_d$.

The composition is demonstrated as follows. $\mathcal{N}_{(np2v \sqcup d)} = \mathcal{N}_{np2v} \sqcup \mathcal{N}_d$. Thus, by juxtaposition $\mathcal{I}_{(np2v \sqcup d)} = \mathcal{J}(\mathcal{I}_{(np2v)}, \mathcal{I}_{(d)})$, such that $d_{1,1j} = d_{1,j} = d_{1j,j} = d_{1,1j,j} = d_0$, $d_{1j} = d_j = wg1_{i_j}$ and $proc = 3d_0$:

$$\mathcal{I}_{(np2v \sqcup d)} = \Big[ \overset{p_{v_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{wg1_{i_j} + 3d_0} \quad \overset{p_{wg_{i_j}}}{wg1_{i_j} + 3d_0} \quad \overset{p_{wc_{i_j}}}{wg1_{i_j} + 3d_0} \quad \overset{p_{wf_{i_j}}}{wg1_{i_j}} \quad \overset{p_{fv_i}}{wg1_{i_j}}$$
$$\overset{p_{proc}}{3d_0} \quad \overset{p_{wd_i}}{6d_0 + 2d_1 + 4wg1_{i_j}} \quad \overset{p_{wpc_{i_1}}}{2d_0 + 2wg1_{i_j}} \quad \overset{p_{wc_{i_1}}}{d_1 + 3d_0} \quad \overset{p_{wpc1_{i_j}}}{2d_0 + d_1 + wg1_{i_j}} \quad \overset{p_{wpc_{i_j}}}{2d_0 + d_1 + wg1_{i_j}}$$
$$\overset{p_{dm_i}}{5d_0 + d_1 + 2wg1_{i_j}} \Big]^T$$

As $\mathcal{I}_{(np2v \sqcup d)} > 0$ and $\mathcal{I}^T_{(np2v \sqcup d)} \times A_{(np2v \sqcup d)} = 0$, in which $A_{(np2v \sqcup d)}$ is the incidence matrix, $\mathcal{N}_{(np2v \sqcup d)}$ is structurally conservative as well as structurally bounded.

### A.18 Non-Preemptive Task Structure with 2 Voltages Block and Task Instance Conclusion Block

$$\mathcal{I}_{(np2v)} = \Big[ \overset{p_{v_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wg_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wc_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg1_{i_j}} \quad \overset{p_{fv_i}}{wg1_{i_j}}$$
$$\overset{p_{proc}}{proc} \Big]^T$$
$$\mathcal{I}_{(c)} = \Big[ \overset{p_{f_i}}{fv_i + wd_i} \quad \overset{p_{fv_i}}{fv_i} \quad \overset{p_{wd_i}}{wd_i} \Big]^T$$

$\mathcal{N}_{(np2v \sqcup c)} = \mathcal{N}_{np2v} \sqcup \mathcal{N}_c$. By juxtaposition $\mathcal{I}_{(np2v \sqcup c)} = \mathcal{J}(\mathcal{I}_{(np2v)}, \mathcal{I}_{(c)})$, such that $fv_i = wg1_{i_j}$:

$$\mathcal{I}_{(np2v \sqcup c)} = \Big[ \overset{p_{v_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wg_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wc_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg1_{i_j}} \quad \overset{p_{fv_i}}{wg1_{i_j}}$$
$$\overset{p_{proc}}{proc} \quad \overset{p_{f_i}}{wg1_{i_j} + wd_i} \quad \overset{p_{wd_i}}{wd_i} \quad wd_i \Big]^T$$

$\mathcal{N}_{(np2v \sqcup c)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(np2v \sqcup c)} > 0$ and $\mathcal{I}^T_{(np2v \sqcup c)} \times A_{(np2v \sqcup c)} = 0$, in which $A_{(np2v \sqcup c)}$ is the respective incidence matrix.

### A.19 Preemptive Task Structure with 2 Voltages Block and Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(p2v_j)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}}) wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}}) wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}}) wg1_{i_j} + proc} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}}) wg1_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}}) wg1_{i_j}}$$
$$\overset{p_{wc_{i_j}}}{(C_{1_{i_j}}) wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}}) wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}}) wg1_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$$\mathcal{I}_{(p2v_s)} = \Big[ \overset{p_{v_{i_s}}}{(C_{1_{i_s}}C_{2_{i_s}})wg1_{i_j}} \quad \overset{p_{wg1_{i_s}}}{(C_{2_{i_s}})wg1_{i_j}} \quad \overset{p_{wc1_{i_s}}}{(C_{2_{i_s}})wg1_{i_j} + proc'} \quad \overset{p_{wf1_{i_s}}}{(C_{2_{i_s}})wg1_{i_j}} \quad \overset{p_{wg_{i_s}}}{(C_{1_{i_s}})wg1_{i_j}}$$
$$\overset{p_{wc_{i_s}}}{(C_{1_{i_s}})wg1_{i_j} + proc'} \quad \overset{p_{wf_{i_s}}}{(C_{1_{i_s}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_s}}C_{2_{i_s}})wg1_{i_j}} \quad \overset{p_{proc}}{proc'} \Big]^T$$

$\mathcal{N}_{(p2v_j \sqcup p2v_s)} = \mathcal{N}_{p2v_j} \sqcup \mathcal{N}_{p2v_s}$. By juxtaposition $\mathcal{I}_{(p2v_j \sqcup p2v_s)} = \mathcal{J}(\mathcal{I}_{(p2v_j)}, \mathcal{I}_{(p2v_s)})$, such that $wg1_{i_j} = (C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}$, $wg1_{i_s} = (C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j}$ and $proc' = proc$:

$$\mathcal{I}_{(p2v_j \sqcup p2v_s)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j} + proc}$$
$$\overset{p_{wf1_{i_j}}}{(C_{2_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}} \quad \overset{p_{wc_{i_j}}}{(C_{1_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j} + proc}$$
$$\overset{p_{wf_{i_j}}}{(C_{1_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}}C_{1_{i_s}}C_{2_{i_s}})wg1'_{i_j}} \quad \overset{p_{proc}}{proc} \quad \overset{p_{v_{i_s}}}{(C_{1_{i_s}}C_{2_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j}}$$
$$\overset{p_{wg1_{i_s}}}{(C_{2_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j}} \quad \overset{p_{wc1_{i_s}}}{(C_{2_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j} + proc} \quad \overset{p_{wf1_{i_s}}}{(C_{2_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j}} \quad \overset{p_{wg_{i_s}}}{(C_{1_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j}}$$
$$\overset{p_{wc_{i_s}}}{(C_{1_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j} + proc} \quad \overset{p_{wf_{i_s}}}{(C_{1_{i_s}}C_{1_{i_j}}C_{2_{i_j}})wg1'_{i_j}} \Big]^T$$

Since $\mathcal{I}_{(np2v \sqcup c)} > 0$ and $\mathcal{I}^T_{(p2v_j \sqcup p2v_s)} \times A_{(p2v_j \sqcup p2v_s)} = 0$, in which $A_{(p2v_j \sqcup p2v_s)}$ is the incidence matrix, $\mathcal{N}_{(p2v_j \sqcup p2v_s)}$ is structurally conservative as well as structurally bounded.

## A.20 Preemptive Task Structure with 2 Voltages Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers three *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$\mathcal{I}_{(p2v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}}$$
$$\overset{p_{wc_{i_j}}}{(C_{1_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$$\mathcal{I}_{(d)} = \Big[ \overset{p_{wd_i}}{3d_0 + 2d_1 + 2d_j + d_{1,1j} + 2d_j + d_{1,j} + d_{1j,j}}$$
$$\overset{p_{wpc_{i_1}}}{d_0 + d_{1j} + d_j + d_{1j,j}} \quad \overset{p_{wc_{i_1}}}{d_1 + d_{1,1j} + d_{1,j} + d_{1,1j,j}}$$
$$\overset{p_{wpc1_{i_j}}}{d_0 + d_1 + d_j + d_{1,j}} \quad \overset{p_{wc1_{i_j}}}{d_{1j} + d_{1,1j} + d_{1j,j} + d_{1,1j,j}}$$
$$\overset{p_{wpc_{i_j}}}{d_0 + d_1 + d_j + d_{1,1j}} \quad \overset{p_{wc_{i_j}}}{d_j + d_{1,j} + d_{1j,j} + d_{1,1j,j}}$$

$$d_0 + d_1 + d_{1j} + d_j + \overset{p_{dm_i}}{d_{1,1j}} + d_{1,j} + d_{1j,j} + d_{1,1j,j} \ ]^T$$

In order to allow the juxtaposition, variable *proc* is replaced by $(2^x/2-1)d_0$, and all variables in $\mathcal{I}_{(d)}$ that cover more than one *waiting for task computation* place (e.g., $d_{1j,j}$) are substituted by $d_0$ . Additionally, variable $d_j$ and $d_{1j}$ are replaced by $(C_{1_{i_j}})wg1_{i_j}$ and $(C_{2_{i_j}})wg1_{i_j}$, respectively. It is assumed that $d_j$ and $d_{1j}$ cover only one *waiting for task computation* place in $\mathcal{I}_{(d)}$ ($p_{wc_{i_j}}$ and $p_{wc1_{i_j}}$, respectively), and $\{p_{wc_{i_j}}, p_{wc1_{i_j}}\} \subseteq P_p \cap P_d$.

The composition is demonstrated as follows. $\mathcal{N}_{(p2v \sqcup d)} = \mathcal{N}_{p2v} \sqcup \mathcal{N}_d$. Thus, by juxtaposition $\mathcal{I}_{(p2v \sqcup d)} = \mathcal{J}(\mathcal{I}_{(p2v)}, \mathcal{I}_{(d)})$, such that $d_{1,1j} = d_{1,j} = d_{1j,j} = d_{1,1j,j} = d_0$, $proc = 3d_0$, $d_1 j = (C_{2_{i_j}})wg1_{i_j}$ and $d_j = (C_{1_{i_j}})wg1_{i_j}$:

$$\mathcal{I}_{(p2v \sqcup d)} = \quad \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j} + 3d_0} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{(C_{1_{i_j}})wg1_{i_j} + 3d_0} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc}}{3d_0}$$

$$6d_0 + 2d_1 + 2\overset{p_{wd_i}}{(C_{2_{i_j}})wg1_{i_j}} + 2(C_{1_{i_j}})wg1_{i_j} \quad 2d_0 + \overset{p_{wpc_{i_1}}}{(C_{2_{i_j}})wg1_{i_j}} + (C_{1_{i_j}})wg1_{i_j}$$

$$\overset{p_{wc_{i_1}}}{d_1 + 3d_0} \quad 2d_0 + \overset{p_{wpc1_{i_j}}}{d_1 + (C_{1_{i_j}})wg1_{i_j}} \quad 2d_0 + \overset{p_{wpc_{i_j}}}{d_1 + (C_{2_{i_j}})wg1_{i_j}}$$

$$5d_0 + d_1 + \overset{p_{dm_i}}{(C_{2_{i_j}})wg1_{i_j}} + (C_{1_{i_j}})wg1_{i_j} \ \Big]^T$$

As $\mathcal{I}_{(p2v \sqcup d)} > 0$ and $\mathcal{I}^T_{(p2v \sqcup d)} \times A_{(p2v \sqcup d)} = 0$, in which $A_{(p2v \sqcup d)}$ is the incidence matrix, $\mathcal{N}_{(p2v \sqcup d)}$ is structurally conservative as well as structurally bounded.

### A.21 Preemptive Task Structure with 2 Voltages Block and Task Instance Conclusion Block

$$\mathcal{I}_{(p2v)} = \quad \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{wc_{i_j}}}{(C_{1_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc}}{proc} \ \Big]^T$$

$$\mathcal{I}_{(c)} = \quad \Big[ \overset{p_{f_i}}{fv_i + wd_i} \quad \overset{p_{fv_i}}{fv_i} \quad \overset{p_{wd_i}}{wd_i} \ \Big]^T$$

$\mathcal{N}_{(p2v \sqcup c)} = \mathcal{N}_{p2v} \sqcup \mathcal{N}_c$. By juxtaposition $\mathcal{I}_{(p2v \sqcup c)} = \mathcal{J}(\mathcal{I}_{(p2v)}, \mathcal{I}_{(c)})$, such that $fv_i = (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}$:

$$\mathcal{I}_{(p2v \sqcup c)} = \quad \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{wc_{i_j}}}{(C_{1_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc}}{proc} \quad \overset{p_{f_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j} + wd_i} \quad \overset{p_{wd_i}}{wd_i} \ \Big]^T$$

Since $\mathcal{I}_{(p2v \sqcup c)} > 0$ and $\mathcal{I}^T_{(p2v \sqcup c)} \times A_{(p2v \sqcup c)} = 0$, in which $A_{(p2v \sqcup c)}$ is the incidence matrix, $\mathcal{N}_{(p2v \sqcup c)}$ is structurally conservative as well as structurally bounded.

## A.22 Deadline Checking Block and Task Instance Conclusion Block

$$\mathcal{I}_{(d)} = \begin{bmatrix} \overset{p_{wd_i}}{(x)d_0 + ... + d_{2,...,x}} & \overset{p_{wpc_{i_1}}}{d_0 + ... + d_{2,...,x}} & \overset{p_{wc_{i_1}}}{d_1 + \cdots + d_{1,...,x}} & \overset{\cdots}{\cdots} & \overset{p_{wpc_{i_j}}}{d_0 + \cdots + d_{1,l}} \\ \overset{p_{wc_{i_j}}}{d_x + ... + d_{1,...,x}} & \overset{p_{dm_i}}{d_0 + \cdots + d_{1,...,x}} \end{bmatrix}^T$$

$$\mathcal{I}_{(c)} = \begin{bmatrix} \overset{p_{f_i}}{fv_i + wd_i} & \overset{p_{fv_i}}{fv_i} & \overset{p_{wd_i}}{wd_i} \end{bmatrix}^T$$

$\mathcal{N}_{(d \sqcup c)} = \mathcal{N}_d \sqcup \mathcal{N}_c$. By juxtaposition $\mathcal{I}_{(d \sqcup c)} = \mathcal{J}(\mathcal{I}_{(d)}, \mathcal{I}_{(c)})$, such that $wd_i = (x)d_0 + ... + d_{2,...,x}$:

$$\mathcal{I}_{(d \sqcup c)} = \begin{bmatrix} \overset{p_{wd_i}}{(x)d_0 + ... + d_{2,...,x}} & \overset{p_{wpc_{i_1}}}{d_0 + ... + d_{2,...,x}} & \overset{p_{wc_{i_1}}}{d_1 + \cdots + d_{1,...,x}} & \overset{\cdots}{\cdots} & \overset{p_{wpc_{i_j}}}{d_0 + \cdots + d_{1,l}} \\ \overset{p_{wc_{i_j}}}{d_x + ... + d_{1,...,x}} & \overset{p_{dm_i}}{d_0 + \cdots + d_{1,...,x}} & \overset{p_{f_i}}{fv_i + (x)d_0 + ... + d_{2,...,x}} & \overset{p_{fv_i}}{fv_i} \end{bmatrix}^T$$

$\mathcal{N}_{(d \sqcup c)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(p2v \sqcup c)} > 0$ and $\mathcal{I}^T_{(d \sqcup c)} \times A_{(d \sqcup c)} = 0$, in which $A_{(d \sqcup c)}$ is the respective incidence matrix.

## A.23 Task Instance Conclusion Block and Join Block

$$\mathcal{I}_{(c)} = \begin{bmatrix} \overset{p_{f_i}}{fv_i + wd_i} & \overset{p_{fv_i}}{fv_i} & \overset{p_{wd_i}}{wd_i} \end{bmatrix}^T$$

$$\mathcal{I}_{(j)} = \begin{bmatrix} \overset{p_{f_1}}{f_1} & \overset{\cdots}{\cdots} & \overset{p_{f_i}}{f_i} & \overset{\cdots}{\cdots} & \overset{p_{f_n}}{f_n} & \overset{p_{end_{spec}}}{f_1 \cdots + f_i + \cdots + f_n} \end{bmatrix}^T$$

$\mathcal{N}_{(c \sqcup j)} = \mathcal{N}_c \sqcup \mathcal{N}_j$. By juxtaposition $\mathcal{I}_{(c \sqcup j)} = \mathcal{J}(\mathcal{I}_{(c)}, \mathcal{I}_{(j)})$, such that $f_i = fv_i + wd_i$:

$$\mathcal{I}_{(c \sqcup j)} = \begin{bmatrix} \overset{p_{f_1}}{f_1} & \overset{\cdots}{\cdots} & \overset{p_{f_i}}{fv_i + wd_i} & \overset{\cdots}{\cdots} & \overset{p_{f_n}}{f_n} & \overset{p_{end_{spec}}}{f_1 + \cdots + (fv_i + wd_i) + \cdots + f_n} & \overset{p_{fv_i}}{fv_i} & \overset{p_{wd_i}}{wd_i} \end{bmatrix}^T$$

As $\mathcal{I}_{(c \sqcup j)} > 0$ and $\mathcal{I}^T_{(c \sqcup j)} \times A_{(c \sqcup j)} = 0$, in which $A_{(c \sqcup j)}$ is the incidence matrix, $\mathcal{N}_{(c \sqcup j)}$ is structurally conservative as well as structurally bounded.

## A.24 Voltage Selection Block and Preemptive Task Structure with Overhead Block

$$\mathcal{I}_{(o)} = \begin{bmatrix} \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} & \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} & \overset{p_{ac_{i_j}}}{wg_{i_j}} \end{bmatrix}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T$$

$$\mathcal{I}_{(v)} = \Big[ \overset{p_{wvs_i}}{wvs_i} \quad \overset{p_{v_{i_1}}}{wvs_i} \quad \overset{\cdots}{\cdots} \quad \overset{p_{v_{i_j}}}{wvs_i} \quad \overset{\cdots}{\cdots} \quad \overset{p_{v_{i_m}}}{wvs_i} \Big]^T$$

$N_{(o \sqcup v} = N_o \sqcup N_v$. By juxtaposition $\mathcal{I}_{(o \sqcup v)} = \mathcal{J}(\mathcal{I}_{(o)}, \mathcal{I}_{(v)})$, such that $wvs_i = (C_{i_j})wg_{i_j}$:

$$\mathcal{I}_{(o \sqcup v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wvs_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{v_{i_1}}}{(C_{i_j})wg_{i_j}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{v_{i_m}}}{(C_{i_j})wg_{i_j}} \Big]^T$$

$\mathcal{N}_{(o \sqcup v)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(o \sqcup v)} > 0$ and $\mathcal{I}^T_{(o \sqcup v)} \times A_{(o \sqcup v)} = 0$, in which $A_{(o \sqcup v)}$ is the respective incidence matrix.

### A.25 Preemptive Task Structure with Overhead Block and Preemptive Task Structure with Overhead Block

$$\mathcal{I}_{(o_j)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T$$

$$\mathcal{I}_{(o_l)} = \Big[ \overset{p_{v_{i_l}}}{(C_{i_l})wg_{i_l}} \quad \overset{p_{wg_{i_l}}}{wg_{i_l}} \quad \overset{p_{wo_{i_l}}}{proc + proc\_idle + wg_{i_l}} \quad \overset{p_{wc_{i_l}}}{proc + proc\_idle + wg_{i_l}} \quad \overset{p_{ac_{i_l}}}{wg_{i_l}}$$

$$\overset{p_{wlc_{i_l}}}{proc + proc\_idle + (C_{i_l})wg_{i_l}} \quad \overset{p_{fv_{i_l}}}{proc\_idle + (C_{i_l})wg_{i_l}} \quad \overset{p_{fv_i}}{(C_{i_l})wg_{i_l}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T$$

$\mathcal{N}_{(o_j \sqcup o_l)} = \mathcal{N}_{o_j} \sqcup \mathcal{N}_{o_l}$. By juxtaposition $\mathcal{I}_{(o_k \sqcup o_j)} = \mathcal{J}(\mathcal{I}_{(o_j)}, \mathcal{I}_{(o_j)})$, such that $wg_{i_j} = (C_{i_l})wg'_{i_j}$ and $wg_{i_l} = (C_{i_j})wg'_{i_j}$:

$$\mathcal{I}_{(o_j \sqcup o_l)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j}C_{i_l})wg'_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{i_l})wg'_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{i_l})wg'_{i_j}}$$

$$\overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{i_l})wg'_{i_j}} \quad \overset{p_{ac_{i_j}}}{(C_{i_l})wg'_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j}C_{i_l})wg'_{i_j}}$$

$$\overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j}C_{i_l})wg'_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j}C_{i_l})wg'_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle}$$

$$\overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{v_{i_l}}}{(C_{i_l}C_{i_j})wg'_{i_j}} \quad \overset{p_{wg_{i_l}}}{(C_{i_j})wg'_{i_l}} \quad \overset{p_{wo_{i_l}}}{proc + proc\_idle + (C_{i_j})wg'_{i_j}}$$

$$\overset{p_{wc_{i_l}}}{proc + proc\_idle + (C_{i_j})wg'_{i_j}} \quad \overset{p_{ac_{i_l}}}{(C_{i_j})wg'_{i_j}} \quad \overset{p_{wlc_{i_l}}}{proc + proc\_idle + (C_{i_l}C_{i_j})wg'_{i_j}}$$

$$\overset{p_{fv_{i_l}}}{proc\_idle + (C_{i_l}C_{i_j})wg'_{i_j}} \ \Big]^T$$

Since $\mathcal{I}_{(o_j \sqcup o_l)} > 0$ and $\mathcal{I}^T_{(o_j \sqcup o_l)} \times A_{(o_j \sqcup o_l)} = 0$, in which $A_{(o_j \sqcup o_l)}$ is the incidence matrix, $\mathcal{N}_{(o_j \sqcup o_l)}$ is structurally conservative as well as structurally bounded.

## A.26 Preemptive Task Structure with Overhead Block and Preemptive Task Structure with Overhead and 2 Voltages Block

$$\mathcal{I}_{(o_j)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \ \Big]^T$$

$$\mathcal{I}_{(o2v_l)} = \Big[ \overset{p_{v_{i_l}}}{(C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{wg1_{i_l}}}{(C_{2_{i_l}})wg1_{i_j}} \quad \overset{p_{wo1_{i_l}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_l}}$$

$$\overset{p_{wc1_{i_l}}}{proc + proc\_idle + (C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{ac1_{i_l}}}{(C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{wlc1_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}}$$

$$\overset{p_{fv1_{i_l}}}{proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle}$$

$$\overset{p_{wg_{i_l}}}{(C_{1_{i_l}})wg_{i_l}} \quad \overset{p_{wo_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}})wg1_{i_l}} \quad \overset{p_{wc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}})wg1_{i_l}} \quad \overset{p_{ac_{i_l}}}{(C_{1_{i_l}})wg1_{i_l}}$$

$$\overset{p_{wlc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{fv_{i_l}}}{proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{fv_i}}{(C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}}$$

$$\overset{p_{wov_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \ \Big]^T$$

$\mathcal{N}_{(o_j \sqcup o2v_l)} = \mathcal{N}_{o_j} \sqcup \mathcal{N}_{o2v_l}$. By juxtaposition $\mathcal{I}_{(o_j \sqcup o2v_l)} = \mathcal{J}(\mathcal{I}_{(o_j)}, \mathcal{I}_{(o2v_l)})$, such that $wg_{i_j} = (C_{1_{i_l}}C_{2_{i_l}})wg'_{i_j}$ and $wg1_{i_l} = (C_{i_j})wg'_{i_j}$:

$$\mathcal{I}_{(o_j \sqcup o2v_l)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j}C_{1_{i_l}}C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_l}}C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg'_{i_j}}$$

$$
\begin{array}{cccc}
\overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_l}} C_{2_{i_l}}) wg'_{i_j}} & \overset{p_{ac_{i_j}}}{(C_{1_{i_l}} C_{2_{i_l}}) wg'_{i_j}} & \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j} C_{1_{i_l}} C_{2_{i_l}}) wg'_{i_j}} & \\
\overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j} C_{1_{i_l}} C_{2_{i_l}}) wg'_{i_j}} & \overset{p_{fv_i}}{(C_{i_j} C_{1_{i_l}} C_{2_{i_l}}) wg'_{i_j}} & \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} & \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \\
\overset{p_{proc_{T_k}}}{proc\_idle} & \overset{p_{proc}}{proc} & \overset{p_{proc\_idle}}{proc\_idle} & \overset{p_{v_{i_l}}}{(C_{1_{i_l}} C_{2_{i_l}} C_{i_j}) wg'_{i_j}} \quad \overset{p_{wg1_{i_l}}}{(C_{2_{i_l}} C_{i_j}) wg'_{i_j}} \\
\overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}} C_{i_j}) wg'_{i_j}} & & \overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_l}} C_{i_j}) wg'_{i_j}} & \overset{p_{ac1_{i_l}}}{(C_{2_{i_l}} C_{i_j}) wg'_{i_j}} \\
\overset{p_{wlc1_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{i_j}) wg'_{i_j}} & & \overset{p_{fv1_{i_l}}}{proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{i_j}) wg'_{i_j}} & \overset{p_{wg_{i_l}}}{(C_{1_{i_l}} C_{i_j}) wg'_{i_j}} \\
\overset{p_{wo_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{i_j}) wg'_{i_j}} & & \overset{p_{wc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{i_j}) wg'_{i_j}} & \overset{p_{ac_{i_l}}}{(C_{1_{i_l}} C_{i_j}) wg'_{i_j}} \\
\overset{p_{wlc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{i_j}) wg'_{i_j}} & & \overset{p_{fv_{i_l}}}{proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{i_j}) wg'_{i_j}} & \\
\overset{p_{wov_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{i_j}) wg'_{i_j}} & \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} & &
\end{array}^T
$$

Since $\mathcal{I}_{(o_j \sqcup o2v_l)} > 0$ and $\mathcal{I}^T_{(o_j \sqcup o2v_l)} \times A_{(o_j \sqcup o2v_l)} = 0$, in which $A_{(o_j \sqcup o2v_l)}$ is the incidence matrix, $\mathcal{N}_{(o_j \sqcup o2v_l)}$ is structurally conservative as well as structurally bounded.

## A.27 Preemptive Task Structure with Overhead Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers three *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$
\mathcal{I}_{(o)} = \left[
\begin{array}{ccccc}
\overset{p_{v_{i_j}}}{(C_{i_j}) wg_{i_j}} & \overset{p_{wg_{i_j}}}{wg_{i_j}} & \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} & \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} & \overset{p_{ac_{i_j}}}{wg_{i_j}} \\
\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j}) wg_{i_j}} & & \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j}) wg_{i_j}} & \overset{p_{fv_i}}{(C_{i_j}) wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} & \\
\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} & \overset{p_{proc_{T_k}}}{proc\_idle} & \overset{p_{proc}}{proc} & \overset{p_{proc\_idle}}{proc\_idle} &
\end{array}
\right]^T
$$

$$
\mathcal{I}_{(d)} = \left[
\begin{array}{cccc}
\overset{p_{wd_i}}{3d_0 + 2d_1 + 2d_{lj} + d_{1,lj} + 2d_j + d_{1,j} + d_{lj,j}} & & & \\
\overset{p_{wpc_{i_1}}}{d_0 + d_{lj} + d_j + d_{lj,j}} & \overset{p_{wc_{i_1}}}{d_1 + d_{1,lj} + d_{1,j} + d_{1,lj,j}} & & \\
\overset{p_{wplc_{i_j}}}{d_0 + d_1 + d_3 + d_{1,j}} & \overset{p_{wlc_{i_j}}}{d_{lj} + d_{1,lj} + d_{lj,j} + d_{1,lj,j}} & & \\
\overset{p_{wpc_{i_j}}}{d_0 + d_1 + d_{lj} + d_{1,lj}} & \overset{p_{wc_{i_j}}}{d_j + d_{1,j} + d_{lj,j} + d_{1,lj,j}} & &
\end{array}
\right]^T
$$

$$d_0 + d_1 + d_{lj} + d_j + \overset{p_{dm_i}}{d_{1,lj}} + d_{1,j} + d_{lj,j} + d_{1,lj,j} \;\Big]^T$$

In order to allow the juxtaposition, variable *proc* is replaced by $(2 \times (2^x/2 - 1) - 1)d_0$, *proc_idle* by $d_0$, and all variables in $\mathcal{I}_{(d)}$ that cover more than one *waiting for task computation* place (e.g., $d_{1j,j}$) are substituted by $2d_0$ . Additionally, variable $d_j$ and $d_{lj}$ are replaced by $wg_{i_j}$, since such variables cover only one *waiting for task computation* place in $\mathcal{I}_{(d)}$ ($p_{wc_{i_j}}$ and $p_{wlc_{i_j}}$, respectively), and $\{p_{wc_{i_j}}, p_{wlc_{i_j}}\} \subseteq P_o \cap P_d$.

The composition is demonstrated as follows. $\mathcal{N}_{(o \sqcup d)} = \mathcal{N}_o \sqcup \mathcal{N}_d$. Thus, by juxtaposition $\mathcal{I}_{(o \sqcup d)} = \mathcal{J}(\mathcal{I}_{(o)}, \mathcal{I}_{(d)})$, such that $proc = 5d_0$,$proc\_idle = d_0$ $d_{1,1j} = d_{1,j} = d_{1j,j} = d_{1,1j,j} = 2d_0$ and $d_{lj} = d_j = wg_{i_j}$:

$$\mathcal{I}_{(o \sqcup d)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{6d_0 + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{6d_0 + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{6d_0 + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{d_0 + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{d_0} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{d_0} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_k}}}{d_0} \quad \overset{p_{proc}}{5d_0} \quad \overset{p_{proc\_idle}}{d_0} \quad \overset{p_{wd_i}}{9d_0 + 4wg_{i_j} + 2d_1} \quad \overset{p_{wpc_{i_1}}}{3d_0 + 2wg_{i_j}} \quad \overset{p_{wc_{i_1}}}{d_1 + 6d_0} \quad \overset{p_{wplc_{i_j}}}{3d_0 + d_1 + wg_{i_j}}$$

$$\overset{p_{wpc_{i_j}}}{3d_0 + d_1 + wg_{i_j}} \quad \overset{p_{dm_i}}{9d_0 + d_1 + 2wg_{i_j}} \;\Big]^T$$

As $\mathcal{I}_{(o \sqcup d)} > 0$ and $\mathcal{I}^T_{(o \sqcup d)} \times A_{(o \sqcup d)} = 0$, in which $A_{(o \sqcup d)}$ is the incidence matrix, $\mathcal{N}_{(o \sqcup d)}$ is structurally conservative as well as structurally bounded.

### A.28 Preemptive Task Structure with Overhead Block and Task Instance Conclusion Block

$$\mathcal{I}_{(o)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \;\Big]^T$$

$$\mathcal{I}_{(c)} = \Big[ \overset{p_{f_i}}{fv_i + wd_i} \quad \overset{p_{fv_i}}{fv_i} \quad \overset{p_{wd_i}}{wd_i} \;\Big]^T$$

$\mathcal{N}_{(o \sqcup c)} = \mathcal{N}_o \sqcup \mathcal{N}_c$. By juxtaposition $\mathcal{I}_{(o \sqcup c)} = \mathcal{J}(\mathcal{I}_{(o)}, \mathcal{I}_{(c)})$, such that $fv_i = (C_{i_j})wg_{i_j}$:

$$\mathcal{I}_{(o \sqcup c)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{f_i}}{(C_{i_j})wg_{i_j} + wd_i} \quad \overset{p_{wd_i}}{wd_i} \ \Big]^T$$

$\mathcal{N}_{(o \sqcup c)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(o \sqcup d)} > 0$ and $\mathcal{I}^T_{(o \sqcup c)} \times A_{(o \sqcup c)} = 0$, in which $A_{(o \sqcup c)}$ is the respective incidence matrix.

## A.29 Preemptive Task Structure with Overhead and 2 Voltages Block and Task Instance Conclusion Block

$$\mathcal{I}_{(o2v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}}$$

$$proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg_{i_j}$$

$$proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$

$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \quad \Big]^T$$

$$\mathcal{I}_{(v)} = \Big[ \overset{p_{wvs_i}}{wvs_i} \quad \overset{p_{v_{i_1}}}{wvs_i} \quad \cdots \quad \overset{p_{v_{i_j}}}{wvs_i} \quad \cdots \quad \overset{p_{v_{i_m}}}{wvs_i} \Big]^T$$

$\mathcal{N}_{(o2v \sqcup v)} = \mathcal{N}_{o2v} \sqcup \mathcal{N}_v$. By juxtaposition $\mathcal{I}_{(o2v \sqcup v)} = \mathcal{J}(\mathcal{I}_{(o2v)}, \mathcal{I}_{(v)})$, such that $wvs_i = (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}$:

$$\mathcal{I}_{(o2v \sqcup v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}}$$

$$proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg_{i_j}$$

$$proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$

$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \quad \overset{p_{wvs_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{v_{i_1}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{v_{i_m}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \Big]^T$$

Since $\mathcal{I}_{(o2v \sqcup v)} > 0$ and $\mathcal{I}^T_{(o2v \sqcup v)} \times A_{(o2v \sqcup v)} = 0$, in which $A_{(o2v \sqcup v)}$ is the incidence matrix, $\mathcal{N}_{(o2v \sqcup v)}$ is structurally conservative as well as structurally bounded.

## A.30 Preemptive Task Structure with Overhead and 2 Voltages Block and Preemptive Task Structure with Overhead and 2 Voltages Block

$$\mathcal{I}_{(o2v_j)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg_{i_j}}$$

$$\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$

$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \Big]^T$$

$$\mathcal{I}_{(o2v_l)} = \Big[ \overset{p_{v_{i_l}}}{(C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{wg1_{i_l}}}{(C_{2_{i_l}})wg1_{i_j}} \quad \overset{p_{wo1_{i_l}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_l}}$$

$$\overset{p_{wc1_{i_l}}}{proc + proc\_idle + (C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{ac1_{i_l}}}{(C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{wlc1_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}}$$

$$\overset{p_{fv1_{i_l}}}{proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle}$$

$$\overset{p_{wg_{i_l}}}{(C_{1_{i_l}})wg_{i_l}} \quad \overset{p_{wo_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}})wg1_{i_l}} \quad \overset{p_{wc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}})wg1_{i_l}} \quad \overset{p_{ac_{i_l}}}{(C_{1_{i_l}})wg1_{i_l}}$$

$$\overset{p_{wlc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{fv_{i_l}}}{proc\_idle + (C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}} \quad \overset{p_{fv_i}}{(C_{1_{i_l}}C_{2_{i_l}})wg1_{i_l}}$$

$$\overset{p_{wov_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \Big]^T$$

$\mathcal{N}_{(o2v_j \sqcup o2v_l)} = \mathcal{N}_{o2v_j} \sqcup \mathcal{N}_{o2v_l}$ By juxtaposition $\mathcal{I}_{(o2v_j \sqcup o2v_l)} = \mathcal{J}(\mathcal{I}_{(o2v_j)}, \mathcal{I}_{(o2v_l)})$, such that $wg_{i_j} = (C_{1_{i_l}}C_{2_{i_l}})wg'_{i_j}$ and $wg_{i_l} = (C_{1_{i_j}}C_{2_{i_j}})wg'_{i_j}$:

$$\mathcal{I}_{(o2v_j \sqcup o2v_l)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}}$$

$$\overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}}$$

$$\overset{p_{ac1_{i_j}}}{(C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}}$$

$$\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \cdots \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$

$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}}$$

$$\overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{ac_{i_j}}}{(C_{1_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}}$$

$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}}$$

$$\overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{1_{i_l}} C_{2_{i_l}})wg'_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle}$$

$$\overset{p_{v_{i_l}}}{(C_{1_{i_l}} C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{wg1_{i_l}}}{(C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{wo1_{i_l}}}{proc + proc\_idle + (C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}}$$

$$\overset{p_{wc1_{i_l}}}{proc + proc\_idle + (C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{ac1_{i_l}}}{(C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}}$$

$$\overset{p_{wlc1_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{fv1_{i_l}}}{proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}}$$

$$\overset{p_{wg_{i_l}}}{(C_{1_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{wo_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}}$$

$$\overset{p_{wc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{ac_{i_l}}}{(C_{1_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}}$$

$$\overset{p_{wlc_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{fv_{i_l}}}{proc\_idle + (C_{1_{i_l}} C_{2_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}}$$

$$\overset{p_{wov_{i_l}}}{proc + proc\_idle + (C_{1_{i_l}} C_{1_{i_j}} C_{2_{i_j}})wg'_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \Big]^T$$

$\mathcal{N}_{(o2v_j \sqcup o2v_l)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(o2v_j \sqcup o2v_l)} > 0$ and $\mathcal{I}^T_{(o2v_j \sqcup o2v_l)} \times A_{(o2v_j \sqcup o2v_l)} = 0$, in which $A_{(o2v_j \sqcup o2v_l)}$ is the respective incidence matrix.

## A.31 Preemptive Task Structure with Overhead and 2 Voltages Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers three *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$\mathcal{I}_{(o2v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg_{i_j}}$$

$$\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$

$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}}$$

$$\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}}$$

$$\overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \Big]^T$$

$$\mathcal{I}_{(d)} = \Big[ \overset{p_{wd_i}}{4d_0 + 3d_{l1j} + 3d_{1j} + 2d_{l1j,1j} + 3d_{lj} + 2d_{l1j,lj} + 2d_{1j,lj} + d_{l1j,1j,lj} + 3d_j}$$

$$+2d_{l1j,j} + 2d_{1j,j} + d_{l1j,1j,j} + 2d_{lj,j} + d_{l1j,lj,j} + d_{1j,lj,j}$$

$$\overset{p_{wplc1_{i_j}}}{d_0 + d_{1j} + d_{lj} + d_{1j,lj} + d_j + d_{1j,j} + d_{lj,j} + d_{1j,lj,j}}$$

$$\overset{p_{wlc1_{i_j}}}{d_{l1j} + d_{l1j,1j} + d_{l1j,lj} + d_{l1j,1j,lj} + d_{l1j,j} + d_{l1j,1j,j} + d_{l1j,lj,j} + d_{l1j,1j,lj,j}}$$

$$\overset{p_{wpc1_{i_j}}}{d_{1j} + d_{l1j,1j} + d_{1j,lj} + d_{l1j,1j,lj} + d_{1j,j} + d_{l1j,1j,j} + d_{1j,lj,j} + d_{l1j,1j,lj,j}}$$

$$\overset{p_{wc1_{i_j}}}{d_{1j} + d_{l1j,1j} + d_{1j,lj} + d_{l1j,1j,lj} + d_{1j,j} + d_{l1j,1j,j} + d_{1j,lj,j} + d_{l1j,1j,lj,j}}$$

$$\overset{p_{wplc_{i_j}}}{d_0 + d_{l1j} + d_{1j} + d_{l1j,1j} + d_j + d_{l1j,j} + d_{1j,j} + d_{l1j,1j,j}}$$

$$\overset{p_{wlc_{i_j}}}{d_{lj} + d_{l1j,lj} + d_{1j,lj} + d_{l1j,1j,lj} + d_{lj,j} + d_{l1j,lj,j} + d_{1j,lj,j} + d_{l1j,1j,lj,j}}$$

$$\overset{p_{wpc_{i_j}}}{d_0 + d_{l1j} + d_{l1j,1j} + d_{1j} + d_{lj} + d_{l1j,lj} + d_{1j,lj} + d_{l1j,1j,lj}}$$

$$\overset{p_{wc_{i_j}}}{d_j + d_{l1j,j} + d_{1j,j} + d_{l1j,1j,j} + d_{lj,j} + d_{l1j,lj,j} + d_{1j,lj,j} + d_{l1j,1j,lj,j}}$$

$$\overset{p_{dm_i}}{d_0 + d_{l1j} + d_{l1j,1j} + d_{1j} + d_{lj} + d_{l1j,lj} + d_{1j,lj} + d_{l1j,1j,lj} + d_j + d_{l1j,j} + d_{1j,j} + d_{l1j,1j,j}}$$

$$+d_{lj,j} + d_{l1j,lj,j} + d_{1j,lj,j} + d_{l1j,1j,lj,j} \Big]^T$$

In order to allow the juxtaposition, variable *proc* is replaced by $(2 \times (2^x/2 - 1) -$

1)$d_0$, *proc_idle* by $d_0$, and all variables in $\mathcal{I}_{(d)}$ that cover more than one *waiting for task computation* place (e.g., $d_{1j,j}$) are substituted by $2d_0$ . Additionally, variable $d_{l1j}$, $d_{lj}$, $d_j$ and $d_{lj}$ are replaced by $(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}$,$(C_{2_{i_j}})wg1_{i_j}$,$(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}$, and $(C_{1_{i_j}})wg1_{i_j}$, respectively, since such variables cover only one *waiting for task computation* place in $\mathcal{I}_{(d)}$ ($p_{wc1_{i_j}}$,$p_{wlc1_{i_j}}$,$p_{wc_{i_j}}$ and $p_{wlc_{i_j}}$), and $\{p_{wc1_{i_j}}, p_{wlc1_{i_j}}, p_{wc_{i_j}}, p_{wlc_{i_j}}\} \subseteq P_{o2v} \cap P_d$.

The composition is demonstrated as follows. $\mathcal{N}_{(o2v \sqcup d)} = \mathcal{N}_{o2v} \sqcup \mathcal{N}_d$. Thus, by juxtaposition $= \mathcal{I}_{(o2v \sqcup d)} = \mathcal{J}(\mathcal{I}_{(o2v)}, \mathcal{I}_{(d)})$, such that $proc = 13d_0$,$proc\_idle = d_0$,$d_{l1j,1j} = d_{l1j,lj} = d_{1j,lj} = d_{l1j,1j,lj} = d_{l1j,j} = d_{1j,j} = d_{l1j,1j,j}d_{lj,j} = d_{l1j,lj,j} = d_{1j,lj,j} = d_{l1j,1j,lj,j} = 2d_0$,$d_{l1j} = (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}$, $d_{1j} = (C_{2_{i_j}})wg1_{i_j}$, $d_{lj} = (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}$,$d_j = (C_{1_{i_j}})wg1_{i_j}$:

$$
\begin{aligned}
\mathcal{I}_{(o2v \sqcup d)} = \Big[ & (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{v_{i_j}}} \quad (C_{2_{i_j}})wg1_{i_j}^{p_{wg1_{i_j}}} \quad 14d_0 + (C_{2_{i_j}})wg1_{i_j}^{p_{wo1_{i_j}}} \quad 14d_0 + (C_{2_{i_j}})wg1_{i_j}^{p_{wc1_{i_j}}} \\
& (C_{2_{i_j}})wg_{i_j}^{p_{ac1_{i_j}}} \quad 14d_0 + (C_{1_{i_j}}C_{2_{i_j}})wg_{i_j}^{p_{wlc1_{i_j}}} \quad d_0 + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{fv1_{i_j}}} \quad d_0^{p_{proc_{T_1}}} \quad \cdots^{\cdots} \quad d_0^{p_{proc_{T_i}}} \\
& \cdots^{\cdots} \quad d_0^{p_{proc_{T_k}}} \quad 13d_0^{p_{proc}} \quad d_0^{p_{proc\_idle}} \quad (C_{1_{i_j}})wg1_{i_j}^{p_{wg_{i_j}}} \quad 14d_0 + (C_{1_{i_j}})wg1_{i_j}^{p_{wo_{i_j}}} \\
& 14d_0 + (C_{1_{i_j}})wg1_{i_j}^{p_{wc_{i_j}}} \quad (C_{1_{i_j}})wg_{i_j}^{p_{ac_{i_j}}} \quad 14d_0 + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{wlc_{i_j}}} \quad d_0 + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{fv_{i_j}}} \\
& (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{fv_i}} \quad 14d_0 + (C_{1_{i_j}})wg1_{i_j}^{p_{wov_{i_j}}} \quad d_0^{p_{proc_{T_i}\_2volt}} \\
& 36d_0 + 3(2C_{1_{i_j}}C_{2_{i_j}} + C_{2_{i_j}} + C_{1_{i_j}})wg1_{i_j}^{p_{wd_i}} \\
& 9d_0 + 3wg1_{i_j}^{p_{wplc1_{i_j}}} \quad 9d_0 + 3wg1_{i_j}^{p_{wpc1_{i_j}}} \quad 9d_0 + 3wg1_{i_j}^{p_{wplc_{i_j}}} \quad 9d_0 + 3wg1_{i_j}^{p_{wpc_{i_j}}} \\
& 23d_0 + (2C_{1_{i_j}}C_{2_{i_j}} + C_{2_{i_j}} + C_{1_{i_j}})wg1_{i_j}^{p_{dm_i}} \Big]^T
\end{aligned}
$$

As $\mathcal{I}_{(o2v \sqcup d)} > 0$ and $\mathcal{I}^T_{(o2v \sqcup d)} \times A_{(o2v \sqcup d)} = 0$, in which $A_{(o2v \sqcup d)}$ is the incidence matrix, $\mathcal{N}_{(o2v \sqcup d)}$ is structurally conservative as well as structurally bounded.

## A.32 Preemptive Task Structure with Overhead and 2 Voltages Block and Task Instance Conclusion Block

$$
\begin{aligned}
\mathcal{I}_{(o2v)} = \Big[ & (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{v_{i_j}}} \quad (C_{2_{i_j}})wg1_{i_j}^{p_{wg1_{i_j}}} \quad proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}^{p_{wo1_{i_j}}} \\
& proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}^{p_{wc1_{i_j}}} \quad (C_{2_{i_j}})wg_{i_j}^{p_{ac1_{i_j}}} \quad proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg_{i_j}^{p_{wlc1_{i_j}}} \\
& proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}^{p_{fv1_{i_j}}} \quad proc\_idle^{p_{proc_{T_1}}} \quad \cdots^{\cdots} \quad proc\_idle^{p_{proc_{T_i}}} \quad \cdots^{\cdots} \quad proc\_idle^{p_{proc_{T_k}}} \quad proc^{p_{proc}} \\
& proc\_idle^{p_{proc\_idle}} \quad (C_{1_{i_j}})wg1_{i_j}^{p_{wg_{i_j}}} \quad proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}^{p_{wo_{i_j}}} \quad proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}^{p_{wc_{i_j}}}
\end{aligned}
$$

$$
\begin{array}{l}
\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \\
\overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \quad \Big]^T \\
\overset{p_{f_i}}{\mathcal{I}_{(c)} = \quad \big[ \; fv_i + wd_i} \quad \overset{p_{fv_i}}{fv_i} \quad \overset{p_{wd_i}}{wd_i} \; \big]^T
\end{array}
$$

$\mathcal{N}_{(o2v \sqcup c)} = \mathcal{N}_{o2v} \sqcup \mathcal{N}_c$. By juxtaposition $\mathcal{I}_{(o2v \sqcup c)} = \mathcal{J}(\mathcal{I}_{(o2v)}, \mathcal{I}_{(c)})$, such that $fv_i = (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}$:

$$
\begin{array}{l}
\mathcal{I}_{(o2v \sqcup c)} = \quad \Big[ \; \overset{p_{v_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}} \\
\overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg_{i_j}} \\
\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \\
\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \\
\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \\
\overset{p_{fv_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \quad \overset{p_{f_i}}{(C_{1_{i_j}}C_{2_{i_j}})wg1_{i_j} + wd_i} \quad \overset{p_{wd_i}}{wd_i} \; \Big]^T
\end{array}
$$

$\mathcal{N}_{(o2v \sqcup c)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(o2v \sqcup c)} > 0$ and $\mathcal{I}^T_{(o2v \sqcup c)} \times A_{(o2v \sqcup c)} = 0$, in which $A_{(o2v \sqcup c)}$ is the respective incidence matrix.

### A.33 Task Instance Conclusion with Inter-task Relations Block and Join Block

$$
\begin{array}{l}
\mathcal{I}_{(cinter)} = \quad \Big[ \; \overset{p_{fv_i}}{pf1_i + rel1_i + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i} \\
\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \; \Big]^T \\
\overset{p_{f_1}}{\mathcal{I}_{(j)} = \quad \big[ \; f_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{f_i}}{f_i} \quad \overset{\cdots}{\cdots} \quad \overset{p_{f_n}}{f_n} \quad \overset{p_{end_{spec}}}{f_1 \cdots + f_i + \cdots + f_n} \; \big]^T
\end{array}
$$

$\mathcal{N}_{(cinter \sqcup j)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_j$. By juxtaposition $\mathcal{I}_{(cinter \sqcup j)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(j)})$, such that $f_i = pf1_i + pf_i$:

$$
\begin{array}{l}
\mathcal{I}_{(cinter \sqcup j)} = \quad \Big[ \; \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i} \\
\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z}
\end{array}
$$

$$\overset{p_{f_1}}{f_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{f_n}}{f_n} \quad \overset{p_{end_{spec}}}{f_1 \cdots + pf1_i + pf_i + \cdots + f_n} \;\Big]^T$$

$\mathcal{N}_{(cinter \sqcup j)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(cinter \sqcup j)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup j)} \times A_{(cinter \sqcup j)} = 0$, in which $A_{(cinter \sqcup j)}$ is the respective incidence matrix.

### A.34 Task Instance Conclusion with Inter-task Relations Block and Non-preemptive Task Structure Block

$$\mathcal{I}_{(cinter)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$

$$\mathcal{I}_{(np)} = \Big[ \overset{p_{v_{i_j}}}{wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup np)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{np}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup np)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(np)})$ and assuming $x$ is the number of inter-task relations in $\mathcal{N}_{(cinter)}$, such that $wg_{i_j} = (z+1)wg'_{i_j}$ and $pf1_1 = rel1_1 = \cdots = rel1_p = \cdots = rel1_z = wg'_{i_j}$:

$$\mathcal{I}_{(cinter \sqcup np)} = \Big[ \overset{p_{fv_i}}{(z+1)wg'_{i_j}} \quad \overset{p_{f_i}}{wg'_{i_j} + pf_i} \quad \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z}$$
$$\overset{p_{rel_1}}{wg'_{i_j} + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{wg'_{i_j} + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{wg'_{i_j} + rel_z} \quad \overset{p_{v_{i_j}}}{(z+1)wg'_{i_j}} \quad \overset{p_{wg_{i_j}}}{(z+1)wg'_{i_j}}$$
$$\overset{p_{wc_{i_j}}}{(z+1)wg'_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(z+1)wg'_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

As $\mathcal{I}_{(cinter \sqcup np)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup np)} \times A_{(cinter \sqcup np)} = 0$, in which $A_{(cinter \sqcup np)}$ is the incidence matrix, $\mathcal{N}_{(cinter \sqcup np)}$ is structurally conservative as well as structurally bounded.

### A.35 Task Instance Conclusion with Inter-task Relations Block and Preemptive Task Structure Block

$$\mathcal{I}_{(cinter)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$

$$\mathcal{I}_{(p)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{wg_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup p)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{np}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup p)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(p)})$ and assuming $z$ is the number of inter-task relations in $\mathcal{N}_{(cinter)}$, such that $wg_{i_j} = (z+1)wg'_{i_j}$ and $pf1_i = rel1_1 = \cdots = rel1_p = \cdots = rel1_z = (C_{i_j})wg'_{i_j}$:

$$\mathcal{I}_{(cinter \sqcup p)} = \Big[ \overset{p_{fv_i}}{(z+1)(C_{i_j})wg_{i_j}} \quad \overset{p_{f_i}}{(C_{i_j})wg_{i_j} + pf_i} \quad \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z}$$

$$\overset{p_{rel_1}}{(C_{i_j})wg_{i_j} + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{(C_{i_j})wg_{i_j} + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{(C_{i_j})wg_{i_j} + rel_z}$$

$$\overset{p_{v_{i_j}}}{(C_{i_j})(z+1)wg'_{i_j}} \quad \overset{p_{wg_{i_j}}}{(z+1)wg'_{i_j}} \quad \overset{p_{wc_{i_j}}}{(z+1)wg'_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(z+1)wg'_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

Since $\mathcal{I}_{(cinter \sqcup np)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup p)} \times A_{(cinter \sqcup p)} = 0$, in which $A_{(cinter \sqcup p)}$ is the incidence matrix, $\mathcal{N}_{(cinter \sqcup p)}$ is structurally conservative as well as structurally bounded.

## A.36 Task Instance Conclusion with Inter-task Relations Block and Preemptive Task Structure with Overhead Block

$$\mathcal{I}_{(cinter)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$

$$\mathcal{I}_{(o)} = \Big[ \overset{p_{v_{i_j}}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{wg_{i_j}}}{wg_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + wg_{i_j}} \quad \overset{p_{ac_{i_j}}}{wg_{i_j}}$$
$$\overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_{i_j}} \quad \overset{p_{fv_i}}{(C_{i_j})wg_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \Big]^T$$
$$\overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup o)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{np}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup o)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(o)})$ and assuming $z$ is the number of inter-task relations in $\mathcal{N}_{(cinter)}$, such that $wg_{i_j} = (z+1)wg'_{i_j}$ and $pf1_i = rel1_1 = \cdots = rel1_p = \cdots = rel1_z = (C_{i_j})wg'_{i_j}$:

$$\mathcal{I}_{(cinter \sqcup o)} = \Big[ \overset{p_{fv_i}}{(z+1)(C_{i_j})wg'_{i_j}} \quad \overset{p_{f_i}}{(C_{i_j})wg'_{i_j} + pf_i} \quad \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z}$$
$$\overset{p_{rel_1}}{(C_{i_j})wg'_{i_j} + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{(C_{i_j})wg'_{i_j} + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{(C_{i_j})wg'_{i_j} + rel_z} \quad \overset{p_{v_{i_j}}}{(C_{i_j})(z+1)wg'_{i_j}}$$
$$\overset{p_{wg_{i_j}}}{(z+1)wg'_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (z+1)wg'_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (z+1)wg'_{i_j}}$$
$$\overset{p_{ac_{i_j}}}{(z+1)wg'_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{i_j})(z+1)wg'_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{i_j})wg_(x+1)wg'_{i_j}}$$
$$\overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc} \quad \overset{p_{proc\_idle}}{proc\_idle} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup o)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(cinter \sqcup o)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup o)} \times A_{(cinter \sqcup o)} = 0$, in which $A_{(cinter \sqcup o)}$ is the respective incidence matrix.

### A.37 Task Instance Conclusion with Inter-task Relations Block and Preemptive Task Structure with Overhead and 2 Voltages Block

$$\mathcal{I}_{(cinter)} = \Big[\ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p} + \cdots + \overset{p_{f_i}}{rel1_z} \quad pf1_i + pf_i$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p} + \cdots + rel_z \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z}\ \Big]^T$$

$$\mathcal{I}_{(o2v)} = \Big[\ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}}$$
$$\overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})wg_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg_{i_j}}$$
$$\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$
$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{wc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}}$$
$$\overset{p_{ac_{i_j}}}{(C_{1_{i_j}})wg_{i_j}} \quad \overset{p_{wlc_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}}$$
$$\overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wov_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \quad \Big]^T$$

$\mathcal{N}_{(cinter \sqcup o2v)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{o2v}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup o)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(o)})$ and assuming $z$ is the number of inter-task relations in $\mathcal{N}_{(cinter)}$, such that $wg_{i_j} = (z+1)wg1'_{i_j}$ and $pf1_i = rel1_1 = \cdots = rel1_p = \cdots = rel1_z = (C_{1_{i_j}} C_{2_{i_j}})wg1'_{i_j}$:

$$\mathcal{I}_{(cinter \sqcup o2v)} = \Big[\ \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{i_j}} \quad \overset{p_{f_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{i_j} + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p} + \cdots + rel_z \quad \overset{p_{rel_1}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{i_j} + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{i_j} + rel_p}$$
$$\overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{i_j} + rel_z} \quad \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})(z+1)wg1'_{i_j}}$$
$$\overset{p_{wo1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})(z+1)wg1'_{i_j}} \quad \overset{p_{wc1_{i_j}}}{proc + proc\_idle + (C_{2_{i_j}})(z+1)wg1'_{i_j}}$$
$$\overset{p_{ac1_{i_j}}}{(C_{2_{i_j}})(z+1)wg1'_{i_j}} \quad \overset{p_{wlc1_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{i_j}}$$
$$\overset{p_{fv1_{i_j}}}{proc\_idle + (C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{i_j}} \quad \overset{p_{proc_{T_1}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_i}}}{proc\_idle} \quad \overset{\cdots}{\cdots} \quad \overset{p_{proc_{T_k}}}{proc\_idle} \quad \overset{p_{proc}}{proc}$$
$$\overset{p_{proc\_idle}}{proc\_idle} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})(z+1)wg1'_{i_j}} \quad \overset{p_{wo_{i_j}}}{proc + proc\_idle + (C_{1_{i_j}})(z+1)wg1'_{i_j}}$$

$$proc + proc\_idle + \overset{p_{wc_{i_j}}}{(C_{1_{i_j}})}(z+1)wg1'_{i_j} \quad \overset{p_{ac_{i_j}}}{(C_{1_{i_j}})}wg_{i_j}$$

$$proc + proc\_idle + \overset{p_{wlc_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})}(z+1)wg1'_{i_j} \quad proc\_idle + \overset{p_{fv_{i_j}}}{(C_{1_{i_j}}C_{2_{i_j}})}(z+1)wg1'_{i_j}$$

$$proc + proc\_idle + \overset{p_{wov_{i_j}}}{(C_{1_{i_j}})}(z+1)wg1'_{i_j} \quad \overset{p_{proc_{T_i}\_2volt}}{proc\_idle} \Big]^T$$

As $\mathcal{I}_{(cinter \sqcup o2v)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup o2v)} \times A_{(cinter \sqcup o2v)} = 0$, in which $A_{(cinter \sqcup o2v)}$ is the incidence matrix, $\mathcal{N}_{(cinter \sqcup o2v)}$ is structurally conservative as well as structurally bounded.

## A.38 Task Instance Conclusion with Inter-task Relations Block and Non-Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(cinter)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$

$$\mathcal{I}_{(np2v)} = \Big[ \overset{p_{v_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wg_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wc_{i_j}}}{wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{wg1_{i_j}} \quad \overset{p_{fv_i}}{wg1_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup np2v)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{np2v}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup np2v)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(np2v)})$ and assuming $x$ is the number of inter-task relations in $\mathcal{N}_{(cinter)}$, such that $wg1_{i_j} = (z+1)wg1'_{i_j}$ and $pf1_i = rel1_1 = \cdots = rel1_p = \cdots = rel1_z = wg1'_{i_j}$:

$$\mathcal{I}_{(cinter \sqcup np2v)} = \Big[ \overset{p_{fv_i}}{(z+1)wg1'_{i_j}} \quad \overset{p_{f_i}}{wg1'_{i_j} + pf_i} \quad \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z}$$
$$\overset{p_{rel_1}}{wg1'_{i_j} + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{wg1'_{i_j} + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{wg1'_{i_j} + rel_z} \quad \overset{p_{v_{i_j}}}{(z+1)wg1'_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(z+1)wg1'_{i_j}}$$
$$\overset{p_{wc1_{i_j}}}{(z+1)wg1'_{i_j} + proc} \quad \overset{p_{wg_{i_j}}}{(z+1)wg1'_{i_j} + proc} \quad \overset{p_{wc_{i_j}}}{(z+1)wg1'_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(z+1)wg1'_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

Since $\mathcal{I}_{(cinter \sqcup np2v)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup np2v)} \times A_{(cinter \sqcup np2v)} = 0$, in which $A_{(cinter \sqcup np2v)}$ is the incidence matrix, $\mathcal{N}_{(cinter \sqcup np2v)}$ is structurally conservative as well as structurally bounded.

## A.39 Task Instance Conclusion with Inter-task Relations Block and Preemptive Task Structure with 2 Voltages Block

$$\mathcal{I}_{(cinter)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$

$$\mathcal{I}_{(p2v)} = \Big[ \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}}$$
$$\overset{p_{wc_{i_j}}}{(C_{1_{i_j}})wg1_{i_j} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})wg1_{i_j}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1_{i_j}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup p2v)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{p2v}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup p2v)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(p2v)})$ and assuming $z$ is the number of inter-task relations in $\mathcal{N}_{(cinter)}$, such that $wg1_{i_j} = (z+1)wg1'_{ij}$ and $pf1_i = rel1_1 = \cdots = rel1_p = \cdots = rel1_z = (C_{1_{i_j}} C_{2_{i_j}})wg1'_{ij}$:

$$\mathcal{I}_{(cinter \sqcup p2v)} = \Big[ \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{ij}} \quad \overset{p_{f_i}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{ij} + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{ij} + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{ij} + rel_p}$$
$$\overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{(C_{1_{i_j}} C_{2_{i_j}})wg1'_{ij} + rel_z} \quad \overset{p_{v_{i_j}}}{(C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{ij}} \quad \overset{p_{wg1_{i_j}}}{(C_{2_{i_j}})(z+1)wg1'_{ij}}$$
$$\overset{p_{wc1_{i_j}}}{(C_{2_{i_j}})(z+1)wg1'_{ij} + proc} \quad \overset{p_{wf1_{i_j}}}{(C_{2_{i_j}})(z+1)wg1'_{ij}} \quad \overset{p_{wg_{i_j}}}{(C_{1_{i_j}})(z+1)wg1'_{ij}}^T$$
$$\overset{p_{wc_{i_j}}}{(C_{1_{i_j}})(z+1)wg1'_{ij} + proc} \quad \overset{p_{wf_{i_j}}}{(C_{1_{i_j}})(z+1)wg1'_{ij}} \quad \overset{p_{fv_i}}{(C_{1_{i_j}} C_{2_{i_j}})(z+1)wg1'_{ij}} \quad \overset{p_{proc}}{proc} \Big]^T$$

$\mathcal{N}_{(cinter \sqcup p2v)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(cinter \sqcup np2v)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup p2v)} \times A_{(cinter \sqcup p2v)} = 0$, in which $A_{(cinter \sqcup p2v)}$ is the respective incidence matrix.

### A.40 Task Instance Conclusion with Inter-task Relations Block and Task Instance Conclusion with Inter-task Relations Block

$$\mathcal{I}_{(cinter_i)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$
$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$
$$\mathcal{I}_{(cinter_j)} = \Big[ \overset{p_{fv_j}}{pf1_i + rel1_j + \cdots + rel1p_j + \cdots + rel1z_j} \quad \overset{p_{f_j}}{pf1_j + pf_j}$$
$$\overset{p_{pwd_j}}{pf_j + rel1_j + \cdots + relp_j + \cdots + relz_j} \quad \overset{p_{rel_{1_j}}}{rel1_j + rel_j} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_{p_i}}}{rel1p_j + relp_j} \quad \overset{\cdots}{\cdots}$$
$$\overset{p_{rel_{z_j}}}{rel1z_j + relz_j} \Big]^T$$

Assume that place renaming function has been applied in net $\mathcal{N}_{cinter_j}$, renaming place $p_{rel_{p_j}}$ to $p_{rel_{p_i}}$. $\mathcal{N}_{(cinter_i \sqcup cinter_j)} = \mathcal{N}_{cinter_i} \sqcup \mathcal{N}_{cinter_j}$. By juxtaposition $\mathcal{I}_{(cinter_i \sqcup cinter_j)} = \mathcal{J}(\mathcal{I}_{(cinter_i)}, \mathcal{I}_{(cinter_j)})$, such that $rel1p_j = rel1p_i$ and $relp_j = relp_i$:

$$\mathcal{I}_{(cinter_i \sqcup cinter_j)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_i + \cdots + rel1p_i + \cdots + rel1z_i} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$

$$pf_i + rel1_i + \cdots + \overset{p_{pwd_i}}{relp_i} + \cdots + relz_i \quad \overset{p_{rel_{1_i}}}{rel1_i + rel_i} \quad \cdots \quad \overset{p_{rel_{p_i}}}{rel1p_i + relp_i} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{rel_{z_i}}}{rel1z_i + relz_i} \quad \overset{p_{fv_j}}{pf1_i + rel1_j + \cdots + rel1p_i + \cdots + rel1z_j} \quad \overset{p_{f_j}}{pf1_j + pf_j}$$

$$pf_j + rel1_j + \cdots + \overset{p_{pwd_j}}{relp_i} + \cdots + relz_j \quad \overset{p_{rel_{1_j}}}{rel1_j + rel_j} \quad \cdots \quad \overset{p_{rel_{z_j}}}{rel1z_j + relz_j} \Big]^T$$

As $\mathcal{I}_{(cinter_i \sqcup cinter_j)} > 0$ and $\mathcal{I}^T_{(cinter_i \sqcup cinter_j)} \times A_{(cinter_i \sqcup cinter_j)} = 0$, in which $A_{(cinter_i \sqcup cinter_j)}$ is the incidence matrix, $\mathcal{N}_{(cinter_i \sqcup cinter_j)}$ is structurally conservative as well as structurally bounded.

### A.41 Task Instance Conclusion with Inter-task Relations Block and Exclusion Pre-Condition Block

$$\mathcal{I}_{(cinter)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$

$$\overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} \quad \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{excl_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \Big]^T$$

$$\mathcal{I}_{(ipree)} = \Big[ \overset{p_{wexcl_i}}{wexcl_i} \quad \overset{p_{wvs_i}}{wexcl_i + excl_{1_i} + \cdots + excl_{p_i} + \cdots + excl_{y_i}} \quad \overset{p_{excl_{1_i}}}{excl_{1_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{excl_{p_i}}}{excl_{p_i}} \quad \overset{\cdots}{\cdots}$$

$$\overset{p_{excl_{y_i}}}{excl_{y_i}} \Big]^T$$

Assume that place renaming function has been applied in net $\mathcal{N}_{cinter}$, renaming place $p_{rel_{p_i}}$ to $p_{excl_{p_i}}$. $\mathcal{N}_{(cinter \sqcup ipree)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{ipree}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup ipree)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(ipree)})$, such that $excl_{p_i} = rel1_i + rel_i$:

$$\mathcal{I}_{(cinter \sqcup ipree)} = \Big[ \overset{p_{fv_i}}{pf1_i + rel1_i + \cdots + rel1p_i + \cdots + rel1z_i} \quad \overset{p_{f_i}}{pf1_i + pf_i}$$

$$pf_i + rel1_i + \cdots + \overset{p_{pwd_i}}{relp_i} + \cdots + relz_i \quad \overset{p_{rel_{1_i}}}{rel1_i + rel_i} \quad \cdots \quad \overset{p_{excl_{p_i}}}{rel1p_i + relp_i} \quad \cdots \quad \overset{p_{rel_{z_i}}}{rel1z_i + rez_i}$$

$$\overset{p_{wexcl_i}}{wexcl_i} \quad \overset{p_{wvs_i}}{wexcl_i + excl_{1_i} + \cdots + rel1_i + rel_i + \cdots + excl_{y_i}} \quad \overset{p_{excl_{1_i}}}{excl_{1_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{excl_{y_i}}}{excl_{y_i}} \Big]^T$$

Since $\mathcal{I}_{(cinter \sqcup ipree)} > 0$ and $\mathcal{I}^T_{(cinter \sqcup ipree)} \times A_{(cinter \sqcup ipree)} = 0$, in which $A_{(cinter \sqcup ipree)}$ is the incidence matrix, $\mathcal{N}_{(cinter \sqcup ipree)}$ is structurally conservative as well as structurally bounded.

### A.42 Task Instance Conclusion with Inter-task Relations Block and Precedence Pre-Condition Block

$$\mathcal{I}_{(cinter)} = \begin{bmatrix} \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} & \overset{p_{f_i}}{pf1_i + pf_i} \\ \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} & \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{prec_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \end{bmatrix}^T$$

$$\mathcal{I}_{(iprep)} = \begin{bmatrix} \overset{p_{wprec_i}}{wprec_i} & \overset{p_{wexcl_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & \overset{p_{prec_{1_i}}}{prec_{1_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{prec_{p_i}}}{prec_{p_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{prec_{x_i}}}{prec_{x_i}} \end{bmatrix}^T$$

Assume that place renaming function has been applied in net $\mathcal{N}_{cinter}$, renaming place $p_{rel_{p_i}}$ to $p_{prec_{p_i}}$. $\mathcal{N}_{(cinter \sqcup iprep)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_{iprep}$. By juxtaposition $\mathcal{I}_{(cinter \sqcup iprep)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(iprep)})$, such that $prec_{p_i} = rel1_i + rel_i$:

$$\mathcal{I}_{(cinter \sqcup iprep)} = \begin{bmatrix} \overset{p_{fv_i}}{pf1_i + rel1_i + \cdots + rel1p_i + \cdots + rel1z_i} & \overset{p_{f_i}}{pf1_i + pf_i} \\ \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + relp_i + \cdots + relz_i} & \overset{p_{rel_{1_i}}}{rel1_i + rel_i} \quad \overset{\cdots}{\cdots} \quad \overset{p_{precp_i}}{rel1p_i + relp_i} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_{z_i}}}{rel1x_i + relx_i} \\ \overset{p_{wprec_i}}{wprec_i} & \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + rel1_i + rel_i + \cdots + prec_{x_i}} & \overset{p_{prec_{1_i}}}{prec_{1_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{prec_{x_i}}}{prec_{x_i}} \end{bmatrix}^T$$

As $\mathcal{I}_{(cinter \sqcup ipree)} > 0$ and $\mathcal{I}^T_{(cinter_i \sqcup iprep)} \times A_{(cinter_i \sqcup iprep)} = 0$, in which $A_{(cinter_i \sqcup iprep)}$ is the incidence matrix, $\mathcal{N}_{(cinter_i \sqcup iprep)}$ is structurally conservative as well as structurally bounded.

### A.43 Task Instance Conclusion with Inter-task Relations Block and Deadline Checking Block

For the sake of understandability, the deadline checking block considers three *waiting for task computation* places in this composition. Nevertheless, the approach is the same for deadline checking blocks containing any number ($x$) of such places.

$$\mathcal{I}_{(cinter)} = \begin{bmatrix} \overset{p_{fv_i}}{pf1_i + rel1_1 + \cdots + rel1_p + \cdots + rel1_z} & \overset{p_{f_i}}{pf1_i + pf_i} \\ \overset{p_{pwd_i}}{pf_i + rel_1 + \cdots + rel_p + \cdots + rel_z} & \overset{p_{rel_1}}{rel1_1 + rel_1} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + rel_p} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + rel_z} \end{bmatrix}^T$$

$$\mathcal{I}_{(d)} = \begin{bmatrix} \overset{p_{wd_i}}{3d_0 + 2d_1 + 2d_{lj} + d_{1,lj} + 2d_j + d_{1,j} + d_{lj,j}} \\ \overset{p_{wpc_{i_1}}}{d_0 + d_{lj} + d_j + d_{lj,j}} & \overset{p_{wc_{i_1}}}{d_1 + d_{1,lj} + d_{1,j} + d_{1,lj,j}} \\ \overset{p_{wplc_{i_j}}}{d_0 + d_1 + d_3 + d_{1,j}} & \overset{p_{wlc_{i_j}}}{d_{lj} + d_{1,lj} + d_{lj,j} + d_{1,lj,j}} \\ \overset{p_{wpc_{i_j}}}{d_0 + d_1 + d_{lj} + d_{1,lj}} & \overset{p_{wc_{i_j}}}{d_j + d_{1,j} + d_{lj,j} + d_{1,lj,j}} \end{bmatrix}^T$$

$$\overset{p_{dm_i}}{d_0 + d_1 + d_{lj} + d_j + d_{1,lj} + d_{1,j} + d_{lj,j} + d_{1,lj,j}}\ \Big]^T$$

Since $P_{cinter} \cap P_d = \{p_{pwd_i}\}$, in which $P_{cinter}$ and $P_d$ are the set of places of the respective Petri nets, only the weight associated with place $p_{pwd_i}$ needs to be adjusted in the P-invariants $\mathcal{I}_{(d)}$ and $\mathcal{I}_{(cinter)}$ (in order to allow the juxtaposition). Firstly, in $\mathcal{I}_{(cinter)}$, $pf_i$ is replaced by the sum of all variables in $\mathcal{I}_{(d)}(p_{pwd_i})$ (with the respective coefficients), which are different from $d_0$ and contains a coefficient greater than 1. Besides, each variable from $rel_1$ to $rel_z$ is replaced by the sum of $d_0''$ (with the same coefficient as $d_0$ in $\mathcal{I}_{(d)}(p_{pwd_i})$) and all variables in $\mathcal{I}_{(d)}(p_{pwd_i})$ whose coefficient is equal to 1. The latter variables are also represented by $d_0''$ in $\mathcal{I}_{(cinter)}$. Additionally, in this composition, $z$ represents the number of inter-task relations in net $\mathcal{N}_{cinter}$. After the summation $(rel_1 + ... + rel_p + ... + rel_z)$, all variables in $\mathcal{I}_{(d)}(p_{pwd_i})$ whose coefficient is equal to 1 are replaced by $d_0$, $d_0$ is substituted by $zd_0'$, and $d_0''$ by $d_0'$.

The composition is demonstrated as follows. $\mathcal{N}_{(cinter \sqcup d)} = \mathcal{N}_{cinter} \sqcup \mathcal{N}_d$. Thus, by juxtaposition $\mathcal{I}_{(cinter \sqcup d)} = \mathcal{J}(\mathcal{I}_{(cinter)}, \mathcal{I}_{(d)})$, such that $pf_i = 2d_1 + 2d_{lj} + 2d_j$ and $rel_1 = \cdots = rel_p = \cdots = rel_z = (3d_0'' + d_0'' + d_0'' + d_0'')$. After the summation $(rel_1 + ... + rel_p + ... + rel_z = 6zd_0'')$, $d_{1,lj} = d_{1,j} = d_{lj,j} = d_0$, $d_0 = zd_0'$, and $d_0''$ is replaced by $d_0'$.

$$\begin{array}{l}
\mathcal{I}_{(cinter \sqcup d)} = \Big[\ \overset{p_{fv_i}}{pf1_i + rel1_i + \cdots + rel1_p + \cdots + rel1_z} \quad \overset{p_{f_i}}{pf1_i + 2d_1 + 2d_{lj} + 2d_j} \\
\overset{p_{pwd_i}}{2d_1 + 2d_{lj} + 2d_j + z6d_0'} \quad \overset{p_{rel_1}}{rel1_1 + 6d_0'} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_p}}{rel1_p + 6d_0'} \quad \overset{\cdots}{\cdots} \quad \overset{p_{rel_z}}{rel1_z + 6d_0'} \quad \overset{p_{wpc_{i_1}}}{2zd_0' + d_{lj} + d_j} \\
\overset{p_{wc_{i_1}}}{d_1 + 3zd_0'} \quad \overset{p_{wplc_{i_j}}}{2zd_0' + d_1 + d_3} \quad \overset{p_{wlc_{i_j}}}{d_{lj} + 3zd_0'} \quad \overset{p_{wpc_{i_j}}}{2zd_0' + d_1 + d_{lj}} \quad \overset{p_{wc_{i_j}}}{d_j + 3zd_0'} \quad \overset{p_{dm_i}}{4zd_0' + d_1 + d_{lj} + d_j}\ \Big]^T
\end{array}$$

$\mathcal{N}_{(cinter \sqcup d)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(cinter \sqcup p2v)} > 0$ and$\mathcal{I}^T_{(cinter \sqcup p2v)} \times A_{(cinter \sqcup d)} = 0$, in which $A_{(cinter \sqcup d)}$ is the respective incidence matrix.

### A.44 Exclusion Pre-Condition Block and Periodic Task Arrival Block

$$\begin{array}{l}
\mathcal{I}_{(ipree)} = \Big[\ \overset{p_{wexcl_i}}{wexcl_i} \quad \overset{p_{wvs_i}}{wexcl_i + excl_{1_i} + \cdots + excl_{p_i} + \cdots + excl_{y_i}} \quad \overset{p_{excl_{1_i}}}{excl_{1_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{excl_{p_i}}}{excl_{p_i}} \\
\overset{\cdots}{\cdots} \quad \overset{p_{excl_{y_i}}}{excl_{y_i}}\ \Big]^T \\
\mathcal{I}_{(a)} = \Big[\ \overset{p_{st_i}}{(\alpha_i + 1)(wvs_i + wd_i)} \quad \overset{p_{wa_i}}{wvs_i + wd_i} \quad \overset{p_{wd_i}}{wd_i} \quad \overset{p_{wr_i}}{wvs_i} \quad \overset{p_{wexcl_i}}{wvs_i}\ \Big]^T;
\end{array}$$

Assume that a place renaming function has been applied in net $\mathcal{N}_a$, renaming place $p_{wvs_i}$ to $p_{wexcl_i}$. $\mathcal{N}_{(ipree \sqcup a)} = \mathcal{N}_{ipree} \sqcup \mathcal{N}_a$. By juxtaposition $\mathcal{I}_{(ipree \sqcup a)} = \mathcal{J}(\mathcal{I}_{(ipree)}, \mathcal{I}_{(a)})$, such that $wvs_i = wexcl_i$:

$$\mathcal{I}_{(ipree \sqcup a)} = \Big[\ \overset{p_{wexcl_i}}{wexcl_i} \quad \overset{p_{wvs_i}}{wexcl_i + excl_{1_i} + \cdots + excl_{p_i} + \cdots + excl_{y_i}} \quad \overset{p_{excl_{1_i}}}{excl_{1_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{excl_{p_i}}}{excl_{p_i}}$$

$$\begin{matrix} & \overset{\cdots}{} & \overset{p_{excl_{y_i}}}{} & & \overset{p_{st_i}}{} & & \overset{p_{wa_i}}{} & \overset{p_{wd_i}}{} & \overset{p_{wr_i}}{} \\ \cdots & & excl_{y_i} & (\alpha_i+1)(wexcl_i+wd_i) & & wexcl_i+wd_i & & wd_i & wexcl_i \ \big] \end{matrix}$$

Since $\mathcal{I}_{(ipree \sqcup a)} > 0$ and $\mathcal{I}^T_{(ipree \sqcup a)} \times A_{(ipree \sqcup a)} = 0$, in which $A_{(ipree \sqcup a)}$ is the incidence matrix, $\mathcal{N}_{(ipree \sqcup a)}$ is structurally conservative as well as structurally bounded.

## A.45 Exclusion Pre-Condition Block and Voltage Selection Block

$$\begin{matrix} \mathcal{I}_{(ipree)} = & \big[ & \overset{p_{wexcl_i}}{} & & \overset{p_{wvs_i}}{} & & \overset{p_{excl_{1_i}}}{} & \overset{\cdots}{} & \overset{p_{excl_{p_i}}}{} \\ & & wexcl_i & wexcl_i+excl_{1_i}+\cdots+excl_{p_i}+\cdots+excl_{y_i} & & & excl_{1_i} & \cdots & excl_{p_i} \\ & \overset{\cdots}{} & \overset{p_{excl_{y_i}}}{} & & & & & & \\ & \cdots & excl_{y_i} \ \big]^T & & & & & & \end{matrix}$$

$$\begin{matrix} \mathcal{I}_{(v)} = & \big[ & \overset{p_{wvs_i}}{} & \overset{p_{v_{i_1}}}{} & \overset{\cdots}{} & \overset{p_{v_{i_j}}}{} & \overset{\cdots}{} & \overset{p_{v_{i_m}}}{} \\ & & wvs_i & wvs_i & \cdots & wvs_i & \cdots & wvs_i \ \big]^T \end{matrix}$$

$\mathcal{N}_{(ipree \sqcup v)} = \mathcal{N}_{ipree} \sqcup \mathcal{N}_v$. By juxtaposition $\mathcal{I}_{(ipree \sqcup v)} = \mathcal{J}(\mathcal{I}_{(ipree)}, \mathcal{I}_{(v)})$, such that $wvs_i = wexcl_i + excl_{1_i} + \cdots + excl_{p_i} + \cdots + excl_{x_i}$:

$$\begin{matrix} \mathcal{I}_{(ipree \sqcup v)} = & \big[ & \overset{p_{wexcl_i}}{} & & \overset{p_{wvs_i}}{} & & \overset{p_{excl_{1_i}}}{} & \overset{\cdots}{} \\ & & wexcl_i & wexcl_i+excl_{1_i}+\cdots+excl_{p_i}+\cdots+excl_{x_i} & & excl_{1_i} & \cdots \\ \overset{p_{excl_{p_i}}}{} & \overset{\cdots}{} & \overset{p_{excl_{x_i}}}{} & & \overset{p_{v_{i_1}}}{} & & \overset{\cdots}{} & \\ excl_{p_i} & \cdots & excl_{x_i} & wexcl_i+excl_{1_i}+\cdots+excl_{p_i}+\cdots+excl_{x_i} & & & \cdots & \\ & & & & \overset{p_{v_{i_j}}}{} & & \overset{\cdots}{} & \\ & & & wexcl_i+excl_{1_i}+\cdots+excl_{p_i}+\cdots+excl_{x_i} & & & \cdots & \\ & & & & \overset{p_{v_{i_m}}}{} & & & \\ & & & wexcl_i+excl_{1_i}+\cdots+excl_{p_i}+\cdots+excl_{x_i} \ \big]^T & & & & \end{matrix}$$

As $\mathcal{I}_{(ipree \sqcup v)} > 0$ and $\mathcal{I}^T_{(ipree \sqcup v)} \times A_{(ipree \sqcup v)} = 0$, in which $A_{(ipree \sqcup v)}$ is the incidence matrix, $\mathcal{N}_{(ipree \sqcup v)}$ is structurally conservative as well as structurally bounded.

## A.46 Exclusion Pre-Condition Block and Precedence Pre-Condition Block

$$\begin{matrix} \mathcal{I}_{(ipree)} = & \big[ & \overset{p_{wexcl_i}}{} & & \overset{p_{wvs_i}}{} & & \overset{p_{excl_{1_i}}}{} & \overset{\cdots}{} & \overset{p_{excl_{p_i}}}{} \\ & & wexcl_i & wexcl_i+excl_{1_i}+\cdots+excl_{p_i}+\cdots+excl_{y_i} & & & excl_{1_i} & \cdots & excl_{p_i} \\ & \overset{\cdots}{} & \overset{p_{excl_{y_i}}}{} & & & & & & \\ & \cdots & excl_{y_i} \ \big]^T & & & & & & \end{matrix}$$

$$\begin{matrix} \mathcal{I}_{(iprep)} = & \big[ & \overset{p_{wprec_i}}{} & & \overset{p_{wexcl_i}}{} & & \overset{p_{prec_{1_i}}}{} & \overset{\cdots}{} & \overset{p_{prec_{p_i}}}{} \\ & & wprec_i & wprec_i+prec_{1_i}+\cdots+prec_{p_i}+\cdots+prec_{x_i} & & & prec_{1_i} & \cdots & prec_{p_i} \\ & \overset{\cdots}{} & \overset{p_{prec_{x_i}}}{} & & & & & & \\ & \cdots & prec_{x_i} \ \big]^T & & & & & & \end{matrix}$$

Assume that a place renaming function has been applied in net $\mathcal{N}_{ipree}$, renaming place $p_{wvs_i}$ to $p_{wexcl_i}$. $\mathcal{N}_{(ipree \sqcup iprep)} = \mathcal{N}_{ipree} \sqcup \mathcal{N}_{iprep}$. By juxtaposition $\mathcal{I}_{(ipree \sqcup iprep)} =$

$\mathcal{J}(\mathcal{I}_{(ipree)}, \mathcal{I}_{(iprep)})$, such that $wexcl_i = wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}$:

$$\mathcal{I}_{(ipree \sqcup iprep)} = \begin{bmatrix} \overset{p_{wexcl_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & & & & & & & & & & & \\ \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & \overset{p_{excl_{1_i}}}{excl_{1_i}} & & & & & & & & & & \\ \overset{\cdots}{\cdots} & \overset{p_{excl_{p_i}}}{excl_{p_i}} & \overset{\cdots}{\cdots} & \overset{p_{excl_{y_i}}}{excl_{y_i}} & \overset{p_{wprec_i}}{wprec_i} & \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & \overset{p_{prec_{1_i}}}{prec_{1_i}} & & & & & \\ \overset{\cdots}{\cdots} & \overset{p_{prec_{p_i}}}{prec_{p_i}} & \overset{\cdots}{\cdots} & \overset{p_{prec_{x_i}}}{prec_{x_i}} & & & & & & & & \end{bmatrix}^T$$

$\mathcal{N}_{(ipree \sqcup iprep)}$ is structurally conservative as well as structurally bounded, as $\mathcal{I}_{(ipree \sqcup v)} > 0$ and $\mathcal{I}^T_{(ipree \sqcup iprep)} \times A_{(ipree \sqcup iprep)} = 0$, in which $A_{(ipree \sqcup iprep)}$ is the respective incidence matrix.

## A.47 Precedence Pre-Condition Block and Periodic Task Arrival Block

$$\mathcal{I}_{(iprep)} = \begin{bmatrix} \overset{p_{wprec_i}}{wprec_i} & \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & \overset{p_{prec_{1_i}}}{prec_{1_i}} & \overset{\cdots}{\cdots} & \overset{p_{prec_{p_i}}}{prec_{p_i}} & \overset{\cdots}{\cdots} & \overset{p_{prec_{x_i}}}{prec_{x_i}} \end{bmatrix}^T$$

$$\mathcal{I}_{(a)} = \begin{bmatrix} \overset{p_{st_i}}{(\alpha_i + 1)(wvs_i + wd_i)} & \overset{p_{wa_i}}{wvs_i + wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wvs_i} & \overset{p_{wprec_i}}{wvs_i} \end{bmatrix}^T;$$

Assume that a place renaming function has been applied in net $\mathcal{N}_a$, renaming place $p_{wvs_i}$ to $p_{wprec_i}$. $\mathcal{N}_{(iprep \sqcup a)} = \mathcal{N}_{iprep} \sqcup \mathcal{N}_a$. By juxtaposition $\mathcal{I}_{(iprep \sqcup a)} = \mathcal{J}(\mathcal{I}_{(iprep)}, \mathcal{I}_{(a)})$, such that $wvs_i = wprec_i$:

$$\mathcal{I}_{(iprep \sqcup a)} = \begin{bmatrix} \overset{p_{wprec_i}}{wprec_i} & \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & \overset{p_{prec_{1_i}}}{prec_{1_i}} & \overset{\cdots}{\cdots} & \overset{p_{prec_{p_i}}}{prec_{p_i}} \\ \overset{\cdots}{\cdots} & \overset{p_{prec_{x_i}}}{prec_{x_i}} & \overset{p_{st_i}}{(\alpha_i + 1)(wprec_i + wd_i)} & \overset{p_{wa_i}}{wprec_i + wd_i} & \overset{p_{wd_i}}{wd_i} & \overset{p_{wr_i}}{wprec_i} \end{bmatrix}$$

Since $\mathcal{I}_{(iprep \sqcup a)} > 0$ and $\mathcal{I}^T_{(iprep \sqcup a)} \times A_{(iprep \sqcup a)} = 0$, in which $A_{(iprep \sqcup a)}$ is the incidence matrix, $\mathcal{N}_{(iprep \sqcup a)}$ is structurally conservative as well as structurally bounded.

## A.48 Precedence Pre-Condition Block and Voltage Selection Block

$$\mathcal{I}_{(iprep)} = \begin{bmatrix} \overset{p_{wprec_i}}{wprec_i} & \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} & \overset{p_{prec_{1_i}}}{prec_{1_i}} & \overset{\cdots}{\cdots} & \overset{p_{prec_{p_i}}}{prec_{p_i}} \\ \overset{\cdots}{\cdots} & \overset{p_{prec_{x_i}}}{prec_{x_i}} \end{bmatrix}^T$$

$$\mathcal{I}_{(v)} = \begin{bmatrix} \overset{p_{wvs_i}}{wvs_i} & \overset{p_{v_{i_1}}}{wvs_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_j}}}{wvs_i} & \overset{\cdots}{\cdots} & \overset{p_{v_{i_m}}}{wvs_i} \end{bmatrix}^T$$

$\mathcal{N}_{(iprep \sqcup v)} = \mathcal{N}_{iprep} \sqcup \mathcal{N}_v$. By juxtaposition $\mathcal{I}_{(iprep \sqcup v)} = \mathcal{J}(\mathcal{I}_{(iprep)}, \mathcal{I}_{(v)})$, such that $wvs_i = wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}$:

$$\mathcal{I}_{(iprep \sqcup v)} = \Big[ \overset{p_{wprec_i}}{wprec_i} \quad \overset{p_{wvs_i}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} \quad \overset{p_{prec_{1_i}}}{prec_{1_i}} \quad \overset{\cdots}{\cdots}$$
$$\overset{p_{prec_{p_i}}}{prec_{p_i}} \quad \overset{\cdots}{\cdots} \quad \overset{p_{prec_{x_i}}}{prec_{x_i}} \quad \overset{p_{v_{i_1}}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} \quad \overset{\cdots}{\cdots}$$
$$\overset{p_{v_{i_j}}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} \quad \overset{\cdots}{\cdots}$$
$$\overset{p_{v_{i_m}}}{wprec_i + prec_{1_i} + \cdots + prec_{p_i} + \cdots + prec_{x_i}} \Big]^T$$

As $\mathcal{I}_{(iprep \sqcup a)} > 0$ and $\mathcal{I}^T_{(iprep \sqcup v)} \times A_{(iprep \sqcup v)} = 0$, in which $A_{(iprep \sqcup v)}$ is the incidence matrix, $\mathcal{N}_{(iprep \sqcup v)}$ is structurally conservative as well as structurally bounded.

APPENDIX B

# MODEL CHECKING

Model checking [115] is a prominent collection of techniques that automatically allow the formal verification of finite-state systems. Such techniques have an important contribution in hard real-time system design, since a system model can be verified for errors and inconsistencies related to the specification. Model checking relies on a tool, namely, model checker, which receives as input a model and a property expected to be satisfied. The tool outputs *true* or *false*, but, in the latter case, the model checker usually provides a counterexample. This appendix is based on [126].

As an example, consider the system depicted in Figure B.1, which transfers water from a tank A into a tank B adopting a pump P. Each tank has two sensors: one to detect if there is no water (Empty) and other one to detect if the tanks is totally filled (Full). The tank level is ok whether it is neither empty nor full. Initially, tanks A and B are empty, and the pump is switched on as soon as tank A has water and tank B is not full. The pump stays on while tank A is not empty and tank B is not full. Similarly, the pump is switched off as soon as tank A becomes empty or tank B becomes full. Adopting such a specification, the system needs to be modeled using a representation compatible with the model checker to allow property verification.

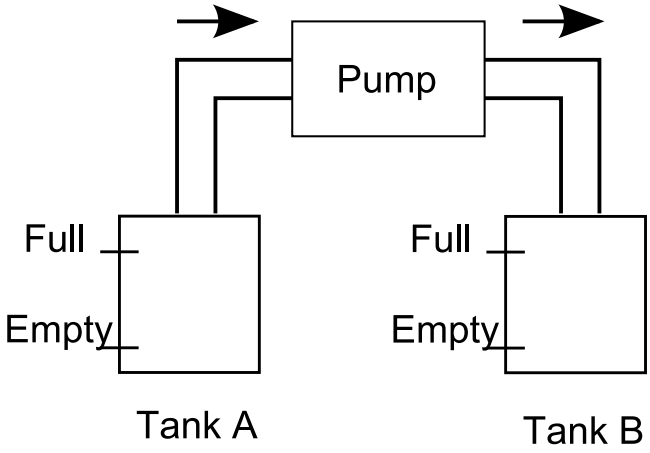


**Figure B.1** Pumping system

Many model-checking tools unfold an input model into a transition system called Kripke structure, and a given property is actually checked against it. However, for the purposes of understanding what a property statement means, a Kripke structure is further unfolded into an infinite tree, in which each path in the tree indicates a possible execution or behavior of the system.

## B.1 PATHS AND FORMULAS

Consider that a state is denoted by an ordered tuple <A,B,P>, where A and B denote the current water level in tank A and B, and P denotes the current pump status. For the sake of illustration, assume the initial state to be <empty,empty,off>. The next state from the initial state could be <empty,empty,off> or <ok,empty,on>. From <ok,empty,on>, the next state could be <ok,empty,on>, <ok,ok,on>, <full,empty,on>, <full,ok,on>, <empty,empty,off>, or <empty,ok,off>. For each of these states, the next possible states could be further calculated.

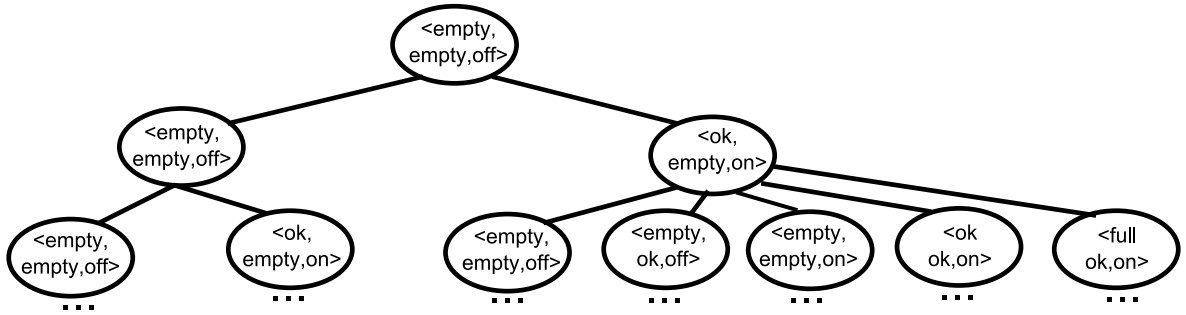


**Figure B.2** Initial part of the execution tree for the pumping system

The states can be arranged in the form of an infinite execution (or computation) tree, in which the root is labeled with the chosen initial state and the children of any state denote the next possible states (see Figure B.2). A system execution is a path in this execution tree, and, indeed, the system may have infinitely many execution paths. The objective of model checking is to examine whether or not the execution tree satisfies a user-given property specification.

A question arises about how someone may specify properties of paths (and their states) in an execution tree. Computation tree logic (CTL) - a branching time temporal logic - is an intuitive notation suitable for this purpose. Without loss of generality, CTL is an extension of the usual Boolean propositional logic (which includes the logical connectives such as and, or, not, implies), where additional temporal connectives are available.

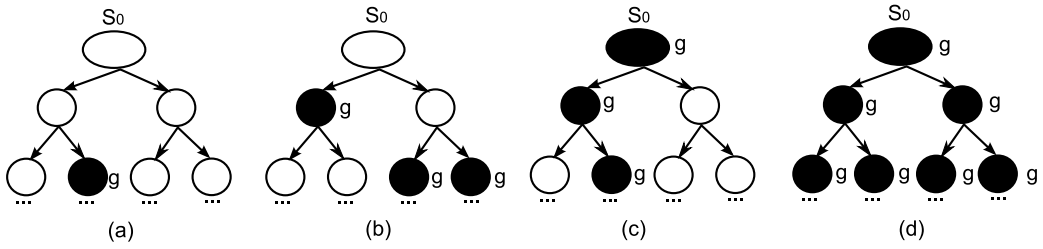


**Figure B.3** Initial part of the execution tree for the pumping system

Table B.1 and Figure B.3 illustrate the intuitive meaning of some of the basic temporal connectives in CTL. In short, E (for some path) and A (for all paths) are path quantifiers for paths beginning from a state. F (for some state) and G (for all states) are state quantifiers for states in a path.

**Table B.1** Some temporal connectives in CTL

| | |
|---|---|
| EX $\varphi$ | *true* in current state if formula $\varphi$ is true in at least one of the next states |
| EF $\varphi$ | *true* in current state if there exists some state in some path beginning in current state that satisfies the formula $\varphi$ |
| EG $\varphi$ | *true* in current state if every state in some path beginning in current state satisfies the formula $\varphi$ |
| AX $\varphi$ | *true* in current state if formula $\varphi$ is true in every one of the next states |
| AF $\varphi$ | *true* in current state if there exists some state in every path beginning in current state that satisfies the formula $\varphi$ |
| AG $\varphi$ | *true* in current state if every state in every path beginning in current state satisfies the formula $\varphi$ |

From a given property and a (possibly infinite) computation tree $T$ (related to the system model), a model-checking algorithm examines $T$ to verify if the property is satisfied. For instance, consider a property AF $g$, in which $g$ is a propositional formula not involving any CTL connectives. Figure B.3(b) shows an example of a computation tree $T$. The property AF $g$ is true for this tree, if there is some state in every path in $T$ starting at $s_0$, such that the formula $g$ is true in that state.

Figure B.3(b) shows that $g$ is true at the root of the left subtree (indicated by the filled circle). Thus, all paths from $s_0$ to the left child (and further down in the left subtree) satisfy the property. Additionally, Figure B.3(b) depicts $g$ is true at all children of the right child of $s_0$ (also indicated by filled circles). Therefore, the property is true for all subtrees of $s_0$ and, so, it's also true at $s_0$.

Figure B.3 summarizes the similar reasoning adopted to check properties stated in other forms, such as EG $g$ and AG $g$. Of course, in practice, the model-checking algorithms are really far more complex than this; they use sophisticated tricks to prune the state space to avoid checking those parts where the property is guaranteed to be true.

Regarding the pump system, the designer may want to check AF (P = off), which states that, for every path beginning at the initial state, there is a state in that path at which the pump is off. This property is trivially true, since, in the initial state, P = off is true.

APPENDIX C

# LIST OF ABBREVIATIONS

| | |
|---|---|
| **CPN** | Coloured Petri Net |
| **DPM** | Dynamic Power Management |
| **DVS** | Dynamic Voltage Scaling |
| **EDF** | Earliest-Deadline First |
| **FCPN** | Free Choice Petri Net |
| **FSM** | Finite-State Machine |
| **HRTS** | Hard Real-Time Systems |
| **I/O** | Input/Output |
| **LCM** | Least Common Multiple |
| **LPEDF** | Low Power Earliest-Deadline First |
| **MOS** | Metal Oxide Semiconductor |
| **RTOS** | Real-Time Operating System |
| **TFCPN** | Time Free-Choice Petri Net |
| **TLTS** | Time Labeled Transition System |
| **TPN** | Time Petri Net |
| **TPNE** | Time Petri Net Extension with Energy Consumption Values and Code Annotations |
| **WCEC** | Worst-Case Execution Cycles |
| **WCET** | Worst-Case Execution Time |