Méthode de conception de systèmes temps réels embarqués multi-coeurs en milieu automobile

Enagnon Cédric Klikpo

THÈSE DE DOCTORAT DE l'UNIVERSITÉ SORBONNE UNIVERSITÉ

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Enagnon Cédric KLIKPO

Pour obtenir le grade de

DOCTEUR de SORBONNE UNIVERSITÉ

Sujet de la thèse :

Méthode de conception de systèmes temps réels embarqués multi-cœurs en milieu automobile

soutenue le 13 mars 2018

devant le jury composé de :

Mme. Alix Munier-Kordon	Directeur de thèse
Mme. Claire Pagetti	Rapporteur
Mme. Maryline Chetto	Rapporteur
Mme. Béatrice Bérard	Examinateur
M. Jean-François Nezan	Examinateur
M. Laurent Pautet	Examinateur
M. Pascal Richard	Examinateur
M. Witold Klaudel	Examinateur

Remerciements

Je remercie Alix Munier-Kordon pour m'avoir fait confiance et pour son encadrement durant ces trois années de thèse. J'ai appris la bonne méthodologie de recherche avec rigueur scientifique et approche formelle. J'ai également grandement apprécié ses qualités humaines.

J'exprime ma profonde gratitude à Maryline Chetto et Claire Pagetti pour avoir accepté de rapporter cette thèse. Merci également aux membres du jury pour l'intérêt qu'ils ont manifesté pour mes travaux de recherche.

Je remercie Witold Klaudel, chef de projet à l'IRT SystemX, pour m'avoir accueilli dans son équipe et me donnée l'opportunité de réaliser cette thèse. Merci à tous mes collèges de l'IRT SystemX pour les moments partagés ensemble. Plus spécialement, je remercie mon référent Aymen Boudguiga pour tous ces conseils, son soutien moral et son dévouement aux relectures. Mon anglais ne sera plus jamais le même grâce à toi.

Merci également à mes collègues du LIP6 : Éric Lao, Olivier Tsiakaka, Youen Lesparre et Jad Khatib. Vous avez élargi mes horizons scientifiques et culturels.

Enfin, je pense très fort à mes parents et à mes frères et sœurs. Les encouragements et le soutien dont ils ont fait preuve ont énormément contribué à l'accomplissement de mon travail.

Ce travail a été effectué dans le cadre des recherches menées au sein de l'IRT SystemX, Paris-Saclay, France, et a ainsi bénéficié d'une aide de l'Etat au titre du programme d'Investissements d'Avenir.

Résumé — La complexité croissante des applications embarquées dans les voitures modernes augmente le besoin de puissance de calcul. Pour répondre à ce besoin, le standard automobile AUTOSAR introduit l'utilisation de plates-formes multi-cœurs. Cependant, l'utilisation du multi-cœur pour des applications temps-réel critique automobile soulève plusieurs problématiques. Notamment, il faut respecter la spécification fonctionnelle et garantir de manière déterministe les échanges de données entre cœurs. Dans cette thèse, nous considérons des systèmes multi-périodiques spécifiés et validés fonctionnellement avec des modèles Matlab/Simulink. Ainsi, nous avons développé un framework pour déployer des applications Matlab/Simulink sur AUTOSAR multi-cœur afin de garantir le déterminisme fonctionnel et temporel tout en exploitant au mieux le parallélisme.

Notre contribution a porté sur trois axes.

Premièrement nous avons identifié les mécanismes d'échanges de données imposés dans le modèle fonctionnel Matlab/Simulink. Nous avons montré que ces mécanismes pouvaient s'exprimer en utilisant le formalisme des Synchronous Dataflow Graph (SDFG). Ce modèle est un excellent outil d'analyse pour exploiter le parallélisme car il est très populaire dans la littérature et largement étudié pour le déploiement d'applications flow de données sur plateforme multi/many-cœur.

Par la suite, nous avons développé des méthodes pour réaliser le flux de données exprimés par le SDFG dans un ordonnancement temps-réel préemptif. Ces méthodes utilisent des résultats théoriques sur les SDFGs pour garantir les contraintes de précédence de manière déterministe sans utiliser des mécanismes de synchronisation bloquants. De cette sorte, nous garantissons à la fois le déterminisme fonctionnel et temporel des applications.

Finalement, nous caractérisons l'impact des contraintes de flux de données sur l'ordonnancement des tâches. Nous proposons une technique de partitionnement qui minimise cet impact. Nous montrons alors que cette technique favorise la construction d'un partitionnement et d'un ordonnancement lorsqu'elle est utilisée pour initialiser des algorithmes de recherche et d'optimisation heuristiques.

Mots clés : AUTOSAR, Matlab/Simulink, SDFG, flux de données, multi-cœur, ordonnancement temps-réel.

Abstract — The increasing complexity of embedded applications in modern cars has increased the need of computing power. To meet this need, the European automotive standard AUTOSAR has introduced the use of multi-core platforms. However, multi-core platform for critical automotive applications raises several issues. In particular, it is necessary to respect the functional specification and to guarantee deterministically the data exchanges between cores. In this thesis, we consider multi-periodic systems specified and validated with Matlab/Simulink. So, we developed a framework to deploy Matlab/Simulink applications on AUTOSAR multi-core. This framework guarantees the functional and temporal determinism and exploits the parallelism.

Our contribution is threefold.

First, we identify the communication mechanisms in Matlab/Simulink. Then, we prove that the dataflow in a multi-periodic Matlab/Simulink system is modeled by a SDFG. The SDFG formalism is an excellent analysis tool to exploit the parallelism. In fact, it is very popular in the literature and it is widely studied for the deployment of dataflow applications on multi/many-core.

Then, we develop methods to realize the dataflow expressed by the SDFG in a preemptive real-time scheduling. These methods use theoretical results on SDFGs to guarantee deterministic precedence constraints without using blocking synchronization mechanisms. As such, both the functional and temporal determinism are guaranteed.

Finally, we characterize the impact of dataflow requirements on tasks. We propose a partitioning technique that minimizes this impact. We show that this technique promotes the construction of a partitioning and a feasible scheduling when it is used to initiate multi-objective research and optimization algorithms.

Keywords: AUTOSAR, Matlab/Simulink, SDFG, dataflow, multi-core, real-time scheduling.

Contents

List of Figures

Table of Acronyms

AAA	*Algorithm-Architecture Adequation*
ACS	*Airbag Control System*
ADL	*Architecture Description Language*
API	*Application Programming Interface*
AUTOSAR	*AUTomotive Open System ARchitecture*
ASAP	*As Soon As Possible*
BSW	*Basic Software*
CAN	*Controller Area Network*
CPU	*Central Processing Unit*
CS	*Client-Server*
CSDFG	*Cyclo-Static Dataflow Graph*
CSV	*Comma-Separated Values*
DAC	*Digital to Analog Converter*
DAG	*Directed Acyclic Graph*
DBP	*Dynamic Buffering Protocol*
DM	*Deadline Monotonic*
DMA	*Direct Memory Access*
DP	*Dynamic Priority*
DPN	*Dataflow Process Network*
EA	*Exclusive areas*
ECU	*Electronic Control Unit*
EDF	*Earliest Deadline First*
ELA	*Electronique et Logiciel pour l'Automobile*
EP	*Expiry Point*
E/E	*Electric/Electronic*

FCCS	*Fuel Cell Control System*
FIFO	*First-In-First-Out*
FJP	*Fixed Job Priority*
FTP	*Fixed Task Priority*
HIL	*Hardware-In-the-Loop*
HSDFG	*Homogeneous Synchronous Dataflow*
IB	*Internal Behavior*
ILP	*Integer Linear Program*
IOC	*Inter-OS-Applications Communication*
IRV	*Inter-Runnables Variables*
I/O	*Input/Output*
LCM	*Least Common Multiple*
LIN	*Local Interconnect Network*
LLF	*Least Laxity First*
LP	*Linear Program*
LTTA	*Loosely Time Triggered Architectures*
MASE	*MAtlab SDFG Extraction*
MBD	*Model-Based Design*
MDG	*Marked Direct Graph*
MOO	*Multi-Objective Optimization*
MPPA	*Massively Parallel Processor Array*
NoC	*Network on Chip*
OEM	*Original Equipment Manufacturers*
OPA	*Optimal Priority Assignment*
OS	*Operating System*
QoS	*Quality of Service*
RAM	*Random-Access Memory*

RM	*Rate Monotonic*
RTA	*Response Time Analysis*
RTB	*Rate Transition Block*
RTE	*Runtime Environment*
RTOS	*Real-Time Operating System*
SBD	*Synchronous Block Diagrams*
SDFG	*Synchronous Dataflow Graph*
SPB	*System Peripheral Bus*
SPM	*Scratchpad memory*
SR	*Sender-Receiver*
SWC	*Software Component*
TTA	*Time Triggered Architectures*
VFB	*Virtual Functional Bus*
WCET	*Worst Case Execution Time*
WCRT	*Worst Case Response Time*
XML	*eXtensible Markup Language*
XSD	*XML Schema Definition*

Chapter 1

Introduction

Context

Innovations in modern cars are mainly focused on embedded electrical and electronic systems. However, the automotive industry is experiencing a rapid technological progress due to the evolution of environmental constraints dictated by Euro standards [39] (to reduce the emission of CO_2) and the continuous addition of new services (*e.g.* connected and autonomous cars). These advances led to increasing complexity of software and processing demands. However, as the computing requirement increases, conventional approaches (*i.e.* increase the number of calculator and/or clocks speed) become ineffective for weight, energy consumption and costs. Instead, an alternate promising approach to increase the computational power is the use of multi-core platforms. Car manufacturers are now considering this approach to meet the need of performance in automotive.

Reflecting this trend, the *AUTomotive Open System ARchitecture* (AUTOSAR) [32] has promoted the use of multi-core platforms by introducing the deployment of multi-core applications. AUTOSAR is a standard that defines the design and development of automotive softwares. However, the use of multi-core for automotive systems raises several issues. In fact, an automotive application is composed of several functions that exchange data. Each application is characterized by a criticality level, where missing the data exchanges and the timing in a highly critical application can be catastrophic.

In the industry, these applications are generally designed and functionally validated with synchronous models such as Matlab/Simulink [21]. These models are further constrained with timing and safety aspects, and implemented on AUTOSAR. However, Matlab/Simulink enforces a sequential execution paradigm along with a parametrized dataflow. Deploying a Matlab/Simulink model on multi-core therefore requires an accurate tracking of the data exchanges in order to implement the validated behavior. For complex systems, transforming such models on multi-core AUTOSAR can be laborious and difficult to prove. As a result, advanced techniques are needed to effectively manage the transformation from Matlab/Simulink to AUTOSAR multi-core. This transformation must guarantee both the functional and temporal determinism of the application, while exploiting parallelism.

Contributions

The main objective of this thesis is to develop a proven methodology to implement deterministic data exchanges on AUTOSAR multi-core. For this purpose, we consider a *Model-Based Design* (MBD) approach, where the functional dataflow is specified by Matlab/Simulink. One can argue that the strict implementation of the specification of Matlab/Simulink over-constrains the system. However, it establishes an automatic framework to help designers during the synthesis of complex critical systems. As such, we are interested in characterizing the functional dataflow and define synthesis methods. The latter must guarantee both the functional and temporal determinism of applications, while making the best use of parallelism.

The first contribution of this thesis is to prove that the dataflow in a multi-periodic Matlab/Simulink system can be modeled by a *Synchronous Dataflow Graph* (SDFG). This formal equivalence between Matlab/Simulink systems and SDFG creates new perspectives for the development of real-time systems (*e.g.* AUTOSAR) on multi/many-core. In fact, SDFG is an excellent analysis tool to exploit the parallelism. It is especially popular in the literature and it is widely studied for the deployment of dataflow applications on multi/many-core [52, 96, 89]. Furthermore, the SDFG model provides a formal characterization of data dependencies. As such, the dataflow can be effectively considered during the configuration of AUTOSAR multi-core. Then, we rely on the compositionality offers by AUTOSAR to focus only on the interactions between Runnables, where the Runnable is the atomic part of an AUTOSAR application.

The second contribution is to exploit the analytical properties of SDFG to build a fast and efficient method to guarantee the functional dataflow in a real-time scheduling without blocking mechanisms. This way, the functional and the temporal determinism are both guaranteed.

The third contribution characterizes the dataflow requirements and measures their impact in partitioned multi-core. Then, we provide a method to perform a mapping that minimizes these impacts. This method is applied to build an initial mapping optimized for dataflow requirements. This initial mapping promotes the construction of a partitioning and a feasible scheduling when it is used to initiate multi-objective research and optimization algorithms. As such, it reduces the number of design iterations and shortens design time.

Figure 1.1 illustrates our approach. In fact, we first identify the dataflow imposed by Matlab/Simulink on the application. Then, we extract and model this dataflow by a *Synchronous Dataflow Graph* (SDFG). Finally, we use the analytical properties of the SDFG formalism to develop tools for partitioning and scheduling for AUTOSAR multi-core.

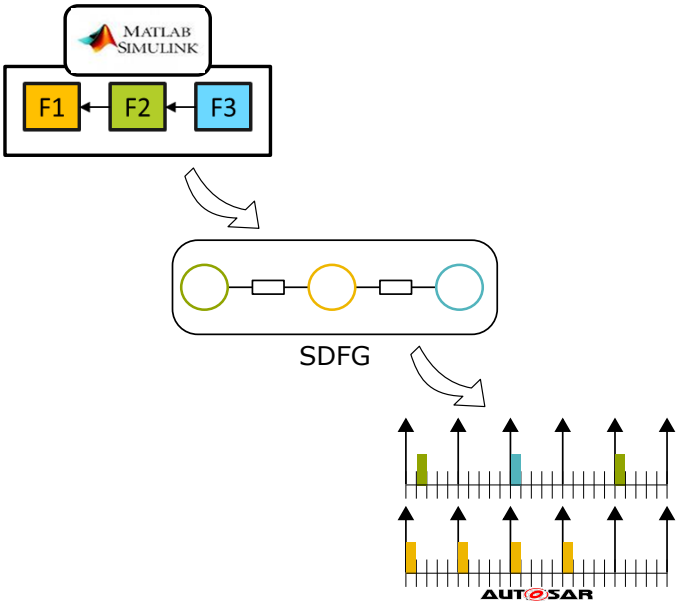


Figure 1.1: Illustration of the approach of this thesis

Work environment

Thesis is carried out at the *Technological Research Institute* (IRT) *SystemX* in cooperation with the *Laboratoire Informatique de Paris 6* (LIP6). The IRT SystemX brings together industrial and academic partners around the topics of digital engineering of future systems in automotive, railroad and avionics. It is organized in several projects in various fields. The work of this thesis is carried out within the project "*Electronique et Logiciel pour l'Automobile* (ELA)". The purpose of this project is to provide operational solutions to new technological and economic challenges in automotive. The ELA project is itself composed of several tasks dealing with different aspects of its objective. My thesis is part of the *"multi-core" task*. This task aims at developing proven methodology and tools to assist the development of predictable implementations on AUTOSAR multi-core. The *"multi-core" task* is performed in partnership with UMPC, the *Original Equipment Manufacturers* (OEM) PSA and Renault, Tier-1 suppliers Valeo and Continental, and Sherpa Engineering.

Thesis outline

Below is summarized the organization and the content of the chapters of this thesis.

Chapter 2 gives the notations and concepts essential for this work. Namely, Chapter 2 presents real-time scheduling theories relevant for this thesis. It also presents the architecture and the methodology to design and configure AUTOSAR applications. In addition, it describes the micro-controller AURIX TriCore TC29x, which is our targeted multi-core architecture.

Chapter 3 details the industrial challenge addressed by this thesis. It also describes our approach and provides a global state of the art that puts into perspective our approach over existing work.

Chapter 4 proves that the dataflow in a multi-periodic Matlab/Simulink specification is modeled by a SDFG. The method to model this dataflow is provided and design rules are defined so that a Matlab/Simulink specification is fully modeled by a SDFG. Finally, the translation from Matlab/Simulink to SDFG is illustrated on a use case of a *Fuel Cell Control System*.

Chapter 5 exploits the analytical properties of the SDFG to establish a technique for temporal isolation between tasks. This technique allows to implement deterministic data dependencies without synchronization mechanisms. A *Linear Program* (LP) is formulated to construct this temporal isolation and a scheduling algorithm that uses the latter to realize deterministic dataflow in single-core is proposed. Additionally, a method to generate random dependent tasks set modeled by the SDFG of Chapter 4 is proposed. These tasks sets generation is used in experiments to evaluate the performances of the approach.

Chapter 6 builds a global method to guarantee deterministic dataflow in partitioned multi-core. This method combines the temporal isolation with the technique of Forget *et al.* [46] to provide a fast and accurate characterization of dataflow requirements in partitioned multi-core. An *Integer Linear Program* (ILP) is formulated to realize a deterministic dataflow with minimal impact on real-time attributes. This ILP is extended to perform a mapping that also minimizes the impact of dataflow requirements. Then, a method to build an initial mapping optimized for dataflow requirements is defined. This initial mapping eases the search of a mapping that takes precedence constraints into account.

Chapter 7 summarizes our approach in a design framework to assist the configuration of AUTOSAR multi-core. This framework uses SDFG to bridge the gap between Matlab/Simulink and AUTOSAR. Furthermore, several tools and an exchange format are provided to automate the framework. This automate framework is applied on the use case of the *Fuel Cell Control System*.

Chapter 8 concludes the thesis and discuss prospects for future work.

Chapter 2

Background

Contents

2.1 Introduction

Electrical and electronic systems in modern cars operate on vehicle functions (*e.g.* engine control), or on occupants security (air bag system) or amenities (*e.g.* air conditioning). They are implemented as real-time embedded applications running on one or several on-board computer system called *Electronic Control Unit* (ECU). This chapter introduces the background necessary to understand the analysis and the implementation of a real-time systems on AUTOSAR multi-core ECU.

In Section 2.2, we present fundamental properties of real-time systems. Then, in Section 2.3, we detail the scheduling real-time systems on both single-core and multi-core. Section 2.4 presents the software architecture of AUTOSAR standard and the principles of interactions between Runnables. Section 2.5 describes the configuration of AUTOSAR applications and defines the real-time task model of AUTOSAR. Finally, Section 2.6 charachterizes the multi-core ECU architecture considered in our work and Section 2.7 concludes the chapter.

2.2 Real-time systems

In this section, we give basic definitions and we detail the concepts essential for the study of real-time systems.

First, in Subsection 2.2.1 we give a general definition of real-time systems. Then, in Subsection 2.2.2, we describe and characterize periodic real-time task model.

2.2.1 Definitions

A real-time system is a computer system that reacts in a timely manner to events from its environment. As such, a real-time system can be defined as follows [106].

Definition 2.1 (Real-time system)
A real-time system is a system whose correctness depends not only on the value of the computation but also on the time at which the latter is produced.

From Definition 2.1, real-time computing is different from fast computing because a real-time system aims to meet the timing of all computations in all possible circumstances [105]. As such, each real-time computation must complete within a proper time called the *deadline*. Consequences associated to missing deadlines define the criticaly level of real-time systems as follows.

Hard real-time systems have strict timing constraints for which all deadlines must be

met. This type of real-time systems is in general associated with critical applications for which missing a deadline can be catastrophic. An example is an *Airbag Control System* (ACS) in the automotive. In fact, ACS monitors various sensors (*e.g.* accelerometers, impact sensors, seat occupancy sensors, *etc.*) in order to detect collisions. When a collision occurs the system triggers a gas inflator/deflator in a timely manner to protect the lives of passengers. For the ACS, it is obvious that a late detection of a collision or a late triggering of the gas inflator/deflator can be catastrophic and cost human lives.

Firm real-time systems admit deadlines miss. Late results do not cause damages but are useless. Guaranteeing deadlines for this type of real-time systems is associated to ensuring a certain *Quality of Service* (QoS). Examples of firm real-time systems can be found in networked and multimedia systems. In fact, a late packet or video frame is in general useless and does not damage the system. However, it decreases the quality of the transmission/reception.

Soft real-time systems also admit deadlines miss. Late results are useful and do not induce damages. However, they lead to performance degradations. This type of real-time systems is in general related to human-machine interactions, where display information remain relevant to the user despite a display delay.

2.2.2 Real-time task model

Let $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a real-time system that consists in a set of n real-time tasks $\tau_{i,i=1\ldots n}$. Each task $\tau_i \in \boldsymbol{\mathcal{T}}$ defines a set of instructions and is executed until completion by the processor. Let $\tau_i[n_i]$ be the n_ith execution or *job* of τ_i, with $n_i \in \mathbb{N}\backslash\{0\}$. The real-time task τ_i can be characterized by the following parameters:

- **computation time** C_i: the *Worst Case Execution Time* (WCET) of τ_i;
- A **release time** or **activation time** $r_i[n_i]$: the time at which job $\tau_i[n_i]$ becomes *active* or ready to execute;
- A **starting date** $s_i[n_i]$: the time at which job $\tau_i[n_i]$ starts its execution;
- An **absolute deadline** $d_i[n_i]$: the maximum time before which job $\tau_i[n_i]$ must complete;
- A **relative deadline** D_i: a constant value that defines the difference between the absolute deadline and the release time of every job, *i.e.* $d_i[n_i] = r_i[n_i] + D_i$, $\forall n_i \geq 1$;
- A **finishing time** $f_i[n_i]$: the time at which job $\tau_i[n_i]$ completes its execution;
- A **job response time** $R_i[n_i]$: $R_i[n_i] = f_i[n_i] - r_i[n_i]$, $\forall n_i \geq 1$;
- A **task worst case response time** R_i: the maximum response time of all jobs. $R_i = \max\limits_{n_i \geq 1} (R_i[n_i]) = \max\limits_{n_i \geq 1} (f_i[n_i] - r_i[n_i])$;

- A minimum **inter-arrival time** or **period** T_i: the minimum time or the time interval between two activations, *i.e.* $T_i = \min_{n_i \geq 1} (r_i[n_i + 1] - r_i[n_i])$;
- A **slack time** X_i: the maximum time the task can be delayed from its activation to complete within its deadline. $X_i = D_i - C_i$;
- A **criticality**: which defines the severity of missing deadlines of jobs. It is related to the criticality of the system and therefore is either hard, firm or soft.

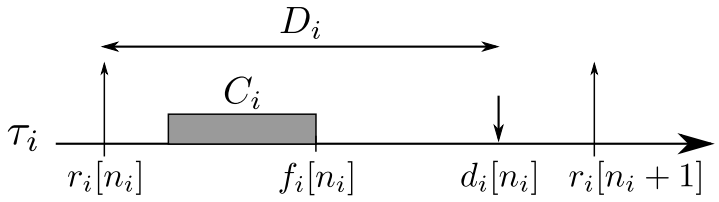


Figure 2.1: Example of real-time task parameters.

Figure 2.1 illustrates some of the real-time parameters defined above. *Up arrows* denote release times and *down arrows* denote absolute deadlines.

Several activation policies exist for real-time tasks. In the rest of this work we consider hard *periodic* real-time tasks. A periodic task is activated at a constant time interval equal to its period. The release time of the first job of a periodic task is called *offset* and is noted $O_i = r_i[1]$. As such, each periodic task $\tau_i \in \boldsymbol{\mathcal{T}}$ is completely characterized by the tuple (T_i, C_i, O_i, D_i). The release time and the absolute deadline of any n_ith job of τ_i is $r_i[n_i] = O_i + (n_i - 1) \cdot T_i$ and $d_i[n_i] = r_i[n_i] + D_i$, respectively. The timing constraints expressed by the relative deadline of τ_i is said to be *implicit* when $D_i = T_i$, *constrained* when $D_i < T_i$ and *arbitrary* when $D_i > T_i$.

Furthermore, a real-time task is said to be a *concrete* task if its offset is explicitly specified as a real-time constraint. Otherwise, the task is said to be an *offset-free* task [51].

Figure 2.2: Sequence of activations of a periodic task

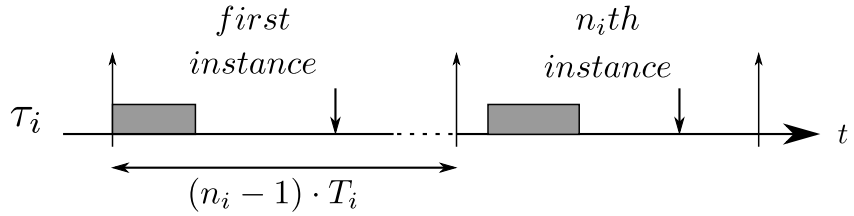


Figure 2.2 illustrates the periodic release of a periodic real-time task. We can notice that the executions are not necessary periodic. In fact, the execution of a job can be delayed at the time of its activation by other tasks (*c.f.* Section 2.3).

Let $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n periodic real-time tasks. $\boldsymbol{\mathcal{T}}$ is said to be a *synchronous* tasks set if all tasks have the same offset. Otherwise, $\boldsymbol{\mathcal{T}}$ is an *asynchronous* tasks set. Furthermore, the *hyperperiod* of $\boldsymbol{\mathcal{T}}$ is defined as the *Least Common Multiple*

(LCM) of all periods. The processor utilization of $\boldsymbol{\mathcal{T}}$ is $U = \sum_{i=1}^{n} U_i$, where $U_i = \frac{C_i}{T_i}$ is the utilization of task τ_i.

Other constraints that can be expressed on real-time tasks are dependency constraints. These constraints express the need to execute tasks or jobs in a certain order. In particular, it is the case when tasks communicate with each other and data exchanges require that producer tasks are executed before consume tasks. At job level, these constraints are referred to as *precedence constraints*. Moreover, tasks that are related by dependency constraints are defined as *dependent tasks* and tasks without dependency constraints are defined as *independent tasks*.

2.3 Real-time scheduling

In this section, we recall several concepts about real-time scheduling of independent tasks. These concepts are used later in this document to study the mapping and scheduling of dependent periodic real-time tasks.

First, Subsections 2.3.1, 2.3.2 and 2.3.3 give general definitions and concepts on real-time scheduling. Then, Subsection 2.3.4 presents approaches for multi-core scheduling.

2.3.1 Definitions

The scheduling of a set of n real-time tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ on single-core consists in deciding the task that is executed by the core at each instant, so that timing constraints are met. To achieve this, the following scheduling strategies can be used:

- Non-preemptive scheduling: once a task starts, it is executed until completion on the core. That is, tasks cannot be preempted and scheduling decisions are taken only at tasks termination or when a new task becomes active while the core is idle.
- Preemptive scheduling: the running task can be interrupted or *preempted* at any time, so that the core can run another task. As a result, scheduling decisions are made each time a new task becomes active or current job completes.

In preemptive scheduling, a priority level π_i is assigned to each task $\tau_i \in \boldsymbol{\mathcal{T}}$, so that the active task with the highest priority is always executed. We distinguish between the following types of algorithms for priority assignment.

- Fixed Task Priority (FTP): each task is assigned a fixed priority that is inherited by its jobs. A scheduling algorithm with FTP is also called *static*. Examples of static scheduling algorithms are *Rate Monotonic* (RM) [76], *Deadline Monotonic* (DM) [75] and the *Optimal Priority Assignment* (OPA) [5].

- Fixed Job Priority (FJP): the priority of each job depends on the state of the system and is assigned at activation of jobs. FJP is also called *task level dynamic* scheduling because jobs of the same task may have different priorities. However, the priority of a single job is fixed and cannot vary over time. *Earliest Deadline First* (EDF) [76] is an example of FJP.
- Dynamic Priority (DP): each job is assigned a priority that is not fixed and can vary over time. Hence, DP is also called *job level dynamic scheduling* algorithm. *Least Laxity First* (LLF) [107] is an example of DP.

Let us consider two real-time tasks $\tau_a(T_a = 3, C_a = 1, O_a = 0, D_a = 3)$ and $\tau_b(T_b = 5, C_b = 2, O_b = 0, D_b = 5)$. Figure 2.3 illustrates a static preemptive scheduling of τ_a and τ_b for a priority assignment such as τ_a has a higher priority than τ_b. Thus, τ_a is executed by the processor at each release. τ_b is executed by the processor only when τ_a is not active. If τ_b is release while τ_a is active or released, the execution of τ_b is delayed until τ_a ends (time 6 to 7). Likewise, if τ_a becomes active while τ_b is executing, τ_b is preempted (time 12) and is resumed after τ_a ends (time 13).

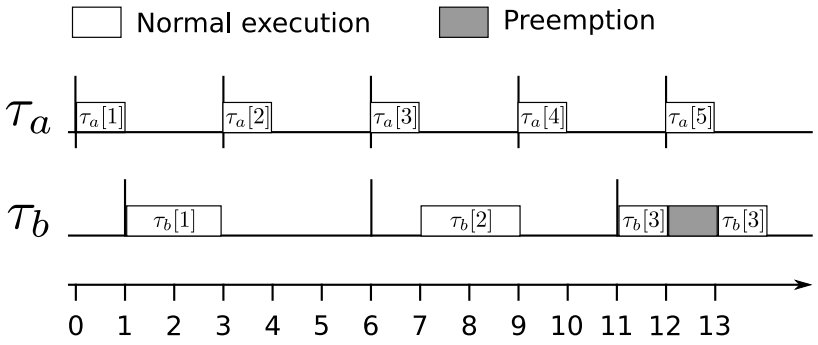


Figure 2.3: Example of a static preemptive execution of periodic real-time tasks on single-core.

2.3.2 Feasibility, schedulability and optimality

Definition 2.2 (Feasibility)
The scheduling of a set of real-time tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *is said to be* feasible *for a given scheduling algorithm if every job of every task* $\tau_i \in \boldsymbol{\mathcal{T}}$ *finishes before its deadline. Formally this is expressed as:*

$$R_i \leq D_i, \quad \forall\ \tau_i \in \boldsymbol{\mathcal{T}} \tag{2.1}$$

Definition 2.3 (Schedulability)
A set of real-time tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *is said to be* schedulable *if there exists at least one algorithm that produces a feasible scheduling.*

The optimality of a scheduling algorithm is given by Definition 2.4.

Definition 2.4 (Optimality)
A scheduling algorithm is said to be optimal *if it always finds a feasible scheduling when one exists.*

In practice, the optimality of a scheduling algorithm is defined according to its scheduling class and the studied task model. For example, in static preemptive scheduling, RM is optimized for synchronous periodic tasks with implicit deadlines [76], DM is optimized for synchronous periodic tasks with constrained deadlines [75] and OPA is optimized for asynchronous periodic tasks with constrained deadlines [5]. EDF and LLF are optimal for constrained-deadline tasks in FJP and DP, respectively.

The feasibility is verified by performing a *schedulability test* that is based on a scheduling condition.

- Sufficient condition: the scheduling is feasible when the condition is verified. However, if the condition is not verified, it does not guarantee that the scheduling is not feasible.
- Necessary condition: the scheduling is not feasible when the condition is not verified. However, if the condition is verified, it does not guarantee that the scheduling is feasible.
- Necessary and sufficient condition: the scheduling is feasible when the condition is verified. Conversely, the scheduling is not feasible when the condition is not verified.

2.3.3 Response time analysis

The *Response Time Analysis* (RTA) is a method that produces a necessary and sufficient schedulability test. It is mainly used to check the feasibility of static preemptive scheduling. RTA consists in computing the *Worst Case Response Time* (WCRT) R_i of each task τ_i and verifying Equation (2.1). More formally, let I_i be the maximum time accumulated by delays and preemptions of highest priority tasks on one execution of τ_i. R_i is expressed as follows:

$$R_i = C_i + I_i, \quad \forall\ \tau_i \in \boldsymbol{\mathcal{T}} \tag{2.2}$$

Joseph et Pandya [62] showed that a task suffers the most of interference when it is release at the same time as highest priority tasks. This instant is called the *critical instant*. Liu and Layland [76] showed that the critical instant is obtained at the first activation in a synchronous tasks sets. Then, they demonstrated that if a task meets its first deadline, it will meet all the subsequents. In that case, the RTA is reduced to the critical instant. This method can also be applied to asynchronous tasks sets if tasks share a critical instants [5].

For sets whose tasks do not share a critical instant, Audsley [5] introduced a *feasibility interval* that is based on one hyperperiod. He demonstrated that if a task meets its deadlines during this interval, it will meet all the subsequents ones. As such, the RTA requires the computation of the response time of every job during the feasibility interval. However, this hyperperiod analysis is not in polynomial time in general [5].

2.3.4 Multi-core scheduling

Multi-core scheduling considers a set of n real-time tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ and a set of q cores. The multi-core scheduling consists in assigning tasks to cores and execute them in order to guarantee the timing constraints. To achieve this, two scheduling decisions are required: the *mapping* of tasks to cores and the *scheduling* of tasks. Mapping consists in assigning each task to a core. The scheduling decides the task that is executed by each core. Accordingly, we distinguish between the following multi-core scheduling approaches.

- Global scheduling: tasks are assigned to core at runtime with respect to a defined algorithm. Moreover, executions of a given task can migrate between cores.

- Partitioned scheduling: each task is allocated off-line to one core and migrations are not allowed. Furthermore, a single core algorithm is used to schedule each core.

- Semi-partitioned scheduling: the allocation mixes between the first two approaches. Some tasks are allowed to migrate while some are not.

In the rest of this work, we consider partitioned scheduling. This scheduling approach has the advantage of dividing multi-core scheduling into several single-core scheduling. As such, the existing results for the latter can be applied. However, the partitioning of tasks to cores is equivalent to a *Bin-Packing* problem, which is NP-hard [30].

Dhall and Liu [36] introduced the use of bin-packing approach to study the partitioned scheduling of real-time systems. For this purpose, they proposed the mapping and scheduling algorithms RM-NF (Rate Monotonic Next-Fit) and RM-FF (Rate Monotonic First-Fit) for independent periodic tasks with implicit deadlines. These algorithms use bin-packing Next-Fit and First-Fit heuristics to partition tasks, respectively. One can refer to Coffman *et al.* [30] for details on bin-packing heuristics. Oh and Son [84] used the same principle and proposed the algorithm RM-BF (Rate Monotonic Best-Fit). The latter uses the bin-packing Best-Fit heuristic to partition tasks. These works [36, 30, 84] lay the foundations of bin-packing approach to the partitioning of real-time systems. Since then, several techniques have emerged. One can see for example [34, 101] for a survey.

2.4 AUTOSAR standard

The *AUTomotive Open System ARchitecture* (AUTOSAR) [32] is an international consortium formed in 2003 between OEM and Tier 1 suppliers. This partnership aims at providing a solution to meet the challenges imposed by the increasing complexity of embedded automotive applications. As such, it defines the AUTOSAR standard, which stands today as the one common industrial standard for the design and development of safety-critical and hard real-time automotive applications. This section describes the general principles and the methodology for designing AUTOSAR applications.

Subsection 2.4.1 gives a brief overview of the software architecture of AUTOSAR. Subsection 2.4.2 details the structure of AUTOSAR applications and characterizes the interactions between elements of the applications. Then, Subsection 2.4.3 outlines the AUTOSAR methodology to design hard real-time systems in automotive.

2.4.1 Software architecture

AUTOSAR software architecture is based on a modular description divided in several layers. In fact, each element in the AUTOSAR architecture is associated with interfaces through which it interacts with others elements of the same and/or different layers. At the highest abstraction level, this architecture is composed of three layers as illustrated in Figure 2.4.

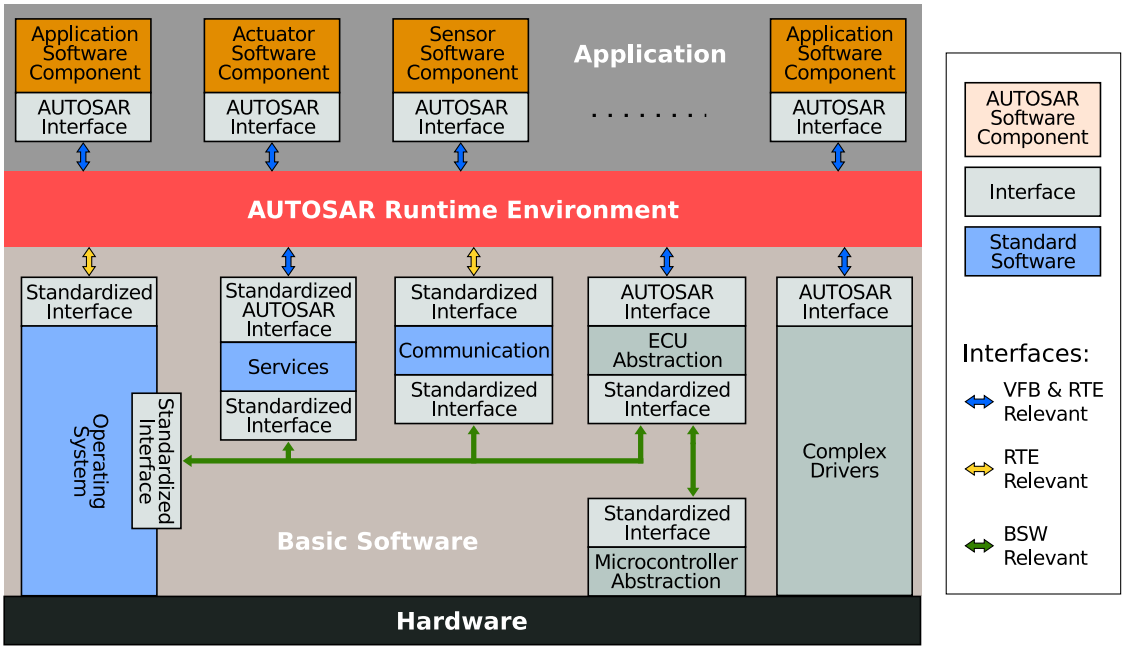


Figure 2.4: AUTOSAR layered software architecture.

The *Application Layer* contains the application codes. The latter are composed of sets of interconnected components called *Software Component* (SWC). Each SWC

encapsulates the implementation of an application functionality, with an abstraction level that is independent of the platform. Examples of SWC from ACS are the subsystem that detect collisions (noted *Collision SWC*) and the one that triggers the gas Inflator (noted *Inflator SWC*).

The *Basic Software* (BSW) is a layer that is strongly connected to the platform. In fact, this layer is responsible of providing infrastructural functionalities to the application layer. As such, the BSW is composed of a *Real-Time Operating System* (RTOS) and a collection of software modules providing access to communication resources, memory, *Input/Output* (I/O), *etc.*

The *Runtime Environment* (RTE) is the central layer of AUTOSAR architecture. This layer is in charge of providing capabilities that enable communications to occur between SWCs, as well as acting as the means by which SWCs access to the BSW.

2.4.2 Application concepts

In AUTOSAR, an application is modeled as a set of interconnected SWCs. Each SWC is a reuse piece of functionality or a large block (*i.e.* an entire application), which is independent of the hardware and available technologies.

Ports and interfaces

Each SWC has well-defined ports through which it interacts with other components. A port specifies one communication interface of a given component. This interface defines the services and/or data that are required and/or provided on the port. For example, the Collision SWC may have one or several ports to interact with the Inflator SWC. The interfaces of these ports will contain the necessary data for the Inflator SWC to command the gas inflation.

Several interfaces can be defined in AUTOSAR, each of which corresponds to a communication mode. The most common are the following.

- *Client-Server* (CS) communication: it involves at least one *client* that requires a service and one *server* that provides this service. CS is a two-way communication because the client sends the request to the server, which sends back the response. This communication is synchronous if the client waits for the response, and asynchronous otherwise.

- *Sender-Receiver* (SR) communication: it consists in a transmission and reception of data elements. These latter are sent by one component and received by one or more components. SR communication is asynchronous in the sense that the sender is not blocked and neither expects nor gets a response from receivers.

Figure 2.5 illustrates an AUTOSAR application consisting in three SWCs. SWC 1 uses one port for CS communications with both SWC 2 and SWC 3. It also uses two separate ports for SR communications with SWC 2 and SWC 3, respectively.

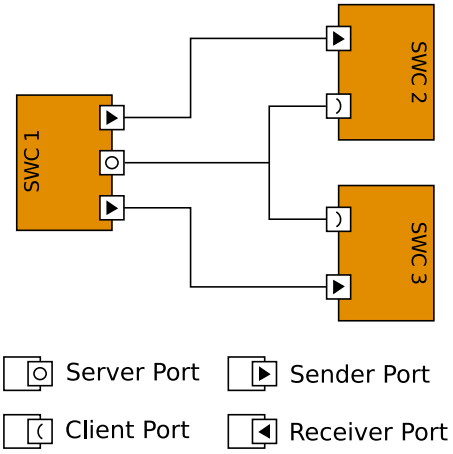


Figure 2.5: Example of an AUTOSAR application software.

The concept of "Runnable"

The Runnable entity is the atomic part of SWCs. It represents a sequence of instructions that describes all or part of the application behavior. That is, a SWC may consist of one or several Runnables. An example of Runnable entity in the Collision SWC is the function that computes the deceleration of the vehicle.

The set of Runnables that composes an atomic SWC is defined as its *Internal Behavior* (IB). As such, Runnables of the same IB interact through typed shared variables implemented using either *Exclusive areas* (EA) or *Inter-Runnables Variables* (IRV). The EA is a section of code in which Runnables cannot access concurrently. The IRV is a SR like buffer, which is reserved to Runnables of the same IB. Moreover, Runnables of different IB interact through the ports of their SWC. Figure 2.6 illustrates an internal behavior of a SWC composed of three Runnables.

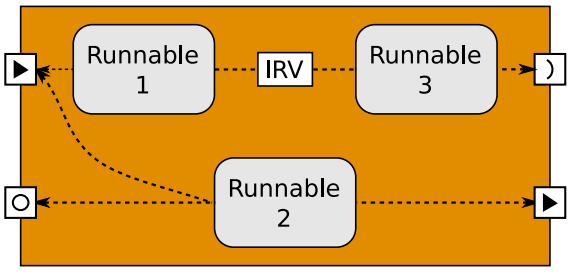


Figure 2.6: Illustration of the internal behavior of a SWC.

On the execution platform, Runnables are executed independently of each other. In fact, a Runnable is started by events from the RTE. Nonetheless, it can also suspend its

execution in order to wait for several RTE events. Accordingly, AUTOSAR defines two categories of Runnables:

- Category 1: this category of Runnables does not have *wait points*. Namely, they do not wait for RTE events. So, they are always completed in a finite time.
- Category 2: this category of Runnables has at least one waiting point.

The main advantage of Runnables of Category 1 is that their WCET can be determined accurately because they do not depend on waiting points (which may rely on unpredictable events). However, Runnables of Category 2 require advanced statistical analysis to determine bounds on their WCET [113]. In this thesis, we consider only Runnables of Category 1 as they allow temporal predictability.

Communications

Communications between Runnables (via port or IRV) are performed via shared buffers. The latter are handled by the RTE, which defines two communication modes.

- Implicit mode: the Runnable uses private local buffers to access the data. In fact, before the Runnable starts, the RTE automatically reads and copies the data from shared to local buffers. Conversely, after the Runnable has terminated, the RTE automatically writes the (different) data from local to shared buffers. Access to these local buffers from the application are performed using implicit RTE APIs. Implicit accesses are always non-blocking.
- Explicit mode: the Runnable accesses directly to shared variables by means of explicit RTE APIs. While explicit sends are always non-blocking, explicit receives can be either blocking or non-blocking. When it is blocking, the Runnable is blocked in a waiting point until the reception of the data.

Implicit read has the advantage of maintaining the same versions of data during the execution of the Runnable. Modifications of shared buffers during this execution do not affect local buffers. In implicit write, intermediate results are keep local and only the last values are updated in the shared buffers of others Runnables. As such, the implicit mode ensures the data coherency of the application. However, it increases the memory by using local buffers. This it is not the case for the explicit mode. However, the latter does not ensure the data coherency, which must be managed by the designer.

2.4.3 AUTOSAR Methodology

The AUTOSAR methodology defines the steps for deploying AUTOSAR softwares on a given hardware architecture. These steps are illustrated in Figure 2.7 and are described as follows.

Step 1: **Description of application softwares** as a composition of interconnected SWCs. The interactions between these SWCs are managed by a *Virtual Functional Bus* (VFB). The latter is a virtual bus to allow communications between SWCs regardless of their ECUs.

Step 2: **Distribution of the application softwares on the hardware architecture**: this step consists in mapping the SWCs on available ECUs. The mapping is constrained by the hardware architecture that expresses system constraints (*e.g.* network capabilities) and ECU constraints (*e.g.* number of processing core).

Step 3: **Configuration of ECU**: for each ECU this step implies the configuration of the RTOS (*e.g.* tasks, scheduling) and the BSW.

2.5 Configuration of AUTOSAR applications

In this section, we study the configuration of an AUTOSAR application. This study is important to understand how an AUTOSAR application runs on an ECU and to define the appropriate real-time task model.

Subsection 2.5.1 defines the task model and the execution mechanisms. Then, Subsection 2.5.2 outlines the configuration steps of an AUTOSAR application.

2.5.1 OS and task execution

Executions of Runnables are performed in the context of hard real-time tasks. The latter are provided by the AUTOSAR RTOS, which defines a partitioned static priority scheduling policy. Moreover, the RTOS provides a concurrent and asynchronous execution paradigm together with a scheduler that organizes the executions sequences.

Task framework

AUTOSAR defines two types of real-time tasks:

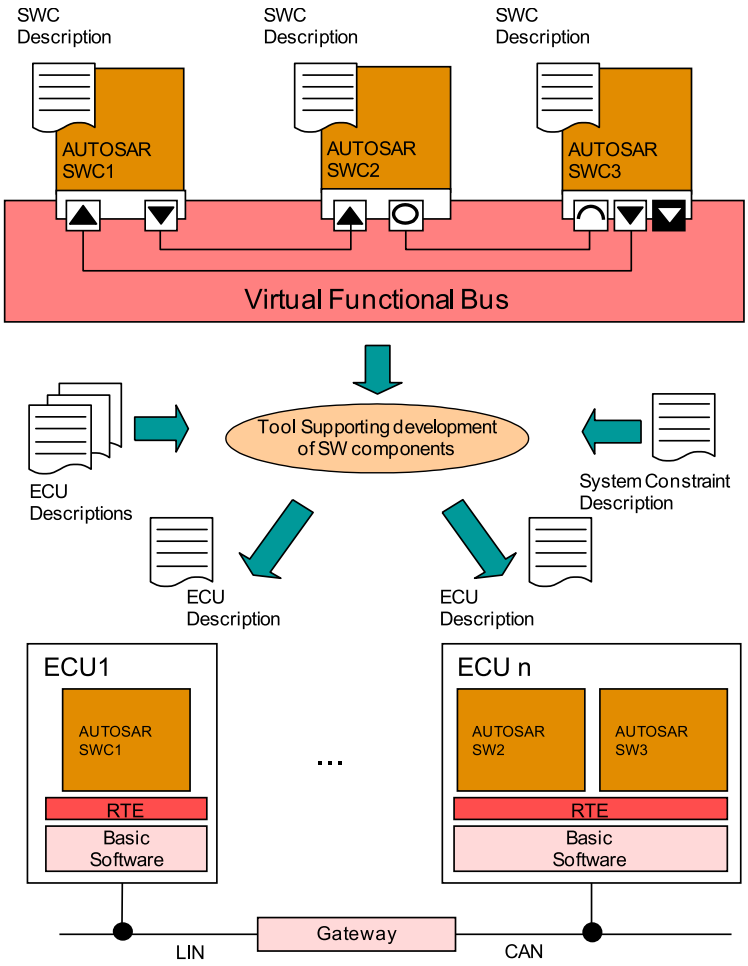


Figure 2.7: AUTOSAR Methodology.

- Basic task: once a basic task has begun executing in the processor, it can exit the latter only at the end of its execution or when it is preempted by higher-priority tasks. As such, a basic task contains only Runnables of Category 1.

- Extended task: beside the specification of a basic task, an extended task can invoke *wait for events* statement. As such, it can exit the processor to enter a *waiting state*. Its execution resumes in the processor only when the required events have occurred. Extended tasks can contain both categories of Runnables.

In the rest of this thesis we consider only basic tasks. The reason of this choice is that basic tasks allow deterministic temporal behavior. This is an important feature when modeling hard real-time systems.

Figure 2.8 shows the state machine of basic tasks. Each basic task that is not activated or released by the RTOS is in the *suspended* state. Once it is activated, the basic task enters the ready state and remains in this state until it is executed by the processor. As soon as it execution starts, the task enters the *running* state. It remains in this state until it is completed or preempted and returns to the suspended or the ready state, respectively.

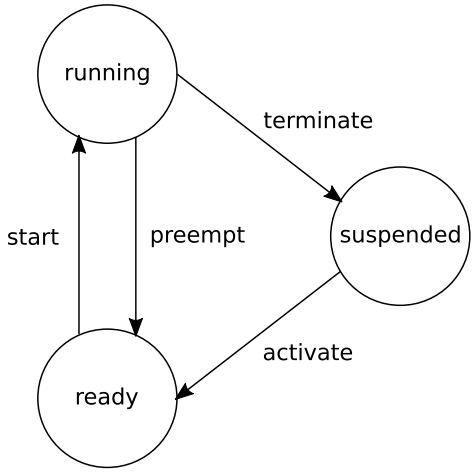


Figure 2.8: State machine of a basic task.

The state machine in Figure 2.8 describes the real-time task model in Section 2.2. In fact, the transition from the suspended state to the ready state corresponds to the activation of one job. The transition from the ready (*resp.* suspended) state to the suspended (*resp.* ready) state corresponds to the starting (*resp.* preemption) of a job. Finally, the transition from the ready state to the suspended state is the termination of a job. Consequently, we use the real-time task specification of Section 2.2 to study AUTOSAR basic tasks. The periodic task model is obtained by using periodic activations.

Furthermore, each basic task can be configured either as preemptive or non-preemptive. The AUTOSAR RTOS allows the coexistence of both preemptive and non-preemptive tasks. However, we consider only preemptive tasks with a static preemptive scheduling. That is, priorities are assigned to basic tasks so that the higher-priority task is always executed.

2.5.2 OS-Application and configuration

Configuring an AUTOSAR application to run on a platform amount to configuring the objects of the RTOS which specify the execution (tasks, alarms, schedule tables, counters, *etc.*). These objects are in general grouped into a collection that forms a cohesive functional unit called *OS-Application*. The RTOS objects that belong to the same OS-Application have access to each other. The right access to objects from other OS-Applications is granted during configuration and is managed by the *Operating System* (OS). For example, communication between SWCs of different OS-Applications are managed by *Inter-OS-Applications Communication* (IOC). The latter is a sender-receiver like communication managed by the OS. Indeed, communications between SWCs of different OS-Applications are managed by the RTE using IOCs.

Accordingly, the configuration of an AUTOSAR application on multi-core consists in the following steps (illustrated in Figure 2.9).

1. Definition of OS-Applications: several OS-Applications can be defined for the same platform. In particular, at least one OS-Application must be defined per core. Then, the RTOS is responsible for scheduling the available processing resources between OS-Applications that share the same core.

2. Mapping of SWCs to OS-Applications: each SWC is statically mapped to one OS-Application.

3. Mapping of Runnables to tasks: several Runnables can execute in the context of one OS task. However, Runnables of the same IB must be mapped to the same OS-Application. The mapping of Runnables to tasks is performed statically during configuration.

4. Scheduling configuration: this step consists in configuring tasks offset and deadline, preemptive or non-preemptive behavior, priorities, tasks activation, events, alarms, and schedule tables, *etc.* Then, tasks are scheduled using the corresponding scheduling policy.

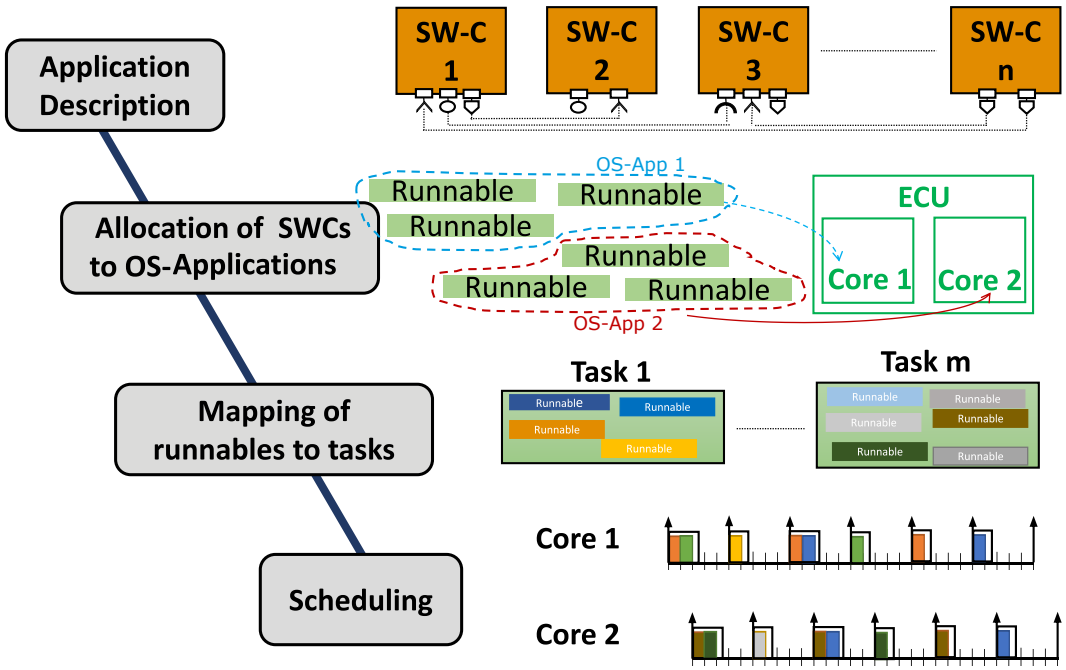


Figure 2.9: Configuration of an AUTOSAR application.

The application configuration is afterward coupled with that of BSW. The whole is compiled for a specific ECU.

2.6 Multi-core Electronic Control Unit

An ECU is an on-board computer system that controls electrical/electronic equipment such as sensors, actuators, displays and speakers. This section describes the architecture of the

multi-core ECU considered in our work.

Subsection 2.6.1 gives general reminder on multi-core microprocessors. Subsection 2.6.2 details the architecture of the micro-controller AURIX TriCore TC29x, which is the multi-core platform uses in ELA project.

2.6.1 Definitions

An ECU is built from a microprocessor, which is composed of one or multiple cores. The simplest multi-core microprocessor architecture is illustrated in Figure 2.10. This architecture is composed of several CPU cores, a memory hierarchy (*i.e.* RAM, Flash) and peripherals that are all connected by an on-chip bus. The cores execute the instructions of the program. Each core may have a *cache memory* or a *Scratchpad memory* (SPM). The cache memory refers to a small and fast local memory that buffers access to a larger but slower and higher latency memory. The SPM refers to a special high-speed memory circuit used to hold small items of data for rapid access. Peripherals are used to interact with the environment. Examples of peripherals are *Digital to Analog Converter* (DAC), *Controller Area Network* (CAN) and Ethernet.

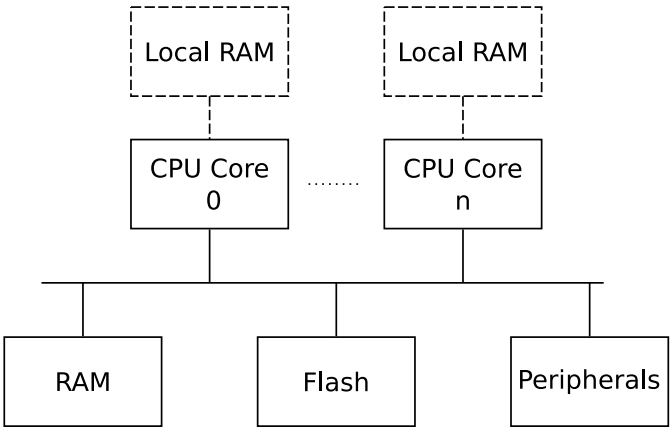


Figure 2.10: Multi-core architecture.

The basic idea behind multi-core architectures is the integration of several independent processing units on a single chip. As such, we can distinguish between the following architectures.

- **Homogeneous multi-core**: it consists of the replicas of the same basic core. That is, all cores in the architecture have the same instruction sets and the same data structures.

- **Heterogeneous multi-core**: it consists of using different types of basic cores. Consequently, cores in this architecture may have different instruction sets and different data structures.

Furthermore, a *Massively Parallel Processor Array* (MPPA) is a many-core architecture that consists in several computing *clusters* that communicate through a *Network on Chip* (NoC). These clusters have the same architecture as in Figure 2.10. Cores have access to local memories of their clusters via the on-ship bus. They access to the memory of other clusters and to peripherals via the NoC. Example of MPPAs are Intel SCC [98], Kalray MPPA-256 [37] and Tilera TilePro64 [10].

2.6.2 AURIX TriCore TC29x micro-controller

The target platform of this work is the micro-controller AURIX TriCore TC29x from Infineon [1]. The architecture of this micro-controller is presented in Figure 2.11. It consists of three cores, each with a dedicated local cache memory and a SPM for both data and instructions separately. The local SPM is accessible by others cores in a non-uniform way. That is, access to local memories are faster than access to others memories. Furthermore, Core 0 is designed for high efficiency, but has a smaller local memory compared to others. Core 1 and Core 2 are identical, but are designed for high performance. The three cores have the same instructions sets and the same data structures. The AURIX TriCore TC29x architecture also contains a shared RAM, a flash memory and several peripherals that are connected via two independent on-chip buses. Namely, the *crossbar* connects each core to high bandwidth modules (*i.e.* cores, RAM, Flash, DMA), and the *System Peripheral Bus* (SPB) connects high bandwidth modules to peripherals. A transparent bridge is used for transfers from the SPB to the Crossbar and vice versa.

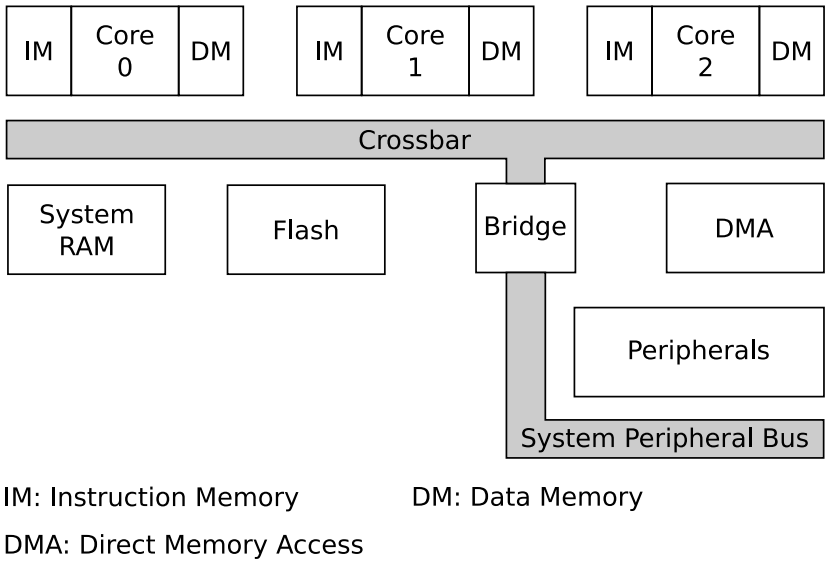


Figure 2.11: Architecture of the AURIX TriCore TC29x micro-controller.

The AURIX TC29x is chosen as reference hardware in the ELA project for the following reasons:

(i) It provides a highly predictable architecture with duplicated memories and crossbars

to avoid resource conflicts. These are key features for the design of safety-critical applications on multi-core platforms [112].

(ii) It is a well known industrial multi-core platform that is used in many automotive application fields [1].

(iii) They are several software development suite for the implementation of multi-core AUTOSAR applications on AURIX TC29x. An example is the Microsar package [81].

In this thesis, we adopt the architecture of the AURIX TC29x as basis of our investigations. Accordingly, we derive the following assumptions:

1. Hardware predictability can be ensured and resource conflicts can be avoided.
2. The multi-core architecture is homogeneous.
3. Cores use shared memories.

Our hardware assumptions outline an homogeneous shared memory architecture. However, our work does not use strong hypotheses on the memory architecture. As such, it can also be applied to MPPA.

2.7 Conclusion

In this chapter, we provided the background to the analysis of real-time implementation of AUTOSAR application on multi-core. In fact, we presented and specified the general principles on the mapping and scheduling real-time systems on partitioned multi-core. Then, we detailed the architecture of AUTOSAR, we presented the principles of interactions between Runnables and we showed how to apply the real-time theory to AUTOSAR applications. We also characterized the multi-core architecture considered in our work.

In the next chapter, we detail our problematic and we describe our approach. Then, we provide relevant related work.

Chapter 3

Motivations and related works

Contents

3.1 Introduction

In the automotive industry, *Model-Based Design* (MBD) is a widely used approach to design embedded applications. This approach consists in designing and specifying applications with high level models like Matlab/Simulink. Then, the applications are implemented on AUTOSAR. However, with the increasing complexity of applications and the adoption of multi-core, elaborated methods are needed to overcome the paradigm gap between Matlab/Simulink and AUTOSAR. This is required to efficiently manage the transformation so that the AUTOSAR multi-core configuration preserves the functional semantics of Matlab/Simulink, while exploiting the parallelism.

In this chapter, we describe our approach to enable implementations that preserve the semantics of data flow on multi-core. This approach is based on the formalism of SDFG [73]. The latter allows us to implicitly exploit parallelism. Furthermore, we present and position our approach compared to related work.

The rest of this chapter is organized as follows. Section 3.2 states the industrial challenge addressed by this work and describes our approach. Section 3.3 presents related work on semantics-preserving approaches, multi-core frameworks and SDFG. Section 3.4 recalls the analytical properties of SDFG used in the thesis. Finally, Section 3.5 concludes the chapter.

3.2 Problem statement and approach

This section provides a brief state of art on the designing process of automotive embedded applications and characterizes the industrial challenge addressed in this thesis. From this, we present our approach to enable a semantics-preserving implementation of the dataflow using the formalism of SDFG. We justify the interest of using SDFG and present our contributions to set a design framework from multi-periodic Matlab/Simulink systems to AUTOSAR multi-core.

3.2.1 Industrial challenge

In the automotive industry, embedded applications are designed using MBD. That is, the functional specification is designed and validated with a Matlab/Simulink [21] model. Thereafter, the latter is transformed into an AUTOSAR software architecture and implemented following the AUTOSAR methodology. Although MBD provides an efficient common framework throughout the design process, moving from the functional specification of Matlab/Simulink to the AUTOSAR configuration is not straightforward. In fact, as illustrated by the V-cycle in Figure 3.1, the deployment of the functional specification on the AUTOSAR platform involves teams from different entities. In fact, the functional specification in Matlab/Simulink is made by domain experts from *Original Equipment*

Manufacturers (OEM), while the AUTOSAR implementation is performed by software experts from Tier 1 suppliers.

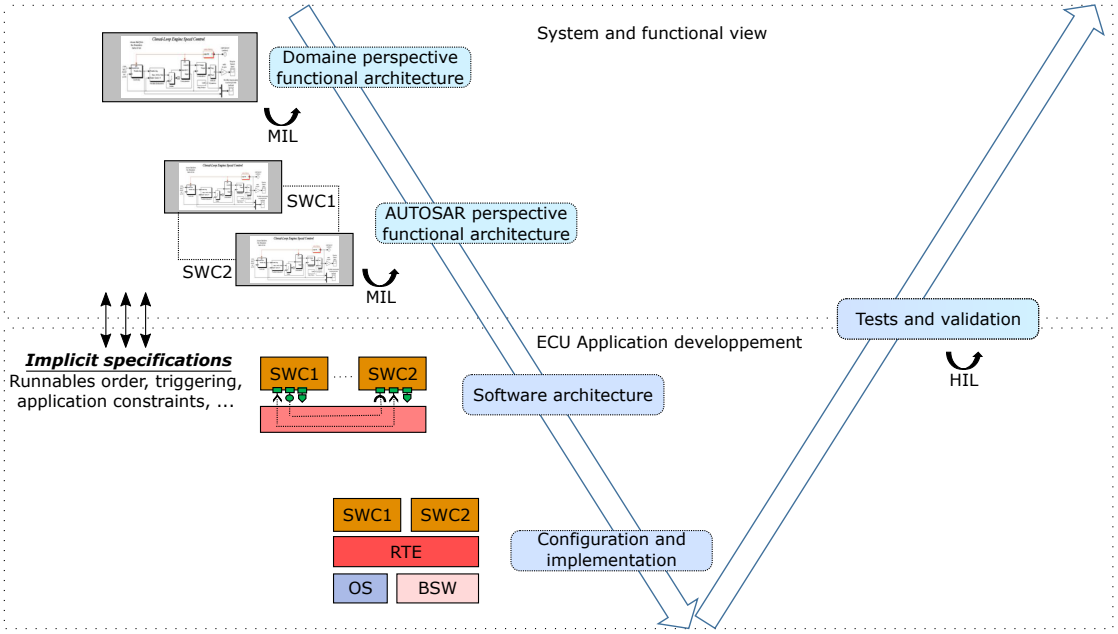


Figure 3.1: V-cycle for the design of applications.

At OEM side, two perspectives of the functional model can be distinguished:

- Domain perspective: it is a Matlab/Simulink model designed by control engineers for functional concerns. The domain perspective is the highest level of representation because it specifies the application with control-command semantics. The functional semantics of this perspective is validated by simulation.
- AUTOSAR perspective: it is a Matlab/Simulink model designed by software engineers according to the domain perspective. In fact, the AUTOSAR perspective is a Matlab/Simulink model structured as an AUTOSAR application software (*i.e.* set of interconnected SWCs). This perspective is validated by simulation with respect to the domain perspective.

At the Tier 1 supplier's, the AUTOSAR software is extracted from the AUTOSAR perspective to perform the AUTOSAR configuration. The result is tested and validated by several *Hardware-In-the-Loop* (HIL) simulations.

To overcome the paradigm gap between Matlab/Simulink and AUTOSAR configuration, OEM and Tiers 1 suppliers exchange information on the operation of the system. These exchanges are in several formats (*e.g.* functional model, textual specifications, business requirements) to guide the Tier 1 supplier towards a valid implementation. However, the process is error-prone and involves teams from different entities. As such, correction of errors are very costly. Furthermore, Matlab/Simulink enforces a certain sequential execution order of applications. In mono-core, this execution order is the specification of the

AUTOSAR configuration. In multi-core, this sequential execution order is not efficient to exploit the parallelism. That is, an accurate characterization of interactions in the Matlab/Simulink functional model is required to ensure the correctness of the implementation and exploit the parallelism of multi-core ECUs.

With the criticality level and the increasing complexity of embedded applications in automotive, the transformation from Matlab/Simulink to AUTOSAR multi-core has become a challenging issue. Advanced techniques are needed to effectively manage this transformation to preserve the functional semantics and exploit the parallelism in multi-core ECUs.

3.2.2 Approach and achievements

This thesis studies the semantic-preserving implementation of the dataflow from the functional model in Matlab/Simulink to the AUTOSAR configuration. For this purpose, we base our investigations on the AUTOSAR perspective of the Matlab/Simulink specification. We assume that this specification is correct by construction and is already formally proven. Hence, we consider a formal approach to construct the AUTOSAR configuration to preserve the functional semantics of the dataflow and exploit the parallelism on a given multi-core ECU.

We assume that each Matlab/Simulink model is validated by a simulation, which enforces the data exchanges in the application. As such, these data exchanges must be implemented deterministically on the ECU to guarantee the validated functionality. On the one hand, Matlab/Simulink allows to generate the semantic-preserving embedded code of Runnables automatically. However, it is the responsibility of the AUTOSAR configuration to implement the correct data exchanges between Runnables. On the other hand, the AUTOSAR configuration must account for real-time constraints as well as non-functional requirements such as performances (*e.g.* total execution time) and multi/many-core constraints (*e.g.* parallelism, mapping).

To get an implementation that complies with the aforementioned requirements, we introduce the use of an abstract representation of the functional dataflow. The interest of this abstraction is to get a formal and accurate tracking of the simulated data exchanges. This allows the extraction of the analytic properties of the functional dataflow to perform the AUTOSAR configuration. We rely on the compositionality offers by AUTOSAR to focus on the interactions between Runnables. Then, we use SDFG [73] as abstract representation. We choose SDFG because of its solid mathematical background and its existing analysis tools for multi/many-core. Moreover, SDFG is an excellent analysis tool to exploit parallelism because it is very popular in the literature and it is largely studied for the deployment of dataflow applications on multi/many-core platform [73].

Our work links between the functional specification of multi-periodic Matlab/Simulink systems and AUTOSAR configuration by extracting the functional dataflow with SDFG. To this end, we perform an in-depth study of the modeling and simulation environment of

Matlab/Simulink in Chapter 4. From this study, we define a set of rules to statically specify the functional behavior of the system. In particular, we rely on the compositionality of AUTOSAR to focus on the interactions between Runnables. We consider that the behavior of each Runnable can be generated as a sequential code in a semantics-preserving way. As such, the functional behavior of the application is determined by the dataflow between Runnables. We proposed a technique to model these interactions by a SDFG. The latter characterizes the precedence constraints such as expressed by Matlab/Simulink.

Furthermore, in Chapter 5 and 6 we exploit the analytical properties of SDFGs to express a real-time task scheduling technique that guarantees the functional dataflow without blocking synchronization mechanisms. As such, both functional and temporal determinism are guaranteed. We characterize the impact of the dataflow requirements on the scheduling and we propose a partitioning technique that minimizes these impacts. We show that this initialization technique promotes the construction of a feasible scheduling. Hence, we propose an integrated framework to use both the SDFG specification and the initialization technique in a semantic-preserving implementation that exploit the parallelism. This framework assists the designer to shorten design time and reduce the number of design iterations.

3.3 Related work

This section presents the related work to our approach and positions our contributions. More precisely, we present related work on synchronous programming models and architecture description languages to lay the basis of deterministic and semantics-preserving methods. Then, we present related work on design framework for multi-core to explore parallelism and facilitate the design process. Finally, we introduce works on multi-core analysis using SDFG.

3.3.1 Synchronous programming models

In *Model-Based Design*, synchronous programming models are widely used to design and specify safety-critical systems [12]. Example of synchronous programming models are synchronous languages such as Esterel [15], Lustre [25] and Signal [13], and design environments like Matlab/Simulink. In the *synchrony hypothesis* of synchronous programming models, time is a sequence of discrete instants so that computations complete instantaneously at each instant. That is, there is no computation between two consecutive time instants. The synchrony hypothesis simplifies the specification and verification of systems by replacing the real time by a logical time. As such, synchronous programming models express deterministically the exact functional behavior. However, to benefit from this specification, the implementation of synchronous models must preserve the synchronous semantics.

The synchronous languages ESTEREL [15], LUSTRE [25] and SIGNAL [13] have addressed the semantic-preserving implementation of synchronous models. In fact, they provide formal approaches to compile a synchronous program to a single sequential code realizing the functional semantics [11]. However, fully sequential codes are not efficient neither for multi-rate systems nor in multi-core platforms [26, 85]. Thus, several works studied the semantics-preserving multi-task implementation of synchronous programs [47].

Among the existing works, Aubry *et al.* [3] translated a SIGNAL [13] program into a graph, where nodes represent basic computations and edges are data dependencies between nodes. Each edge is characterized by a clock indicating when the data is present. Thus, according to a partition given by designers, they partition the initial graph into a set of subgraphs. This graph transformation mainly consists in characterizing the communication needs between partitions and verifying that no circuit is created. The subgraphs are thereafter translated into sequential codes. Girault and Nicollin [48], and Girault *et al.* [49] automatically distributed a LUSTRE [25] program into several tasks. They used a clock partition provided by the designer to automatically generate the task associated to the computations driven by each clock.

To ensure that the distributed code has the semantics of the centralized program, authors in [3] and [48, 49] have used blocking communication mechanisms to synchronize tasks. However, scheduling policies with blocking mechanisms are not sustainable. In fact, a system can become unschedulable online (when execution in less than the WCET), while it has been proved offline to be schedulable [53]. Thus, Sofronis *et al.* [102] and Caspi *et al.* [26] studied multi-task implementation of synchronous programs on single-core and proposed a lock-free inter-task buffering protocol called *Dynamic Buffering Protocol* (DBP). Zeng and Di Natale [117] extended the protocol to multi-core.

Works of Aubry *et al.* [3], Girault and Nicollin [48], Girault *et al.* [49], Sofronis *et al.* [102] and Caspi *et al.* [26] laid foundations for semantics-preserving multi-task and distributed implementation of synchronous programs. However, they do not address neither the real-time scheduling nor the mapping, which are out of their scope. That is, the SYNDEX [72] environment provides both a language and a framework to help designers in optimizing the implementation of real-time applications on multiprocessor. SYNDEX implements the *Algorithm-Architecture Adequation* (AAA) methodology, where applications are modeled by *Directed Acyclic Graph* (DAG) called *software graphs* and hardwares are modeled by undirected graphs called *hardware graphs*. These graphs exhibit both the potential parallelism of the application algorithm and the available parallelism in the platform. Then, the implementation is formalized as a graph transformations to distribute and schedule the application on the platform. Several heuristics are proposed to explore solutions that optimize the response time and resources allocation, while satisfying real-time constraints [103, 54].

Matlab/Simulink models can be translated into synchronous programs [24, 13, 15], so that above mentioned works can apply. However, neither synchronous languages [15, 25, 13] nor SYNDEX [72] allow to exploit the compositionality offers by AUTOSAR, to focus on the interactions between Runnables and exploit their parallelism. In our work, we

assume that tasks code can be generated in a semantic-preserving way. As such, we focus only on the semantic-preserving implementation of the interactions between tasks. In this sense, our approach is more akin to works on *Architecture Description Language* (ADL).

3.3.2 Architecture Description Languages

An *Architecture Description Language* (ADL) provides a higher abstraction layer than above-mentioned synchronous languages. In fact, an ADL focuses on interactions between high-level components like tasks, instead of all basic computations.

The multi-periodic synchronous dataflow language PRELUDE [44], is an ADL to design and assemble interacting locally mono-periodic synchronous systems into a globally multi-periodic synchronous system. The PRELUDE program is thereafter transformed into a set of dependent tasks related by precedence constraints expressed at job level [45]. On single-core, PRELUDE compiler implements deterministic communications between tasks without synchronization mechanisms. This is done by ensuring that consumer jobs always starts after producer jobs and that the producers has a higher priority than consumers [46]. On multi-core, PRELUDE uses explicit synchronization mechanisms, but prevents scheduling anomaly issues by forcing the execution of tasks to take the WCET [85].

GIOTTO [58] provides a logical abstract programming framework for the implementation of embedded control systems. In GIOTTO, a control application consists of periodic tasks that communicate through *ports*. Each task has a logical start time and a logical end time corresponding to the start time and the end time of its execution period, respectively. GIOTTO logical abstraction does not specify when, where, and how the actual computation is performed on the platform. However, the time when the input (*resp.* output) ports are updated is determined and is equal to the logical start (*resp.* end) time. By making communications independent of the actual execution start time and end time, GIOTTO enables a functionality determinism of the implementation.

The OASIS [27, 77, 78] environment provides a framework for implementing multi-task safety-critical real-time systems. In OASIS, an application is a set of parallel communicating tasks called *agents*. Each agent is an execution entity composed of a sequential number of processing operations. Each processing operation is associated with an execution interval defining its earliest start time and its latest end time. Then, an OASIS task is modeled by an execution graph called *state-transition diagram*, where each node expresses a temporal constraint (*e.g.* start date, deadline) and each edge is a basic processing operation between two nodes. The temporal dates associated to nodes define the synchronization points of agents. In fact, agents communicate through non-blocking mechanisms implemented either as temporal variables or messages. A temporal variable is a shared memory where modifications are made visible only at synchronization points, while messages has explicit definition of visibility dates. Hence, by combining temporal constraints on executions and communications, OASIS ensures a temporal and behavioral determinism of the real-time implementation.

Our work is similar to PRELUDE, GIOTTO and OASIS as we rely on the compositionality offers by AUTOSAR to focus on the interactions between Runnables. We consider that the behavior of each Runnable can be generated as a sequential code using a semantics-preserving approach. Therefore, the behavior of the application is determined by the interactions (*i.e.* data exchanges) between Runnables in the functional specification.

However, our approach differs from the latter because we study the deterministic semantic-preserving implementation of the dataflow by modeling the functional dataflow with SDFG. We rely on the solid mathematical background on SDFG to study the mapping and scheduling of control applications designed and validated with Matlab/Simulink.

3.3.3 Frameworks for multi-core

The adoption of multi-core architectures and the growing complexity of embedded features have encouraged the development of several integrated design frameworks. Among others, the AMALTHEA project [2] provides an open source development platform for automotive multi-core systems. The AMALTHEA methodology [115] covers several design steps such as *Electric/Electronic* (E/E) architecture modeling, functional modeling, software modeling (*i.e.* intermediate abstraction), mapping and scheduling. Each step is implemented by a custom or an existing tool. For example, the functional model can be provided by Matlab/Simulink. The proposed framework enables interoperability between the steps by providing converters and tools for exchanges between steps [115]. In addition, the design flow of AMALTHEA can be tailored so that some steps can be ignored at the convenience of the user. For example, the functional modeling step can be ignored in AMALTHEA methodology.

The parMERASA project [88] investigated the parallelization and timing-predictable executions of hard real-time applications on multi-core. First, they split the application code into a set of *activities* to expose the maximum degree of parallelism [61]. Then, they agglomerate parallel activities to reduce the granularity of the parallelism to a reasonable level. This agglomeration is formulated as an optimization problem to reduce the makespan (total execution of the program), while reducing the costs of blocking synchronization and communications.

Our approach is different from those mentioned above as we target the semantic-preserving implementation of Matlab/Simulink specifications. In this purpose, we developed a SDFG model that simplifies the specification of the functional dataflow. This model provides a high level abstraction with strong mathematical background for the analysis of the semantics-preserving implementation of the dataflow on multi/many-core.

3.3.4 Multi-core analysis with Synchronous Dataflow Graphs

The mapping and scheduling on multi/many-core using SDFG [73] have been studied for decades. Works of Marchetti and Munier [79] demonstrate a sufficient condition on the existence of a scheduling for a SDFG. Munier [83] proposed a method to construct a scheduling that gives the maximum throughput. Marchetti and Munier [80] studied the periodic scheduling of SDFGs. These works provide the major mathematical properties on SDFG, which are the basis of the work of this thesis.

Several authors have used SDFG to study the mapping, the routing and the execution problem of applications on parallel and distributed architectures [52, 96, 89]. The AccessCore toolchain is a recent example of a compiler that transforms programs written in ΣC [4] into SDFG. ΣC is an enriched form of C programming language with some communication mechanisms to facilitate the programming of massively parallel applications. The SDFG obtained from ΣC is thereafter used to perform mapping and scheduling analysis in order to run the application on Kalray MPPA-256 [37]. However, they do not address hard real-time aspects.

For streaming applications, Bamakhrama and Stefanov [7, 8] introduced the hard real-time scheduling of applications modeled by *Cyclo-Static Dataflow Graph* (CSDFG) [17]. The CSDFG is an extension of SDFG where tasks can have multiple computation phase. However, they considered that the real-time attributes of tasks (periods and deadlines) are not expressed as input constraints. Thus, they compute these attributes to guarantee the maximum throughput [7] and to minimize the latency [8] of the application.

The work of this thesis target control applications, which often contain cyclic dependencies representing feed-back loops. Consequently, our SDFG model also contain cycles. Furthermore, the real-time attributes of tasks are driven by the dynamic of the physical system. As a result, the real-time attributes are input constraints that we guarantee during our mapping and scheduling analysis.

3.4 Synchronous Dataflow Graphs

In this section, we review some basic definitions on the SDFG formalism and we detail the analytical properties used in the thesis.

3.4.1 Basic definitions and notations

The *Synchronous Dataflow Graph* (SDFG) formalism was introduced by Lee and Messerschmitt [73] to model communications between tasks in dataflow applications. A SDFG is a directed graph $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ characterized as follows:

- $\boldsymbol{\mathcal{T}}$: is a set of nodes, where each node models a SDFG task $\tau_i \in \boldsymbol{\mathcal{T}}$.
- $\boldsymbol{\mathcal{A}}$: is a set of edges, where each edge $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$ models a *First-In-First-Out* (FIFO) buffer between two SDFG tasks τ_i and τ_j.
- $\boldsymbol{\mathcal{M}}$: is a set of initial markings $\boldsymbol{\mathcal{M}} = \{M_0(a) \mid a \in \boldsymbol{\mathcal{A}}\}$, where $M_0(a)$ is a non-negative number that represents the initial amount of tokens in the buffer a. A token denotes a data send/received via the buffer.
- in_a: is the *production rate* that denotes the amount of tokens produced in $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$ at each activation of τ_i.
- out_a: is the *consumption rate* that denotes the amount of tokens consumed in $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$ at each activation of τ_j.
- gcd_a: is the greatest common divisor of in_a and out_a for every buffer $a \in \boldsymbol{\mathcal{A}}$.

Note that the concept of SDFG tasks is not restricted to real-time tasks or periodic tasks. It also includes non real-time tasks and non periodic tasks. As a result, tasks parameters such as release times and periods are not equivalent for SDFG tasks and real-time tasks.

The SDFG formalism provides a static description of the dataflow of applications by enforcing that all executions of each task have the same behavior. As such, the amount of tokens produced (*resp.* consumed) in the buffer at each execution is fixed. Consequently, the dataflow is completely predictable.

Figure 3.2 illustrates a SDFG buffer $a = (\tau_1, \tau_2)$ between two tasks τ_1 and τ_2. There is an initial amount of $M_0(a) = 5$ tokens. Each execution of τ_1 produces $in_a = 3$ more tokens in the buffer. Each execution of τ_2 consumes $out_a = 5$ tokens from the buffer.

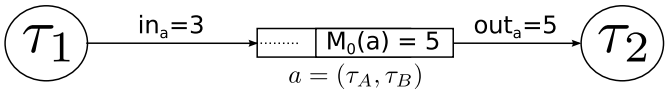
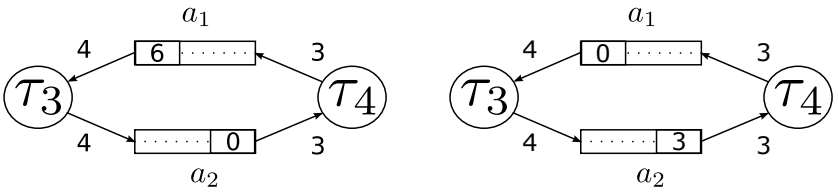


Figure 3.2: Example of SDFG with two tasks τ_1 and τ_2 connected by the buffer $a = (\tau_1, \tau_2)$.

3.4.2 Liveliness

The execution of a SDFG task τ_i depends on the presence or the absence of tokens in buffers. In fact, the amount of token in a buffer is always non-negative and the consumer task can execute only if there are enough tokens in the buffer. As consequence, each task can execute only when there are enough tokens in all it input buffers. In the example of Figure 3.2, task τ_1 has no input buffer and can run infinitely. However, τ_2 can run only when there are enough tokens in $a = (\tau_1, \tau_2)$.



(a) Example of alive graph. (b) Example of deadlock graph.

Figure 3.3: Example of alive and deadlocked graphs.

The liveliness is an important property that indicates whether all tasks of a SDFG can be executed infinitely often during the dataflow execution. A graph is said to be *alive* if all its tasks can be executed as many times as required. Otherwise, the graph is said to be *deadlocked*.

Figure 3.3(a) illustrates an example of an alive SDFG. In fact, as there are 6 initial tokens in buffer a_1, τ_3 executes once. This execution consumes 4 tokens from a_1 and produces 4 tokens in a_2. τ_3 cannot execute again because there are only 2 remaining tokens in a_1, which is not sufficient for its execution. However, τ_4 can execute once because there are now 4 tokens in a_2. This execution consumes 3 tokens from a_2 and produces 3 tokens in a_1. τ_4 cannot execute again because there is only 1 remaining token in a_2. However, τ_3 can execute once because there are now 5 tokens in a_1. We can repeat this reasoning indefinitely to check that there are always enough tokens so that τ_3 or τ_4 can executes.

The example of Figure 3.3(b) illustrates a deadlocked SDFG. In fact, τ_4 executes once because there are 3 initial tokens in a_2. This execution consumes 3 tokens from a_2 and produces 3 tokens in a_1. There are now 3 tokens is a_1 and no remaining token in a_2. As such, neither τ_4 nor τ_3 can execute because there are not enough tokens in a_1 and a_2, respectively.

Checking the liveliness of a SDFG is a difficult problem widely studied in the dataflow community [74]. In this work, we consider the sufficient condition of liveliness demonstrated by Marchetti and Munier-Kordon [79]. This condition is expressed by Theorem 3.1. The normalized SDFG in this theorem is a SDFG where the production and consumption rates corresponding to a given task are identical. This is because rates are associated to buffers and not tasks.

Theorem 3.1 (Sufficient condition of liveliness)
Let $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ be a normalized SDFG. $\boldsymbol{\mathcal{G}}$ is alive if for every cycle μ in $\boldsymbol{\mathcal{G}}$ the initial marking satisfies Equation (3.1).

$$\sum_{a \in \mu} M_0(a) > \sum_{a \in \mu} (out_a - gcd_a) \tag{3.1}$$

The SDFGs in Figure 3.3 are normalized. Thus, we can verify that the SDFG in Figure 3.3(a) satisfies the condition of Equation 3.1. Indeed, $\sum_{a \in \mu} M_0(a) = M_0(a_1) + M_0(a_2) = 6$ and $\sum_{a \in \mu}(out_a - gcd_a) = (4-1) + (3-1) = 5$.

3.4.3 Precedence constraints

Precedence constraints define the relationships between executions of a producer task and those of a consumer task through a SDFG buffer.

Definition 3.1 (Precedence constraints)
A buffer $a = (\tau_i, \tau_j)$ with an initial marking $M_0(a)$ models a precedence constraint from the n_ith job of τ_i to the n_jth job of τ_j if:

(1) $\tau_j[n_j]$ is the first execution of τ_j that requires a token produced by $\tau_i[n_i]$ and

(2) $\tau_j[n_j]$ can only be executed immediately after $\tau_i[n_i]$.

We denote by $\tau_i[n_i] \rightarrow \tau_j[n_j]$ the precedence constraint from $\tau_i[n_i]$ to $\tau_j[n_j]$ and by $\tau_i \rightarrow \tau_j$ the set of precedence constraints between τ_i and τ_j.

Figure 3.4 illustrates the execution model of the SDFG of Figure 3.2. The number 5 (in blue) circled represents the initial amount of tokens. *Top numbers* indicate the cumulative amount of tokens in the buffer after each execution of τ_1. For example, the first execution $\tau_1[1]$ produces 3 tokens in the buffer, which are added to the initial 5 to give the top number 8. *Down numbers* indicate the cumulative amount of tokens needed before each execution of τ_2. For example, the first execution $\tau_2[1]$ requires 5 tokens in the buffer. The second execution $\tau_2[2]$ requires 5 more tokens, which is accumulated to the previous to give the down number 10. *Arrows* indicate precedence constraints between executions of τ_1 and τ_2. For example, *arrow 1* indicates the precedence constraint $\tau_1[2] \rightarrow \tau_2[2]$. In fact, after $\tau_1[1]$ there are only 8 accumulated tokens in the buffer. This is not sufficient for the execution of $\tau_2[2]$. However, after $\tau_1[2]$ there are 11 accumulated tokens in the buffer. $\tau_2[2]$ can then execute for the first time only after $\tau_1[2]$.

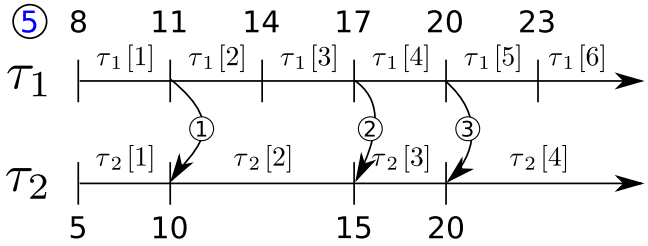


Figure 3.4: Example of precedence constraints between tasks τ_1 and τ_2 connected by the buffer $a = (\tau_1, \tau_2)$ of Figure 3.2.

The principle of precedence constraints is a key feature of SDFG. This principle is used throughout this thesis. Theorem 3.2 reminds the necessary and sufficient condition on the existence of precedence constraints between executions of two tasks. Theorem 3.2 was demonstrated by Marchetti and Munier-Kordon [80].

Theorem 3.2
A buffer SDFG $a = (\tau_i, \tau_j)$ models a precedence constraint between the n_ith execution of τ_i

and the n_jth execution of τ_j if and only if:

$$in_a > M_0(a) + in_a \cdot n_i - out_a \cdot n_j \geq \max(in_a - out_a, 0) \tag{3.2}$$

The condition of Equation (3.2) can be verified on the precedence constraint $\tau_1[2] \rightarrow \tau_2[2]$ (*arrow 1*) in Figure 3.4. This is done by replacing $in_a = 3$, $out_a = 5$, $M_0(a) = 5$, $n_i = 2$ and $n_j = 2$. So, we get $3 > 5 + 3 * 2 - 5 * 2 \geq \max(3 - 5, 0)$, which is equivalent to $3 > 1 \geq 0$.

3.4.4 Scheduling of Synchronous Dataflow Graphs

In general, the scheduling of a SDFG is based on the starting date of tasks. However, we based our analysis rather on the release date because we target a preemptive scheduling.

Definition 3.2 (Scheduling of SDFG)
Scheduling a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ consists of fixing the release time of each job according to a given strategy. More formally, it consists in setting the release time $r_i^{\mathcal{G}}[n_i]$ of any n_ith execution of every task $\tau_i \in \boldsymbol{\mathcal{T}}$.

Definition 3.3 (Periodic scheduling of SDFG)
A periodic scheduling of a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ is a scheduling that defines a period $T_i^{\mathcal{G}}$ *and a first execution staring date $r_i^{\mathcal{G}}[1]$ for each task $\tau_i \in \boldsymbol{\mathcal{T}}$. As such, the release time of $\tau_i[n_i]$ in the periodic scheduling is $r_i^{\mathcal{G}}[n_i] = r_i^{\mathcal{G}}[1] + (n_i - 1) \cdot T_i^{\mathcal{G}}$.*

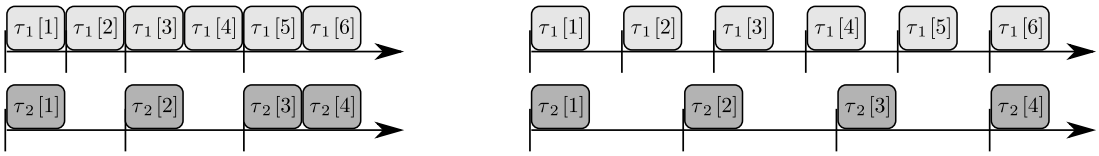
In the scheduling theory of SDFG, the execution time of a given task is the same for all its executions. However in our context, a preemptive execution of a given task τ_i is performed in a time interval, which is not the same for all its executions (*i.e.* different preemption scenarios per execution). So, we apply the scheduling theory of SDFG in our context by replacing the execution time of each task τ_i by an *execution interval* $D_i^{\mathcal{G}}$ greater than the maximum time intervals of the executions of τ_i. We define the scheduling of SDFG for our context in Definition 3.4

Definition 3.4 (Scheduling of SDFG for preemptive tasks)
Scheduling a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ for preemptive tasks consists in setting the execution interval for each task and fixing the release times of each job according to a given strategy.

From Definition 3.4, the execution interval $D_i^{\mathcal{G}}$ of each task τ_i is not a constant in our context. It is rather decided by the scheduling algorithm of the graph.

Let us assume arbitrary execution intervals for the tasks in Figure 3.2. Figure 3.5(a) illustrates a scheduling that uses precedence constraints to set the release times of tasks. Indeed, this scheduling sets the release times so that tasks are executed *As Soon As Possible* (ASAP). Namely, as soon as there are enough tokens in the buffer $a = (\tau_1, \tau_2)$. As such,

the scheduling executes τ_1 continuously. τ_2 is executed only after the end of the execution time of the job of τ_1 with a precedence constraint. For example, $\tau_2[3]$ is scheduled after $\tau_1[4]$ finishes because of the precedence constraint $\tau_1[4] \rightarrow \tau_2[3]$. Figure 3.5(b) illustrates a periodic scheduling of the same graph. This scheduling uses the same initial release time for τ_1 and τ_2, but enforces different scheduling periods.



(a) ASAP scheduling. (b) Periodic scheduling.

Figure 3.5: Examples of scheduling of the SDFG of Figure 3.2.

Definition 3.5 (Validity of the scheduling of SDFG)
A scheduling of a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ is said to be valid if the release times satisfy the precedence constraints.

The validity condition of the scheduling of a SDFG is characterized by Theorem 3.3.

Theorem 3.3
Let us consider a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ such as each job of $\tau_i \in \boldsymbol{\mathcal{T}}$ completes within the interval defined by its execution interval $D_i^{\mathcal{G}}$. The scheduling of $\boldsymbol{\mathcal{G}}$ is valid if for every buffer $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$ and every precedence constraint $\tau_i[n_i] \rightarrow \tau_j[n_j]$:

$$r_j^{\mathcal{G}}[n_j] \geq r_i^{\mathcal{G}}[n_i] + D_i^{\mathcal{G}} \tag{3.3}$$

Proof. If $r_j^{\mathcal{G}}[n_j] < r_i^{\mathcal{G}}[n_i] + D_i^{\mathcal{G}}$ then $\tau_j[n_j]$ can start executing before $\tau_i[n_i]$ completes. This does not guarantee the precedence constraint $\tau_i[n_i] \rightarrow \tau_j[n_j]$ because $\tau_j[n_j]$ must start only after $\tau_i[n_i]$ finishes. Otherwise, there are not enough tokens in the buffer for $\tau_j[n_j]$. As consequence, $r_j^{\mathcal{G}}[n_j]$ must be greater or equal to $r_i^{\mathcal{G}}[n_i] + D_i^{\mathcal{G}}$. □

The ASAP scheduling of Figure 3.5(a) is valid because each job of τ_2 starts only after the execution interval of the job of τ_1 with a precedence constraint. However, the periodic scheduling of Figure 3.5(b) is not valid because $\tau_2[3]$ starts executing before the end of the execution interval of $\tau_1[4]$. Therefore, the precedence constraint $\tau_1[4] \rightarrow \tau_2[3]$ is not guaranteed.

Next theorem, proved by Marchetti and Munier-Kordon [80], expresses the validity condition of periodic scheduling of SDFG.

Theorem 3.4
Let us consider the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. A periodic scheduling of $\boldsymbol{\mathcal{G}}$, which enforces a scheduling period $T_i^{\mathcal{G}}$ for every task $\tau_i \in \boldsymbol{\mathcal{T}}$, is valid if a rational $K = \frac{T_i^{\mathcal{G}}}{in_a} = \frac{T_j^{\mathcal{G}}}{out_a}$ exists for

every buffer $a \in \boldsymbol{\mathcal{A}}$, so that Equation (3.4) is verified.

$$r_j^{\mathcal{G}}[1] - r_i^{\mathcal{G}}[1] \geq D_i^{\mathcal{G}} + K \cdot (out_a - M_0(a) - gcd_a) \tag{3.4}$$

The validity condition of the scheduling of SDFG ensures the precedence constraints of the graph. It guarantees that consumptions are always made before productions. In fact, Theorem 3.3 and Theorem 3.4 enforce executions of consumers to start only after the end of executions of producers. This allows a predictive implementation of dataflow because each execution of a consumer has the guarantee of the presence of its input data.

3.5 Conclusion

The deterministic implementation of Matlab/Simulink models on AUTOSAR multi-core is an emerging issue in the automotive industry. In this chapter, we formulate the problem of semantic-preserving implementation of the dataflow in a multi-periodic Matlab/Simulink model on AUTOSAR multi-core. We relied on the compositionality offers by AUTOSAR to focus on the interactions between Runnables. We use SDFG [73] as abstract representation of the functional dataflow. Compared to related work, we provide a framework for semantics-preserving implementation of the dataflow in hard real-time system using a formalism adapted for multi-core.

In next chapter, we study the method to extract the functional dataflow in a multi-periodic Matlab/Simulink system by SDFG.

Chapter 4

Modeling multi-periodic Simulink systems by SDFG

Contents

4.1 Introduction

In the industry, Matlab/Simulink [21] is a specification and simulation tool widely used to design control systems. The functional semantics of a Matlab/Simulink system is validated by a simulation, which imposes a certain dataflow. So, the implementation of a Matlab/Simulink system on a multi-core real-time platform must guarantee this functional dataflow. In particular, the data exchanges between applications running on different cores must conform to the synchronous model of Matlab/Simulink, even if cores are not synchronized. As such, an accurate tracking of the data exchanges is required. In addition, the transformation must account for real-time and non-functional constraints such as performance, multi/many-core constraints (*e.g.* parallelism and resources), reliability, *etc.*

In this chapter, we demonstrate that the dataflow in a multi-periodic Matlab/Simulink system can be formally modeled by a SDFG. We assume that the semantics of the system is correct and has been formally proven. Hence, we describe statically and accurately the functional dataflow by exploiting the periods and the data exchanges in the Matlab/Simulink simulation. Our transformation from Matlab/Simulink to SDFG uses mathematical equations in $\mathcal{O}(1)$. It does not need complex algorithmic and the resulting graph is equivalent in size to the Matlab/Simulink block diagram.

The rest of this chapter is organized as follows. Section 4.2 presents related work concerning synchronous multi-periodic models for multi-core, the modeling of multi-periodic systems and AUTOSAR multi-core applications. In Section 4.3, we first describe the architecture of Matlab/Simulink systems. Then, we perform an in-depth study of the simulation by exploring several configuration parameters and we characterize the main communication mechanisms. Section 4.4 is devoted to the modeling of multi-periodic Matlab/Simulink systems by SDFGs. Several static properties of the resulting SDFG are given in Section 4.5. Finally, Section 4.6 illustrates the transformation on an industrial use case of a *Fuel Cell Control System*. Section 4.7 concludes the chapter.

4.2 Related work

The Matlab/Simulink design environment belongs to synchronous programming models [12] with synchronous languages such as ESTEREL[15], LUSTRE [25] and SIGNAL [13]. Approaches of semantics preserving implementation of such models on both multi-core and distributed architectures have been discussed in Section 3.3. Thus, Caspi *et al.* [24] applied these approaches to the semantics-preserving implementation of synchronous models on *Time Triggered Architectures* (TTA) [70]. The latter are distributed architectures built upon a synchronous bus with a global clock. Tripakis *et al.* [108] have also proposed such technique in the less constrained *Loosely Time Triggered Architectures* (LTTA) [14]. LTTA is characterized by a communication mechanism where no global clock synchronization is required. Despite that TTA and LTTA are promising approaches, they do not address specifically multi-periodic systems and can lead to implementations that are not optimized

with respect to the multi-periodicity.

More recently, Forget *et al.* [45] have proposed an approach based on the multi-periodic synchronous dataflow language PRELUDE [85]. The latter allows to express finely multi-periodic aspects. Their work is interesting because it handles the process from PRELUDE specification to multi/many-core implementation [86]. However, the methodology of Pagetti *et al.* [86] does not formally specify the equivalence between communication mechanisms in Matlab/Simulink and those in PRELUDE.

Our approach allows to formalize a number of communication mechanisms with SDFG. The latter is used as a semantics-preserving abstraction that keeps the specificities of the system, while providing analysis tools for multi-periodic systems on multi-core. As such, it provides a formal method to move towards the aforementioned strategies for preserving-semantics implementation on multi/many-core.

Several studies have also investigated the implementation Matlab/Simulink systems on AUTOSAR multi-core. So, Zeng and Di Natale [117] suggested a number of implementation strategies for certain communication mechanisms of Matlab/Simulink in AUTOSAR multi-core. Their study demonstrates that it is possible to implement Matlab/Simulink mechanisms on AUTOSAR multi-core. However, they do not take into consideration communications modeling. The latter allows to perform high level analysis with respect to mapping, allocation, resources utilization and memory overhead. Having a high level model with strong mathematical background and software analysis tools such as SDFG provides additional optimization capabilities, which would not be possible if we were limited to the transcription of communication mechanisms.

Höttger *et al.* [60] proposed an abstract description of AUTOSAR systems. This abstraction considers only the communication links between the elements of the system. Then, they applied parallelization approaches for multi-core partitioning. However, their abstraction is somewhat far from the original specification of the system because it does not consider how communications are made. In addition, they do not rely on synchronous specification such as Matlab/Simulink. Our description is more interesting as it is closer to the initial specification and it provides a refinement of communication models. The latter is used to have better multi-core analysis methods.

Concerning the formalism, Richard *et al.* [94] have developed the principle of *generalized dependencies* to describe synchronous multi-periodic systems. Their equations are similar to those of SDFG. However, they have not established the formal equivalence between their model and SDFG. Moreover, their formalism fails to model all the communication mechanisms that we consider in Matlab/Simulink.

Tripakis *et al.* [108] made the link between synchronous model and *Marked Direct Graph* (MDG) [31]. MDG is a special case of SDFG also known as *Homogeneous Synchronous Dataflow* (HSDFG) [73]. A HSDFG is a SDFG with all production and consumption rates are equal to 1. Despite SDFG is known to be convertible into MDG for multi-periodic systems, Marchetti and Munier [80] have shown that this conversion is not in polynomial

size and suffers from performance issues. Thereby, MDG is not efficient to model large multi-periodic applications.

Guesmi and Hasnaoui [56] succeeded in linking Matlab/Simulink and SDFG. However, their link concerns only the structure and there is no detailed study of the synchronous semantics. Their abstraction does not account for the real communication mechanisms imposed by the functional semantics of Matlab/Simulink.

Authors in [28, 20, 40] have proposed translation strategies from Matlab/Simulink to SDFG by taking communications into account. Chessa [28] implemented a translation of single rate Matlab/Simulink models to HSDFGs. Authors in [20] and [40] proposed translation of multi-periodic Matlab/Simulink systems to SDFG. However, those works used only harmonic periods. Our translation consider multi-periodic systems with both harmonic and non-harmonic periods.

4.3 Matlab/Simulink functional specification

The Matlab/Simulink design environment is a graphical modeling editor, simulator and code generator. It is used to design, simulate, implement and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing. Matlab/Simulink has the ability to model multi-domain systems. As such, it enables the co-simulation of both functional and physical models to validate the behavior of the system. In this section, we consider the functional specification of multi-periodic systems in Matlab/Simulink.

In Subsection 4.3.1, we describe Matlab/Simulink synchronous block diagrams and we detail its temporal aspects. In Subsection 4.3.2, we perform an in-depth study of Matlab/Simulink simulation. This study comprises the definition of several simulation parameters, the description of simulation processes and steps, and the characterization of the main communication mechanisms.

4.3.1 Synchronous block diagrams

A Matlab/Simulink system is described by hierarchical *Synchronous Block Diagrams* (SBD). The latter is composed of a set of communicating functional blocks connected by signals. Each block has dedicated input and output ports through which it communicates with others. A block in this SBD is either a basic block from Matlab/Simulink library (*e.g.* add, a filter, unit delay, s-function) or a grouping of several blocks into a *subsystem* block. The internal structure of a subsystem block is also described as a SBD. In fact, subsystems provide a graphical and hierarchical organization for the system. Two categories of subsystems exist: *virtual* and *atomic*. Virtual subsystems are ignored (*i.e.* flattened) by Matlab/Simulink during simulation, whereas atomic subsystems are evaluated in block

(*i.e.* as a single unit).

Figure 4.1 illustrates a Matlab/Simulink model of a conditional Multiply-add operation. This model has three input ports (In1, In2 and In3) and one output port (Out). The top hierarchy SBD is composed of an atomic subsystem (Multiply-add), a virtual subsystem (If-else) and an unit delay (Delay). The latter connects the output of the Multiply-add block with one input of the If-else block. The Multiply-add block realizes an unconditional multiply-by-4-and-add operation. Its internal structure is described under another hierarchy of SBD. This hierarchy is composed of two input ports (a and b), one output port (c), a 4-gain multiplier (Gain) and an add block (Add). The internal structure of the If-else block is also described under another hierarchy of SBD. The latter is composed of three input ports (u, x1 and x2), an If-condition block, two If-action blocks (If-action1 and If-action2) and a merge block (Merge). The If-condition block activates If-action1 or If-action2 depending on whether the input port u is greater than 0, respectively. Each If-action block transfers its input to the output and the merging of the two forms the output of the subsystem.

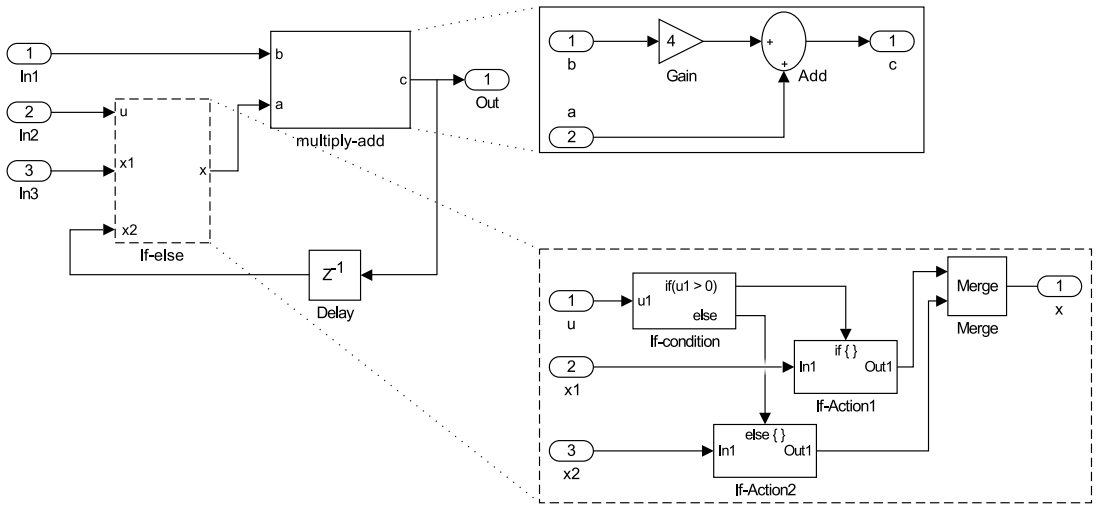


Figure 4.1: Simulink model of a conditional Multiply-add operation.

Matlab/Simulink defines a time-based SBD. In fact, signals are quantities that change over time and are defined for all points in time. The output values of some blocks are functions of previous values of temporal variables called *states*. Computing the outputs of these blocks therefore requires saving the states at each time. For example, the Delay block in Figure 4.1 has an internal state variable that captures the value of its input signal at a given point in time for computing its output signal at the next point in time. Figure 4.2 illustrates a graphic model of blocks with state variables, where x represents the set of state variables of the block.

Each block in the SBD describes a set of equations called *block methods*. These equations define the relationships between the input signals, the output signals and the state variables of the block. So, the execution of a block (in simulation) consists of evaluating its block methods and updating the values of its output signals and its state variables, respectively. For the example of Figure 4.1, the subsystem Multiply-add is characterized

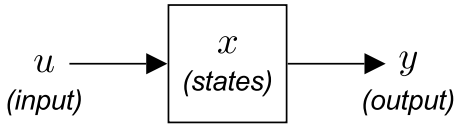


Figure 4.2: A graphical representation of a blocks with internal state variables.

only by an output equation or *output method* $c = 4b + a$. This subsystem does not have state variables. Hence, its output signal c depends directly on its inputs signals a and b. In the same example, the Delay block is characterized by a state variable $x_{Delay} = Input$ and an output method $Output = x_{Delay}$. Note that the output method is always evaluated before the state variable is updated. As a result, the output signal of the Delay block depends only on its state variable x_{Delay}, which is the previous value of its input signal.

In Matlab/Simulink, the relationship between input and output ports defines two types of functional blocks:

- Direct feedthrough blocks: direct feedthrough defines a block that has one or more output ports that depend directly on one or more input ports. Furthermore, independently of the internal structure, an atomic subsystem is direct feedthrough. In Figure 4.1, all blocks except the unit Delay are direct feedthrough.

- Non-direct feedthrough blocks: non-direct feedthrough defines a block whose output ports depend only on the state variables. In Figure 4.1, the Delay block is an example of a non-direct feedthrough block.

The distinction between direct feedthrough and non-direct feedthrough blocks is a key characteristic of the simulation of Matlab/Simulink.

4.3.2 Simulation: ordered time-based block evaluation

The simulation of a Matlab/Simulink block diagram consists in propagating signals over time. This is achieved by computing the output ports of all blocks in time. Thus, the result of the simulation is obtained by stepping the time at a rate defined by a *step size*. Namely, the logical time advances in simulation by adding one time step to the current time. Among the existing stepping techniques in Matlab/Simulink (*e.g.* variable steps and fixed step), the *discrete fixed-step* is the one used to design the functional specification of embedded applications. This stepping technique computes the time of the next step by adding a fixed step size to the current time.

At some point in the simulation, a block is executed and the new values of its output signals are propagated only when certain execution conditions are satisfied. If these conditions are not met, the block is not executed and its output signals hold their values. Matlab/Simulink defines three types of execution conditions and block:

- Unconditional blocks or subsystems: the block is assigned a *sampling time* that indicates when it is executed during the simulation. This sample time can be periodic (given by the designer) or continuous. Periodic blocks are executed during the simulation only at time steps that are multiple of their periods. Continuous blocks use Ordinary Differential Equations to determine dynamically their execution time during the simulation [21]. The Multiply-add subsystem in Figure 4.1 is an example of unconditionally executed blocks.

- Conditional subsystems: the subsystem is executed only when an external control signal enables or triggers its execution. Examples of conditional subsystems are *Enabled subsystems* (execute at a time step if the control signal is positive), *Triggered subsystems* (execute each time a trigger event occurs), *Function-call Subsystems* (execute each time a function-call event occurs).

- Logically-executed subsystems: the subsystem is executed one or more times at the current time step when it is enabled by a signal from a control block. A control block implements control logic such as if-action blocks.

In the rest of this study, we consider only discrete blocks. In fact, continuous blocks are mainly used to model physical environments, while embedded applications are modeled by discrete blocks. In particular, this is the case for Matlab/Simulink applications in AUTOSAR perspective [6]. In addition, we use the periodic assumption to characterize the simulation because its execution scenarios cover those of conditionally and logically executed subsystems.

Figure 4.3 shows a conceptual block diagram representing a Matlab/Simulink system containing four periodic blocks A, B, C and D, which have the sample times 80ms, 40ms, 30ms, and 50ms, respectively.

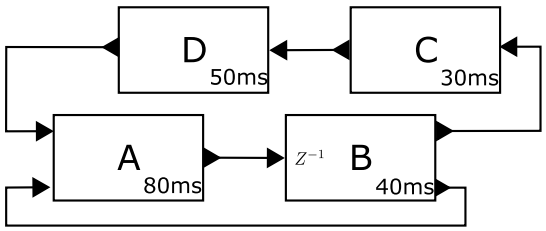


Figure 4.3: Example of Simulink block diagram with annotated sample times.

Before the simulation starts, the simulation engine determines the order in which to execute blocks at each time step. This block execution ordering is called the *sorted order*. The sorted order of unconditional blocks and subsystems is constructed by combining two rules: the first applies to direct feedthrough blocks and the second applies to non-direct feedthrough blocks. In fact, on the one hand, outputs of any direct feedthrough block depend directly on its input ports. Hence, any direct feedthrough block must be executed after the blocks driving its input ports. Otherwise, its outputs can not be calculated and its output signals cannot be propagated. As a result, the simulation of direct

feedthrough blocks is performed in the topological order. On the other hand, outputs of any non-direct feedthrough block do not depend directly on its input ports. As such, non-direct feedthrough blocks can be executed in any order as long as each precedes the direct feedthrough blocks that it drives. Then, the sorted order of conditional and logically-executed blocks and subsystems is derived from the sorted order of the unconditional blocks or subsystems that drive their execution conditions. Finally, at each time step, the simulation engine scans the sorted order and executes (in order) blocks that verify their execution conditions.

For cyclic dependencies, Matlab/Simulink requires at least one non-direct feedthrough block in any topological dependency cycle. In fact, non-direct feedthrough blocks break direct dependency cycles because they act as delay blocks (in terms of data transfer).

In Figure 4.3, let us assume that blocks A, C and D are direct feedthrough and block B is non-direct feedthrough. The relative order of blocks A, C and D follows their topological dependency. Since B is a non-direct feedthrough and its outputs do not depend directly on the current value of A (*i.e.* B has a delay semantics), B must be executed before A. The corresponding sorted order for this system is $B \rightarrow C \rightarrow D \rightarrow A$.

The execution of a block at each time step is divided into two ordered stages. The first stage is to evaluate the output methods and the second stage is to evaluate the state methods. Then, the dataflow semantics of the simulation is dictated by the order in which these stages are evaluated relatively to others. In fact, the simulation engine uses the concept of *simulation group* to define blocks sets that are executed together through each stage. Namely, output methods of blocks in the same group are evaluated together, followed by the evaluation of their state methods. Simulation groups are transparent for designers. However, Matlab/Simulink environment provides two simulation modes (*single-tasking* and *multitasking*) that define implicitly how groups are formed. As a result, the interpretation and the functional specification of a given SBD depend of simulation modes.

The single-tasking mode defines only one simulation group. That is, all blocks are processed together through each stage of the simulation. More precisely, at each time step, the output methods of all blocks that verify the execution conditions are first evaluated in the sorted order. Then, their state variables are updated in the same way. Figure 4.4 illustrates a single-tasking simulation of the system of Figure 4.3. In this illustration, at time step 0ms the output methods of all blocks are first computed together ($B \rightarrow C \rightarrow D \rightarrow A$) followed by the computation and the updating of their state variables in the same order. The same goes to simulation time step 30ms except that only block C is to be executed, and so on.

In the simulation, we define as *direct* communication the data exchange mechanism where a block can always use the data produced at the same time step by the block driving its input. For example, direct data exchange is illustrated in Figure 4.4 by block C which consumes at time step 0s the data produced at the same time step by block B. This is because the output of C depends directly on its input port which is fed by the output of block B and the latter is updated before the former.

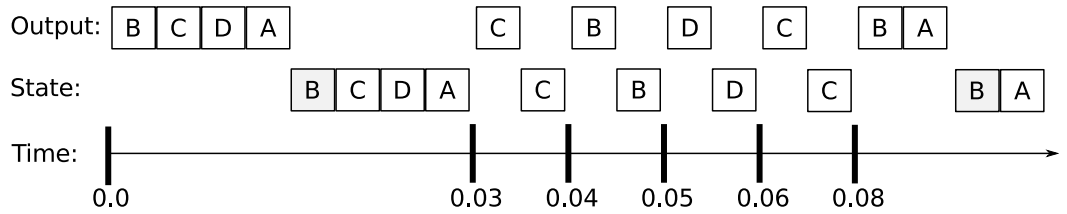


Figure 4.4: Example of single-tasking simulation.

As well, we define as *delayed* communication the data exchange mechanism where a block does not use the latest data produced by the block driving its input at each time step. For example, a delayed data exchange is illustrated in Figure 4.4 by block B which does not consume at time step 0.0s the data produced at the same time step (*i.e.* the latest) by block A. This is because the output of B depends only on its state variable, which is in an initial state. Furthermore, the communication from block B to block C in the simulation of Figure 4.4 illustrates a special case of delayed communication. In fact, the latter is between blocks at different periods. In that case, the updating of the input port (and thus the state variable) with the latest value of the output of block A is controlled by a *Rate Transition Block* (RTB). The latter delays this updating by one cycle of the producer block A. So, at steps 0s and 0.08s the inputs and the state variable of block B (gray background) are not updated with the latest output of block A. As such, the output of block B at time step 0.04s is computed with the same initial value (of the state variable) as at time step 0s. Then, its state variable is updated with the value of the output of block C (which was computed at time step 0s). This value is used to compute the output of block B at time step 0.08s.

Note that the illustration of Figure 4.4 is conceptual and actually the single-tasking mode of Matlab/Simulink allows harmonic periods only.

In the multitasking mode, blocks at the same sample time are associated to one simulation group. That is, at each time step each group is executed as in single-tasking. However, it is the simulation group with the highest rate that is executed first, followed by the group with the next highest rate, and so on. Figure 4.5 illustrates the multitasking simulation of the system of Figure 4.3. We distinguish four groups, which are from highest to lowest rates: $\{C\} \rightarrow \{B\} \rightarrow \{D\} \rightarrow \{A\}$. In the conceptual multitasking simulation in Figure 4.5 we can notice that at each time step, the simulation stages of each group are processed separately from others.

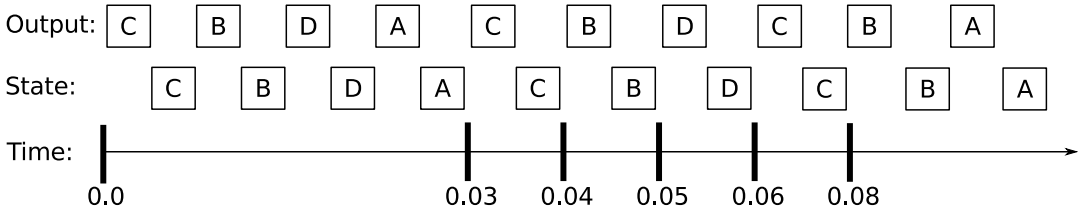


Figure 4.5: Example of multitasking simulation.

Besides direct and delayed communication mechanisms we define the *hybrid* commu-

nication mechanism in multitasking mode. This mechanism combines both direct and delayed communications. The hybrid communication mechanism is specific for data exchanges between blocks belonging to different groups. In fact, the hybrid communication is used when the data transfer is from a slow block to a fast block, but delayed communication is not required or not possible. In such configuration, when blocks are evaluated at the same time step the delayed mechanism is used. This is because the slow block driving the input is executed after the fast one. However, when the blocks are not evaluated at the same time step the direct mechanism is used because the fast block takes the latest output produced by the slow one.

The data transfer from block B to block C in Figure 4.5 is an example of hybrid communication. Block C is faster than block B. When blocks B and C are fired at the same time step 0ms, C is executed before B and the communication is delayed. At time step 0.06s the execution of block C uses direct communication because it takes the output of the previous execution of block B (at time step 0.05s).

Communication mechanisms between blocks at different rates are always ensured by a RTB (both in single-tasking and in multitasking). The latter are special blocks for which the input methods and the output methods can be evaluated at different rates to realize the aforementioned communication mechanisms [99]. The choice of the RTB for communication between blocks of different periods can be done manually by the designer or automatically by Matlab/Simulink. In the latter case, the configuration parameter "*Automatically handle rate transition for data transfer*" must be checked. Then, the designer can ask Matlab/Simulink to use deterministic RTB either always, whenever possible or never. The term deterministic is used in Matlab/Simulink to refer to configurations where the tool can generate codes realizing the data exchanges deterministically.

There exists five main types of RTB [99, 21]:

- ZOH: the Zero-Order-Hold is a deterministic RTB that implements direct communications from fast to slow blocks at harmonic periods.
- 1/Z: the 1/Z is a deterministic RTB that implements delayed communications from slow to fast block at harmonic periods.
- Buf: this RTB block implements direct communications from fast to slow blocks at arbitrary periods.
- Db-buf: this RTB implements hybrid communications from slow to fast block at arbitrary periods.
- NoOp: this RTB implements direct communications from fast to slow blocks at arbitrary periods.

We can notice that the communication mechanism between pair of producer/consumer blocks in a discrete Matlab/Simulink model can be specified statically without simulating.

In fact, the dataflow can be extracted for the Matlab/Simulink model by relying on simulation parameters (*e.g.* simulation mode, RTB), blocks types (virtual, feedthrough) and sample times.

4.3.3 The AUTOSAR perspective

The AUTOSAR perspective is a Matlab/Simulink blocks diagram organized as an AUTOSAR application software. That is, the layers are composed of sets of SWCs that interact through well defined ports. The specification of SWCs and Runnables in such models are defined by the designers. However, the design suite Embedded Coder provides the following rules to let engineers map Matlab/Simulink models to AUTOSAR component descriptions [6]:

- SWCs are designed as virtual subsystems
- Runnables are designed as function-call subsystems with periodic function calls

The AUTOSAR perspective allows to generate semantics-preserving embedded code for Runnables automatically. However, the implementation of data exchanges and the communication mechanisms between Runnables is left at the charge of the AUTOSAR configuration.

4.4 Modeling dataflow in Simulink by SDFG

In this section, we investigate the modeling of the dataflow in a discrete multi-periodic Matlab/Simulink system by a SDFG.

Let us assume that each periodic block i in Matlab/Simulink is modeled by a SDFG task τ_i. The latter is characterized by a period T_i which corresponds to the sample time of block i. According to the simulation of i, the logical start date (or time step) of the n_ith execution of τ_i is $(n_i - 1) \cdot T_i$. For each producer/consumer pair (τ_i, τ_j), we denote by $gcd_{i,j}$ the greatest common divisor of T_i and T_j.

In Subsections 4.4.1, 4.4.2 and 4.4.3, we model direct, delayed and hybrid communication mechanisms between pair of producer/consumer (τ_i, τ_j) by a SDFG buffer $a = (\tau_i, \tau_j)$, respectively. We also prove the characteristics of these buffers using the periods of blocks. Subsection 4.4.4 provides the process to model a multi-periodic Matlab/Simulink system by a SDFG.

4.4.1 Modeling direct communication

In direct communication, producers are always executed before consumers. The transfer of data is therefore performed with the producer execution that starts at the same time or just before the execution of the consumer.

Figure 4.6 illustrates the data dependency model of direct communication between executions of a producer τ_i and a consumer τ_j. Arrow 1 indicates a data dependency between $\tau_i[n_i - 1]$ and $\tau_j[n_j - 1]$, where producer and consumer start at the same date. Arrows 2 and 3 exhibit data dependencies between $\tau_i[n_i]$ and $\tau_j[n_j]$ and between $\tau_i[n_i + 2]$ and $\tau_j[n_j + 1]$, respectively. For these dependencies the transferred data are the latest produced by the execution of τ_i.

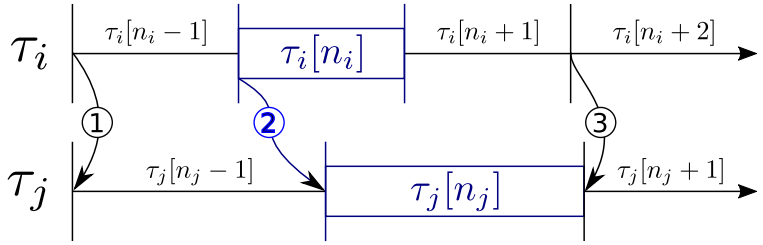


Figure 4.6: Principle of data transfer in direct communication.

The data dependency constraints imposed by direct communication mechanisms depend on the logical start and end dates of executions. These constraints are characterized by Definition 4.1.

Definition 4.1
Let us consider the n_ith execution of a producer τ_i and the n_jth execution of a consumer τ_j. A data dependency exists between $\tau_i[n_i]$ and $\tau_j[n_j]$ for direct communication iff*:*

Condition 1: $\tau_j[n_j]$ starts at the same date or after the start date of $\tau_i[n_i]$;

Condition 2: $\tau_j[n_j - 1]$ starts strictly before the start date of $\tau_i[n_i]$;

Condition 3: $\tau_j[n_j]$ starts strictly before the start date of $\tau_i[n_i + 1]$.

In Figure 4.7, we use SDFG tasks τ_C and τ_D to illustrate the data dependencies of direct communication from block C to D (Figure 4.3). All data dependencies in this illustration verify the conditions of Definition 4.1. In particular, the dependency $\tau_C[1]$ to $\tau_D[1]$ verifies that executions start at the same date. Constraint $\tau_C[4]$ to $\tau_D[3]$ verifies that $\tau_D[3]$ starts after $\tau_C[4]$ but before $\tau_C[5]$ (conditions 1 and 3) and $\tau_D[2]$ starts before $\tau_C[4]$ (condition 2).

We deduce the following lemma from Definition 4.1.

Lemma 4.1
Let (τ_i, τ_j) be a producer/consumer pair with respective periods T_i and T_j. A data dependency constraint exists between the n_ith execution of τ_i and the n_jth execution of τ_j for

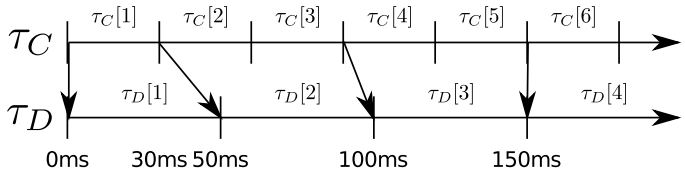


Figure 4.7: Direct communication model from 30ms periodic SDFG task τ_C to 50ms periodic SDFG task τ_D.

direct communication iff*:*

$$T_i > (T_j - gcd_{i,j}) + n_i \cdot T_i - n_j \cdot T_j \geq \max(T_i - T_j, 0) \tag{4.1}$$

Proof. The conditions of Definition 4.1 can be expressed as follows:

1. Condition 1 is equivalent to:
   $(n_i - 1) \cdot T_i \leq (n_j - 1) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j \leq T_i - T_j$
2. Condition 2 is equivalent to:
   $(n_i - 1) \cdot T_i > (n_j - 2) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j > T_i - 2 \cdot T_j$
3. Condition 3 is equivalent to:
   $n_i \cdot T_i > (n_j - 1) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j > -T_j$

Adding T_j to the combination of the three equations yields to:

$$T_i \geq T_j + n_i \cdot T_i - n_j \cdot T_j > \max(T_i - T_j, 0) \tag{4.2}$$

$gcd_{i,j}$ divides the terms of Equation (4.2). Hence, $gcd_{i,j}$ can be subtracted to the middle term of Equation (4.2) so that we obtain Equation (4.1). □

We express the SDFG model of direct communication mechanisms in Theorem 4.1.

Theorem 4.1
Direct communication between a producer/consumer pair (τ_i, τ_j) is modeled by a buffer $a = (\tau_i, \tau_j)$, where production and consumption rates are $in_a = T_i$ and $out_a = T_j$, respectively. The initial marking is $M_0(a) = T_j - gcd_a$.

Proof. The equivalence between data dependency constraints of direct communication in Equation (4.1) and precedence constraints of SDFG in Equation (3.2) proves the theorem. □

Data exchanges from τ_C to τ_D use direct communication. According to Theorem 4.1, it is modeled by the SDFG buffer of Figure 4.8. In that case $in_a = T_C = 30$, $out_a = T_D = 50$, $gcd_a = 10$ and $M_0(a) = 50 - 10 = 40$.

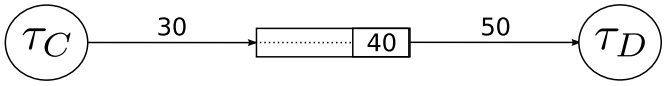


Figure 4.8: SDFG model of direct communication from 30ms periodic SDFG task τ_C to 50ms periodic SDFG task τ_D.

4.4.2 Modeling delayed communication

In delayed communication, the consumer is always executed before the producer at each time step in simulation. In addition, the data transfer is delayed by one period of the producer and it occurs only after the start date of the next execution of the latter. As such, the data dependency constraints imposed by the delayed communication depend on the logical start dates of executions. These constraints are illustrated in Figure 4.9 and are characterized by Definition 4.2.

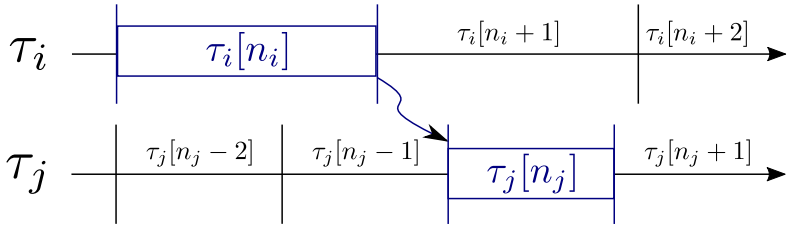


Figure 4.9: Principle of data transfer in delayed communication.

Definition 4.2
Let us consider the n_ith execution of a producer τ_i and the n_jth execution of a consumer τ_j. A data dependency exists between $\tau_i[n_i]$ and $\tau_j[n_j]$ for delayed communication iff*:*

Condition 4: $\tau_j[n_j]$ starts at the same date or after the start date of $\tau_i[n_i+1]$;

Condition 5: $\tau_j[n_j-1]$ starts strictly before the start date of $\tau_i[n_i+1]$;

Condition 6: $\tau_j[n_j]$ starts strictly before the start date of $\tau_i[n_i+2]$.

In Figure 4.10, we use SDFG tasks τ_A and τ_B to illustrate the data dependencies of delayed communication from block A to B (Figure 4.3). All the dependencies in this illustration verify the conditions of Definition 4.2. For example, let us consider the dependence $\tau_A[1]$ to $\tau_B[3]$. $\tau_B[3]$ starts concurrently with the end of the cycle of $\tau_A[1]$ at 80ms, but before the end of the cycle of $\tau_A[2]$ at 160ms; conditions 4 and 6 are checked. $\tau_B[2]$ starts at 40ms. That is, it starts before the end of the cycle of $\tau_A[1]$; Condition 5 is checked. In delayed communication, although $\tau_B[2]$ starts after $\tau_A[1]$ there is no dependence from $\tau_A[1]$ to $\tau_B[2]$ because $\tau_B[2]$ starts before the end of the cycle of $\tau_A[1]$.

We deduce the following lemma from Definition 4.2.

Lemma 4.2
Let (τ_i, τ_j) be a producer/consumer pair with respective periods T_i and T_j. A data depen-

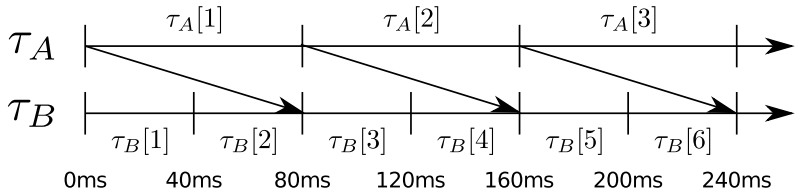


Figure 4.10: Delayed communication model from 80ms periodic SDFG task τ_A to 40ms periodic SDFG task τ_B.

dency constraint exists between the n_ith execution of τ_i and the n_jth execution of τ_j for delayed communication iff:

$$T_i > T_i + T_j - gcd_{i,j} + n_i \cdot T_i - n_j \cdot T_j \geq \max(T_i - T_j, 0) \tag{4.3}$$

Proof. The conditions of Definition 4.2 can be expressed as follows:

1. Condition 4 is equivalent to:
   $n_i \cdot T_i \leq (n_j - 1) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j \leq -T_j$
2. Condition 5 is equivalent to:
   $n_i \cdot T_i > (n_j - 2) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j > -2 \cdot T_j$
3. Condition 6 is equivalent to:
   $(n_i + 1) \cdot T_i > (n_j - 1) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j > -T_i - T_j$

Adding $T_i + T_j$ to the combination of the three equations yields to:

$$T_i \geq T_i + T_j + n_i \cdot T_i - n_j \cdot T_j > \max(T_i - T_j, 0) \tag{4.4}$$

$gcd_{i,j}$ divides the terms of Equation (4.4). Then, Equation (4.3) is derived by subtracting $gcd_{i,j}$ to the middle term of Equation (4.4). □

We express the SDFG model of delayed communication in Theorem 4.2.

Theorem 4.2
Delayed communication between a producer/consumer pair (τ_i, τ_j) is modeled by a buffer $a = (\tau_i, \tau_j)$, where production and consumption rates are $in_a = T_i$ and $out_a = T_j$, respectively. The initial marking is $M_0(a) = T_i + T_j - gcd_a$.

Proof. The equivalence between data dependency constraints of delayed communication in Equation (4.3) and precedence constraints of SDFG in Equation (3.2) proves the theorem. □

Data exchanges from τ_A to τ_B use direct communication. According to Theorem 4.2, it is modeled by the SDFG buffer of Figure 4.11. In that case $in_a = T_A = 80$, $out_a = T_B = 40$, $gcd_a = 40$ and $M_0(a) = 80 + 40 - 40 = 80$.

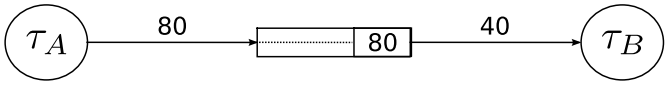


Figure 4.11: SDFG model of delayed communication from 80ms periodic SDFG task τ_A to 40ms periodic SDFG task τ_B.

4.4.3 Modeling hybrid communication

Hybrid communication is related to the mechanism where the consumer is executed before the producer at a given time step. In addition, the data transfer is not delayed by one period of the producer but can occur directly at the end of the execution of the latter. As such, the data used in this mechanism are those produced by the execution of the producer that starts strictly before the execution of the consumer. However, there is no data transfer when the executions of producer and consumer have the same start date because the latter is executed before the former. The data dependency constraints imposed by hybrid communication are illustrated in Figure 4.12 and are characterized by Definition 4.3.

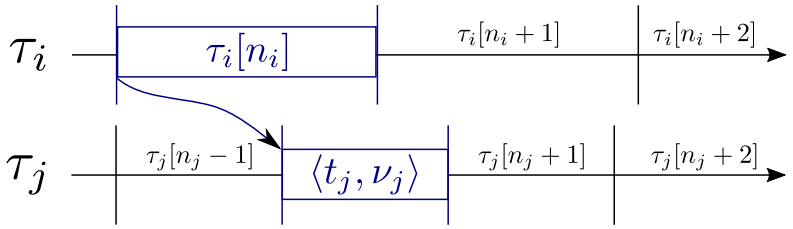


Figure 4.12: Principle of data transfer in hybrid communication.

Definition 4.3
Let us consider the n_ith execution of a producer τ_i and the n_jth execution of a consumer τ_j. A data dependency exists between $\tau_i[n_i]$ and $\tau_j[n_j]$ for hybrid communication iff*:*

Condition 7: $\tau_j[n_j]$ starts strictly after the start date of $\tau_i[n_i]$;

Condition 8: $\tau_j[n_j - 1]$ starts before or at the same start date as $\tau_i[n_i]$;

Condition 9: $\tau_j[n_j]$ starts before or at the same start date as $\tau_i[n_i + 1]$.

In Figure 4.13, we use SDFG tasks τ_B and τ_C to illustrate the data dependencies of hybrid communication from block B to C (Figure 4.3). All the dependencies in this illustration verify the conditions of Definition 4.3. That is, the executions of τ_B starts strictly before those of τ_C. For example, the dependency $\tau_B[1]$ to $\tau_C[2]$ verifies that $\tau_C[2]$ starts strictly after $\tau_B[1]$ and before $\tau_B[2]$ (conditions 7 and 9). And $\tau_C[2]$ starts concurrently with $\tau_B[1]$ (condition 8).

Lemma 4.3
Let (τ_i, τ_j) be a producer/consumer pair with respective periods T_i and T_j. A data dependency constraint exists between the n_ith execution of τ_i and the n_jth execution of τ_j for

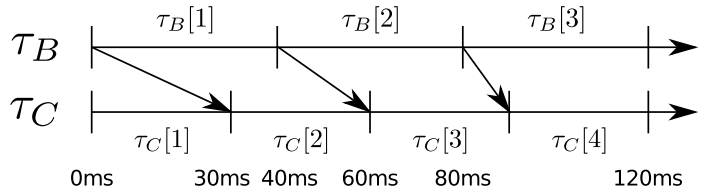


Figure 4.13: Hybrid communication model from 40ms periodic SDFG task τ_B to 30ms periodic SDFG task τ_C.

hybrid communication iff*:*

$$T_i > T_j + n_i \cdot T_i - n_j \cdot T_j \geq \max(T_i - T_j, 0) \tag{4.5}$$

Proof. The conditions of Definition 4.3 can be expressed as follows:

1. Condition 7 is equivalent to:
   $(n_i - 1) \cdot T_i < (n_j - 1) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j < T_i - T_j$

2. Condition 8 is equivalent to:
   $(n_i - 1) \cdot T_i \geq (n_j - 2) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j \geq T_i - 2 \cdot T_j$

3. Condition 9 is equivalent to:
   $n_i \cdot T_i \geq (n_j - 1) \cdot T_j \Longleftrightarrow n_i \cdot T_i - n_j \cdot T_j \geq -T_j$

We get the required Equation (4.5) by adding T_j to the combination of the aforementioned three equations. □

The SDFG model of hybrid communication is given by Theorem 4.3.

Theorem 4.3
Hybrid communication between a producer/consumer pair (τ_i, τ_j) is modeled by a buffer $a = (\tau_i, \tau_j)$, where production and consumption rates are $in_a = T_i$ and $out_a = T_j$, respectively. The initial marking is $M_0(a) = T_j$.

Proof. The equivalence between data dependency constraints of hybrid communication in Equation (4.5) and precedence constraints of SDFG in Equation (3.2) proves the theorem. □

Data exchanges from τ_B to τ_C use hybrid communication. According to Theorem 4.3, it is modeled by the SDFG buffer of Figure 4.14. In that case $in_a = T_B = 40$, $out_a = T_C = 30$ and $M_0(a) = 30$.

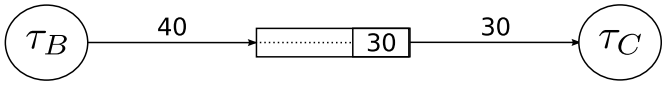


Figure 4.14: SDFG model of hybrid communication from 40ms periodic SDFG task τ_B to 30ms periodic SDFG task τ_C.

4.4.4 Translation process

The SDFG formalism requires atomicity properties and a static behavior for SDFG tasks [73]. As such, all Matlab/Simulink blocks that verify these conditions can be modeled by SDFG tasks. This requires the definition of both the principle of atomicity and static behavior for SDFG tasks that model Matlab/Simulink blocks.

Definition 4.4 (Atomicity)
An atomic *SDFG task is a Matlab/Simulink basic block or an upper level atomic subsystem.*

In Definition 4.4, an upper level atomic subsystem is an atomic subsystem that is not included in the hierarchy of another atomic subsystem. The assumption of considering only upper level atomic subsystems is justified by the fact that atomic subsystems are usually generated as embedded functions. As such, the upper level atomic subsystem is the main function that call the subroutines in its internal structure.

Definition 4.5 (Static behavior)
A static *SDFG task is a Matlab/Simulink basic block or atomic subsystem that has a uniform sequence of execution during simulation.*

In Definition 4.4, a uniform sequence of execution means that the block is executed at fixed time intervals during simulation. In Matlab/Simulink, this can be achieved by unconditional blocks or subsystems with periodic sample times, triggered subsystems with periodic triggers and function-call subsystems with periodic function calls.

We use Definition 4.6 to construct the structure of the SDFG of a Matlab/Simulink multi-periodic system.

Definition 4.6 (Construction of the structure)
Given a Matlab/Simulink block diagram, all basic blocks or upper level atomic subsystems with periodic executions are modeled by SDFG tasks, except for data transfer blocks that are modeled by SDFG buffers.

Definition 4.6 is derived from Definitions 4.4 and 4.5. However, we choose not to model data transfer bocks such as unit delay blocks and RTB. In fact, these blocks only perform data transfers using appropriate mechanisms. For this reason, we chose to model their behaviors by SDFG buffers. Definition 4.6 provides the structure of the SDFG of the system. Then, the characteristics of the SDFG buffers are provided by Theorems 4.1, 4.2 and 4.3.

We assume that all blocks in the Matlab/Simulink system of Figure 4.3 are either basic blocks or upper level atomic subsystems. By applying Definition 4.6 and Theorems 4.1, 4.2 and 4.3 we get the SDFG model of Figure 4.15.

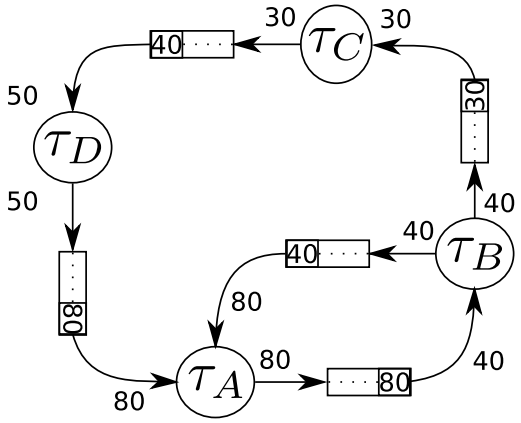


Figure 4.15: Structure of the SDFG of the Matlab/Simulink system of Figure 4.3.

We also apply Definition 4.6 to the system of Figure 4.1. We model the upper level atomic subsystem Multiply-add by a SDFG task. We go into the internal structure of the If-else virtual subsystem since it does not verify the atomicity property. In this internal structure, only the If-condition block is modeled by a SDFG task. The two If-action blocks and the Merge block cannot be modeled by SDFG tasks. This is because they are logically-executed (*i.e.* the static behavior is not verified). However, we cannot go into their internal structures (they are atomic blocks). Consequently, these blocks and their internal hierarchies cannot be modeled by SDFG tasks.

We derive Property 4.1 from Definition 4.6 to characterize Matlab/Simulink block diagrams for which all blocks are modeled by SDFG tasks.

Property 4.1
A Matlab/Simulink block diagram for which all blocks are modeled by SDFG tasks has all its upper level atomic subsystems and blocks that have periodic executions.

Property 4.1 is not verified by the system of Figure 4.1. In fact, the If-action blocks are upper level atomic blocks that do not have periodic executions. However, the model can be modified in order to verify the property. This is done by simply configuring the If-else block as an atomic subsystem. In doing so, the latter becomes an unconditional atomic subsystem at the same sample rate as the If-condition block in its internal structure. Consequently, both the Multiply-add and the If-else block are now modeled by SDFG tasks. The communication from the If-else to the Multiply-add is modeled by a SDFG buffer with direct communication. The communication from the Multiply-add to the If-else block through the unit Delay is modeled by a SDFG buffer with delayed communication. Communication with the system environment is not modeled. Then, we get the SDFG of Figure 4.16 by assuming that these blocks have the same sample time at 10ms. τ_{MA} and τ_{IF} are the SDFG tasks associated to subsystems Multiply-add and If-else, respectively.

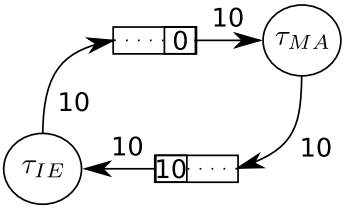


Figure 4.16: SDFG of the system of Figure 4.1 according to Definition 4.6.

4.5 Static properties

In the SDFG of Theorems 4.1, 4.2 and 4.3 tokens are not directly associated to data. They rather represent the time periods at which data are produced and consumed. In fact, for every buffer $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$, the production rate $in_a = T_i$ and the consumption rate $out_a = T_j$ define the intervals of production and consumption of data according to the periods of tasks, respectively. In this section, we show that the SDFG allows to identify statically and precisely the correct semantics of the data involved in the communication between pair of producer/consumer in the Matlab/Simulink simulation. This result provides efficient analysis and implementation strategies for Matlab/Simulink systems.

A data dependency constraint indicates that the execution of the consumer reads the data produced by the execution of the producer associated to that constraint. When an execution of a producer is not associated with any dependency constraint, the result of that execution is not used in communication. Likewise, when an execution of a consumer is not associated with any dependency constraint, that execution requires the same data as the previous one. Consequently, there is no need for communication.

For every producer/consumer pair (τ_i, τ_j) let us denote by $k = \frac{T_j}{T_i} = \frac{out_a}{in_a}$ the periods factor. Actually, we can notice that when the producer is faster than the consumer, data produced by some executions of the producer are not related to any data dependency constraints. However, all executions of the consumer are concerned. Theorem 4.4 provides the data dependency conditions for $k > 1$.

Theorem 4.4
Let us consider a buffer $a = (\tau_i, \tau_j)$. Let $S(a)$ be a set of pairs of executions $(n_i, n_j) \in \mathbb{N}\backslash\{0\} \times \mathbb{N}\backslash\{0\}$ such that there exists a data precedence constraint from $\tau_i[n_i]$ to $\tau_j[n_j]$. If $in_a < out_a$ then $S(a) = \{(f(n_j), n_j), n_j \in \mathbb{N}\backslash\{0\}\}$, where:

$$\begin{array}{rccl} f & : & \mathbb{N}\backslash\{0\} & \rightarrow & \mathbb{N}\backslash\{0\} \\ & & x & \mapsto & f(x) = \lceil \frac{out_a \cdot x - M_0(a)}{in_a} \rceil \end{array} \tag{4.6}$$

Proof. If $in_a < out_a$ then $\max(in_a - out_a, 0) = 0$. Equation (3.2) is equivalent to $in_a + out_a \cdot n_j - M_0(a) > in_a \cdot n_i \geq out_a \cdot n_j - M_0(a)$. This equation divided by in_a becomes $1 + \frac{out_a \cdot n_j - M_0(a)}{in_a} > n_i \geq \frac{out_a \cdot n_j - M_0(a)}{in_a}$. Since $n_i \in \mathbb{N} \setminus \{0\}$, we get $n_i = f(n_j) = \lceil \frac{out_a \cdot n_j - M_0(a)}{in_a} \rceil$. □

In the example of Figure 4.8, $in_a = 30$, $out_a = 50$, $M_0(a) = 40$ and $k = \frac{50}{30} > 1$. Thus, all executions of τ_D are involved in the communication but not all of τ_C. When applying Theorem 4.4 to the first four executions of τ_D we get the pairs of executions $(1,1),(2,2),(4,3)$ and $(6,4)$. These latter correspond to the dependency constraints illustrated in Figure 4.7.

Reciprocally, when τ_i is slower or equal to τ_j ($k \leq 1$), some executions of the consumer are not related to data dependency constraints, whereas all executions of the producer are concerned. Theorem 4.5 provides the data dependency conditions for $k \leq 1$.

Theorem 4.5
Let us consider a buffer $a = (\tau_i, \tau_j)$. Let $S(a)$ be a set of pairs of executions $(n_i, n_j) \in \mathbb{N}\backslash\{0\} \times \mathbb{N}\backslash\{0\}$ such that there exists a data precedence constraint from $\tau_i[n_i]$ to $\tau_j[n_j]$. If $in_a \geq out_a$ then $S(a) = \{(n_i, f(n_i)), n_i \in \mathbb{N}\backslash\{0\}\}$, where:

$$\begin{array}{rccl} f & : & \mathbb{N}\backslash\{0\} & \rightarrow & \mathbb{N}\backslash\{0\} \\ & & x & \mapsto & f(x) = \lfloor \frac{M_0(a)+in_a\cdot(x-1)}{out_a} \rfloor + 1 \end{array} \tag{4.7}$$

Proof. If $in_a \geq out_a$ then $\max(in_a - out_a, 0) = in_a - out_a$. Equation (3.2) is equivalent to $in_a - M_0(a) - in_a \cdot n_i > -out_a \cdot n_j \geq in_a - out_a - M_0(a) - in_a \cdot n_i$. This equation divided by $-out_a$ becomes $\frac{M_0(a)+in_a\cdot(n_i-1)}{out_a} < n_j \leq \frac{M_0(a)+in_a\cdot(n_i-1)}{out_a} + 1$. Assuming that $n_j \in \mathbb{N}\backslash\{0\}$, $n_j = f(n_i) = \lfloor \frac{M_0(a)+in_a\cdot(n_i-1)}{out_a} \rfloor + 1$. □

In the example of Figure 4.11 $in_a = 80$, $out_a = 40$, $M_0(a) = 80$ and $k = \frac{40}{80} < 1$. All executions of τ_A are involved in the communication but not all of τ_B. When applying Theorem 4.5 to the first three executions of τ_A, we get the pairs of executions $(1,3),(2,5),(3,7)$. These latter correspond to the dependency constraints illustrated in Figure 4.10.

When $M_0(a) = T_j - gcd_a$ (*i.e.* direct communication), Equations (4.6) and (4.7) express the *generalized precedence constraints* of Richard *et al.* [94]. That is, our approach extends their work and represents the system in a generic fashion. Namely, we allow modeling several multi-periodic communication mechanisms. In addition, Theorems 4.4 and 4.5 determine locally the relationships between pair of producer/consumer. They allow to isolate communications precisely without simulating the graph. This provides a better abstract understanding of communications compared to Matlab/Simulink. Furthermore, the SDFG expresses the data exchanges that must be implemented to preserve the functional semantics enforced by Matlab/Simulink.

Let us take each buffer $a = (\tau_i, \tau_j)$ separately. Let $\mathcal{N}_{i,j} = \text{lcm}(in_a, out_a)$ be the least common multiple of in_a and out_a. In a multi-periodic system, the data dependency constraints between pair of producer/consumer τ_i and τ_j is repetitive on $\mathcal{N}_{i,j}$. The number of executions of τ_i (*resp.* τ_j) on $\mathcal{N}_{i,j}$ is equal to $\frac{\mathcal{N}_{i,j}}{in_a}$ (*resp.* $\frac{\mathcal{N}_{i,j}}{out_a}$). Theorem 4.6 indicates the number of read and write operations involved in communications during the interval $\mathcal{N}_{i,j}$.

Theorem 4.6
Let $a = (\tau_i, \tau_j)$ be a buffer modeling a communication from τ_i to τ_j. The number of read and write operations involved in the communication during the interval $\mathcal{N}_{i,j}$ is equal to $2 \times \frac{\mathcal{N}_{i,j}}{\max(in_a, out_a)}$.

Proof. When $in_a < out_a$, Theorem 4.4 states that there are exactly $\frac{\mathcal{N}_{i,j}}{out_a}$ read operations associated to the same number of write operations during the interval $\mathcal{N}_{i,j}$ (*i.e.* all consumer executions are involved in communication). Reciprocally when $in_a \geq out_a$, Theorem 4.5 states that there are exactly $\frac{\mathcal{N}_{i,j}}{in_a}$ write operations associated to the same number of read operations during the interval $\mathcal{N}_{i,j}$ (*i.e.* all producer instances are involved in communication). The combination of the two cases proves the theorem. □

Theorem 4.6 provides better data stream characterization for the buffers of the SDFG. This result can be used by approaches where applications are modeled by *Dataflow Process Network* (DPN). In fact, communications are represented by weighted arcs in DPN formalism. The weights characterize the data streams and are used to study the mapping and routing of applications on many-core platforms [104].

Based on the characteristics highlighted by Theorem 4.6, communications analysis and implementation policies can be optimized by extracting only those necessary for the operation of the system. As a comparison, if we consider all communications between a pair of producer/consumer τ_i and τ_j during each interval $\mathcal{N}_{i,j}$, we will perform $\frac{\mathcal{N}_{i,j}}{in_a} + \frac{\mathcal{N}_{i,j}}{out_a}$ operations. However, only $2 \times \frac{\mathcal{N}_{i,j}}{\max(in_a, out_a)}$ are necessary to operate the system. Implementing only the read and write operations that are necessary for the respect of the functional semantics of Matlab/Simulink can reduce the possible impact of unnecessary data transmission, especially on shared memory or NoC.

In Figure 4.14, τ_B and τ_C have different periods and τ_B is slower that τ_C. Buffer $a = (\tau_B, \tau_C)$ is characterized by a production rate $in_a = 40$, a consumption rate $out_a = 30$ and $\mathcal{N}_{B,C} = \text{lcm}(40, 30) = 120$. There are exactly $\frac{120}{40} = 3$ executions of τ_B and $\frac{120}{30} = 4$ executions of τ_C during the interval $\mathcal{N}_{B,C}$. If we consider that each execution of τ_B performs a write operation and each execution of τ_C performs a read operation, 7 read/write operations will happen in total. However, with Theorem 9.3 we know that only $2 \times \frac{120}{\max(40,30)} = 6$ read/write operations are necessary to operate the system. That is, τ_C should not perform one read operation, otherwise that read could affect the semantics.

4.6 SDFG modeling of a Fuel Cell Control System

In this section, we present a case study of a *Fuel Cell Control System* (FCCS) that we model by SDFG. This case study is a multi-periodic Matlab/Simulink system at AUTOSAR perspective. It is designed according to Property 4.1 and the simulation is configured as discrete fixed-step in single-tasking mode.

First, Subsection 4.6.1 describes the fuel cell system and presents the main blocks of the FCCS. Then, in Subsection 4.6.2, we perform the translation of FCCS to SDFG by applying Definition 4.6 and Theorems 4.1, 4.2 and 4.3.

4.6.1 Description of a *Fuel Cell Control System*

A fuel cell is a system used to power the motor in certain range of electric cars. It is a device that directly transforms the chemical energy of a fuel into electrical one by electrochemical reactions. The simplest fuel cell system allows to provide water and electricity from hydrogen and oxygen.

A cell is composed of an anode charged with hydrogen and a cathode charged with oxygen arising from the air. The anode and the cathode are separated by an electrolyte which prevents the passage of electrons. The oxidation reaction of hydrogen and the air reduction reaction create an electrons flow from the anode to the cathode through an external circuit. This flow produces the electrical energy that powers the engine. Figure 4.17 illustrates the operating principle of a hydrogen-air fuel cell.

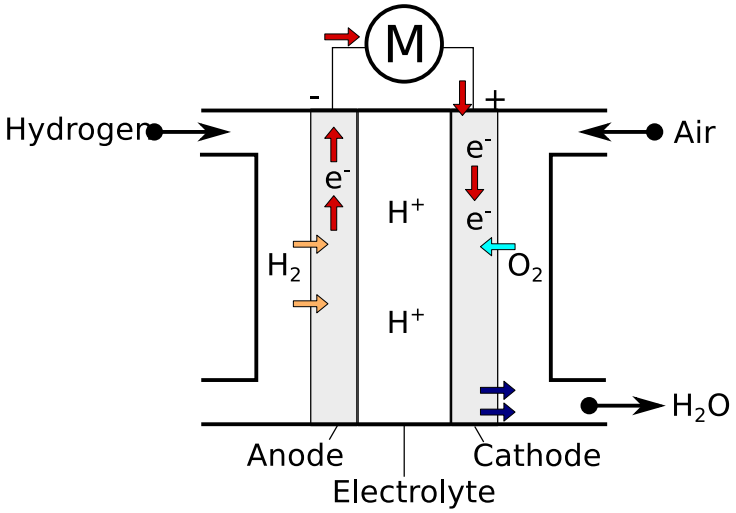


Figure 4.17: Operating principle of a fuel cell.

The *Fuel Cell Control System* (FCCS) is the management system of a fuel cell device composed of a large number of cells connected in series and parallel. Series connections allow to obtain a required tension and parallel connections allows to obtain a necessary power. Each cell works as long as it is supplied with a reagent at a certain temperature. The FCCS of our case study (Figure 4.18) is a control and command system designed in Matlab/Simulink to meet these requirements.

The FCCS in Figure 4.18 is decomposed into seven periodic blocks that control the physical system of the fuel cell in Figure 4.19. The block *Air* commands an air compressor and an air valve to regulate the flow rate and the air pressure in cells. The block $H2$ regulates the flow rate and the pressure of the hydrogen circuit. This is achieved by

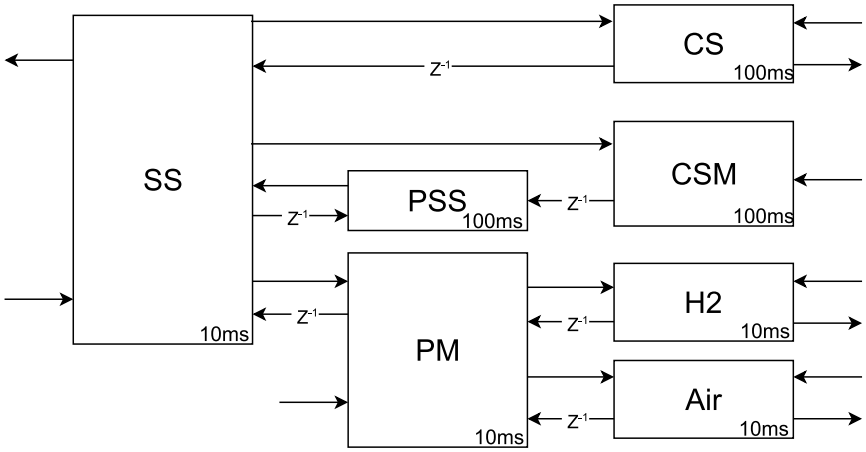


Figure 4.18: Fuel Cell Control System block diagram in Simulink.

controlling a $H2$ recirculation pump and a $H2$ valve allowing the oxygen to exit the tank. The block Cooling System (CS) is responsible of the nominal operating temperature of cells. For this purpose, CS controls a fan, a cooling pump and valve that allow the coolant to flow. The block PM is a power module management block. It computes the flow commands of $H2$ and Air circuits according to the operating modes of the vehicle (start, stop, and running) and the required levels of voltage and power. Blocks Cell Status Monitoring (CSM) and *Pre-System Supervisor* (PSS) supervise the status of cells in order to ensure a nominal voltage and detect failures. The System Supervisor (SS) coordinates the entire system according to the state of the vehicle (start, speed, acceleration) and feedback from other blocks. SS in turn gives feedback on the dashboard.

Blocks CSM, PSS and CS operate at a lower period 100ms compared to the period 10ms of other blocks. This difference is justified by the fact that temperature variates slowly compared to pressure and flow rate. However, despite the electrical dynamic (voltages) of cells is very fast, 100ms is sufficient for supervision.

4.6.2 SDFG modeling of the Fuel Cell Control System

We apply the methodology of transforming a Matlab/Simulink complete system to a SDFG on the industrial use case FCCS. The top level hierarchy of the FCCS is composed of seven blocks (Figure 4.18). We apply Definition 4.6 and we associate a SDFG task to each block of Figure 4.18. As such, we create one SDFG buffer for each pair of producer/consumer blocks. We compute the production rate, the consumption rate and the initial marking of buffers by applying Theorems 4.1 and 4.2. We use one SDFG buffer to model signals and data transfer blocks between each pair of producer/consumer blocks. This is because we are only interested in the communication mechanisms. Therefore, the actual number of signals and data transfer blocks are irrelevant as long as they imply the same communication mechanism. As such, each arrow in Figure 4.18 represents the set of signals and data transfer blocks that imply the same communication mechanism between each pair of producer/consumer bocks. Arrows that represent delayed communications are labeled

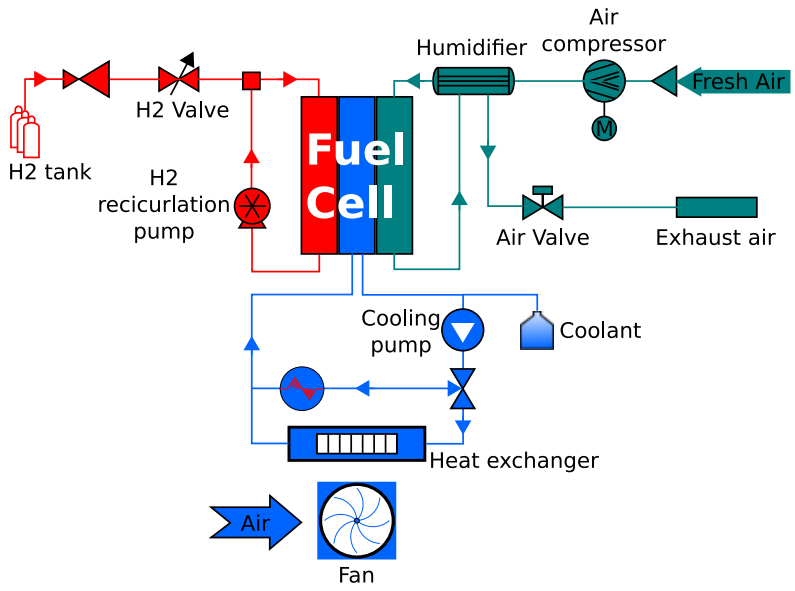


Figure 4.19: Physical system of the FCCS.

with Z^{-1}. Others arrows represent direct communications.

To illustrate the computation of production/consumption rates, let us consider the arrow from block *SS* at period 10ms to *PSS* at period 100ms. The communication mechanism of this arrow is delayed and we associate the buffer $a = (SS, PSS)$. According to Theorem 4.2 the production rate is $in_a = 10$, the consumption rate is $out_a = 100$, $gcd_a = 10$ and the initial marking is $M_0 = 100 + 10 - 10 = 100$. The resulting SDFG in Figure 4.20 is equivalent in size to the initial SBD in Figure 4.18.

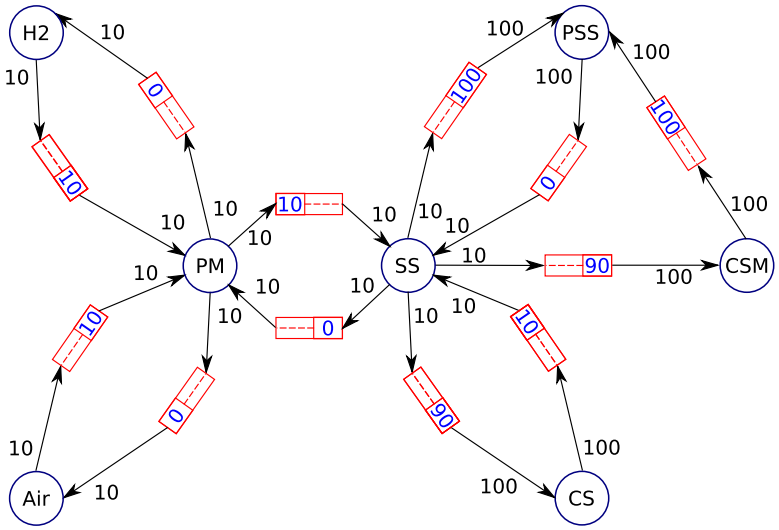


Figure 4.20: SDFG model of Fuel Cell Control System.

Notice that data streams in the multi-periodic part of Figure 4.20 (*i.e.* *SS*, *PSS*, *CSM* and *CS*) can be significantly reduced. In fact, buffers that connect *SS* to one of *PSS*, *CSM* and *CS* involves two different production and consumption rates. For each of these buffers, let us denote by n_{rw} and n_{rw_op} the total number of read and write operations, and the

number of read and write operations necessary to operate the system, respectively. During one interval $\mathcal{N}_{i,j} = 100$, we have $n_{rw} = \frac{100}{10} + \frac{100}{100} = 11$ and $n_{rw\_op} = 2 \times \frac{100}{\max(100,10)} = 2$. Thereby, for each of the aforementioned buffers the data streams is reduced by 82%. This information can be later used to analyze buffers sizing and bus utilization reduction.

The first consequence of having modeled FCCS by SDFG is that we have a static and well-defined characterization of communications among blocks. This enables to transform the Matlab/Simulink descriptions into SDFG formalism, which is very common to optimize the execution and implementation on multi/many-core architectures. The SDFG model is an entry point to the existing studies and tools [89, 96] that use SDFG to provide solutions for mapping, routing and the analysis of certain execution parameters on multi/many-core.

4.7 Conclusion

In this chapter, we performed an in-depth study of Matlab/Simulink functional specification. We also proved that the dataflow between a pair of periodic blocks in Matlab/Simulink is modeled by a SDFG buffer. Then, we showed that an entire multi-periodic Matlab/Simulink system can be modeled by a SDFG following a certain design rules.

The proposed SDFG characterizes the communications finely by using closed mathematical formulas. The advantages of this SDFG compared to other approaches are its complete, concise and precise formalism. That is, it catches exactly the functional synchronous semantics of systems, while having a size equivalent to the number of communication pairs. The representation is not on the hyper-period and all dependency constraints are expressed by mathematical formulas that do not use algorithmic processing and are scalable.

The approach developed in this chapter allows to have an automatic transformation from multi-periodic Matlab/Simulink towards the SDFG formalism, with a fine and complete description of communications. This transformation is not expensive from an algorithmic perspective, but establishes a complete chain that makes the link between designers of Matlab/Simulink applications and specific SDFG analysis tools. This is very interesting because on the one hand Matlab/Simulink is widely used in industry. On the other hand, SDFG is a well known formalism in academia [89, 96]. This chapter provides an original framework to bridge works of several communities, *i.e.* SDFG and real-time communities.

The results of this chapter is the first step toward the implementation. In next chapter, we use the analytical properties of SDFG to construct an AUTOSAR compliant preemptive scheduling that implements the functional dataflow.

Chapter 5

Dependencies and preemptive scheduling on single-core

Contents

5.1 Introduction

In this chapter, we introduce a static preemptive scheduling method on single-core. This scheduling method enables a deterministic implementation of the dataflow enforced by Matlab/Simulink. As such, we can get an AUTOSAR implementation that complies with the functional specification validated in simulation.

We assume that the Matlab/Simulink application verifies Property 4.1 and is fully modeled by the SDFG of Chapter 4. For simplicity, we also assume that the nodes of this SDFG are implemented as real-time tasks and are scheduled by the RTOS with a static preemptive policy. Then, we exploit the scheduling of the SDFG to establish a *temporal isolation* that realizes the dataflow of Matlab/Simulink. In fact, the temporal isolation sets execution intervals to tasks so that consumer jobs execute only after the completion of its producer jobs. This ensures that the preemptive executions of tasks can use the same input data as enforced by the functional specification.

For this study, we consider both periodic concrete and offset-free tasks [51] with constrained deadlines. We formulate a *Linear Program* (LP) and propose several heuristics to find the graph scheduling that establishes the temporal isolation. We use the latter to adjust or set tasks offset and deadline in order to transform the dependent tasks set into an independent one. Hence, we easily get rid of the dependency constraints and we use existing static preemptive scheduling algorithms of independent tasks.

The temporal isolation technique sets a global implementation flow from the functional specification of Matlab/Simulink to the real-time implementation analysis on AUTOSAR. Furthermore, it allows to schedule dependent tasks without requiring synchronization mechanisms, making the dataflow implementation completely characterized. The proposed temporal isolation heuristics find a scheduling solution in 76 percent of cases. As such, our method provides a fast technique to deal with dependencies in large multi-periodic systems.

This chapter also introduces a method to generate random dependent real-time tasks set modeled by the SDFG of Chapter 4. This method has three main objectives. The first is to generate alive SDFG. The second is to generate tasks parameters so that tasks sets are not trivially infeasible due to dataflow constraints. The third is to provide dependent real-time tasks sets that can be used for experiential assessments.

The rest of this chapter is organized as follows. Section 5.2 presents related works on the static preemptive scheduling of both independent and dependent real-time tasks, and the scheduling of SDFG. Section 5.3 characterizes both the valid preemptive scheduling that enables a deterministic dataflow implementation and the SDFG scheduling that sets the temporal isolation. Then, in Section 5.4, we formulate a linear program to build the temporal isolation. We also detail the algorithm to build a valid preemptive scheduling on single-core. Section 5.5 presents the method of generating random dependent tasks sets, which are used in the experiments of Section 5.6. Finally, Section 5.7 concludes the chapter and gives some perspectives on the temporal isolation technique.

5.2 Related works

We are interested in the static preemptive scheduling on single-core that ensures a deterministic implementation of dataflow. That is, we handle both preemptive scheduling and data dependencies.

In the context of static preemptive scheduling on single-core, RM [76], DM [75] and OPA [5] are optimal for synchronous implicit-deadline, synchronous constrained-deadline and asynchronous constrained-deadline tasks, respectively. For offset-free tasks [51], Goossens and Devillers [50] showed the non-optimality of the aforementioned policies. Then, Goossens [51] and Mathieu *et al.* [55] proposed several near-optimal approaches based on OPA to schedule offset-free tasks. Although these policies use static preemptive scheduling, they do not address the deterministic dataflow implementation. In this chapter, we propose an approach that transforms dependent tasks into independent ones, so that those policies can be used.

Forget *et al.* [46] addressed the deterministic dataflow implementation by proposing an adaptation of DM and OPA to schedule dependent synchronous and asynchronous periodic tasks, respectively. They modeled the system by tasks graph and they proposed a scheduling approach based on the precedences encoding of Chetto *et al.* [29]. The latter consists in adjusting task parameters and priorities to realize dependencies on single-core. Their approach produces the optimal scheduling algorithm on single-core. However, for multi-periodic and cyclic dependencies it requires the unfolding of the graph, which is not in polynomial size and suffers from performance issues [80].

Compared to existing works on real-time scheduling of dependent tasks, our approach relies instead on the SDFG model of the dataflow. We use the SDFG scheduling to propose several heuristics that transform the dependent tasks into independent ones without unfolding the graph. This allows to handle efficiently multi-periodic and cyclic dependencies in huge systems. In addition, automotive embedded softwares often contain cyclic dependencies, which represent feed-back loops in control applications. The periods of tasks in such applications are driven by the dynamic of the physical system, and are not necessary identical nor harmonics. With the increasing complexity of automotive embedded features, such applications become more common and there is a big interest in addressing their scheduling efficiently.

Bamakhrama and Stefanov [7, 8] scheduled a CSDFG [17] as an asynchronous set of periodic real-time tasks. However, their work considers only acyclic CSDFG and their scheduling is not constrained by real-time attributes. Our approach considers both cyclic and acyclic SDFGs. In addition, we use the real-time attributes of tasks as constraints that we guarantee in our scheduling. For this purpose, we exploit the mathematical properties of the SDFG [79, 83, 80] to construct a graph scheduling that guarantees precedence constraints and real-time constraints. Then, we use this graph scheduling to transform the dependent real-time tasks set with strict timing constraints into an independent set. We schedule the independent tasks using the existing scheduling algorithms for real-time

tasks.

Authors in [57, 114, 71] investigated the refinement of execution intervals for the scheduling of SDFG. Given a priority ordering, Hausmans *et al.* [57] and Wilmanns *et al.* [114] compute execution interval bounds that take into account interferences from higher-priority tasks. Then, they analyze the impact of processor sharing on the scheduling of the graph. Kurtin *et al.* [71] refined the execution interval estimation by an iterative process that converges to a minimum offset and a maximum finish time for each task. Those approaches are different from ours as they evaluate the execution intervals of tasks on the preemptive single-core to produce a graph scheduling that takes it into account. In our approach, we use heuristics that rely on the real-time timing constraints of tasks to build a graph scheduling. The latter imposes the execution intervals that tasks scheduling must conform to. As such, we establish a fast technique to deal with dependency constraints in preemptive scheduling.

Furthermore, at our knowledge, this is the first time SDFG is used to construct a temporal isolation that constraints the real-time execution.

5.3 Valid scheduling of dependent real-time tasks

Let $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n dependent periodic real-time tasks modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. In this section, we define and characterize the preemptive scheduling of dependent real-time tasks set $\boldsymbol{\mathcal{T}}$ that allows a deterministic semantics-preserving dataflow implementation.

Definition 5.1 (Valid preemptive scheduling)
Let $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n dependent periodic real-time tasks modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. A valid preemptive scheduling of $\boldsymbol{\mathcal{T}}$ guarantees both the real-time constraints of tasks and the dataflow constraints of $\boldsymbol{\mathcal{G}}$.

From Definition 5.1, a valid preemptive scheduling of $\boldsymbol{\mathcal{T}}$ can be characterized by a valid scheduling of $\boldsymbol{\mathcal{G}}$ (*c.f.* Definition 3.5) that verifies the real-time constraints of tasks. Theorem 5.1 expresses the scheduling of $\boldsymbol{\mathcal{G}}$ that verifies the real-time constraints of tasks.

Theorem 5.1 (SDFG scheduling with real-time constraints)
Let us consider a set of dependent periodic real-time tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task $\tau_i \in \boldsymbol{\mathcal{T}}$ is characterized by the real-time attributes O_i, C_i, T_i and D_i. $\boldsymbol{\mathcal{T}}$ is modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. A scheduling of $\boldsymbol{\mathcal{G}}$ guarantees the real-time constraints if it is a periodic scheduling that defines a period $T_i^{\mathcal{G}}$ *and a first execution staring date $r_i^{\mathcal{G}}[1]$ for each task $\tau_i \in \boldsymbol{\mathcal{T}}$ such as:*

(1) $T_i^{\mathcal{G}} = T_i$

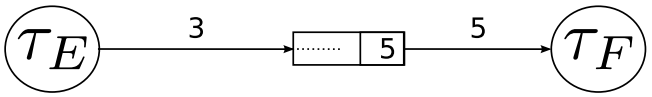
(2) Equation 5.1 is verified:

$$r_i^{\mathcal{G}}[1] \geq O_i \tag{5.1a}$$

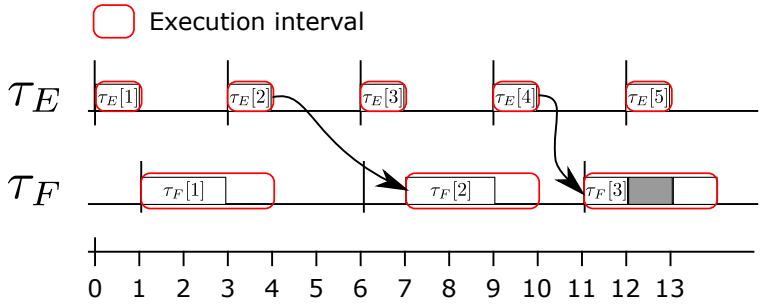
$$r_i^{\mathcal{G}}[1] + D_i^{\mathcal{G}} \leq O_i + D_i \tag{5.1b}$$

Proof. Let us assume that the execution interval for the first job of τ_i in the scheduling of the SDFG verifies the real-time constraints. That is, this execution interval is between O_i and $O_i + D_i$. This is expressed formally by $r_i^{\mathcal{G}}[1] < r_i[1]$ and $r_i^{\mathcal{G}}[1] + D_i^{\mathcal{G}} \leq r_i[1] + D_i$, with $r_i[1] = O_i$. If the scheduling of the SDFG verifies the real-time constraints, the execution interval of the n_ith job of τ_i also verifies the real-time constraints. Therefore, we get $r_i^{\mathcal{G}}[n_i] \geq r_i[n_i]$ and $r_i^{\mathcal{G}}[n_i] + D_i^{\mathcal{G}} \leq r_i[n_i] + D_i[n_i]$, where $r_i[n_i] = O_i + (n_i - 1) \cdot T_i$. However, as $D_i^{\mathcal{G}}$ is a constant, the graph scheduling is periodic with the period $T_i^{\mathcal{G}} = T_i$ for each task $\tau_i \in \boldsymbol{\mathcal{T}}$. This proves the theorem. □

The SDFG in Figure 5.1(a) models the dependencies between the periodic real-time tasks $\tau_E(O_E = 0, C_E = 1, D_E = 3, T_E = 3)$ and $\tau_F(O_F = 1, C_F = 2, D_F = 5, T_F = 5)$. Figure 5.1(b) illustrates both a valid periodic scheduling of the SDFG of Figure 5.1(a) and the static preemptive scheduling of τ_E and τ_F on single-core. This valid periodic scheduling assigns the initial starting dates $r_E^{\mathcal{G}}[1] = 0$ and $r_F^{\mathcal{G}}[1] = 1$, and the execution intervals $D_E^{\mathcal{G}} = 1$ and $D_F^{\mathcal{G}} = 3$ to τ_I and τ_F, respectively. As such, the precedence constraints and the real-time constraints are verified in the scheduling of the SDFG.



(a) Example of two dependent real-time tasks modeled by a SDFG graph.



(b) Illustration of preemptive executions and execution intervals.

Figure 5.1: Example of preemptive executions inclusion in graph scheduling.

Note that in Figure 5.1(b), the initial release times and the execution intervals in the scheduling of the SDFG are different from the offsets and the deadlines of tasks. It was required to ensure the validity condition of the scheduling of the SDFG (*c.f.* Theorem 3.3).

As such, it is possible to construct a valid periodic scheduling of $\boldsymbol{\mathcal{T}}$ by adjusting the real-time attributes of tasks according to a valid scheduling of $\boldsymbol{\mathcal{G}}$ that verifies the real-time constraints. Thus, we deduce the following corollary from Theorem 5.1.

Corollary 5.1.1
Let $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of n dependent periodic real-time tasks modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. A valid preemptive scheduling of $\boldsymbol{\mathcal{T}}$ can be obtained by modifying the offsets and deadlines of the real-time tasks according to a valid periodic scheduling of $\boldsymbol{\mathcal{G}}$ that verifies the real-time constraints.

5.4 Construction of a valid preemptive scheduling

In this section, we use Corollary 5.1.1 to construct a valid preemptive scheduling of a set of dependent periodic real-time tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. To this end, we formulate a *Linear Program* to build a valid periodic scheduling of $\boldsymbol{\mathcal{G}}$. We use this graph scheduling to adjust the offset and deadline of tasks, so that we transform the dependent tasks into independent ones. Hence, any feasible preemptive scheduling of the latter produces a valid preemptive scheduling of the former.

In Subsection 5.4.1, we describe the variables and constraints of the LP. In Subsection 5.4.2, we propose several heuristics as objective functions for the LP. These heuristics aim are used to find a periodic scheduling of the SDFG that guarantees the existence of a feasible preemptive scheduling of the independent tasks set. Finally, in Subsection 5.4.3 we define the algorithm to construct a valid preemptive scheduling on single-core.

5.4.1 Variables and constraints of the LP

Let us consider the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ that models the dependent tasks set $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Finding a valid periodic scheduling of $\boldsymbol{\mathcal{G}}$ consists in setting the first execution starting date $r_i^{\mathcal{G}}[1]$, the deadline or execution interval $D_i^{\mathcal{G}}$, and the period T_i of every task. It follows that the variables of the LP are $r_i^{\mathcal{G}}[1]$ and $D_i^{\mathcal{G}}$ for every SDFG task $\tau_i \in \boldsymbol{\mathcal{T}}$. To ease the readability, in the sequel we refer to $r_i^{\mathcal{G}}[1]$ by $r_i^{\mathcal{G}}$.

For concrete task, the constraints on the first execution starting dates and the execution intervals are expressed by Equation 5.1. In fact, Equation 5.1 exploits the offsets of concrete tasks to guarantee the real-time constraints.

For an offset-free task, the offset is not a constraint. Thus, it is only required that the first execution starting date $r_i^{\mathcal{G}}$ is greater than zero. A such, the constraints on the first execution starting dates and the execution intervals are expressed by Equation 5.2 in the case of offset-free tasks.

$$r_i^{\mathcal{G}} \geq 0 \tag{5.2a}$$

$$C_i \leq D_i^{\mathcal{G}} \leq D_i \tag{5.2b}$$

Moreover, for both concrete and offset-free tasks, the execution interval of a task $\tau_i \in \boldsymbol{\mathcal{T}}$ cannot be less than the worst case execution time C_i. This constraint is expressed by Equation (5.3).

$$D_i^{\mathcal{G}} \geq C_i \tag{5.3}$$

In both concrete and offset-free tasks, the constraint of Equation (5.4) expresses the validity condition. In fact, since we are concerned with valid periodic graph scheduling, the condition of Equation (3.4) in Theorem 3.4 must be verified. However, from Theorem 5.1 the period of each task $\tau_i \in \mathcal{T}$ in the graph scheduling is T_i. Also, from the SDFG model defined in Chapter 4, the production and consumption rates of each buffer $a = (\tau_i, \tau_j) \in \mathcal{A}$ are $in_a = T_i$ and $out_A = T_j$, respectively. Equation (5.4) is obtained by replacing $K = \frac{T_i}{in_a} = \frac{T_j}{out_A} = 1$ in Equation (3.4).

$$r_j^{\mathcal{G}} - r_i^{\mathcal{G}} \geq D_i^{\mathcal{G}} + out_a - M_0(a) - gcd_a \tag{5.4}$$

5.4.2 Objective function for execution intervals

Our goal is to find execution intervals that result in a preemptive execution of tasks within the graph scheduling. The easiest way to achieve this is by assigning the maximum allowed execution interval D_i to each SDFG task τ_i. However, the dependency constraints do not always allow to assign the maximum allowed execution intervals to tasks.

In fact, let us consider the dependent periodic offset-free tasks $\tau_G(T_G = 3, C_G = 1, D_G = 2)$, $\tau_H(T_H = 2, C_H = 0.5, D_H = 2)$ and $\tau_I(T_I = 5, C_I = 1, D_I = 5)$ modeled by the SDFG of Figure 5.2. We extract for instance the dependency constraints $\tau_G[1] \rightarrow \tau_H[1]$, $\tau_H[1] \rightarrow \tau_I[1]$ and $\tau_I[1] \rightarrow \tau_G[3]$. We assign the initial execution starting dates $r_G^{\mathcal{G}} = 0$ and $r_H^{\mathcal{G}} = 2$. With respect to the precedence constraints, we also assign the maximum allowed execution intervals $D_G^{\mathcal{G}} = D_G = 2$ and $D_H^{\mathcal{G}} = D_H = 2$ to τ_G and τ_H, respectively. From dependency $\tau_H[1] \rightarrow \tau_I[1]$, we can assign the minimum initial execution starting date $r_I^{\mathcal{G}} = 4$ to τ_I. However, $D_I^{\mathcal{G}} = D_I = 5$ cannot be used otherwise $r_I^{\mathcal{G}} + D_I^{\mathcal{G}} = 9 \not\leq r_G^{\mathcal{G}} + 2 \cdot T_G = 6$ and the validity condition of $\tau_I[1] \rightarrow \tau_G[3]$ will not be respected. Consequently, at least one execution interval cannot be set to the maximum.

Nonetheless, tasks do not strictly require their maximum execution intervals. In fact, in a preemptive scheduling tasks with high priorities suffer less from preemption than low priority ones. Thus, highest priority tasks generally need less execution interval than lowest priority ones. In the example of Figure 5.2, assuming τ_I is given the highest priority, its

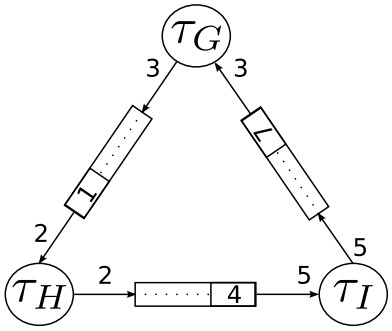


Figure 5.2: Example of dependent tasks with constrained execution intervals.

worst case response time is $R_I = C_I = 1$. Then, the execution interval of τ_I can be set to $D_I^{\mathcal{G}} = 1$. Combined with previous defined values, the precedence constraint $\tau_I[1] \rightarrow \tau_G[3]$ is now respected. As consequence, our goal is to assign the maximum possible execution interval to each task.

Classic approach

The assignment of the maximum possible execution intervals can be achieved by minimizing the difference between the maximum allowed execution interval D_i and the assigned one $D_i^{\mathcal{G}}$ for every task τ_i. In classical approach, this is expressed by minimizing the weighted maximum of the differences, where the weight denotes the importance of minimizing the difference for a given task. More formally, let ρ_i be the weight of task τ_i. The objective function is formulated as follows:

$$\min \left(\max_{\boldsymbol{\tau}} \left(\rho_i \cdot (D_i - D_i^{\mathcal{G}}) \right) \right) \tag{5.5}$$

However, this approach does not provide a precise minimization of the difference on each task.

Gradient approach

Let $f(D_1^{\mathcal{G}}, D_2^{\mathcal{G}}, \ldots, D_n^{\mathcal{G}})$ be a function such as its partial derivative with respect to $D_i^{\mathcal{G}}$ is $\frac{\partial f}{\partial D_i^{\mathcal{G}}} = \rho_i \cdot (D_i - D_i^{\mathcal{G}})$. Here, $\frac{\partial f}{\partial D_i^{\mathcal{G}}}$ measures the weighted (ρ_i) gradient (*i.e.* the difference) between the maximum allowed execution interval D_i and the assigned one $D_i^{\mathcal{G}}$ for tasks τ_i. The gradient of f is formally expressed by Equation (5.6).

$$\overrightarrow{\nabla} f \left| \begin{array}{l} \rho_1 \cdot (D_1 - D_1^{\mathcal{G}}) \\ \rho_2 \cdot (D_2 - D_2^{\mathcal{G}}) \\ \vdots \\ \rho_n \cdot (D_n - D_n^{\mathcal{G}}) \end{array} \right. \tag{5.6}$$

The total differential $\mathrm{d}f = \sum \frac{\partial f}{\partial D_i^{\mathcal{G}}}$ measures the combined gradient of the execution interval on all tasks. Note that the more the execution interval $D_i^{\mathcal{G}}$ is closer to the deadline D_i, the smaller is $\frac{\partial f}{\partial D_i^{\mathcal{G}}}$ and the smaller is $\mathrm{d}f$. Consequently, we use $\mathrm{d}f$ as the cost function of the linear program and our objective is to minimize it. The objective function of the gradient approach is expressed by Equation (6.10). This function combines the differences on all tasks so that to have a more effective minimization on each.

$$\min(\sum_{\mathcal{T}} \rho_i \cdot (D_i - D_i^{\mathcal{G}})) \tag{5.7}$$

In the objective function of Equation (6.10), the weight ρ_i of a task $\tau_i \in \mathcal{T}$ defines the importance of minimizing its gradient. If ρ_i has the highest value, the gradient of τ_i is minimized first. The gradient of the task with the next highest weight is minimized next, and so on. In the previous example, the weight of τ_I must be smaller than those of τ_G and τ_H. As such, the linear program assigns the maximal execution intervals $D_G^{\mathcal{G}} = 3$ and $D_H^{\mathcal{G}} = 2$ to τ_G and τ_H, respectively. Then, the remaining execution interval $D_I^{\mathcal{G}} = 1$ is assigned to τ_I.

We propose several heuristics method to assign weights to tasks.

- The heuristic of Equation (5.8) minimizes gradients fairly. That is, it does not give any importance to the minimization of a gradient compared to others.

$$\rho_i = 1, \quad \forall\ \tau_i \in \mathcal{T} \tag{5.8}$$

- The heuristic of Equation (5.9) assumes that the importance of gradients varies with deadlines. In fact, let us consider tasks $\tau_G(T_G = 3, C_G = 1, D_G = 2)$, $\tau_H(T_H = 2, C_H = 0.5, D_H = 2)$ and $\tau_I(T_I = 5, C_I = 1, D_I = 5)$ of Figure 5.2. A gradient of 1 for each task entails the execution intervals $D_G^{\mathcal{G}} = 1$, $D_H^{\mathcal{G}} = 1$ and $D_I^{\mathcal{G}} = 4$. Hence, the same value of gradient produces tight execution intervals for tasks with small deadlines. Consequently, tasks with small deadlines must be given small gradients because the less is the deadline, the more meaningful is the gradient.

$$\rho_i = \frac{1}{D_i} \quad \forall\ \tau_i \in \mathcal{T} \tag{5.9}$$

- The heuristic of Equation (5.10) uses the same principle as above, except that the reasoning is made on the slack time rather than on the deadline.

$$\rho_i = \frac{1}{D_i - C_i} \quad \forall\ \tau_i \in \mathcal{T} \tag{5.10}$$

5.4.3 Algorithm to construct a valid preemptive scheduling

The valid periodic scheduling of the SDFG from the LP defines initial starting dates and execution intervals that guarantee the validity condition of Theorem 3.3. By adjusting the real-time parameters (offsets and deadlines) of the real-time tasks according to the latter, we transform the initial dependent tasks set into an independent one. Hence, we construct a valid preemptive scheduling of the dependent tasks set by finding a feasible preemptive scheduling of the independent set. This entails finding a priority assignment that produces a feasible preemptive scheduling.

Figure 5.3 illustrates a valid periodic scheduling of the SDFG of Figure 5.1(a). This graph scheduling assigns the initial starting dates $r_G^{\mathcal{G}} = 0$, $r_H^{\mathcal{G}} = 2$ and $r_I^{\mathcal{G}} = 2.5$, and the execution intervals $D_G^{\mathcal{G}} = 2$, $D_H^{\mathcal{G}} = 0.5$ and $D_I^{\mathcal{G}} = 2.5$ to τ_G, τ_H and τ_I, respectively. Accordingly, τ_G can only be assigned the highest priority in a static preemptive scheduling on single-core. This is because any preemption will overflow its execution outside its assigned execution interval. Then, we verify that the priority ordering $\pi_H > \pi_G > \pi_I$ schedules tasks within the execution intervals defined by the graph scheduling. We use this technique to construct a valid preemptive scheduling for a set of dependent periodic real-time tasks modeled by a SDFG on single-core.

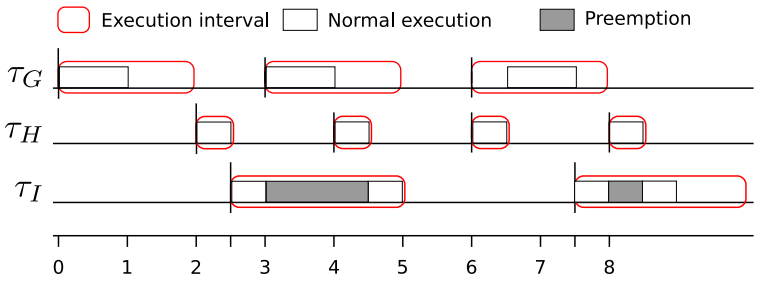


Figure 5.3: Valid periodic scheduling of the SDFG of Figure 5.2.

Let us denote by o_i^* and d_i^* the adjusted offset and deadline of the real-time task τ_i. Then, we have $o_i^* = r_i^{\mathcal{G}}$ and $d_i^* = D_i^{\mathcal{G}}$.

At this step, any priority assignment that produces a feasible scheduling of the resulting independent tasks set can be used. In this work, we chose the OPA [5] policy because it is the optimal static preemptive scheduling algorithm for independent asynchronous tasks on single-core.

Algorithm 1 describes the method to construct a valid preemptive scheduling of a dependent periodic real-time tasks set $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ modeled by a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ on single-core. The originality of this algorithm is in the function FIND_PERIODIC_SCHEDULE(). In fact, it is this function that constructs the valid periodic scheduling of the SDFG by solving the LP with the heuristics presented above. Accordingly, the dependent tasks set is transformed into an independent one. Then, OPA is used to find a feasible priority assignment for the static preemptive scheduling on single-core.

Algorithm 1 Construction of valid preemptive scheduling of dependent periodic tasks set $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ modeled by a SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$.

```
scheduled ← false
SG = FIND_PERIODIC_SCHEDULE(G)
if SG then
    for τ_i ∈ T do
        o_i^* ← r_i^G
        d_i^* ← D_i^G
    end for
    priorities = OPA(T)
    if |priorities| = n then
        for τ_i ∈ T do
            π_i ← priorities(τ_i)
        end for
        scheduled ← true
    end if
end if
if scheduled = false then No valid preemptive schedule found
end if
```

5.5 Dependant task sets generation

In this section, we address the generation of random dependent tasks sets of cardinality n. We developed this generation method in order to evaluate the construction of a valid preemptive scheduling on a wide range of dependent tasks sets. The random tasks sets are created so that their dependency model follows the dataflow model of Chapter 4. Our generation method follows three steps so that the generate tasks sets are not trivially infeasible.

In Subsection 5.5.1, we explain the method to generate random real-time tasks sets. In Subsection 5.5.2, we describe the generation of a dependency model using a random generated SDFG. Then, in Subsection 5.5.3 we generate the offsets so that the tasks set is not trivially infeasible.

5.5.1 Tasks sets generation

The first step in the generation of the dependent real-time tasks set consists in defining the period, the WCET and the deadline of tasks.

Periods

We choose the periods of tasks randomly within a sample that is composed of commonly used period values in real applications. The goal of this generation method is to generate tasks with representative periods while limiting the size of the hyperperiod. This is because the hyperperiod defines the interval of the feasibility analysis in OPA [5]. However, the time complexity of the latter increases exponentially with the hyperperiod [5]. In this thesis, we use a sample composed of periods 10ms, 20ms, 30ms, 50ms and 100ms.

Utilization and WCET

We adopt the utilization based technique of Bini and Buttazo [18] to set the WCET of tasks. First, we choose the total utilization U randomly between 0.6 and 1. Then, we distribute U uniformly over tasks using the *UUniFast* method [18]. The WCET of each task is afterward derived from its utilization and its period.

Deadlines

Throughout this thesis, we consider only implicit tasks for which deadlines are equal to periods. We believe that the experimental results can be extended to constrained tasks.

5.5.2 Dependency model generation

The second step of our method is the generation of the dependency model. It consists in defining a SDFG such as:

- each node is associated to a real-time task generated in Subsection 5.5.1, and
- SDFG tasks and buffers parameters comply with the dependency model of Chapter 4.

To achieve this, we generate a random SDFG composed of n nodes using the SDFG generation tool Turbine [19]. Turbine generates connected SDFG and allows to specify several settings such as tasks degree (*i.e.* number of buffers incident to tasks). However, the SDFG tasks and buffers parameters generated by Turbine do not comply with the dependency model defined in Chapter 4. We adjust these parameters as follows.

SDFG tasks parameters

First, we assign the generated real-time attributes to the generated SDFG tasks. Then, following the dependency model defined in Chapter 4, we set the production and consumption rates equal to periods.

Buffers parameters

Each buffer of the graph is randomly assigned an initial marking that characterizes one communication mechanism between direct, delayed and hybrid. However, to avoid deadlock graphs, which are trivially infeasible, we break each strict cyclic dependency by inserting at least one buffer with delayed or hybrid communication.

In fact, let us consider a cycle μ in the SDFG. Let us assume that every buffer $a \in \mu$ uses the direct communication. From Theorem 4.1, we get $\sum_{a\in\mu} M_0(a) = \sum_{a\in\mu} out_a - gcd_a$, which does not verify the sufficient condition of liveliness (Theorem 3.1). Hence, we cannot decide on the liveliness of the SDFG.

Now, let us assume that one buffer $a_0 \in \mu$ uses the delayed communication. From Theorem 4.1 and Theorem 4.2, we get $\sum_{a\in\mu} M_0(a) = \left(\sum_{a\in\mu\setminus a_0}(out_a - gcd_a)\right) + (out_{a_0} + in_{a_0} - gcd_{a_0})$. That is, the sufficient condition of liveliness in Theorem 3.1 is verified. As consequence, the SDFG is alive. The same reasoning is used for hybrid communication.

We invoke Turbine after assigning the parameters of all tasks and buffers to check and ensure that the generated SDFG is deadlock free.

5.5.3 Offsets generation

The third and last step of our method is the generation of offsets of tasks, so that the tasks set is not trivially infeasible. This is because the offsets of dependent concrete tasks cannot be totally random. In fact, let τ_i and τ_j be two dependent tasks related by the precedence constraint $\tau_i[n_i] \rightarrow \tau_j[n_j]$. This precedence requires to execute $\tau_j[n_j]$ after $\tau_i[n_i]$ completes. However, this is possible only if the execution of $\tau_j[n_j]$ (after that of $\tau_i[n_i]$) can complete before its absolute deadline. More formally, a precedence constraint $\tau_i[n_i] \rightarrow \tau_j[n_j]$ is possible only if Equation 5.11 is verified. Otherwise, $\tau_j[n_j]$ cannot execute after $\tau_i[n_i]$ and before its absolute deadline $r_j[n_j] + D_j$. Lemma 5.1 gives the condition to select the offsets of two dependent tasks.

$$r_i[n_i] + C_i \leq r_j[n_j] + D_j - C_j \tag{5.11}$$

Lemma 5.1
Let $a = (\tau_i, \tau_j)$ be a buffer that models the precedence constraints between two concrete tasks $\tau_i(T_i, C_i, O_i, D_i)$ and $\tau_j(T_j, C_j, O_j, D_j)$. $M_0(a)$ is the initial marking of a. The precedence constraints enforced by the buffer $a = (\tau_i, \tau_j)$ are possible only if O_i and O_j verify Equation (5.12).

$$O_i \leq O_j + T_i - T_j - C_i - C_j + D_j + M_0(a) + \min(0, T_j - T_i) \tag{5.12}$$

Proof. From Equation 5.11, any precedence constraint $\tau_i[n_i] \rightarrow \tau_j[n_j]$ enforced by the

buffer $a = (\tau_i, \tau_j)$ is possible only if $r_i[n_i] + C_i \leq r_j[n_j] + D_j - C_j$. This is equivalent to $O_i + (n_i - 1) \cdot T_i + C_i \leq O_j + (n_j - 1) \cdot T_j + D_j - C_j$. Then, we get $O_i \leq O_j + T_i - T_j - C_i - C_j + D_j + (-n_i \cdot T_i + n_j \cdot T_j)$. However, from Equation (3.2) $-n_i \cdot T_i + n_j \cdot T_j \leq M_0(a) + \min(0, T_j - T_i)$. This proves the lemma. □

According to Lemma 5.1, we generate random offsets between 0 and tasks period so that Equation (5.12) is verified for all their predecessors (*i.e.* tasks that produce their data).

5.6 Experiments

In this section, we use dependent tasks sets generated randomly to evaluate our approach of constructing a valid preemptive scheduling on single-core.

In Subsection 5.6.1, we compare heuristics to see the one that gives the best results depending on system utilization and cardinality. In Subsection 5.6.2, we evaluate our method compared to the optimal algorithm [46] for concrete tasks and acyclic graphs. Finally, in Subsection 5.6.3, we study some timing aspects.

5.6.1 Comparison of heuristics

In the following analyses, we define the objective functions of Equation (5.13) and Equation (5.14). These functions are obtained from the classical approach by replacing the weight ρ_i by 1 and $\frac{1}{D_i - C_i}$ in Equation (5.5), respectively. These objective functions serve to demonstrate the interest of the gradient approach.

$$\min(\max(D_i - D_i^{\mathcal{G}})) \tag{5.13}$$

$$\min(\frac{1}{D_i - C_i} \cdot \max(D_i - D_i^{\mathcal{G}})) \tag{5.14}$$

In the experiments of Figure 5.4, we vary the total utilization from 0.65 to 0.95 (abscissa). We generate for each utilization a set of 10 graphs. Each graph is composed of 100 tasks with a degree from 1 to 10. Then, we apply Algorithm 1 on graphs and we represent the average percentage of tasks that are scheduled by each objective function.

We can observe in Figure 5.4 that independently of utilizations, the objective functions based on the gradient approach produce better scheduling results than the ones based on the classical approach. Moreover, heuristic $\rho_i = \frac{1}{D_i - C_i}$ produces the best results.

In order to analyze the influence of the number of tasks, we repeat the same experiments, but with decreasing number of tasks. We obtained similar results to those of Figure 5.4.

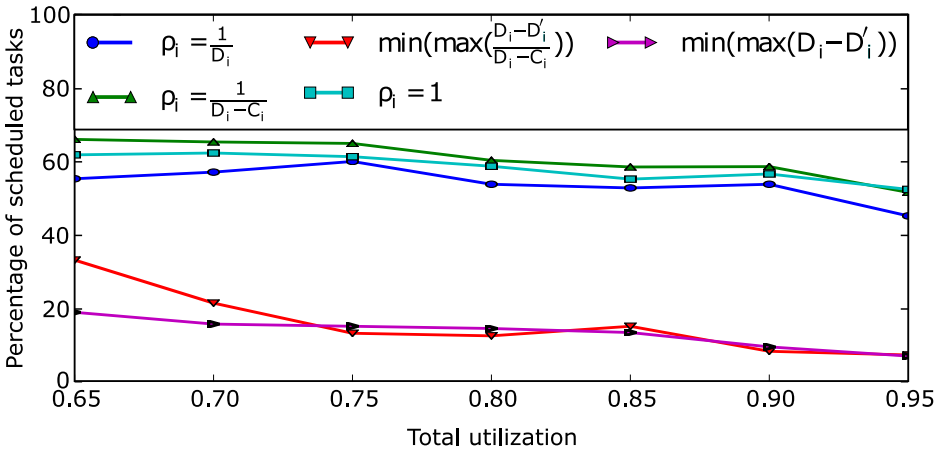


Figure 5.4: Scheduling test for random graphs composed of 100 tasks.

Throughout these experiments, the average percentage of tasks that are scheduled by Algorithm 1 varies from 50 to 70 percent. We observe a slight drop when utilization increases. Nevertheless, the number of tasks shows no real impact on the performance of heuristics with regards to scheduling.

5.6.2 Evaluation of the method

In Figure 5.4, the average percentage of scheduled tasks is less that 70 percent. However, we do not know if the generated graphs have feasible scheduling. To evaluate our temporal isolation technique, we repeat the experiments using acyclic graphs modeling concrete dependent tasks. The latter are fully scheduled by the optimal algorithm of Forget *et al.* [46]. The scheduling results are shown in Figure 5.5.

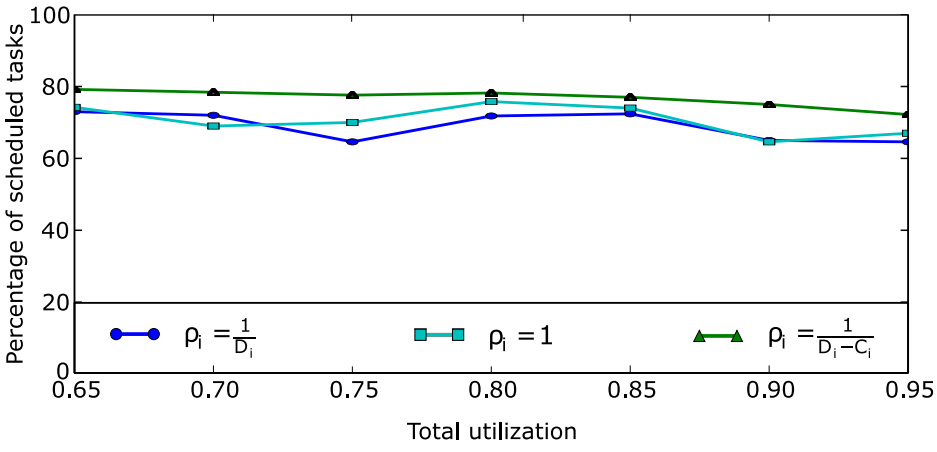


Figure 5.5: Evaluation of the scheduling method compared to the optimal algorithm [46] for acyclic dependent tasks on single-core.

Compared to the optimal algorithm [46], heuristic $\rho_i = \frac{1}{D_i - C_i}$ that produces the best results, schedules an average of 76.8 percent of tasks with a standard deviation of 2.24. Tasks that are not scheduled come both from the structure of graphs and the heuristic.

In fact, let us consider the system of Figure 5.6. It is composed of four dependent tasks τ_H, τ_I, τ_J and τ_K having the same offset and the same period 10ms. We assume that $C_H = 1$, $C_I = 2$, $C_J = 1$ and $C_K = 2$. Since $D_I - C_I = 8 < D_H - C_H = 9$ (*resp.* $D_K - C_K = 8 < D_J - C_J = 9$), τ_I and τ_K are given their maximum execution interval 8, while τ_H and τ_J are given their minimum execution interval 1. That is, none of the latter allows preemptions. So, they are infeasible because they have the same offset. However, adding 1 unit to the execution interval of τ_H or τ_J allows preemption and the system becomes feasible.

To address such a case, the temporal isolation need further refinement. The latter consists in an iterative scheduling analysis to determine tasks that requires more execution interval. This refinement is similar to those of [57, 114, 71], which are not in polynomial size. Our approach rather provides a fast technique to deal with dependencies in multi-periodic systems. As such, it provides a fast and good estimate of the impact of dataflow constraints on real-time scheduling. These are interesting features that we use later to address multi-core mapping efficiently.

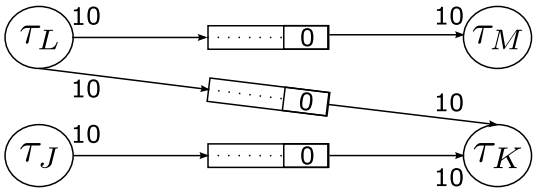


Figure 5.6: Example of not scheduled graph

5.6.3 Timing analyses

The complexity of Algorithm 1 (in terms of computing) is driven by OPA, which is exponential in function of the number of tasks. In fact, Algorithm 1 can be divided in two main parts. The first is related to the function FIND_PERIODIC_SCHEDULE(). This function constructs the valid periodic scheduling of the SDFG and transforms the dependent tasks set into an independent set. The second concerns OPA(), which finds the priority assignment to schedule the independent tasks. In the experiments of Figure 5.7, we measure the computing time of each part on random graphs with increasing tasks number.

The results confirm that OPA is time consuming compared to the construction of a valid periodic scheduling of the SDFG. The algorithmic added to transform the dependent tasks into independent ones is therefore less penalizing than a conventional preemptive scheduling of independent real-time tasks.

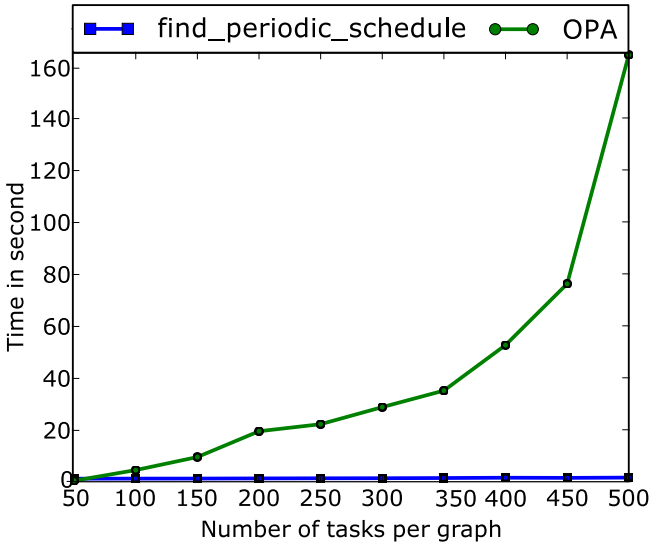


Figure 5.7: Computing time analyses

5.7 Conclusion

In this chapter, we proposed a temporal isolation technique to enable a deterministic implementation of the dataflow without blocking synchronization mechanisms. This technique uses the graph scheduling to adjust real-time tasks offset and deadline. As such, the dependent real-time tasks set is transformed into an independent one. Then, we exploit the existing policies for independent tasks to schedule the latter on single-core.

Furthermore, the temporal isolation technique enables to treat fairly large applications in a reasonable time. It provides a fast technique to deal with dependencies in large multi-periodic systems, along with a good estimate of the impact of dataflow requirements on real-time scheduling. However, the overall process is still limited in terms of computation by the real-time scheduling analysis, which is a classic real-time issue.

This chapter establishes an implementation flow starting from the functional specification of Matlab/Simulink to the scheduling analysis on AUTOSAR. In the next chapter, we exploit the temporal isolation technique to build a method to guarantee the dataflow in multi-core scheduling.

Chapter 6

Characterization of dataflow in partitioned multi-core

Contents

6.1 Introduction

The mapping and scheduling of dependent periodic tasks on multi-core is NP-complete [97]. Therefore, heuristic algorithms are used to reduce the problem complexity [22, 116, 41, 111, 43, 42]. One important aspect when mapping and scheduling dependent tasks in multi-core is to guarantee of the dataflow. This aspect has been addressed in the literature using several approaches. Authors in [41, 111, 82, 95, 87] reduced the complexity by considering only mapping without precedence constraints between cores. In [22, 116, 43, 42, 100, 93, 23, 91, 92, 90], the authors considered precedence constraints between cores. So, they perform iterative and costly scheduling analysis after each mapping decision to verify and ensure the dataflow. To avoid this iterative and costly analysis, authors in [95, 111, 110] separated the mapping from the dataflow requirement. They verify and ensure the latter once for a given mapping. However, experiments of Wang et al. [110] showed that a mapping can be rejected because of strict precedence constraints between cores. Furthermore, with the increasing complexity of automotive embedded features, configurations with multi-core dataflow requirements become more common. Hence, there is a big interest in addressing multi-core dataflow in the mapping of large industrial applications efficiently.

In this chapter, we consider the mapping and scheduling of multi-periodic systems on a fixed number of identical cores. We assume that applications are modeled according to the SDFG specification of Chapter 4 and each node can be implemented as a real-time task. Then, we target a partitioned static scheduling on AUTOSAR multi-core. We address the mapping and scheduling problem from a dataflow perspective and we provide a fast and accurate characterization of dataflow requirements. To do this, we realize the deterministic dataflow by combining the temporal isolation technique of Chapter 5 with the technique of Forget et al. [46]. In fact, the temporal isolation can be used for both single-core and multi-core because it is independent of the mapping of tasks. However, when the tasks are on the same core, the precedence constraints can be realized more accurately using the technique of Forget *et al.* [46]. The latter technique is optimal to realize dataflow on single-core by means of a priority ordering. For this reason, we use the technique of Forget *et al.* [46] to realize the dataflow when priority ordering is possible. Nonetheless, the technique of Forget et al. [46] requires a non polynomial graph unfolding [80] to manage cyclic dependencies on single-core. We rather introduce a cycle breaking technique that uses temporal isolation to manage these cyclic dependencies efficiently.

In addition, we propose an ILP formulation to adjust the offset and deadline of tasks to realize the dataflow. We extend the ILP to perform a mapping that minimizes the impact of dataflow requirements on task parameters. The latter builds an initial partial mapping optimized for dataflow requirements on the entire application. This initial mapping is intended to initialize heuristic mapping approaches and to ease the search of a mapping that takes precedence constraints into account. Then, we define a mapping algorithm that takes dataflow requirements into account during the mapping process. The latter is introduced to analyze the impact of the initialization optimized for dataflow requirements on the overall mapping and scheduling process.

Furthermore, we define a fast and accurate characterization of dataflow requirements in multi-periodic systems on partitioned multi-core. We show that the initial mapping optimized for dataflow requirements improves the scheduling performances by an average of 11.92 percent more tasks, while it reduces the computation time of the mapping and scheduling process. As such, this chapter sets a global implementation flow from Matlab/Simulink to the real-time implementation analysis on AUTOSAR multi-core.

The rest of this chapter is organized as follows. Section 6.2 presents some related works on the mapping and scheduling on multi-core and many-core. In Section 6.3, we detail our method to guarantee the dataflow and the cycle breaking technique. In Section 6.4, we formulate the ILP to perform tasks parameters adjustment in single-core. In Section 6.5, we extend this ILP to multi-core so that it also performs a mapping optimized for dataflow requirements. Then, in Section 6.6, we detail the method to build an initial mapping optimized for dataflow requirements. In Section 6.7, we implement a bin packing worst-fit decreasing algorithm that considers dataflow requirements during its mapping and scheduling process. In Section 6.8, we evaluate the performance of the approach on random dependent tasks sets. Finally, conclusions and future works are given in Section 6.9.

6.2 Related works

The problem of mapping and scheduling real-time applications on multi-core and many-core has been intensively studied in literature. Shin and Peng [100] proposed a branch and bound technique to partition dependent tasks to minimize response times. Ramamritham [93] developed several heuristics for mapping dependent periodic tasks to minimize communication costs. Panić *et al.* [87] used bin packing with a worst-fit decreasing heuristic to map dependent and independent tasks with the objective of load balancing. Carle *et al.* [23] and Puffitsch *et al.* [91] studied the mapping and scheduling of dependent real-time tasks to many and multi-core. They exploited low-level hardware details to provide mapping and scheduling heuristics that allow an efficient use of communication resources [23] and avoid contention [91]. Those approaches differ from ours as we do not consider the costs of communications between cores. We rather focus on constructing a mapping that guarantees a deterministic dataflow and minimizes the impact of the latter in multi-periodic systems. However, our formulation can be used in conjunction with those works to take these impacts into account during their mapping processes.

Puffitsch *et al.* [92] and Perret *et al.* [90] studied constraint programming for the mapping and scheduling of dependent tasks on many and multi-core. They also considered communication scheduling and data mapping. Their approach is fundamentally different from ours because they considered non-preemptive tasks. They used the graph unfolding technique to ensure the precedence constraints and the schedulability of the system. In our work, we consider preemptive scheduling and we use linear expressions to model precedence constraints. As such, we do not use the graph unfolding to construct a mapping that minimizes the impact of dataflow requirements.

Hladik *et al.* [59] proposed a constraint programming formulation with a separate mapping and scheduling. Their algorithm proves the nonexistence of a solution when it cannot find one. They did not consider dependent tasks and they exploited the relative simplicity of the scheduling in that case along with filtering algorithms to speedup the search. In this chapter, we mainly focus on dependent tasks and the deterministic implementation of dataflow. This work proposes an initial mapping to ease the search of a mapping with a feasible scheduling that guarantees a deterministic dataflow.

Buttazzo *et al.* [22] and Wu *et al.* [116] proposed a method to guarantee deterministic dataflow that is similar to ours. They considered dynamic priority scheduling and adapted the offsets and deadlines assignment of Chetto *et al.* [29] to ensure precedence constraints. Then, they proposed several bin packing best-fit heuristic algorithms that use critical paths to perform the mapping. The critical path is the dependency path with the most computation time. Their work is different from ours because they build a feasible mapping that minimizes the number of cores. This work builds an initial mapping optimized for dataflow requirements on a fixed number of cores, to ease the search of a feasible mapping. Nevertheless, we extend their heuristic algorithm to multi-periodic systems to define a mapping algorithm that takes dataflow requirements into account during the mapping process

Monot *et al.* [82] studied the mapping of AUTOSAR applications on multi-core. They get rid of multi-core dataflow requirements by assuming no dependency between tasks on different cores. In fact, they clustered all dependent tasks to obtain a set of independent clustered tasks. Then, these latter are mapped to cores. However, with the increasing complexity of automotive applications, this assumption cannot always be taken. Recently, Wang *et al.* [110] proposed a refined task parameters adjustment to increase the schedulability of a given mapping. Their experiments showed that a mapping can be rejected because of strict precedence constraints between cores. Consequently, the impact of precedence constraints between cores must be taking into account during mapping.

Other works have proposed several ILP formulations for mapping. Tuncali *et al.* [109] proposed an optimal formulation for non-preemptive single-rate systems. They have also implemented several heuristics to improve the efficiency of their approach. The work of Saidi et al. [95] is based on AUTOSAR and it considers the minimization of inter-core communications. Becker *et al.* [9] proposed another formulation to avoid memory access contention. However, they do not address the dataflow.

Several authors proposed meta-heuristics approaches to find good mapping solutions. Faragardi *et al.* [41] used evolutionary algorithms. Wang et al. [111] proposed a Tabou search technique. Feljan *et al.* [43], and Feljan and Carlson [42] considered random and heuristics local searches to find a mapping. However, each of these meta-heuristics requires an initialization that can be provided by our approach.

To the best of our knowledge, this is the first time dataflow requirements on multi-periodic systems in partitioned multi-core are formally characterized and used to initialize mapping algorithms.

6.3 Method to guarantee deterministic dataflow

In this section, we assume that the mapping is known and we describe the principle of our method to guarantee a deterministic dataflow.

First, Subsection 6.3.1 details the techniques to ensure precedence constraints in partitioned static scheduling. Then, in Subsection 6.3.2, we introduce the cycle breaking technique to manage cyclic dependencies efficiently.

6.3.1 Techniques to ensure precedence constraints

In partitioned multi-core, precedence constraints are expressed either between tasks on the same core or between tasks on different cores. The temporal isolation technique of Chapter 5 can be used to realize the dataflow in both case. This is because the temporal isolation adjusts the offset and deadline of tasks, so that productions always happen before consumptions and there is no overlap between production and consumption jobs. More formally, let o_i^* and d_i^* be the adjusted offset and deadline of a task τ_i. In temporal isolation o_i^* and d_i^* verify Equation (6.1). In fact, Equation (6.1a) is simply the definition of a precedence constraint, where $\tau_j[n_j]$ cannot start its execution before the end of the execution interval allocated to $\tau_i[n_i]$. In Equation (6.1b), execution intervals are allocated in regard to the real-time attributes of tasks. That is, the execution interval allocated to any task τ_i cannot exceed the initial interval D_i minus the difference $o_i^* - O_i$.

$$o_j^*[n_j] \geq o_i^*[n_i] + d_i^*, \quad \forall\, \tau_i[n_i] \rightarrow \tau_j[n_j] \tag{6.1a}$$

$$d_i^* \leq D_i - (o_i^* - O_i), \quad \forall\, \tau_i \tag{6.1b}$$

To illustrate the temporal isolation technique, let us consider the system modeled by the SDFG of Figure 6.1. This system is composed of three tasks $\tau_N(O_N = 1, C_N = 5, D_N = 10, T_N = 10)$, $\tau_O(O_O = 0, C_O = 3, D_O = 10, T_O = 10)$ and $\tau_P(O_P = 9, C_P = 2, D_P = 10, T_P = 10)$. The SDFG exhibits a cyclic dependency characterized by the precedence constraints $\tau_N[n] \rightarrow \tau_O[n] \rightarrow \tau_P[n] \rightarrow \tau_N[n+1]$. We assume that τ_N and τ_O are on the same core (*i.e.* core 0) and τ_P is on another one (*i.e.* core 1). We use the temporal isolation to ensure the dependencies between τ_O and τ_P, and between τ_P and τ_N. We guarantee that executions of τ_O and τ_P do not overlap by adjusting the relative deadline of τ_O to $d_O^* = 8$, and the offset and relative deadline of τ_P to $o_P^* = 9$ and $d_P^* = 2$, respectively. Then, precedence constraints $\tau_P[n] \rightarrow \tau_N[n+1]$ are guaranteed with the same parameters. Figure 6.2 illustrates the corresponding execution intervals.

When the communicating tasks are on the same core and in addition producer jobs and consumer jobs overlap, the precedence constraints are realized accurately using the

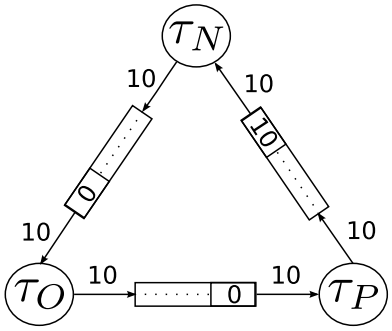


Figure 6.1: Example of system with cyclic dependency.

technique of Forget *et al.* [46]. The latter is the optimal technique in that case. It consists in adjusting tasks offsets so that consumer jobs cannot start before producer ones. The relative deadlines are also adjusted accordingly. Then, the dataflow is ensured by using a higher priority for the producer task. As such, producer jobs are always executed before consumer jobs. In the technique of Forget *et al.* [46] the adjusted offset and relative deadline verify Equation (6.2). In fact, Equation (6.2a) simply defines that $\tau_j[n_j]$ cannot start its execution before $\tau_i[n_i]$. In Equation (6.2b), the relative deadline is adjusted according to the offset. That is, the relative deadline of any task τ_i is equal to the relative deadline D_i minus the difference $o_i^* - O_i$.

$$o_j^*[n_j] \geq o_i^*[n_i], \quad \forall\, \tau_i[n_i] \rightarrow \tau_j[n_j] \tag{6.2a}$$

$$d_i^* = D_i - (o_i^* - O_i), \quad \forall\, \tau_i \tag{6.2b}$$

In the example of Figure 6.1, we illustrate the technique of Forget *et al.* [46] on the precedence constraints $\tau_N[n] \rightarrow \tau_O[n]$ on core 0. We first prevent τ_O from starting before τ_N by adjusting its offset to $o_O^* = \max(o_N^*, o_O^*) = 1$ and its deadline to $d_O^* = 10 - (0 - 1) = 9$. Then, we assign a higher priority to τ_N with respect to τ_O. The corresponding real-time scheduling is illustrated on Figure 6.2. In this scheduling, we can observe that τ_O is always executed before τ_N, so that precedence constraints $\tau_N[n] \rightarrow \tau_O[n]$ are ensured on core 0.

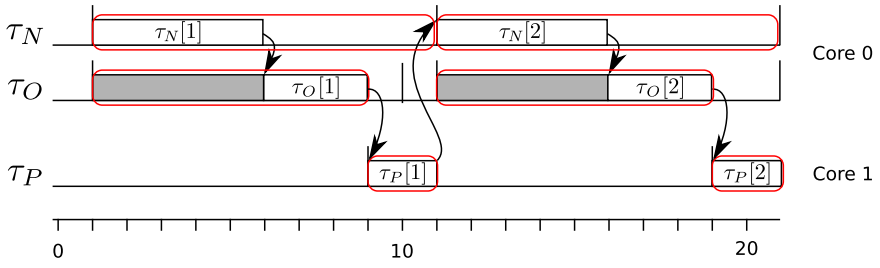


Figure 6.2: Scheduling of the system of Figure 6.1 on two cores.

In summary, we use the priority ordering of the technique of Forget *et al.* [46] to realize the deterministic dataflow when tasks are on the same core and addition producer jobs and

consumer jobs overlap. Otherwise, we use the temporal isolation of Chapter 5 to realize the deterministic dataflow.

6.3.2 Cycle breaking technique

The scheduling of Figure 6.2 succeeds in managing the cyclic dependency because of temporal isolation. When a cyclically dependent tasks set is on the same core, the priority ordering of the technique of Forget *et al.* [46] requires a graph unfolding to choose the execution order. This latter mainly consists in choosing the task that is executed first and whose priority is the highest. Then, the order of the remaining tasks follows the dependency order.

In the example of Figure 6.1, let us consider the real-time attributes $\tau_N(O_N = 0, C_N = 5, D_N = 10, T_N = 10)$, $\tau_O(O_O = 0, C_O = 3, D_O = 10, T_O = 10)$ and $\tau_P(O_P = 9, C_P = 2, D_P = 10, T_P = 10)$. We assume that the tasks are mapped to the same core. Following Equation (6.2), the precedence constraints do not require parameters adjustment. Then, we can verify that the cyclic dependency is managed with the technique of Forget *et al.* [46] only when τ_P is given the highest priority, followed by τ_N and τ_O. That is, only the priority ordering $\pi_N < \pi_O < \pi_P$ produces a feasible scheduling with a deterministic dataflow. This scheduling and the dataflow is illustrated in Figure 6.3.

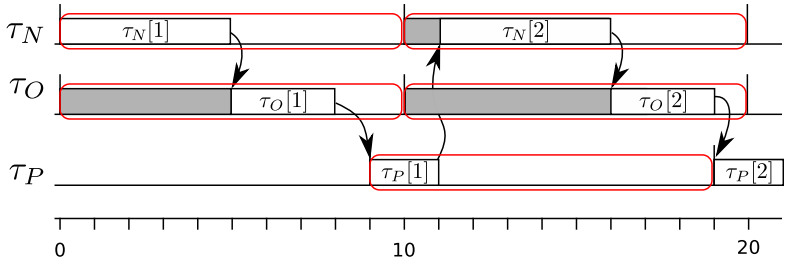


Figure 6.3: Scheduling of the system of Figure 6.1 on single-core using priority ordering

For a large system, with several cyclic dependencies, the graph unfolding is not in polynomial size and suffers from performance issues [80]. Instead, we provide an efficient way to manage cyclic dependencies by introducing a *cycle breaking* technique. The latter consists in setting a temporal isolation for at least one dependency in any cycle. As such, the precedence constraints induced by that dependency is guaranteed de-facto. Others precedence constraints of the cycle are guaranteed by the technique of Forget *et al.* [46] with a priority order starting from the consumer task of the temporal isolated dependency.

Figure 6.4 illustrates the cycle breaking technique on the scheduling of the system of Figure 6.1. In that scheduling, we set a temporal isolation for the dependency $\tau_O \rightarrow \tau_P$ by adjusting $o_P^* = 10$ and $d_P^* = 9$. Precedence constraints $\tau_N[n] \rightarrow \tau_O[n]$ and $\tau_P[n] \rightarrow \tau_N[n+1]$ are ensured by the technique of Forget *et al.* [46] using the priority ordering $\pi_N < \pi_O < \pi_P$. This configuration produces the single-core static preemptive scheduling of Figure 6.4, where all precedence constraints are verified.

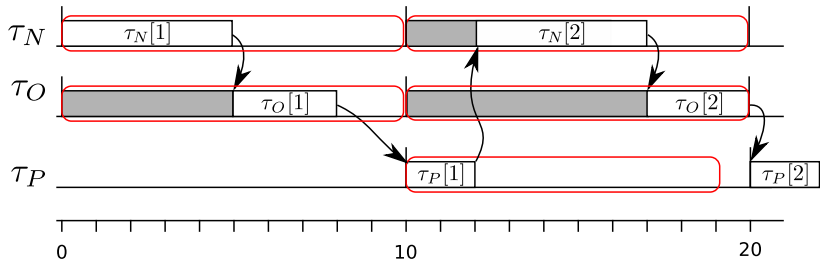


Figure 6.4: Scheduling of the system of Figure 6.1 on single-core using the cycle breaking technique.

The cycle breaking technique provides a fast approach to manage cyclic dependencies in multi-periodic systems. We introduce this technique to build an integrated method to adjust tasks parameters and to enable a deterministic dataflow in partitioned multi-core.

6.4 ILP formulation for parameters adjustment in single-core

In this section, we formulate an ILP to adjust the parameters of tasks to enable a deterministic dataflow on single-core. This ILP does not explicitly consider the real-time scheduling of tasks. It rather minimizes the impact of precedence constraints on tasks parameters. As such, it reduces the impact of dataflow requirements.

First, Subsection 6.4.1 defines the set of variables of the ILP. Then, in Subsection 6.4.2, we describe and characterize the constraints used in the formulation. Finally, in Subsection 6.4.3 we propose an objective function to minimize the impact of dataflow requirements.

6.4.1 Variables of the ILP

Let us consider a set of n dependent tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$.

We define the following set of variables for each task $\tau_i \in \boldsymbol{\mathcal{T}}$.

- o_i^* and d_i^*: are positive variables that characterize the adjusted offset and deadline, respectively.
- γ_i: is a variable associated to each task τ_i. We define this set of variables $\gamma_{i,i=1\ldots n}$ to detect cycles in $\boldsymbol{\mathcal{G}}$.

We also define the following set of variables for every buffer $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$.

- β_{ij}: is a binary variable that indicates whether the dependency is realized using temporal isolation or not. Formally, this is expressed as:

$$\beta_{ij} = \begin{cases} 1, & \text{if temporal isolation} \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

- d^*_{ij}: is a positive variable that corresponds to the value of the execution interval of the producer task. This variable is defined only for linearizion purposes.

6.4.2 Constraints of the ILP

We divide the constraints of the ILP into the following categories.

Real-time constraints

Each task $\tau_i \in \boldsymbol{\mathcal{T}}$ is characterized by both an offset O_i and a relative deadline D_i. The real-time constraints on the adjusted offset o^*_i and the adjusted relative deadline d^*_i of concrete tasks and offset-free tasks are the same as in Equation (5.1) and Equation (5.2), respectively.

Cycle breaking constraints

The cycle breaking technique requires that a temporal isolation is set for at least one buffer of every cyclic dependency. Let us denote by μ a cycle in the SDFG. The cycle breaking technique is formally expressed as follows:

$$\sum_{a=(\tau_i,\tau_j)\in\mu} \beta_{ij} \geq 1, \quad \forall \mu \tag{6.4}$$

Equation (6.4) is implemented in the ILP by the constraint of Equation (6.5), where $m = |\boldsymbol{\mathcal{A}}|$ denotes the number of buffers in $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$.

$$\gamma_j - \gamma_i \geq \frac{1}{m} - \beta_{ij}, \quad \forall a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}} \tag{6.5}$$

In fact, let us consider Theorem 6.1, which provides the necessary and sufficient condition on the existence of the set of variables $\gamma_{i,i=1\ldots n}$. Theorem 6.1 was proved in [33] (c.f. *Theorem* 24.9).

Theorem 6.1 (Existence of $\gamma_{i,i=1\ldots n}$)
Let $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}})$ be a directed graph with n nodes and m edges. Each node τ_i is associated with a variable γ_i and each edge $a = (\tau_i, \tau_j)$ is associated with a value B_{ij}. Let us consider

the system of difference constraints expressed by Equation (6.6). There exists a solution for this system if and only if for every cycle μ in $\boldsymbol{\mathcal{G}}$, $\sum_{a=(\tau_i,\tau_j)\in\mu} B_{ij} \leq 0$.

$$\gamma_j - \gamma_i \geq B_{ij}, \quad \forall a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}} \tag{6.6}$$

Equation (6.5) is obtained by replacing B_{ij} by $\frac{1}{m} - \beta_{ij}$ in Equation (6.6). In fact, according to Theorem 6.1 any feasible solution of the ILP verifies that $\sum_{a=(\tau_i,\tau_j)\in\mu}(\frac{1}{m} - \beta_{ij}) \leq 0$ for all cycle μ in $\boldsymbol{\mathcal{G}}$. This equation is equivalent to $\sum_{a=(\tau_i,\tau_j)\in\mu} \beta_{ij} \geq \sum_{a=(\tau_i,\tau_j)\in\mu} \frac{1}{m}$, which is also equivalent to Equation (6.4) because $0 < \sum_{a=(\tau_i,\tau_j)\in\mu} \frac{1}{m} \leq 1$ and β_{ij} are binary variables.

Precedence constraints

Our method to guarantee the deterministic dataflow uses both temporal isolation technique of Chapter 5 and the scheduling technique of Forget *et al.* [46]. In single core, the temporal isolation realizes precedence constraints only to break cyclic dependency. The technique of Forget *et al.* [46] realizes the precedence constraints in all other cases.

From Chapter 5, we derive that precedence constraints enforced by a buffer $a = (\tau_i, \tau_j)$ is realized by temporal isolation if Equation (6.7) is verified. In fact, this equation is obtained from Equation (5.4) by using variables o_i^*, d_i^* and d_{ij}^* as defined for this ILP. As a result, Equation (6.7) expresses precedence constraints in such a way that consumer jobs can start only after the end of the execution interval of producer jobs they depend on.

$$o_j^* - o_i^* \geq d_{ij}^* + out_a - M_0(a) - gcd_a \tag{6.7}$$

The technique of Forget *et al.* [46] requires two steps to realize precedence constraints. The first step adjusts tasks parameters to prevent consumer jobs from starting before producer jobs. The second step assigns highest priorities to producer tasks. In this ILP, we consider only the parameters adjustment because it characterizes the impact of precedence constraints. Then, we achieve this by removing d_{ij}^* from Equation (6.7). In fact, when $d_{ij}^* = 0$, Equation (6.7) expresses a constraint such as consumer jobs cannot start before producer jobs, which is the required behavior for the technique of Forget *et al.* [46].

Accordingly, we use Equation (6.7) to characterize precedence constraints for both temporal isolation [68] and the technique of Forget et al. [46]. We differentiate between the two cases by using the expression of d_{ij}^* in Equation (6.8). That is, when temporal isolation is required (i.e. $\beta_{ij} = 1$), we get $d_{ij}^* = d_i^*$. Otherwise, $\beta_{ij} = 0$ in Forget et al. [46] case, and we get $d_{ij}^* = 0$.

$$d_{ij}^* = \beta_{ij} \cdot d_i^* \tag{6.8}$$

This expression of d_{ij}^* consists in the product of the continue variable d_i^* and the binary variable β_{ij}. It can be linearized using the constraints of Equation (6.9). In fact, when

$\beta_{ij} = 1$, Equation (6.9) enforces that $d^*_{ij} = d^*_i$. When $\beta_{ij} = 0$, it enforces that $d^*_{ij} = 0$, which is exactly the specification of d^*_{ij}.

$$d^*_{ij} \geq d^*_i + (\beta_{ij} - 1) \cdot D_i \tag{6.9a}$$

$$\beta_{ij} \cdot D_i \geq d^*_{ij} \tag{6.9b}$$

$$d^*_i \geq d^*_{ij} \tag{6.9c}$$

6.4.3 Objective function of the ILP

Parameters adjustment mainly consists in reducing tasks offsets and relative deadlines to guarantee the dataflow. However, shortening deadlines can adversely impact the schedulability [16]. For that reason, the goal of this ILP is to perform parameters adjustment that minimizes this impact.

To achieve this goal, we use the objective function of Equation (6.10), which was proposed in Subsection 5.4.2. This objective function aims at assigning the maximum deadlines to tasks by minimizing the difference between the initial and the adjusted deadlines. The minimization grants a preference to tasks with small slack time (i.e. $D_i - C_i$). That is, the difference is minimized first for the task with the smallest slack time. Then, it is minimized for the next task with the next smallest slack time, and so on.

$$\min \left(\sum_{\boldsymbol{\tau}} \frac{1}{D_i - C_i} (D_i - D^*_i) \right) \tag{6.10}$$

6.5 ILP formulation for parameters adjustment in multi-core and mapping

In this section, we extend the ILP of Section 6.4 to adjust parameters in multi-core and to perform a mapping that minimizes the impact of dataflow requirements. As such, the objective function is the same as in previous formulation.

First, Subsection 6.5.1 defines the set of variables and tuning parameter of the ILP. Then, in Subsection 6.5.2, we describe and characterize the constraints used in the formulation.

6.5.1 Variables and tuning parameter of the ILP

We consider the mapping of a set of n dependent tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ on $q < n$ identical cores. The dependencies between tasks are modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$.

In addition to the variables defined in Subsection 6.4.1, we define the following set of variables for each task $\tau_. \in \boldsymbol{\mathcal{T}}$

- x_{ik}: are q binary variables that fix the mapping of task τ_i to core k. Formally, this is expressed as:

$$x_{ik} = \begin{cases} 1, & \text{if } \tau_i \text{ is mapped to core } k \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

Likewise, in addition to the variables defined in Subsection 6.4.1, we define the following set of variables for each buffer $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$.

- α_{ij}: is a positive variable that determines if two communicating tasks are mapped to the same core. Formally, this is expressed as:

$$\alpha_{ij} = \begin{cases} 1, & \text{if same core} \\ 0, & \text{otherwise} \end{cases} \tag{6.12}$$

Furthermore, we define $U_{\max} \leq 1$ as a tunable constant of the ILP. This constant indicates the maximum utilization per core.

6.5.2 Constraints of the ILP

In this formulation, we define the following constraints in addition to the ones defined in Subsection 6.4.2.

Utilization constraint

Each core has a limited utilization defined by $U_{\max}$. This constraint is expressed as follows.

$$\sum_{i=1}^{n} U_i \cdot x_{ik} \leq U_{\max} \tag{6.13}$$

Mapping constraints

Each task $\tau_i \in \boldsymbol{\mathcal{T}}$ must be mapped to exactly one core. Equation (6.14) expresses this constraint.

$$\sum_{k=0}^{q-1} x_{ik} = 1, \quad \forall \tau_i \in \boldsymbol{\mathcal{T}} \tag{6.14}$$

Colocation constraints

For every buffer $a = (\tau_i, \tau_j) \in \boldsymbol{\mathcal{A}}$, variable α_{ij} indicates if the communicating tasks are mapped to the same core. In Equation (6.15), we use the binary variables x_{ik} and x_{jk} to derive α_{ij}.

$$x_{ik} - x_{jk} + 1 \geq \alpha_{ij} \geq x_{ik} + x_{jk} - 1, \quad \forall a = (\tau_i, \tau_j) \in \mathcal{A}, \forall k < q \tag{6.15}$$

In fact, when τ_i and τ_j are mapped to the same core k, $x_{ik} = x_{jk} = 1$ and we get $\alpha_{ij} = 1$. On the contrary, when they are mapped to different cores, the case $x_{ik} = 0$ and $x_{jk} = 1$ appears. The latter leads to the constraint $\alpha_{ij} \leq 0$. However, α_{ij} is a positive variable, which implies that $\alpha_{ij} = 0$. Moreover, for all other cases such as $x_{ik} = x_{jk} = 0$, we get the constraints $1 \geq \alpha_{ij} \geq -1$. These constraints are trivial and do not influence the value of α_{ij}. Then, from Equation (6.15), α_{ij} can be declared as a real variable.

Furthermore, temporal isolation is required to realize precedence constraints between cores. This is enforced by Equation (6.16). In fact, this equation enforces $\beta_{ij} = 1$ (*i.e.* temporal isolation) when $\alpha_{ij} = 0$ (*i.e.* tasks are on different cores). However, when $\alpha_{ij} = 1$, Equation (6.16) does not enforce the value of β_{ij}. In that case, the value of β_{ij} is either enforced by the cycle breaking constraints or managed by the objective function.

$$\beta_{ij} \geq 1 - \alpha_{ij}, \quad \forall a \in \boldsymbol{\mathcal{A}} \tag{6.16}$$

6.6 Initial mapping optimized for dataflow requirements

In this section, we propose a method to construct an initial feasible or partial mapping optimized for dataflow requirements. The partial mapping is composed of a subset of tasks that are mapped and scheduled. The initial mapping is intended to initialize mapping algorithms, to ease the search of a mapping that takes precedence constraints into account.

Our method builds a mapping optimized for dataflow requirements by using the ILP of Section 6.5. Since this ILP minimizes the impact of precedence constraints on tasks, we obtain a mapping optimized for dataflow requirements on the entire application. However, as this mapping does not consider the real-time scheduling of tasks, it can lead to a scheduling that is not feasible. As such, we verify the scheduling on each core using the algorithm of Forget *et al.* [46]. Then, we derive the initial mapping from the mapping of tasks that are scheduled.

Let us consider the static preemptive scheduling of a set of n dependent periodic tasks $\boldsymbol{\mathcal{T}} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ on partitioned multi-core with q identical cores ($q < n$). Dependencies between tasks are modeled by the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$. Let us denote by $\boldsymbol{\mathcal{T}}_k$ the set of tasks allocated to core k. Algorithm 2 describes the method to construct the initial

mapping optimized for dataflow requirements.

Algorithm 2 Algorithm to construct an initial mapping.

```
1: function MAPPING_INITIALIZATION(G)
2:     {T_k, k = 1, ..., q} = ILP_MAPPING(G)
3:     for k = 1, ..., q do
4:         SCHEDULE(T_k)
5:         for all τ_i ∈ T_k do
6:             if τ_i is not scheduled then
7:                 T_k ← T_k \ {τ_i}
8:             end if
9:         end for
10:    end for
11:    return {T_k, k = 1, ..., q}
12: end function
```

The originality of Algorithm 2 is the function ILP_MAPPING(), which computes a mapping optimized for dataflow requirements using the ILP of Section 6.5. Function SCHEDULE() computes the scheduling on each core using the algorithm of Forget *et al.* [46]. This algorithm assigns priorities from lowest to highest. It ensures that producer tasks are given highest priorities than consumer tasks. The initial mapping is derived from the mapping of tasks that have a feasible priority.

6.7 Heuristic mapping with precedence constraints

In this section, we specify a mapping algorithm that takes dataflow requirements into account during the mapping and scheduling process. This algorithm will serve to analyze the impact of the initial mapping optimized for dataflow requirements.

First, in Subsection 6.7.1 we define a variant of the ILP of Section 6.5 to perform a fast parameters adjustment for a given mapping. Then, in Subsection 6.7.2 we present a bin packing algorithm with worst-fit decreasing heuristic that takes dataflow requirements into account during the search.

6.7.1 Fast parameters adjustment

To ensure dataflow requirements for a given mapping, we define a variant of the ILP of Section 6.5. This variant removes the following variables and constraints from the formulation of Section 6.5:

- binary variables x_{ik} and α_{ij} which are not needed when the mapping is known,
- mapping and colocation constraints which do no apply to fixed mapping.

Accordingly, only real-time constraints, cycle breaking constraints and precedence constraints are used. The objective function remains the same, so that adjusted parameters minimize the impact of dataflow requirements.

This variant is interesting as it provides a fast and efficient approach to set temporal isolation and parameters adjustment for the algorithm of Forget et al. [46]. As such, it guarantees the dataflow during the evaluation of the scheduling after each mapping decision.

6.7.2 Heuristic mapping algorithm

The mapping algorithm that we suggest is based on bin packing worst-fit decreasing heuristic. It extends the algorithm of Buttazzo *et al.* [22] to multi-periodic systems. The algorithm proceeds by selecting tasks in decreasing order of utilization. Then, at each step of the search, it assigns the unmapped task with the highest utilization to one core using worst-fit heuristic. However, a task is assigned to a core only if the scheduling of this core is still feasible after the task is added. When the current task cannot be assigned to any core, the algorithm moves to the next task with the next highest utilization. This process is repeated until all tasks are mapped or no task can be mapped.

The specificity of this mapping approach, compared to standard bin packing worst-fit decreasing, is the use of the ILP variant. In fact, the latter is invoked at each mapping decision so that the real-time parameters are adjusted before checking the scheduling. As such, the impact of dataflow requirements is taken into account throughout the mapping process. In addition, we ensure that the mapping of the current task does not affect the scheduling of tasks that have already been mapped to other cores. This is done by enforcing temporal isolation for all dependencies between tasks that are mapped and tasks that are not. Hence, if the current task is mapped, it does not affect temporal isolation with tasks on other cores. Consequently, their scheduling is not affected.

Let $\boldsymbol{\mathcal{T}}'$ be the set of tasks that are not mapped. Algorithm 3 gives the pseudo code of the mapping method. Function Sorted_by_Decreasing_utilization() returns the set of tasks ordered by decreasing utilizations. Function Ilp_Adjustment() uses the ILP variant to build parameters adjustment for both techniques [68, 46]. Then, the function Schedule() tests the scheduling on a given core using the algorithm of Forget et al. [46]. The latter returns true only if the mapping is feasible. Otherwise, it returns false.

6.8 Experiments and performance measurements

In this section, we evaluate the initial mapping on random dependent tasks sets.

First, in Subsection 6.8.1 we explain the generation of random dependent tasks sets for multi-core. Then, in Subsection 6.8.2 we evaluate the scheduling and the runtime

Algorithm 3 Heuristic mapping algorithm

```
1: function HEURISTIC_WF_MAPPING(G)
2:     {T_k, k = 0...q-1} ← INITIAL_MAPPING(T)
3:     T' ← SORTED_BY_DECREASING_UTILIZATION(T)
4:     unscheduled ← φ                                   ▷ Unscheduled tasks

5:     for τ_i ∈ T' \ unscheduled do
6:         k ← least occupied core
7:         repeat
8:             T_k ← T_k ∪ {τ_i}
9:             T ← ILP_ADJUSTMENT(T)
10:            feasible ← SCHEDULE(T_k)
11:            if feasible then
12:                T' ← T' \ {τ_i}
13:            else
14:                T_k ← T_k \ {τ_i}
15:                k ← next least occupied core
16:            end if
17:        until feasible = true or no core remains
18:        if no core remains then
19:            unscheduled ← unscheduled ∪ {τ_i}
20:        end if
21:    end for

22: end function
```

performance of the initial mapping optimized for dataflow requirements.

6.8.1 Dependent tasks sets generation in multi-core

In the experiments of this section, we use the method of Section 5.5 to generate dependent tasks. However, we change the approach for generating the utilization of tasks. This change consists in choosing the total processor utilization U randomly between 1 and the number of cores. Then, we distribute the latter uniformly over tasks using the *RandFixedSum* function of Emberson *et al.* [38].

We do not use *UUniFast* method [18] in multi-core because when the total utilization is greater than 1, *UUniFast* generates task utilizations greater than 1. These tasks are indeed not schedulable. Davis and Burns [35] developed and extension of *UUniFast* called *UUnifast-Discard*. The latter consists in using *UUniFast* and discarding task sets that contain tasks with utilizations greater than 1. However, this algorithm is inefficient when the value of U approaches $\frac{n}{2}$ [38]. Though this is not our case, we rather use *RandFixed-Sum* [38]. The latter is an efficient method to distribute the processor utilization uniformly over tasks in multi-core.

6.8.2 Scheduling performance and runtime

For the following analysis, we introduce an initial mapping optimized for load balancing. The latter is obtained using the same approach as in Section 6.6. That is, we replace the ILP by a standard bin packing worst-fit decreasing algorithm that builds an initial load balanced mapping. We suggest this initial mapping optimized for load balancing in order to illustrate the interest of the initial mapping optimized for dataflow requirements. This choice is justified by the fact that most mapping algorithms consider load balancing with several mapping concerns such as jitter minimization [111], inter-core communication minimization [41, 111, 43, 42] and parallelization [87]) in their mapping and scheduling process. Then, in each experiment, we run Algorithm 3 both without initialization (labeled "H"), with the initialization optimized for load balancing (labeled "lb-H") and with the initialization optimized for dataflow requirements (labeled "df-H'').

In the experiments in Figure 6.5 we consider a mapping and scheduling on $q = 4$ cores. We vary the system utilization from 1.5 to 3.5 in abscissa and we generate a set of 5 graphs for each utilization. Each graph is composed of 50 tasks with a degree from 1 to 3. We limit the maximum utilization per core to $U_{\max} = 1$. Figure 6.5 represents the average percentage of tasks that are scheduled for each initialization.

We observe in Figure 6.5 that initialization with the mapping optimized for dataflow requirements produces better scheduling results. Algorithm 3 without initialization schedules on average 75.2 of tasks. With the initialization optimized for dataflow requirements (*resp.* for load balancing) the average increases to 87.12 percent (*resp.* 76.1 percent) of tasks.

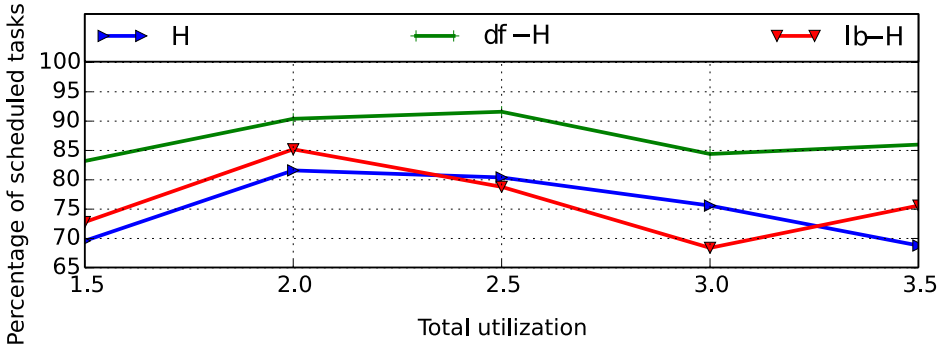


Figure 6.5: Scheduling performance

That is, the initialization optimized for dataflow requirements increases the scheduling of Algorithm 3 by an average of 11.92 percent more tasks, while the initialization optimized for load balancing increases it only by an average of 0.9 percent more tasks. This result confirms the impact of dataflow requirements on the feasibility of mapping. It also shows the interest of taking dataflow requirements into account during the mapping of dependent tasks. Furthermore, it motivates the need for a good initial mapping optimized for dataflow requirements.

Figure 6.6 uses the same experiments as before, except that we measure the computation times of the whole mapping process (i.e. initialization + Algorithm 3). Then, we observe that initializing Algorithm 3 with the mapping optimized for dataflow requirements does not increase the overall computing time. On the contrary, the latter is reduced. This result may seem surprising because the initialization optimized for load balancing requires less computing time than the one optimized for dataflow requirements (Figure 6.7). However, the latter produces a partial mapping with an average of 57.44 percent of tasks, while the former contains only 34.64 percent of tasks (Figure 6.8). As such, the computing time of Algorithm 3 is mostly reduced with the initial mapping optimized for dataflow requirements. This is because the latter maps more tasks and faster than the required incremental scheduling analysis [5], which is in exponential time.

However, the runtime reduction is possible only if the resolution of the ILP is faster than the incremental scheduling analysis. In experiments of Figure 6.9, we study the time needed to get the optimal solution of the ILP. We find that this time increases with both the number of tasks and the number of cores. Nonetheless, the magnitudes are not uniform but depend on the structure of graphs (Figure 6.9(a)).

The experiments in Figure 6.9(b) show that the density of graphs (*i.e.* the connectivity of tasks) influences greatly on the complexity of finding the optimal solution of the ILP. In fact, in Figure 6.9(b) we set the number of tasks to 100 and the number of cores to 4. We vary the maximum degree of tasks from 2 to 6 in abscissa and we generate a set of 5 graphs for each. We represent the average time needed to find the optimal solution in ordinate. The results show that the complexity of finding the optimal solution increases exponentially with the density of graphs.

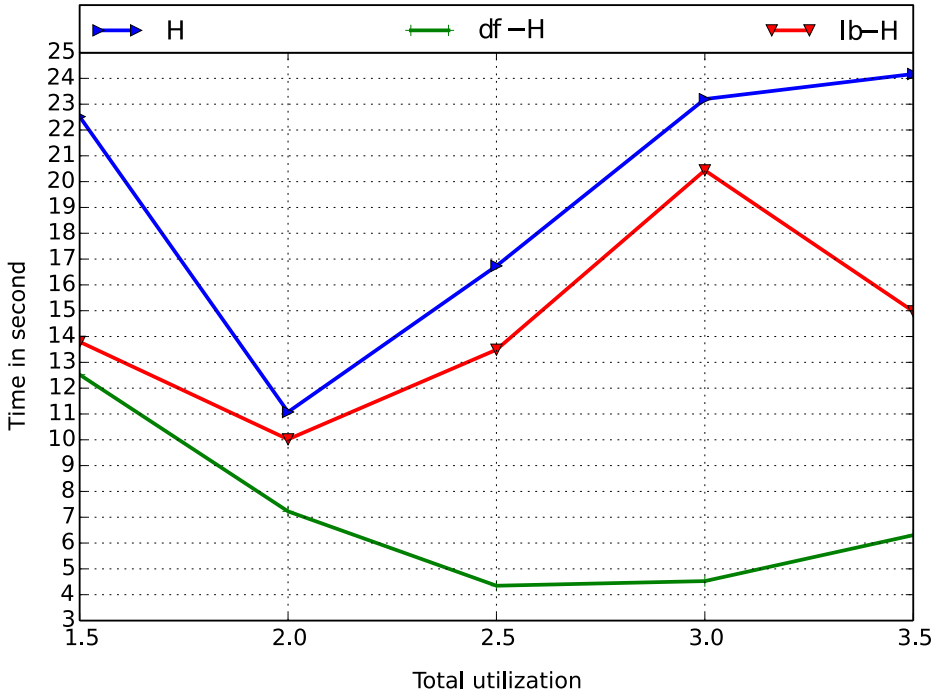


Figure 6.6: Runtime analyses.

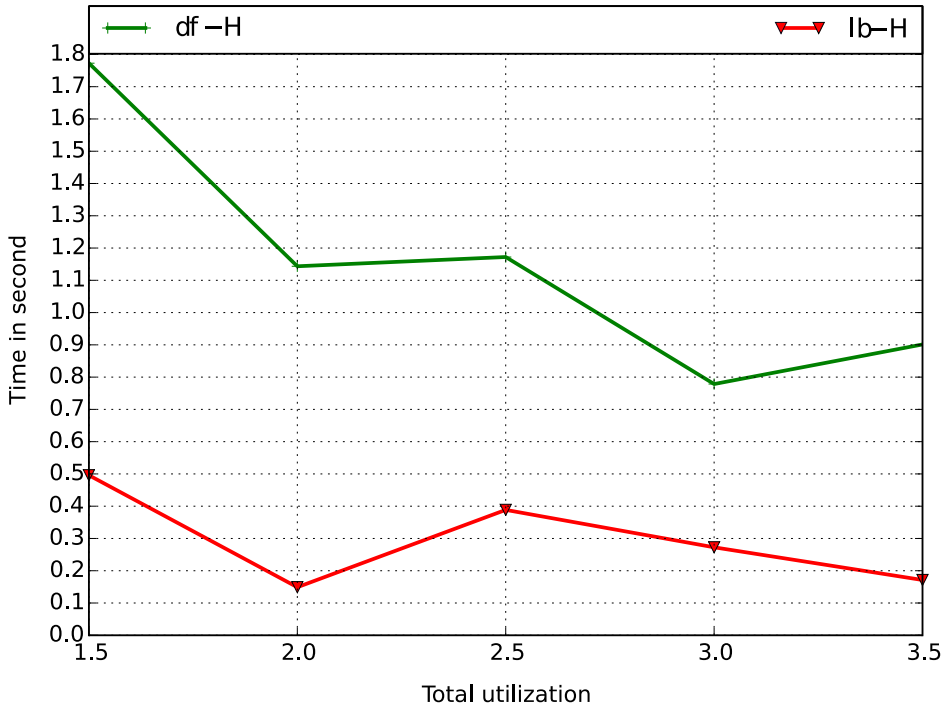


Figure 6.7: Computing time of each initial partial mapping

The size of our ILP is polynomial. In fact, Equations (6.3) and (6.11) define m and $n \cdot q$ binary variables, respectively. Since $m = \mathcal{O}(n^2)$, the number of binary variables is in $\mathcal{O}(n^2)$. In addition, Equations (6.13), (6.14), (6.15) and (6.16) define q, n, $q \times n^2$ and n^2 constraints on the binary variables. That is, the number of constraints on binary variables is in $\mathcal{O}(q \times n^2)$. As such, the size of the ILP is $\mathcal{O}(n^2)$. The exponential time resolution

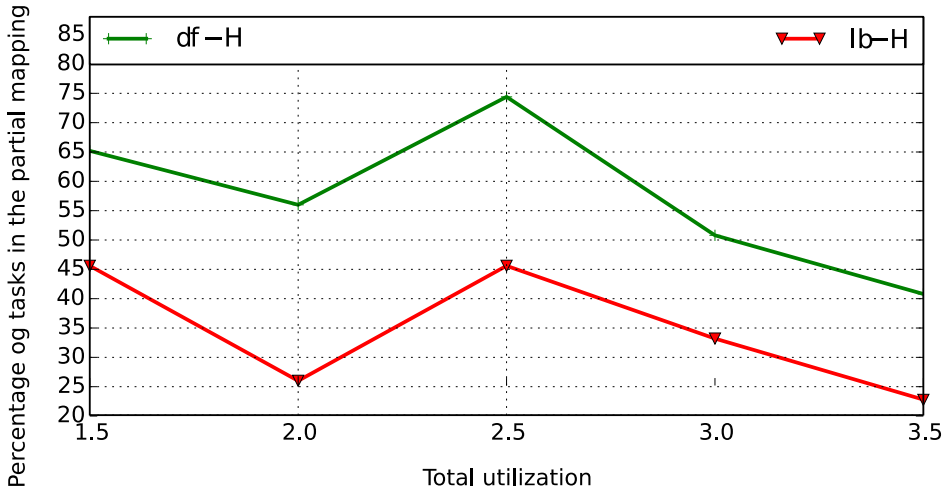


Figure 6.8: Percentage of tasks in each initial partial mapping

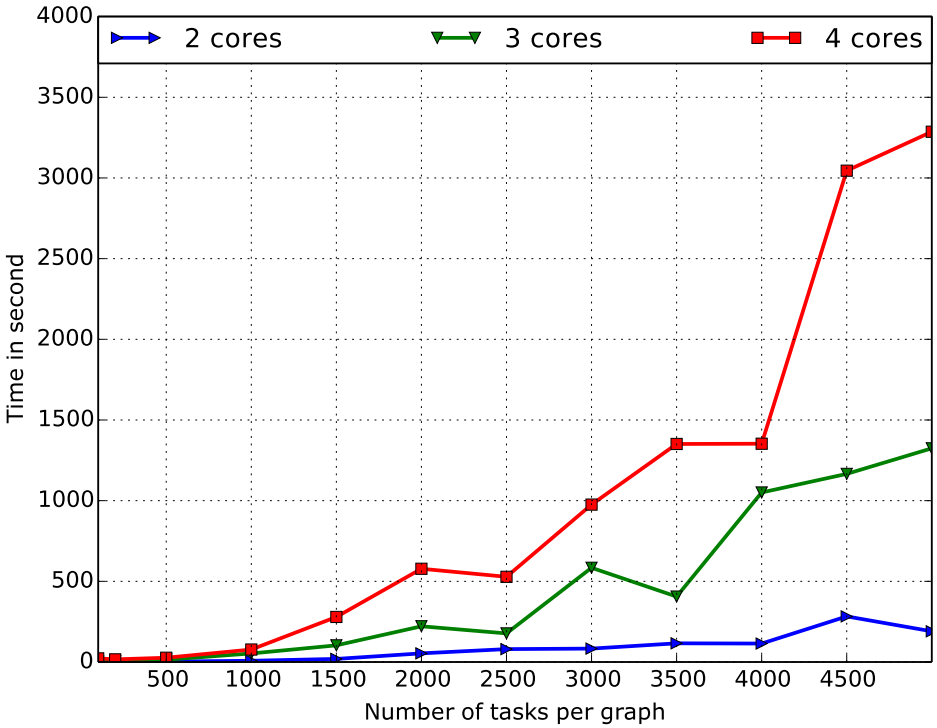
observed when the density increases in Figure 6.9 is due to the combinatorial explosion of the number of possible states for the values of the variables. In fact, when the density increases, a single mapping decision affects all the adjacent buffers.

6.9 Conclusion

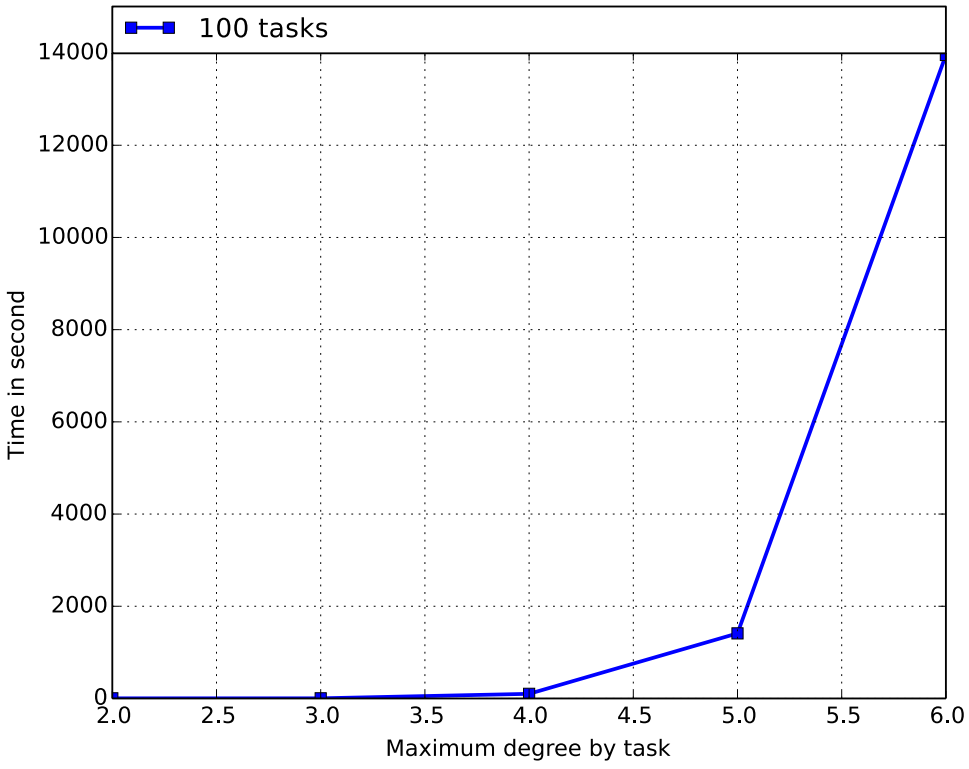
In this chapter, we considered the mapping and scheduling problem of multi-periodic systems on a fixed number of identical cores. We addressed this problem from a dataflow perspective to propose a fast and accurate characterization of dataflow requirements in partitioned multi-core. To do this, we introduced a method that guarantees a deterministic dataflow by combining the temporal isolation technique developed in Chapter 5 with the technique of Forget *et al.* [46]. The latter provides the optimal technique to realize deterministic dataflow on single-core using priority ordering. Then, we introduced a cycle breaking technique to manage cyclic dependencies in multi-periodic systems efficiently. As such, we built an integrated method to adjust tasks parameters and enable a deterministic dataflow in partitioned multi-core.

Furthermore, we formulated an ILP that adjusts the parameters of tasks to implement deterministic dataflow while minimizing the impact of the latter. We also extended this ILP to perform a mapping optimized for dataflow requirements. This ILP builds an initial partial or feasible mapping optimized for dataflow requirements. The latter initializes mapping algorithms and ease the search of a mapping that takes precedence constraints into account.

This chapter demonstrates the interest of taking dataflow requirements into account during the mapping and scheduling. The proposed approach constructs an initial mapping optimized for dataflow requirements, which increases the scheduling performances and can reduce the computation time of the mapping and scheduling process.



(a) Effect of the number of tasks and cores.



(b) Effect of the density of graphs.

Figure 6.9: Computing time to find the optimal solution of the ILP.

This work allows the establishment of an implementation flow from Matlab/Simulink to the mapping and the scheduling analysis on partitioned AUTOSAR. An interesting

perspective consists in combining the initial mapping with other mapping algorithms in order to build a complete implementation flow. The latter will consider both dataflow guaranteeing and optimization concerns such as jitter [111] and inter-core communication minimization [41, 111, 43, 42], and parallelization [87]. In next chapter we use this result to provide a design framework from Matlab/Simulink to AUTOSAR multi-core.

Chapter 7

Design framework from Simulink to AUTOSAR

Contents

7.1 Introduction

This chapter uses the contributions of previous chapters to bridge the gap between Matlab/Simulink and the configuration of AUTOSAR multi-core. We achieve this by defining a design framework that provides a formal link from Matlab/Simulink to AUTOSAR. As such, our contributions are downstream of Matlab/Simulink, but upstream of the mapping and scheduling on AUTOSAR multi-core. In fact, we extract the dataflow in the Matlab/Simulink specification by a SDFG using the method of Chapter 4. Then, we characterize the dataflow requirements to implement deterministic dataflow on AUTOSAR multi-core using the methods of Chapters 5 and 6. However, we leave the configuration step to the designer so that it can choose the configuration techniques best suited to its system. This is because the configuration of a given application in the industry is subjected to subjective preferences. Nonetheless, we assist this configuration by providing an initial mapping optimized for dataflow requirements. The latter is an excellent entry point for the configuration because it improves the scheduling performances, while it can reduce the computing time of the configuration process. Furthermore, we specify an *XML Schema Definition* and several tools to automate the design framework. We apply this framework on the use case FCCS.

The rest of this chapter is organized as follows. Section 7.2 describes our framework using the results of previous chapters. In Section 7.3, we describe an exchange format and the tools implemented to automate the configuration process. Then, in Section 7.4, we illustrate the design framework on a fine grain model of the FCCS. Discussion and conclusions are given in Section 7.5.

7.2 Framework description

Our framework is composed of several steps as illustrated in Figure 7.1.

The first step consists in designing and validating the functional specification in Matlab/Simulink. The latter must verify the rules defined in Section 4.4 to allow the formal characterization of dataflow requirements. Namely, every Runnable must be modeled by a SDFG task and communications between each pair of Runnables must be characterized by an initial marking. Once these rules are verified, the next step extracts the SDFG of the validated Matlab/Simulink specification. This SDFG specifies the dataflow that must be implemented. As such, it provides the formal link between Matlab/Simulink and AUTOSAR.

The characterization of the dataflow requirements provided by Section 6.3 expresses the constraints on the AUTOSAR configuration to ensure a deterministic dataflow on partitioned multi-core. However, industrial applications are also constrained by real-time specifications and several requirements such as jitter [111], response time [100] and inter-core communication [41, 111, 43, 42] minimization, data mapping [110] and parallelism [87].

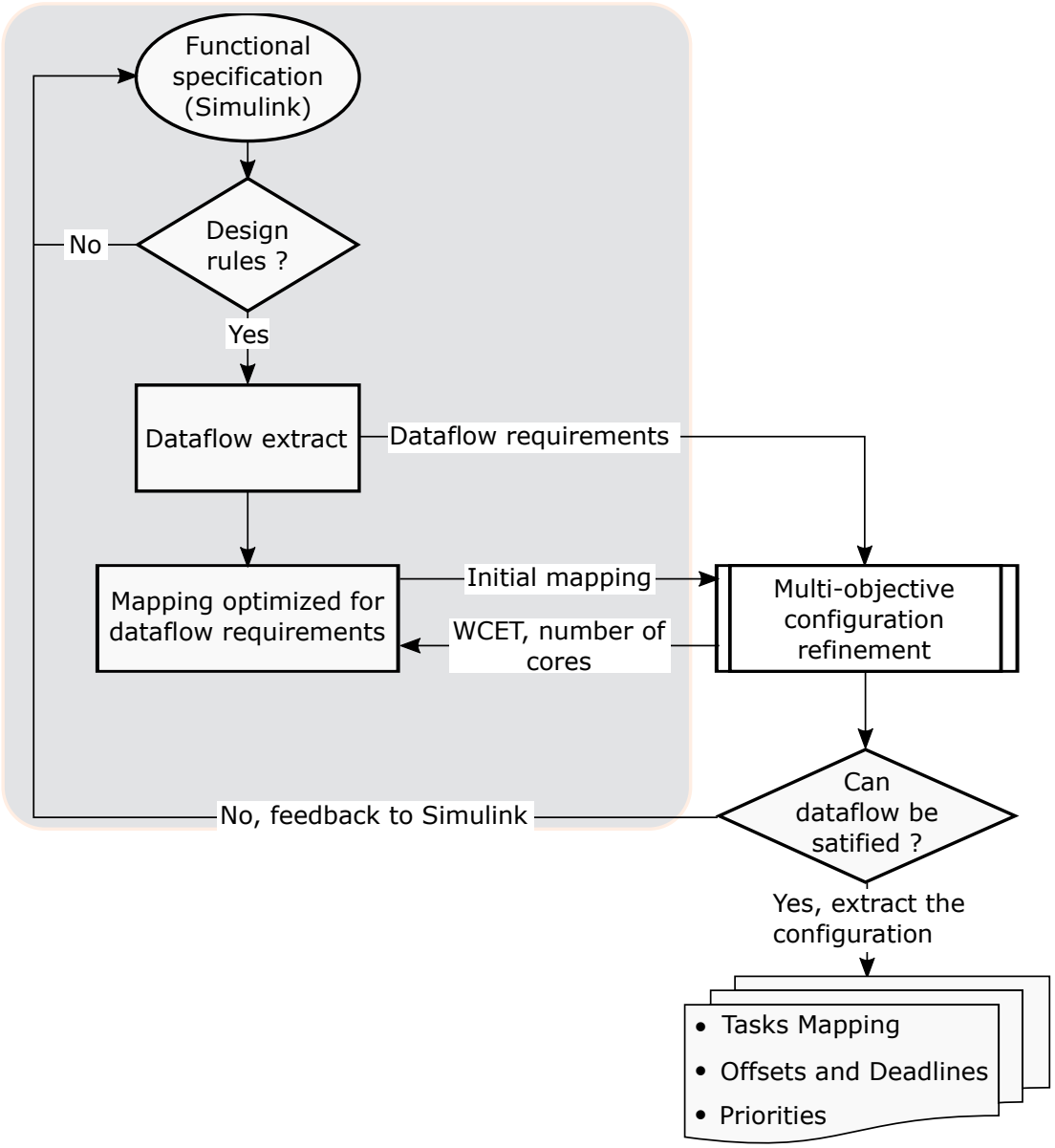


Figure 7.1: Proposed design framework.

To meet all these requirements, the AUTOSAR configuration actually relies on a *Multi-Objective Optimization* (MOO) problem. The latter uses more than one criteria to be optimized simultaneously. However, the difficulty in a MOO problem is the presence of conflicting objectives. For example, the minimization of inter-core communications may cause overloading of cores and increase response times. For such nontrivial MOO problem, no single solution exists that simultaneously optimizes each objective. In fact, there exists a (possibly infinite) number of *Pareto* optimal solutions. A solution is Pareto optimal if none of the objectives can be improved in value without degrading some of the others. As such, all Pareto optimal solutions are considered equally good.

In the industry, the "best" configuration for an application is subjected to subjective preferences. For this reason, we leave the configuration step to the designer to choose the

appropriate configuration techniques. However, we define a step to assist this configuration on AUTOSAR multi-core. This assistance is to perform an initial mapping optimized for dataflow requirements using the extracted SDFG, the WCET of Runnables and the number of cores. This initial mapping optimized for dataflow requirements assists the configuration because it improves the scheduling performances, while it can reduce the computing time of the process. As such, it is an excellent entry point for the configuration to reduce the number of design iterations and shortens the design time.

After the configuration step, if some dataflow requirements cannot be met, the information is feedback to adapt the Matlab/Simulink specification. Otherwise, the configuration (*i.e.* the tasks mapping, the adjusted offset and deadline of tasks, and the priorities of tasks) is extracted.

7.3 Exchange format and tool suite

We specify an *XML Schema Definition* (XSD) to facilitate the exchanges between the steps of the framework. In fact, this XSD defines the elements and the structures to describe the SDFG (*i.e.* SDFG tasks, buffers, initial markings), the hardware constraints (*i.e.* number of cores, maximum utilization) and the WCET of Runnables. Then, we develop the *MAtlab SDFG Extraction* (MASE) tool in outsourcing by Sherpa Engineering. MASE is a Matlab script that checks the design rules on a Matlab/Simulink specification and automatically extracts the SDFG. It provides this SDFG in the XML format specified by the XSD. Moreover, MASE includes additional information on SDFG tasks (*e.g.* Runnables names, SWCs) and SDFG Buffers (*e.g.* data types). These information are required by downstream tools, for example to constraint the mapping of Runnables of the same SWCs to the same core.

Furthermore, we develop a python script to retrieve this XML file and perform the mapping optimized for dataflow requirements. The rest of the tools developed in the thesis are python scripts to facilitate interoperability. To date, the output format of the configuration is a CSV file describing the mapping, the offsets, the deadlines and the priorities in the initial mapping.

The tools mentioned above will be available on the website of the ELA project[1].

7.4 Illustration on the Fuel Cell Control System

We illustrate our framework on a fine grain Matlab/Simulink specification of the FCCS. The latter verifies the rules defined in Section 4.4. This specification is composed of 24 Runnables at 10ms and 9 Runnables at 100ms, for a total of 33 Runnables spread

[1] `http://www.irt-systemx.fr/project/ela`

over 12 SWCs. We extract the SDFG $\boldsymbol{\mathcal{G}} = (\boldsymbol{\mathcal{T}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$ of this specification using MASE, where Runnables are modeled by SDFG tasks. We assume that each Runnable can be implemented by an offset-free task. However, we do not have the real estimation of the WCET. As such, we generate random WCET using *RandFixedSum* [38]. Then, we consider the architecture of the micro-controller TC29x [1] presented in Section 2.6, with $q = 3$ cores.

For the mapping, we ensure that Runnables of the same SWC are mapped to the same core (as required by AUTOSAR) by associating the binary variables x_{ik} (*c.f.* Equation 6.11) to SWCs (*i.e.* $i = 1 \dots 12$ and $j = 0 \dots 2$). In fact, these variables fix the mapping of Runnables to cores. As a result, Runnables of the same SWC are characterized by the same mapping variables x_{ik}, so that they are mapped to the same core. Then, we perform the mapping optimized for dataflow requirements using the ILP of Section 6.5.

For the configuration, we just target a feasible mapping. That is, we use the mapping optimized for dataflow requirements directly. We check the feasibility using the method of Section 6.3, where priority ordering realizes dependency only when temporal isolation is not used. Then, if the mapping is feasible, we extract the configuration. Otherwise, we identify the dataflow requirements associated to tasks that are not scheduled.

In the following experiments, we vary the utilization U of the FCCS from 2 to 2.5 in abscissa. We generate a set of 10 random Runnables parameters for each utilization. We implement a load balancing with 5 percent margin by constraining core utilization to $U_{\max} = \frac{U}{3} \cdot 1.05$. In Figure 7.2, we show the percentage of feasible mapping obtained using different real-time scenarios for each system utilization (labeled "$Iter - 1$"). We call this percentage the *acceptance ratio*. Moreover, to illustrate feedback to the functional specification when the configuration is not feasible, we soften the dataflow requirements associated to tasks that are not scheduled. This softening consists in enforcing the communications from theses tasks to use the delayed mechanism. Then, we repeat the mapping and the scheduling analysis once. The acceptance ratio after that iteration is also shown in Figure 7.2 (labeled "$Iter - 2$").

We observe in Figure 7.2 that softening dataflow requirements improves the acceptance ratio for the FCCS. Despite the fact that this approach is naive, it illustrates that dataflow requirements can be used to adapt the functional specification and to enhance the implementation.

For experiments that produce feasible scheduling, one step remains before the implementation on the ECU. This step consists in the implementation of the communications between tasks. In fact, our work characterizes the dataflow and guarantee the deterministic implementation using scheduling approach. However, the AUTOSAR RTOS does not directly provide features for the data exchanges mechanisms. That is, it remains to adapt the communication in AUTOSAR to implement the mechanisms of Matlab/Simulink.

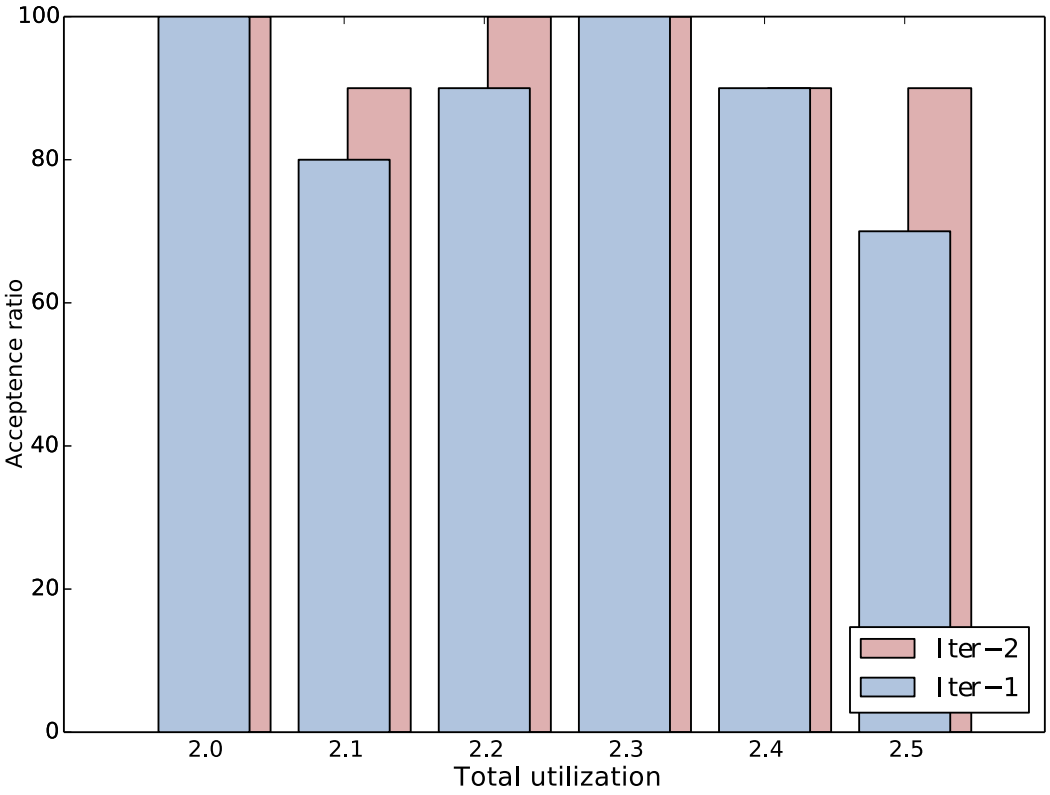


Figure 7.2: Acceptance ratio.

7.5 Discussion and Conclusion

This chapter established a design framework to bridge the gap between Matlab/Simulink and the implementation on AUTOSAR. That, is it provides a formal link between Matlab/Simulink and the implementation analysis on AUTOSAR multi-core. In fact, the dataflow of Matlab/Simulink is extracted and modeled by a SDFG using the method developed in Chapter 4. Then, the method proposed in Chapter 6 characterizes the dataflow requirements to implement deterministic dataflow on AUTOSAR multi-core. However, since in the industry the configuration of an application is subjected to subjective preference, we leave the configuration to the designer to choose the appropriate techniques. Nonetheless, we assist this configuration by providing an initial mapping optimized for dataflow requirements. The latter is an excellent entry point for the configuration to reduce the number of design iterations and shortens the design time. If some dataflow requirements cannot be met by the configuration, the information on the dataflow requirements that are not verified are feedback to the Matlab/Simulink specification.

Furthermore, we specified a XSD and several tools to automate the design framework. To this end, we assume that Runnables are implemented as real-time tasks. However, this is not a limitation because this hypothesis is used only to study the mapping and the deterministic dataflow guarantee in partitioned multi-core. As such, Runnables to tasks mapping can be performed later during the configuration [64, 16].

Chapter 8

Conclusion

Summary

This thesis studied the deterministic implementation of data exchanges in AUTOSAR multi-core. We considered a MBD approach, where the dataflow is specified by Matlab/Simulink. So, we relied on the compositionality offers by AUTOSAR to focus only on the interactions between Runnables. Then, we targeted the implementation which guarantees the functional and temporal determinism of data exchanges between Runnables, while making the best use of parallelism.

Our work proves that the dataflow in a multi-periodic Matlab/Simulink system is modeled by a SDFG. We characterized this SDFG and we provided the method to model the dataflow in a Matlab/Simulink specification by a SDFG. This SDFG is not on the hyperperiod and all dependency constraints are expressed by closed mathematical formulas. As such, the translation of the Matlab/Simulink specification to SDFG is scalable. Moreover, the formal equivalence proven between Matlab/Simulink and SDFG opens new perspectives for the development of real-time systems on multi/many-core. This is because the formalism of SDFG is very popular in the literature and it is widely studied for the deployment of dataflow applications on multi/many-core.

We exploited the theory of SDFG to build a real-time scheduling technique that guarantees the implementation of the functional dataflow without blocking mechanisms. This technique uses the scheduling of the SDFG to transform the dependent tasks into independent ones by adjusting their real-time attributes. The proposed temporal isolation is performed using a LP. As such, it provides a fast technique to deal with dependencies in multi-periodic systems.

Furthermore, we combined the technique of temporal isolation with the technique of Forget *et al.* [46] to build a global method that guarantees deterministic dataflow in partitioned multi-core. In fact, the technique of Forget *et al.* [46] is optimal to realize deterministic dataflow on single-core. As a result, the combination of both technique provides a fast and accurate characterization of dataflow requirements in partitioned multi-core. Moreover, we provided a method to build an initial mapping optimized for dataflow requirements. The experiments prove that this initial mapping increases the scheduling performances and can reduce the computation time of the mapping and scheduling process.

Finally, we proposed a design framework which uses SDFG to bridge the gap between Matlab/Simulink and the configuration of AUTOSAR multi-core. This configuration is constrained by the dataflow requirements characterized using our method to guarantee deterministic dataflow. Then, the mapping optimized for dataflow requirements is provided to initialize multi-objective optimization methods to promote the construction of a partitioning and a feasible scheduling. We claim that this reduces the number of design iterations and shortens the design time.

Perspectives

Below are discussed prospects for future work in the continuity of this thesis.

Combination of communication mechanisms

The Direct, Delayed and Hybrid mechanisms are the basic communication mechanisms in Matlab/Simulink. In fact, it is possible to combine these basis mechanism to create new ones. The resulting mechanisms of some combinations are obvious. For example, the combination of two delay blocks leads to a mechanism where the data transfer is delayed by two periods of the producer. However, for other combinations, the resulting mechanisms are not so obvious and may even depend on the order of the combination. An example is the combination of delay and hybrid mechanisms. Future work could investigate the combination of communication mechanisms in Matlab/Simulink. Then, provide the SDFG model of the latter.

Implementation strategies

Our characterization of dataflow requirements defines the conditions to for the deterministic implementation of data exchanges between Runnables. However, the implementation must ensure that only the proper data are consumed. As such, following our work, the choice of the suitable communication modes and tasks configuration in AUTOSAR should be considered.

Communication costs

In multi/many-core systems, communications between cores are actually costly. In fact, because of the bus and/or the NoC sharing, inter-core communications take longer. As a result, the dataflow requirements associated with inter-core communications have an even greater impact on tasks. This aspect should be taken into account effectively for the characterization of dataflow requirements and in the initial mapping.

Publications

International conference with reading committee

- Enagnon Cedric Klikpo and Alix Munier-Kordon. "Characterization of dataflow requirements in partitioned multi-core." In: *Submitted* ()
- Enagnon Cedric Klikpo, Jad Khatib, and Alix Munier-Kordon. "Modeling multi-periodic simulink systems by synchronous dataflow graphs." In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*. IEEE. 2016, pp. 1–10
- Enagnon Cédric Klikpo and Alix Munier-Kordon. "Preemptive scheduling of dependent periodic tasks modeled by synchronous dataflow graphs." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 77–86
- Jad Khatib et al. "Computing latency of a real-time system modeled by Synchronous Dataflow Graph." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 87–96

National Workshop

- Enagnon Cedric Klikpo, Jad Khatib, and Alix Munier-Kordon. "Modélisation SDF des communications Simulink." In: *École d'été Temps Réel (ETR'2015)* (20015)
- Enagnon Cedric Klikpo et al. "Modélisation des communications d'un système temps-réel par un SDFG." in: *École d'été Temps Réel (ETR'2015)* (20015)

Bibliography

[1] Infineon Technologies AG. *Highly integrated and performance optimized 32-bit microcontrollers for automotive and industrial applications.* URL: `http://www.infineon.com/` (cit. on pp. 22, 23, 111).

[2] *AMALTHEA project homepage.* URL: `http://amalthea-project.org/` (cit. on p. 32).

[3] Pascal Aubry, Paul Le Guernic, and Sylvain Machard. "Synchronous distribution of Signal programs." In: *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,* vol. 1. IEEE. 1996, pp. 656–665 (cit. on p. 30).

[4] Pascal Aubry et al. "Extended Cyclostatic Dataflow Program Compilation and Execution for an Integrated Manycore Processor." In: *Proceedings of the International Conference on Computational Science.* 2013 (cit. on p. 33).

[5] Neil C Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.* Citeseer, 1991 (cit. on pp. 9, 11, 12, 69, 76, 78, 102).

[6] *AUTOSAR Support in MATLAB and Simulink - Automotive Industry Standards - MATLAB & Simulink.* URL: `https://fr.mathworks.com/solutions/automotive/standards/autosar.html` (cit. on pp. 47, 51).

[7] Mohamed Bamakhrama and Todor Stefanov. "Hard-real-time scheduling of data-dependent tasks in embedded streaming applications." In: *Proceedings of the ninth ACM international conference on Embedded software.* ACM. 2011, pp. 195–204 (cit. on pp. 33, 69).

[8] Mohamed A Bamakhrama and Todor Stefanov. "Managing latency in embedded streaming applications under hard-real-time scheduling." In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis.* ACM. 2012, pp. 83–92 (cit. on pp. 33, 69).

[9] Matthias Becker et al. "Contention-free execution of automotive applications on a clustered many-core platform." In: *28th Euromicro Conference on Real-Time Systems (ECRTS).* IEEE. 2016, pp. 14–24 (cit. on p. 88).

[10] Shane Bell et al. "Tile64-processor: A 64-core soc with mesh interconnect." In: *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International.* IEEE. 2008, pp. 88–598 (cit. on p. 22).

[11] A. Benveniste et al. "The synchronous languages 12 years later." In: *Proceedings of the IEEE* 91.1 (2003-01) (cit. on p. 30).

[12] Albert Benveniste and Gérard Berry. "The Synchronous Approach to Reactive and Real-time Systems." In: *Readings in Hardware/Software Co-design.* Ed. by Giovanni De Micheli, Rolf Ernst, and Wayne Wolf. Norwell, MA, USA: Kluwer Academic Publishers, 2002, pp. 147–159 (cit. on pp. 29, 42).

[13] Albert Benveniste, Paul Le Guernic, and Christian Jacquemot. "Synchronous Programming with Events and Relations: The SIGNAL Language and Its Semantics." In: *Sci. Comput. Program.* 16.2 (Sept. 1991) (cit. on pp. 29, 30, 42).

[14] Albert Benveniste et al. "A protocol for loosely time-triggered architectures." In: *Embedded Software*. Springer. 2002, pp. 252–265 (cit. on p. 42).

[15] Gérard Berry and Georges Gonthier. "The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation." In: *Sci. Comput. Program.* 19.2 (Nov. 1992) (cit. on pp. 29, 30, 42).

[16] Antoine Bertout, Julien Forget, and Richard Olejnik. "Minimizing a real-time task set through Task Clustering." In: *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*. ACM. 2014, p. 23 (cit. on pp. 95, 112).

[17] Greet Bilsen et al. "Cycle-static dataflow." In: *IEEE Transactions on signal processing* 44.2 (1996), pp. 397–408 (cit. on pp. 33, 69).

[18] Enrico Bini and Giorgio C Buttazzo. "Biasing effects in schedulability measures." In: *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. IEEE. 2004, pp. 196–203 (cit. on pp. 78, 101).

[19] Bruno Bodin et al. "Fast and efficient dataflow graph generation." In: *Proceedings of the 17th International Workshop on Software and Compilers for Embedded Systems*. ACM. 2014, pp. 40–49 (cit. on p. 78).

[20] Pontus Boström and Jonatan Wiik. "Contract-based verification of discrete-time multi-rate Simulink models." In: *Software & Systems Modeling* 15.4 (2016), pp. 1141–1161 (cit. on p. 44).

[21] Michael W. Browne et al. "Simulink user's guide." In: *Sage Focus Editions* 154 (1993) (cit. on pp. 1, 26, 42, 47, 50).

[22] Giorgio Buttazzo, Enrico Bini, and Yifan Wu. "Partitioning real-time applications over multicore reservations." In: *Transactions on Industrial Informatics* 7.2 (2011), pp. 302–315 (cit. on pp. 86, 88, 99).

[23] Thomas Carle et al. "Static mapping of real-time applications onto massively parallel processor arrays." In: *14th International Conference on Application of Concurrency to System Design (ACSD)*. IEEE. 2014, pp. 112–121 (cit. on pp. 86, 87).

[24] Paul Caspi et al. "From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications." In: *ACM Sigplan Notices*. Vol. 38. 7. ACM. 2003 (cit. on pp. 30, 42).

[25] Paul Caspi et al. "LUSTRE: a declarative language for real-time programming." In: *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1987 (cit. on pp. 29, 30, 42).

[26] Paul Caspi et al. "Semantics-preserving multitask implementation of synchronous programs." In: *ACM Transactions on Embedded Computing Systems (TECS)* 7.2 (2008), p. 15 (cit. on p. 30).

[27] Damien Chabrol et al. "Deterministic Distributed Safety-Critical Real-Time Systems within the Oasis Approach." In: *IASTED PDCS*. 2005, pp. 260–268 (cit. on p. 31).

[28] Dominik Chessa. "Conception and Implementation of Parallelism Analyses in MATLAB/SIMULINK Models for programming Embedded Multicore-Systems." In: *Bachelorarbeit, Technische Universität München* (2011) (cit. on p. 44).

[29] Houssine Chetto, Maryline Silly, and T Bouchentouf. "Dynamic scheduling of real-time tasks under precedence constraints." In: *Real-Time Systems* 2.3 (1990), pp. 181–194 (cit. on pp. 69, 88).

[30] Edward G Coffman Jr, Michael R Garey, and David S Johnson. "Approximation algorithms for bin packing: A survey." In: *Approximation algorithms for NP-hard problems*. PWS Publishing Co. 1996, pp. 46–93 (cit. on p. 12).

[31] Frederic Commoner et al. "Marked directed graphs." In: *Journal of Computer and System Sciences* 5.5 (1971), pp. 511–523 (cit. on p. 43).

[32] AUTOSAR Consortium et al. "AUTOSAR Release 4.0." In: *Specification of multicore OS architecture v1* (2009) (cit. on pp. 1, 13).

[33] Thomas H.. Cormen et al. *Introduction to algorithms*. Vol. 6. MIT press Cambridge, 2001 (cit. on p. 93).

[34] Robert I Davis and Alan Burns. "A survey of hard real-time scheduling for multiprocessor systems." In: *Computing surveys (CSUR)* 43.4 (2011), p. 35 (cit. on p. 12).

[35] Robert I Davis and Alan Burns. "Priority assignment for global fixed priority preemptive scheduling in multiprocessor real-time systems." In: *2009 30th IEEE Real-Time Systems Symposium (RTSS)*. IEEE. 2009, pp. 398–409 (cit. on p. 101).

[36] Sudarshan K Dhall and Chung Laung Liu. "On a real-time scheduling problem." In: *Operations research* 26.1 (1978), pp. 127–140 (cit. on p. 12).

[37] Benoît Dupont de Dinechin et al. "A distributed run-time environment for the kalray mppa®-256 integrated manycore processor." In: *Procedia Computer Science* 18 (2013) (cit. on pp. 22, 33).

[38] Paul Emberson, Roger Stafford, and Robert I Davis. "Techniques for the synthesis of multiprocessor task sets." In: *Proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010)*. 2010, pp. 6–11 (cit. on pp. 101, 111).

[39] *EU CO2 standards for passenger cars and light-commercial vehicles | International Council on Clean Transportation*. URL: `http://www.theicct.org/publications/eu-co2-standards-passenger-cars-and-light-commercial-vehicles` (visited on 11/06/2017) (cit. on p. 1).

[40] Maher Fakih and Sebastian Warsitz. "Automatic SDF-based Code Generation from Simulink Models for Embedded Software Development." In: *arXiv preprint arXiv:1701.04217* (2017) (cit. on p. 44).

[41] Hamid Reza Faragardi et al. “An efficient scheduling of AUTOSAR runnables to minimize communication cost in multi-core systems.” In: *7th International Symposium on Telecommunications (IST)*. IEEE. 2014, pp. 41–48 (cit. on pp. 86, 88, 101, 106, 108).

[42] Juraj Feljan and Jan Carlson. “Task allocation optimization for multicore embedded systems.” In: *40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2014, pp. 237–244 (cit. on pp. 86, 88, 101, 106, 108).

[43] Juraj Feljan, Jan Carlson, and Tiberiu Seceleanu. “Towards a model-based approach for allocating tasks to multicore processors.” In: *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2012, pp. 117–124 (cit. on pp. 86, 88, 101, 106, 108).

[44] Julien Forget et al. “A multi-periodic synchronous data-flow language.” In: *High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE*. IEEE. 2008, pp. 251–260 (cit. on p. 31).

[45] Julien Forget et al. “Dynamic priority scheduling of periodic tasks with extended precedences.” In: *16th IEEE Conference on Emerging Technologies & Factory Automation, 2011*. IEEE. 2011 (cit. on pp. 31, 43).

[46] Julien Forget et al. “Scheduling dependent periodic tasks without synchronization mechanisms.” In: *2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE. 2010, pp. 301–310 (cit. on pp. 4, 31, 69, 80, 81, 86, 90, 91, 94, 97–99, 104, 113).

[47] Alain Girault. “A survey of automatic distribution method for synchronous programs.” In: *International workshop on synchronous languages, applications and programs, SLAP*. Vol. 5. 2005 (cit. on p. 30).

[48] Alain Girault and Xavier Nicollin. “Clock-driven automatic distribution of Lustre programs.” In: *Embedded Software*. Springer. 2003, pp. 206–222 (cit. on p. 30).

[49] Alain Girault, Xavier Nicollin, and Marc Pouzet. “Automatic rate desynchronization of embedded reactive programs.” In: *ACM Transactions on Embedded Computing Systems (TECS)* 5.3 (2006), pp. 687–717 (cit. on p. 30).

[50] J. Goossens and R. Devillers. “The Non-Optimality of the Monotonic Priority Assignments for Hard Real-Time Offset Free Systems.” In: *Real-Time Systems* 13.2 (1997), pp. 107–126 (cit. on p. 69).

[51] Joël Goossens. “Scheduling of Offset Free Systems.” In: *Real-Time Systems* 24.2 (2003), pp. 239–258 (cit. on pp. 8, 68, 69).

[52] Michael I Gordon et al. “A stream compiler for communication-exposed architectures.” In: *ACM SIGPLAN Notices*. Vol. 37. 10. ACM. 2002 (cit. on pp. 2, 33).

[53] Ronald L. Graham. “Bounds on multiprocessing timing anomalies.” In: *SIAM journal on Applied Mathematics* 17.2 (1969), pp. 416–429 (cit. on p. 30).

[54] Thierry Grandpierre and Yves Sorel. "From algorithm and architecture specifications to automatic generation of distributed real-time executives: a seamless flow of graphs transformations." In: *Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2003. MEMOCODE'03.* IEEE. 2003, pp. 123–132 (cit. on p. 30).

[55] Mathieu Grenier, Joël Goossens, and Nicolas Navet. "Near-optimal fixed priority preemptive scheduling of offset free systems." In: *14th International Conference on Real-Time and Networks Systems (RTNS'06).* 2006, pp. 35–42 (cit. on p. 69).

[56] Kaouther Guesmi and Salem Hasnaoui. "Translating of MATLAB/SIMULINLK model to synchronous dataflow graph for parallelism analysis and programming embedded multicore systems." In: *9th International Design & Test Symposium, 2014.* IEEE. 2014, pp. 156–161 (cit. on p. 44).

[57] Joost PHM Hausmans et al. "Dataflow analysis for multiprocessor systems with non-starvation-free schedulers." In: *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems.* ACM. 2013, pp. 13–22 (cit. on pp. 70, 82).

[58] Thomas Henzinger, Benjamin Horowitz, and Christoph Kirsch. "Giotto: A time-triggered language for embedded programming." In: *Embedded software.* Springer. 2001, pp. 166–184 (cit. on p. 31).

[59] Pierre-Emmanuel Hladik et al. "Solving a real-time allocation problem with constraint programming." In: *Journal of Systems and Software* 81.1 (2008), pp. 132–149 (cit. on p. 88).

[60] Robert Höttger, Lukas Krawczyk, and Burkhard Igel. "Model-Based Automotive Partitioning and Mapping for Embedded Multicore Systems." In: *ICPDSSE 2015: International Conference on Parallel, Distributed Systems and Software Engineering, Istanbul, Turkey, (Jan 26-27, 2015)* 1.1 (2014), p. 957 (cit. on p. 43).

[61] Ralf Jahr, Mike Gerdes, and Theo Ungerer. "On Efficient and Effective Model-based Parallelization of Hard Real-Time Applications." In: *MBEES.* 2013, pp. 50–59 (cit. on p. 32).

[62] Mathai Joseph and Paritosh Pandya. "Finding response times in a real-time system." In: *The Computer Journal* 29.5 (1986), pp. 390–395 (cit. on p. 11).

[63] Jad Khatib et al. "Computing latency of a real-time system modeled by Synchronous Dataflow Graph." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems.* ACM. 2016, pp. 87–96 (cit. on p. 115).

[64] Fouad Khenfri, Khaled Chaaban, and Maryline Chetto. "A novel heuristic algorithm for mapping AUTOSAR runnables to tasks." In: *Pervasive and Embedded Computing and Communication Systems (PECCS), 2015 International Conference on.* IEEE. 2015, pp. 1–8 (cit. on p. 112).

[65] Enagnon Cedric Klikpo, Jad Khatib, and Alix Munier-Kordon. "Modeling multi-periodic simulink systems by synchronous dataflow graphs." In: *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*. IEEE. 2016, pp. 1–10 (cit. on p. 115).

[66] Enagnon Cedric Klikpo, Jad Khatib, and Alix Munier-Kordon. "Modélisation SDF des communications Simulink." In: *École d'été Temps Réel (ETR'2015)* (20015) (cit. on p. 115).

[67] Enagnon Cedric Klikpo and Alix Munier-Kordon. "Characterization of dataflow requirements in partitioned multi-core." In: *Submitted* () (cit. on p. 115).

[68] Enagnon Cédric Klikpo and Alix Munier-Kordon. "Preemptive scheduling of dependent periodic tasks modeled by synchronous dataflow graphs." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 77–86 (cit. on pp. 94, 99, 115).

[69] Enagnon Cedric Klikpo et al. "Modélisation des communications d'un système temps-réel par un SDFG." In: *École d'été Temps Réel (ETR'2015)* (20015) (cit. on p. 115).

[70] Hermann Kopetz and Günther Bauer. "The time-triggered architecture." In: *Proceedings of the IEEE* 91.1 (2003) (cit. on p. 42).

[71] Philip S Kurtin, Joost PHM Hausmans, and Marco JG Bekooij. "Combining offsets with precedence constraints to improve temporal analysis of cyclic real-time streaming applications." In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2016, pp. 1–12 (cit. on pp. 70, 82).

[72] Christophe Lavarenne et al. "The SynDEx software environment for real-time distributed systems design and implementation." In: *European Control Conference*. Vol. 2. 1991, pp. 1684–1689 (cit. on p. 30).

[73] E.A. Lee and D.G. Messerschmitt. "Synchronous data flow." In: *Proceedings of the IEEE* 75.9 (1987-09) (cit. on pp. 26, 28, 33, 39, 43, 58).

[74] Youen Lesparre. "Evaluation de l'affectation des tâches sur une architecture à mémoire distribuée pour des modèles flot de données." PhD thesis. Paris, UPMC, 2017 (cit. on p. 35).

[75] Joseph Y-T Leung and Jennifer Whitehead. "On the complexity of fixed-priority scheduling of periodic, real-time tasks." In: *Performance evaluation* 2.4 (1982), pp. 237–250 (cit. on pp. 9, 11, 69).

[76] Chung Laung Liu and James W Layland. "Scheduling algorithms for multiprogramming in a hard-real-time environment." In: *Journal of the ACM (JACM)* 20.1 (1973), pp. 46–61 (cit. on pp. 9–11, 69).

[77] Stéphane Louise et al. "OASIS project: deterministic real-time for safety critical embedded systems." In: *Proceedings of the 10th workshop on ACM SIGOPS European workshop*. ACM. 2002, pp. 223–226 (cit. on p. 31).

[78] Stephane Louise et al. "The oasis kernel: A framework for high dependability real-time systems." In: *High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium on*. IEEE. 2011, pp. 95–103 (cit. on p. 31).

[79] Olivier Marchetti and Alix Munier-Kordon. "A sufficient condition for the liveness of weighted event graphs." In: *European Journal of Operational Research* 197.2 (2009), pp. 532–540 (cit. on pp. 33, 35, 69).

[80] Olivier Marchetti and Alix Munier-Kordon. "Cyclic Scheduling for the Synthesis of Embedded Systems." In: *Introduction to scheduling* (2009), pp. 135–164 (cit. on pp. 33, 36, 38, 43, 69, 86, 91).

[81] *MICROSAR - AUTOSAR Basic Software*. URL: `https://vector.com/vi_microsar_en.html` (cit. on p. 23).

[82] Aurélien Monot et al. "Multicore scheduling in automotive ECUs." In: *Embedded Real Time Software and Systems-ERTSS 2010*. 2010 (cit. on pp. 86, 88).

[83] A. Munier-Kordon. *Régime asymptotique optimal d'un graphe d'évènement temporisé généralisé: application à un problème d'assemblage*. Vol. 25. 5. RAIRO-Automatique Productique Informatique Industrielle, 1993 (cit. on pp. 33, 69).

[84] Yingfeng Oh and Sang H Son. "Tight performance bounds of heuristics for a real-time scheduling problem." In: *Submitted for Publication* (1993) (cit. on p. 12).

[85] Claire Pagetti et al. "Multi-task implementation of multi-periodic synchronous programs." In: *Discrete Event Dynamic Systems* 21.3 (2011), pp. 307–338 (cit. on pp. 30, 31, 43).

[86] Claire Pagetti et al. "The ROSACE case study: From simulink specification to multi/many-core execution." In: *2014 24th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2014, pp. 309–318 (cit. on p. 43).

[87] Miloš Panić et al. "Runpar: An allocation algorithm for automotive applications exploiting runnable parallelism in multicores." In: *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*. ACM. 2014, p. 29 (cit. on pp. 86, 87, 101, 106, 108).

[88] *parMERASA | Multi-Core Execution of Parallelised Hard Real-Time Applications Supporting Analysability*. URL: `https://www.parmerasa.eu/` (cit. on p. 32).

[89] Maxime Pelcat et al. "An open framework for rapid prototyping of signal processing applications." In: *EURASIP journal on embedded systems* 2009 (2009) (cit. on pp. 2, 33, 66).

[90] Quentin Perret et al. "Mapping hard real-time applications on many-core processors." In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM. 2016, pp. 235–244 (cit. on pp. 86, 87).

[91] Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. "Mapping a multi-rate synchronous language to a many-core processor." In: *19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2013, pp. 293–302 (cit. on pp. 86, 87).

[92] Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. "Off-line mapping of multi-rate dependent task sets to many-core platforms." In: *Real-Time Systems* 51.5 (2015), pp. 526–565 (cit. on pp. 86, 87).

[93] Krithi Ramamritham. "Allocation and scheduling of precedence-related periodic tasks." In: *IEEE Transactions on Parallel and Distributed Systems* 6.4 (1995), pp. 412–420 (cit. on pp. 86, 87).

[94] Pascal Richard, Francis Cottet, and Michaël Richard. "On-line scheduling of real-time distributed computers with complex communication constraints." In: *Proceedings. Seventh IEEE International Conference on Engineering of Complex Computer Systems, 2001*. IEEE. 2001, pp. 26–34 (cit. on pp. 43, 61).

[95] Salah Eddine Saidi et al. "An ILP approach for mapping autosar runnables on multi-core architectures." In: *Proceedings of the Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. ACM. 2015, p. 6 (cit. on pp. 86, 88).

[96] Stuijk Sander, Geilen Marc, and Basten Twan. "SDF^ 3: SDF For Free." In: vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006 (cit. on pp. 2, 33, 66).

[97] Vivek Sarkar. *Partitioning and scheduling parallel programs for execution on multiprocessors*. Tech. rep. Stanford Univ., CA (USA), 1987 (cit. on p. 86).

[98] *SCC External Architecture Specification (EAS)*. Many-core Applications Research Community. URL: `https://communities.intel.com/docs/DOC-5044` (cit. on p. 22).

[99] Sebastien Dupertuis. *Rate Transition Block (RTB)*. July 2014. URL: `http://www.idsc.ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Embedded-Control-Systems/RTB.pdf` (cit. on p. 50).

[100] Kang G Shin and Dar-Tzen Peng. *Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-time Systems*. International Computer Science Institute, 1988 (cit. on pp. 86, 87, 108).

[101] Amit Kumar Singh et al. "Mapping on multi/many-core systems: survey of current and emerging trends." In: *Proceedings of the 50th Annual Design Automation Conference*. ACM. 2013, p. 1 (cit. on p. 12).

[102] Christos Sofronis, Stavros Tripakis, and Paul Caspi. "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling." In: *Proceedings of the 6th ACM & IEEE International conference on Embedded software*. ACM. 2006, pp. 21–33 (cit. on p. 30).

[103] Yves Sorel. "Massively parallel computing systems with real time constraints: the" Algorithm Architecture Adequation" methodology." In: *Proceedings of the First International Conference on Massively Parallel Computing Systems, 1994*. IEEE. 1994, pp. 44–53 (cit. on p. 30).

[104] Oana Stan et al. "A GRASP metaheuristic for the robust mapping and routing of dataflow process networks on manycore architectures." In: *4OR* (2015) (cit. on p. 62).

[105] John A. Stankovic. "Misconceptions about real-time computing: A serious problem for next-generation systems." In: *Computer* 21.10 (1988), pp. 10–19 (cit. on p. 6).

[106] John A Stankovic and Krithi Ramamritham. *Hard real-time systems*. [sn], 1988 (cit. on p. 6).

[107] Project MAC (Massachusetts Institute of Technology). Engineering Robotics Group and ML Dertouzos. *Control robotics: The procedural control of physical processes*. 1973 (cit. on p. 10).

[108] Stavros Tripakis et al. "Implementing synchronous models on loosely time triggered architectures." In: *IEEE Transactions on Computers* 57.10 (2008) (cit. on pp. 42, 43).

[109] Cumhur Erkan Tuncali, Georgios Fainekos, and Yann-Hang Lee. "Automatic parallelization of Simulink models for multi-core architectures." In: *12th International Conferen on Embedded Software and Systems (ICESS)*. IEEE. 2015, pp. 964–971 (cit. on p. 88).

[110] Wenhao Wang, Fabrice Camut, and Benoît Miramond. "Generation of schedule tables on multi-core systems for AUTOSAR applications." In: *Design and Architectures for Signal and Image Processing (DASIP), 2016 Conference on*. IEEE. 2016, pp. 191–198 (cit. on pp. 86, 88, 108).

[111] Wenhao Wang et al. "Optimizing Application Distribution on Multi-Core Systems within AUTOSAR." In: *8th European Congress on Embedded Real Time Software and Systems (ERTS)*. 2016 (cit. on pp. 86, 88, 101, 106, 108).

[112] Reinhard Wilhelm and Jan Reineke. "Embedded systems: Many cores—Many problems." In: *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*. IEEE. 2012, pp. 176–180 (cit. on p. 23).

[113] Reinhard Wilhelm et al. "The worst-case execution-time problem—overview of methods and survey of tools." In: *ACM Transactions on Embedded Computing Systems (TECS)* 7.3 (2008), p. 36 (cit. on p. 16).

[114] Philip S Wilmanns et al. "Buffer sizing to reduce interference and increase throughput of real-time stream processing applications." In: *2015 IEEE 18th International Symposium on Real-Time Distributed Computing*. IEEE. 2015, pp. 9–18 (cit. on pp. 70, 82).

[115] Carsten Wolff et al. "AMALTHEA—Tailoring tools to projects in automotive software development." In: *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on*. Vol. 2. IEEE. 2015, pp. 515–520 (cit. on p. 32).

[116] Yifan Wu, Zhigang Gao, and Guojun Dai. "Deadline and activation time assignment for partitioned real-time application on multiprocessor reservations." In: *Journal of Systems Architecture* 60.3 (2014), pp. 247–257 (cit. on pp. 86, 88).

[117] Haibo Zeng and M. Di Natale. “Mechanisms for guaranteeing data consistency and flow preservation in AUTOSAR software on multi-core platforms.” In: 2011 6th IEEE International Symposium on Industrial Embedded Systems (SIES). 2011-06, pp. 140–149 (cit. on pp. 30, 43).