Fast and Accurate Performance Models for Probabilistic Timing Analysis of SDFGs on MPSoCs

Hai-Dang Vu

▶ **To cite this version:**

Hai-Dang Vu. Fast and Accurate Performance Models for Probabilistic Timing Analysis of SDFGs on MPSoCs. Electronics. Université de Nantes, 2021. English. NNT : . tel-03171880

HAL Id: tel-03171880
https://hal.science/tel-03171880

Submitted on 17 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

THESE DE DOCTORAT DE

L'UNIVERSITE DE NANTES
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Électronique

Par
Hai Dang VU

Fast and Accurate Performance Models for Probabilistic Timing Analysis of SDFGs on MPSoCs

Thèse présentée et soutenue à Nantes, le 04/03/2021
Unité de recherche : IETR UMR CNRS 6164

Rapporteurs avant soutenance :

Liliana CUCU-GROSJEAN	Chargée de Recherche/HDR, INRIA, Paris
François VERDIER	Professeur, Université Côte d'Azur

Composition du Jury :

Présidente	Laurence PIERRE	Professeur, Université Grenoble Alpes
Examinateurs :	Liliana CUCU-GROSJEAN	Chargée de Recherche/HDR, INRIA, Paris
	François VERDIER	Professeur, Université Côte d'Azur
Dir. de thèse :	Sébastien PILLEMENT	Professeur, Université de Nantes
Co-encadrant :	Sébastien LE NOURS	Maître de Conférences, Université de Nantes

ACKNOWLEDGEMENT

Before you read my thesis, I would like to tell you about the people who have accompanied me throughout my wonderful PhD journey.

First of all, I would like to thank my supervisor, Mr. Sébastien PILLEMENT, Professor at the Université de Nantes. He has given to me the right instructions at the right moments to prevent me from going to the wrong ways.

I would like to thank my co-supervisor, Mr. Sébastien LE NOURS, Lecturer at the Université de Nantes. He asked me a lot of interesting questions to not only help me to understand better about my topic, but also create new ideas.

I would like to express my sincere gratitude to all members of the jury. I would like to thank Madam Liliana CUCU-GROSJEAN, Researcher at INRIA, Paris and Mr. François VERDIER, Researcher at LEAT, Université Côte d'Azur for having accepted the responsibility of rapporteurs. I express my gratitude to Madam Laurence PIERRE, Professor at the Université Grenoble Alpes, to have participated in the jury.

Special thanks to the secretary, Sandrine CHARLIER at IETR Polytech Nantes for taking care of all my administrative issues during my thesis with her ultimate kindness.

I would like to thank all colleagues at IETR for interesting time to discuss about working, to exchange their ideas, their cultures during the coffee time. I also would like to thank to my colleagues at OFFIS: Mr. Kim GRUETTNER and Mr. Ralf STEMMER who have been working together in the Pssim4MC project.

To my dear family in Vietnam, I would like to express my gratitude and love to my father Hai Duong VU, my mother Thi Tam DOAN, my younger brother Hai Thien Long VU, my sister and brother-in-law Thi Thuong Thuong VU and Minh Hai DANG, my nephew Vu Binh Nguyen DANG, my niece Vu Thao Nguyen DANG and finally my love Thi Diem Huong BUI who always love me unconditionally. They gave me a lot of motivations and encouragements to finish my thesis from thousands of miles away in Vietnam.

I would like to thank all my dear friends who supported me during my time in Nantes.

And finally, I would like to conclude this section by a poem of Walter Savage Landor (1775-1864):

I strove with none, for none was worth my strife.
Nature I loved, and, next to nature, Art;
I warm'd both hands before the fire of Life;
It sinks, and I am ready to depart.

Table of Contents

List of acronyms

AADL	Architecture Analysis and Design Language
AHB	Advanced High-performance Bus
ALTL	Adaptive Linear Temporal Logic
AMBA	Advanced Microcontroller Bus Architecture
AP	Atomic Proposition
ARM	Advanced Risc Machine
ATLM	Arbitrated Transaction Level Model
BC	Best Case
BIP	Behavior Interaction Priority
BFM	Bus Functional Model
BLTL	Bounded Linear Temporal Logic
BRAM	Block Random Access Memory
CAN	Controller Area Network
COST	Commercial Off-The-Shelf
CPU	Central Processing Unit
CTL	Computation Tree Logic
CTMC	Continuous Time Markov Chains
DMA	Direct Memory Access
DOL	Distributed Operation Layer
DSE	Design Space Exploration
DSP	Digital Signal Processors
DTMC	Discrete Time Markov Chains
ESL	Electronic System Level
ESPAM	Embedded System-level Platform synthesis and Application Mapping
ET	Event Triggered
FCFS	First Come First Served
FIFO	First In First Out
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Arrays
FPU	Floating Point Unit
FSM	Finite State Machine

GCSL Goal and Contract Specification Language
GPP General Purpose Processors
GPU Graphical Processing Unit
GSL GNU Scientific Library
GSM Global System for Mobile Communication
GUI Graphical User Interface
HW Hartware
IO Input Output
IP Intellectual Property
JPEG Joint Photographic Experts Group
JTAG Joint Test Action Group
KPN Kahn Process Network
LTL Linear Temporal Logic
LRU Least-Recently Used
LUT Lookup Table
MA Markov Automata
MAC Medium Access Control
MAG Monitor and Aspect Advice Generator
MDP Markov Decision Process
ML Message Level
MoC Model of Computation
MoA Model of Architecture
MoP Model of Performance
MoS Model of Structure
MP3 MPEG-1/2 Audio Layer III
MPA Modular Performance Analysis
MPSoC Multiprocessor System-on-Chip
MUL Multiplier
MUV Model Under Verification
NoC Network on Chip
NPTA Network of Priced Timed Automata
PA Probabilistic Automata
PCTL Probabilistic Computation Tree Logic
PE Processing Element
PLASMA Platform for Learning and Advanced Statistical Model checking Algorithms
PLRU Pseudo Least-Recently Used
PTA Priced Timed Automata

RML Reactive Module Language
ROM Result Oriented Modeling
RR Round Robin
RTL Register Transfer Level
RTOS Real-Time Operating System

SADF Scenario Aware Dataflow
SBIP Stochastic Behavior Interaction Priority
SCE System-on-Chip Environment
SCHM Single Chip Heterogeneous Multiprocessors
SCM Statistical Contention Model
SCL Stochastic Contention Level
SDF Synchronous Dataflow
SDFG Synchronous Dataflow Graph
SLDL System level design language
SMC Statistical Model Checking
SPRT Sequential Probability Ratio Test
SLD System Level Design
SUA System Under Analysis
SSP Single Sampling Plan
SW Software
SystemILA System Integrated Logic Analyzer

TA Timed Automata
TAS Task Allocation Scheduling
TDMA Time Division Multiple Access
TL Transaction Level
TLM Transaction-Level Model
TMC Time Measurement Controller
TMU Time Measurement Unit
TPN Timed Petri Net
TT Time Triggered

UML Unified Modeling Language

VHDL Very-high-speed integrated circuits hardware description language

WLAN	Wireless Local Area Network
WCRT	Worst Case Response Time
WFS	Wave Field Synthesis
WCET	Worst Case Execution Time
WMTL	Weighted Metric Temporal Logic
WC	Worst Case
XML	Extensible Markup Language

List of Figures

List of Tables

Résumé

Motivations

Les systèmes multi-processeurs sur puce (MPSoC) sont largement appliqués dans de nombreux domaines en raison de leur ratio performance/consommation d'énergie élevé. La vérification des propriétés temporelles de ces systèmes au plus tôt dans le processus de conception est très importante. Cette vérification vise à garantir que les contraintes de temps sont pleinement respectées. Cependant, les ressources partagées au sein des MPSoCs peuvent impliquer de fortes variations du temps d'exécution du logiciel et compliquent donc la prévision des propriétés temporelles de ces systèmes. Par conséquent, il est essentiel de créer un modèle de performance capable de capturer les effets des ressources partagées. De plus, il faut choisir une approche d'analyse appropriée qui propose non seulement un bon compromis entre la précision et le temps d'analyse, mais aussi permette une bonne scalabilité. Cette thèse vise à étudier l'adoption de méthodes de modélisation et d'analyse probabilistes pour améliorer l'efficacité des approches d'analyse temporelle des systèmes MPSoC.

Contexte de la thèse

Les systèmes multiprocesseurs système sur puce (MPSoC) sont apparus dans de nombreux domaines en quelques décennies. Un système MPSoC se compose de trois parties principales: une application, un système d'exploitation et une plate-forme matérielle. L'application peut être divisée en plusieurs tâches exécutées sur une plate-forme matérielle. Le système d'exploitation est composé des logiciels du système (les services de gestion de la mémoire, l'ordonnanceur, etc.) et des pilotes d'abstraction. La plate-forme matérielle se compose de différents composants tels que des éléments de traitement (processing elements, PEs), des bus, des mémoires et des périphériques. Dans un système MPSoC, ces composants matériels sont conçus sur une même puce. Un MPSoC homogène contient plusieurs PE du même type et offre des capacités de parallélisation des applications. Un MPSoC hétérogène présente différents types de PE conçus pour des fonctionnalités spécifiques. La complexité des systèmes MPSoCs augmente rapidement dans les deux parties logicielles et matérielles. Du côté matériel, les interactions entre différents composants matériels accédant aux ressources partagées compliquent la prévision des comportements du système. Du côté logiciel, les fonctionnalités des applications sophistiquées avec des millions de lignes de code nécessitent d'énormes efforts pour être testées et vérifiées. La vérification de

la performance de ces systèmes est donc indispensable car elle permet de vérifier le respect des fonctionnalités mais aussi des exigences dites non fonctionnelles. Lors de l'exécution du programme, les ressources logicielles et matérielles influencent le temps d'exécution du programme. Dans cette thèse, nous nous concentrons principalement sur l'étude des influences des ressources partagées sur l'analyse temporelle des systèmes MPSoC.

Dans les systèmes MPSoC, les éléments de traitement partagent des ressources du système telles que des bus, des mémoires (mémoire globale, caches privées), des caches partagées, etc. Les accès simultanés à ces ressources partagées provoquent des interférences qui entraînent des délais supplémentaires dans le temps d'exécution des tâches. Ces ressources partagées influent sur les comportements temporels des systèmes MPSoC.

Les interférences des accès concurrents aux résources partargées causent des difficultés à prévoir la performance du système. Le temps d'exécution d'une tâche peut varier en fonction du logiciel de la tâche et des ressources matérielles où la tâche est exécutée.

— Du point de vue logiciel, le temps d'exécution d'une tâche dépend de ses données d'entrée. Différentes données d'entrée peuvent être traitées en suivant différents chemins d'exécution à travers le programme, et donc fournir des temps d'exécution différents.

— Du point de vue matériel, la variabilité du temps d'exécution est causée par les interférences entre les tâches accédant aux ressources partagées.

Étant donné que de nombreuses ressources matérielles sont impliquées dans l'exécution de la tâche, le temps d'exécution de la tâche devient plus difficile à prévoir. La caractérisation de ces temps élémentaires est cruciale pour la prévision des comportements temporels du système.

Différentes approches ont été proposées pour estimer la performance du système, par exemple: les approches basées sur la simulation, les approches formelles, les approches probabilistes.

Les approches basées sur la simulation testent partiellement les propriétés du système en se basant sur un ensemble limité de stimuli. Dans les approches existantes, des modèles d'architectures matériel-logiciel sont formés en combinant un modèle d'application et un modèle de plate-forme. Au début de la phase de conception, une description complète des fonctionnalités de l'application n'est pas obligatoire et des modèles de charge de travail de l'application sont utilisés. Un modèle de charge de travail exprime les charges de calcul et de communication (par exemple, le temps d'exécution, la consommation d'énergie, le coût de la mémoire) qu'une application provoque lorsqu'elle est exécutée sur les ressources de la plateforme. Les modèles de performances capturés sont générés sous forme de descriptions exécutables et simulés. Le temps d'exécution de chaque primitive de chargement est approximé comme un délai. Les délais sont généralement estimés à partir de mesures sur des prototypes réels ou de l'analyse de simulations de bas niveau. Les approches basées sur la simulation nécessitent une analyse approfondie de l'architecture dans divers scénarios de travail possibles. Les modèles d'architecture créés ne peuvent pas être simulés de façon exhaustive et les pires scénarios de travail peuvent être difficilement identifiés

et évalués. Un autre problème important concerne la précision des modèles créés. Enfin, avec la complexité croissante des plates-formes MPSoC, l'exécution des modèles de simulation nécessite plus de temps de simulation.

Différentes approches formelles ont été proposées pour analyser les systèmes multicœurs et fournir des bornes de temps. Ces approches formelles sont généralement classées comme méthodes analytiques et méthodes basées sur l'état. Les méthodes analytiques ont l'avantage d'être évolutives pour analyser des systèmes à grande échelle. Par contre, ces méthodes analytiques font abstraction de nombreux détails de fonctionnement du système analysé, tels que les protocoles d'arbitrage complexes ou les dépendances de tâches de communication inter-processeurs, ce qui conduit à des résultats pessimistes par rapport aux résultats des méthodes basées sur l'état. Les méthodes basées sur les états representent le système consideré comme un système de transition (états et transitions). Étant donné qu'elles reflètent les états de fonctionnement réels du comportement du système, des résultats plus précis peuvent être obtenus par rapport aux méthodes analytiques. De nombreuses approches récentes pour l'analyse temporelle du logiciel sur des architectures multicoeur sont construites sur des techniques d'analyse basées sur l'état. Les approches basées sur les états permettent une analyse exhaustive des propriétés du système au prix d'effort de modélisation et d'analyse importants.

Les approches probabilistes sont une combinaison de modèles probabilistes et de techniques d'analyse. Dans le contexte des systèmes embarqués, ils représentent un moyen de capturer la variabilité du système. La variabilité provient principalement de la sensibilité du système à l'environnement et des effets de bas niveau des plate-formes matérielles. Les modèles probabilistes peuvent être utilisés pour capturer de manière appropriée cette variabilité. Les modèles probabilistes sont des extensions du système de transition et permettent de prendre en compte les variations des temps d'exécution et des transitions d'état.L'analyse de ces modèles probabilistes permet d'obtenir des mesures quantitatives. Les approches probabilistes qui combinent la simulation et les approches formelles est un bon compromis entre la précision et les efforts d'exploration. Le model-checking statistique (SMC) a été proposé comme alternative aux approches formelles pour éviter une exploration exhaustive du modèle de l'espace d'états. SMC fait référence à une série de techniques utilisées pour explorer une sous-partie de l'espace d'états et fournit une estimation sur la performance du système. SMC désigne un ensemble de techniques statistiques présentant les avantages suivants:

— SMC est basé sur une sémantique formelle des systèmes qui permet de vérifier les propriétés comportementales. SMC répond à des questions qualitatives (Est-ce que la probabilité pour un modèle de satisfaire une propriété donnée est supérieure ou égale à un certain seuil?) et des questions quantitatives (quelle est la probabilité pour un modèle de satisfaire une propriété donnée?).

— Il nécessite simplement un modèle exécutable du système qui peut être simulé et vérifié par

rapport aux propriétés basées sur l'état exprimées en logiques temporelles. Les exécutions observées sont traitées pour décider avec une certaine confiance si le système satisfait une propriété donnée.

— En tant qu'approche basée sur la simulation, SMC demande moins de mémoire et de temps que les approches exhaustives. SMC peut donc être considéré comme un compromis entre les tests et la vérification formelle.

SMC permet d'approcher des systèmes qui ne peuvent être évalués avec une approche exhaustive. Il nécessite simplement un modèle de simulation du système, qui peut être vérifié par rapport aux propriétés basées sur l'état. Cependant, l'adoption de techniques SMC pour l'analyse des systèmes MPSoC a rarement été envisagée. Cela peut s'expliquer par les efforts considérables déployés pour mettre en place cette approche. La création de modèles probabilistes reste donc une tâche difficile et n'est pas bien prise en charge pour les systèmes MPSoC. Cette thèse vise à fournir des lignes directrices pratiques pour faciliter la création de modèles probabilistes et favoriser l'adoption d'approches d'analyse probabiliste pour les systèmes multicœurs.

Contributions

Dans le cadre de cette thèse, nous cherchons à étudier l'adoption de méthodes de modélisation probabiliste et d'analyse temporelle des systèmes MPSoC. Nos principales contributions sont:

1. Cette thèse vise à fournir des lignes directrices pratiques pour faciliter l'adoption de méthodes probabilistes dans le flot de conception au niveau système. Ce flot de travail est basé sur trois parties principales. (1) Une approche basée sur la mesure est d'abord utilisée pour caractériser l'exécution d'une application sur une plateforme réelle. Cette caractérisation se fait à la fois dans les parties calcul et communication, (2) L'approche modélise à la fois les parties logicielles et matérielles et se base sur un langage de modélisation de niveau système. Cette modélisation capte la variation du temps d'exécution en utilisant les approches probabilistes, (3) Une approche d'analyse probabiliste est basée sur le model checking statistique (SMC) qui estime la probabilité que notre modèle probabiliste puisse satisfaire une propriété temporelle.
2. Au niveau transactionnel, le processus de simulation reste lent, ce qui pose des difficultés pour analyser des systèmes complexes. Nous proposons un modèle de communication au niveau message d'un bus multiprocesseur pour fournir des résultats de simulation rapides mais précis. Le modèle proposé a montré une accélération significative de la simulation par rapport au modèle au niveau transactionnel (TLM) sans dégrader la précision de l'analyse.
3. Une *approche d'analyse probabiliste* utilisant la méthode de model checking statistique est mise en place. Cette approche SMC explore partiellement l'espace d'état du système,

tout en rendant possible pour limiter la probabilité de faire une erreur sur les prédictions en contrôlant le nombre d'exécutions de simulation à l'aide d'algorithmes statistiques. Dans cette analyse, les effets des paramètres des algorithmes statistiques sur les résultats d'analyse sont étudiés.

Contenu du manuscrit

Ce manuscrit est organisé comme suit:

— Chapitre 1 introduit les motivations, le contexte de la thèse, les contributions proposées.

— Chapitre 2 donne un aperçu de l'état de l'art, la comparaison des approches et des méthodes d'amélioration de la vitesse de simulation.

— Chapitre 3 présente notre démarche de travail, comme le modèle de l'architecture (MoA), le modèle de calcul (MoC), l'infrastructure de mesure et le modèle de simulation. Quelques premières études de cas sont présentées et les résultats obtenus avec l'environnement développé sont donnés.

— Chapitre 4 introduit le modèle de communication au niveau des messages. Ce modèle de haut niveau améliore le modèle de simulation à la fois en précision et en vitesse de simulation. Les résultats de la simulation montrent une accélération significative par rapport au modèle de niveau transactionnel (TLM).

— Chapitre 5 présente l'approche d'analyse temporelle probabiliste. Nous visons à démontrer l'efficacité de l'approche de vérification des modèles statistiques pour l'analyse temporelle des MPSoCs. Une étude plus approfondie sur différents algorithmes statistiques est fournie dans ce chapitre. Une extension de l'architecture matérielle avec cache privé activé pour chaque tuile est présentée. Les résultats d'analyse d'un tel système montrent un bon niveau de précision.

— Chapitre 6 résume les contributions et fournit différentes perspectives et les possibles travaux futurs.

Chapter 1

Introduction

Multi-processor system-on-chip (MPSoC) systems are widely applied in many domains because of their high performance/energy consumption ratio. A key property to allow early performance evaluation of such systems in the design process is timing predictability. It represents the capability to predict timing properties of MPSoC in order to verify that timing constraints are met. However, shared resources in MPSoC can cause high variations of software execution time and thus make timing prediction of such systems a challenging task. Therefore, it is crucial to create performance models that can capture the shared resources effects. In this chapter, we first introduce the context of our research. Then we discuss different analysis approaches which are currently being considered for timing analysis. Finally, we present our main contributions and the organization of this manuscript.

1.1 Timing predictability issues in multi-processor system-on-chip

1.1.1 Hardware and software resources of multi-processor system-on-chip

Multi-processor system-on-chip (MPSoC) systems have emerged in many domains in few past decades. We define a MPSoC system as composed of three main parts: an application, an embedded software and a hardware platform. Application can be divided into multiple tasks which are executed on a hardware platform. Embedded software is composed from system softwares (operating system, memory management services, arbitration policies, etc.,) and abstraction drivers. Hardware platform consists of different components such as processing elements (PEs), buses, memories and peripherals. In MPSoC system, multiple PEs and other hardware components are designed on a same chip. In Fig. 1.1, we present an overview of a MPSoC system. A four-task application is executed on a hardware platform consisting of three processing elements that communicates to a shared memory and peripherals via an interconnection.

We can classify a MPSoC system as *homogeneous* or *heterogeneous* depending on the type of PEs. A *homogeneous MPSoC* contains multiple PEs which are of same type. Homogeneous MPSoCs are designed to execute a same application on their PEs for parallelism perspectives. *Heterogeneous MPSoC* have different types of PEs which are designed for specific functionalities.

For example, these PEs can be General Purpose Processors (GPPs), Digital Signal Processors (DSPs), Graphical Processing Units (GPUs), hardware accelerators.

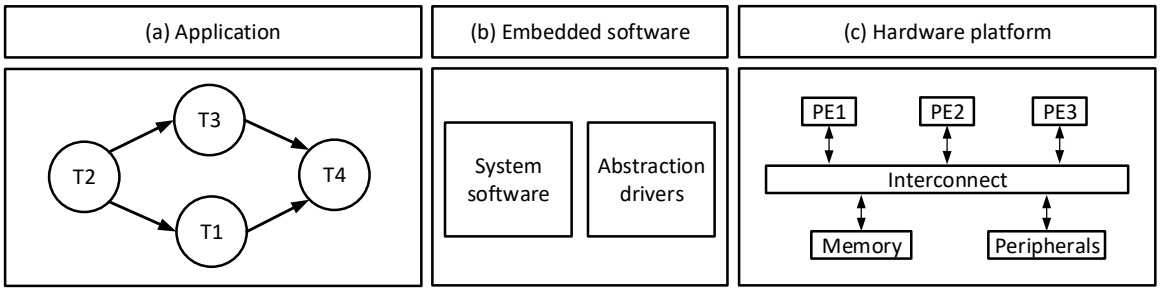


Figure 1.1 – Overview of a multiprocessor system-on-chip consisting of an application (a), an embedded software (b) and a hardware platform (c).

Along with the growth of system complexity, designers have to consider several challenges in the design process. Different design challenges were discussed by Teich [1] and Nouri [2]:

1. *Heterogeneous system-on-chip technology* challenges the designers to integrate a heterogeneous system into a single multi-billion transistor system-on-chip. The current tendency to get higher performance is to put more transistors on smaller size of chip. However, the higher density of transistor on a chip requires much more higher efforts for physical design and optimization of different aspects such as energy consumption, temperature, etc.
2. *Hardware/software complexity*: From the hardware side, interactions between different complex hardware components accessing shared resources cause the unpredictability of system behaviors. From the software side, the functionalities of sophisticated applications with millions lines of code need enormous efforts to be tested and verified.
3. *Hardware/software verification* checks in the design process whether the MPSoCs can meet its functionalities and requirements. Since MPSoCs become more and more complex, the performance verification of such system is inevitable. In [3], Radetzki presented two aspects which has to be taken into account in the performance verification: *functional* and *extra-functional* properties. *Functional* properties are verified to check the system functionality correctness by analyzing possible testing scenarios to detect potential problems. This avoids costly iterations in the design/manufacturing process. *Extra-functional* properties consist of the timing behavior, power consumption, reliability, security, etc. Early performance verification reduces not only the design time but also save the time-to-market windows.
4. *Uncertain environment effects*: Embedded systems are designed for specific functionalities where they can continuously interact with their environment. Thus their performance get

influenced by different uncertain and unpredictable environment contexts. Therefore, the environment uncertainties should be taken into account in the design process.

In the program execution, both software and hardware resources influence the program execution time. In this thesis, we mainly focus on studying the shared resources influences to the timing analysis of MPSoC systems. Thus in the next Section, we aim to present such influences.

1.1.2 Influence of shared resources on timing predictability of MPSoC systems

In MPSoC systems, the processing elements share system resources such as buses, memories, caches, etc. These shared resources provide different influences to timing behaviors of MPSoC systems. Fig. 1.2 presents different types of shared resources existing in a MPSoC system. A *private cache* is accessible only by each PE. A *shared cache* is used by all the PEs. Both private and shared caches are used to temporarily store data to reduce the large latency between the PE and main memory. The PEs access to the shared caches and main memory via an interconnect.

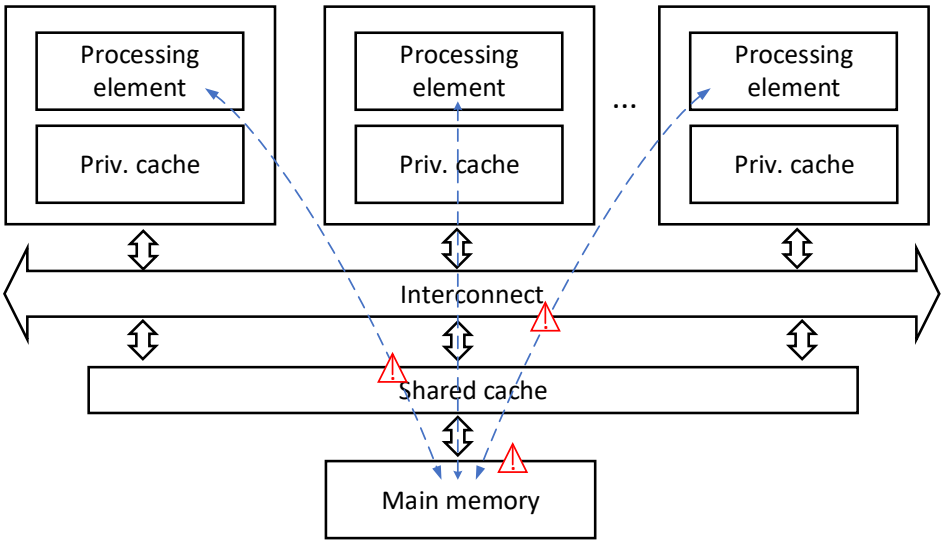


Figure 1.2 – Different types of shared resources in MPSoCs: buses, memories (main memory, private caches), shared caches. Interferences occur while the processing elements access to the shared resources at same time.

Concurrent accesses to shared resources cause interferences. These inteferences lead to undesired additional delays in the application execution time. Different mechanisms causing additional delays in Commercial off-the-shelf (COST) multi-processors were presented by Kotaba et al. in [4]. For example, contention on shared bus can come from concurrent accesses of multiple cores, or other device, such as IO, DMA, etc. In shared memory, interleaved accesses by multiple

cores cause additional delays for cores. For dynamic memory, additional delays can come from memory refresh.

In [5], Abel et al. classified shared resources into two classes: *bandwidth resources* and *storage resources*. The *bandwidth resource* corresponds to the situation when several tasks access to this resource at the same time. However, this resource only allows one access at a time. An arbitration policy is thus used to manage the order of access of these tasks. It allows one task to access to this resource and delays the others. One example of bandwidth resource is *shared bus*. The arbitration policies are divided into three main classes:

— *Time-driven arbitration* uses a predefined bus schedule which assigns time slot of fixed size to particular PEs (e.g., *Time Division Multiple Access* (TDMA)).

— *Event-driven arbitration* decides at run-time, which PE is granted the access to the resource during the next time slot. These decisions usually depend on the access histories of all PEs (e.g., *Round Robin* or *First Come First Serve* (FCFS)).

— *Hybrid arbitration* splits their arbitration period into segments of fixed length. Static segments use *Time-driven resource arbitration* and dynamic segments use *Event-driven resource arbitration* (e.g., *FlexRay* a bus protocol used in the automotive industry of this strategy).

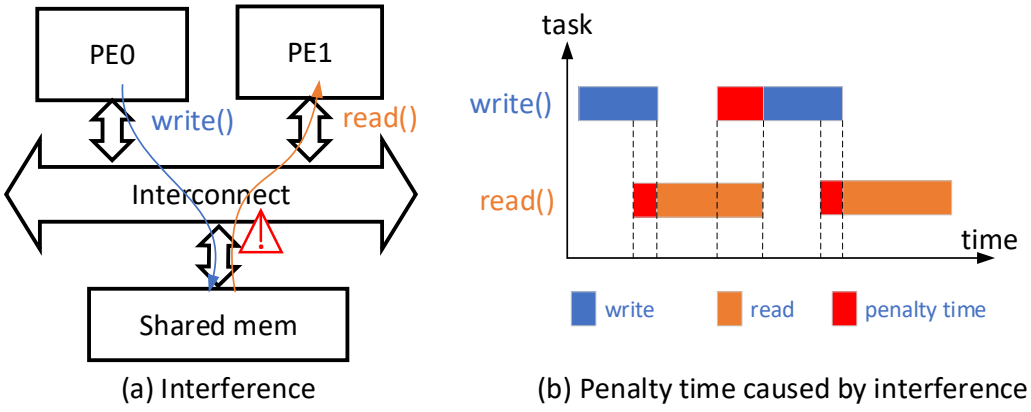


Figure 1.3 – Interference on bus to access to the shared memory (a) causes penalty time (b).

Fig. 1.3 presents the interferences caused by concurrent accesses to the bus. In Fig. 1.3 (a), two PEs attempt to access the shared memory via an interconnect. One PE tries to write data to the shared memory and another one tries to read data from the shared memory. The bus arbitration policy in this case is First-Come-First-Served. Then in Fig. 1.3 (b), the execution process of these two PEs is presented. The first write function is executed and the first read function is delayed. The second write function is also delayed until the end of the first read

function. Similarly, the second read function is delayed by the second write function. The delay duration caused by the bus interference is called *penalty time*.

The *storage resource* relates to the situation when one task changes the state of the shared resource which is being used by a second task. This situation causes an additional delay for the second task. The examples of storage resource are *shared memories* and *shared caches*. Multiple accesses to main memory via shared cache might cause penalty delays. In Fig. 1.4 (a), we present an example in which two PEs access to main memory via a shared cache. *Cache hit* occurs when PE0 attempts to read data from the main memory and this data is available on shared cache. For PE1, the data is not available on the shared cache (i.e, *cache miss*) and it has to load data

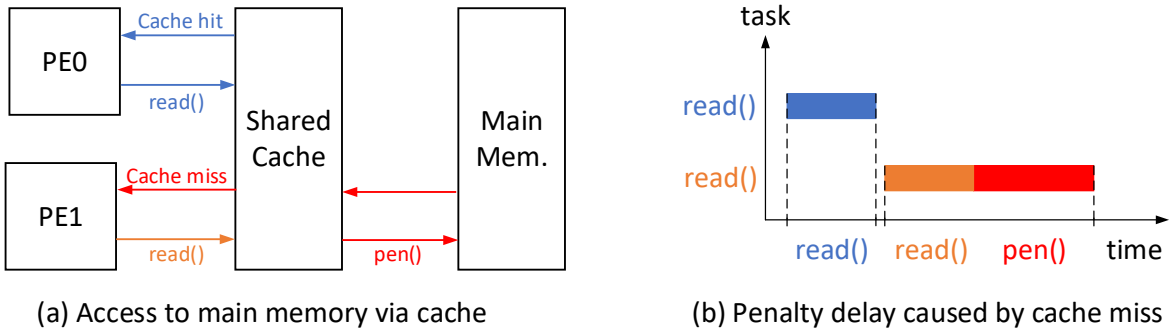


Figure 1.4 – Accessing to the main memory via cache (a) and a penalty delay caused by a cache miss (b) leads to longer execution time.

from main memory. In Fig. 1.4 (b), the execution time to read data of two PEs is presented. The first read function needs only the read time from the cache. The second read function gets slower execution time with additional penalty delay caused by the cache miss. Different cache replacement policies are used to reduce cache misses. These policies help the cache to identify which data should be removed to make space for new data that needs to be added. Furthermore, they improve both precision and efficiency of a cache analysis. In [6], Monniaux et al. presented in detail some following cache replacement policies.

— *Least-Recently Used* (LRU) evicts the data block least recently used when a cache miss occurs. On a miss, the oldest block is replaced by a new one, which has age 0 and the ages of all other blocks are incremented.

— *Pseudo-LRU* (PLRU) improves the performance of the LRU algorithm by evicting data block using approximate measures of age rather than maintaining the exact age of every data block in the cache.

— *First-In-First-Out* (FIFO) evicts the data block in the order they were added without depending on how often or how many times they were accessed before.

In the performance verification of MPSoC systems, timing behaviors are among priorities to be analyzed, especially the task execution time. As summarized by Mitra et al. in [7], the task execution time can variate depending on both software tasks and hardware resources where the tasks are executed.

— From the software perspective, the execution time of a task depends on its input. Different *input data* can be processed following different execution paths through the program leading to different execution times.

— From hardware perspective, the variability in the execution time is caused by interferences between tasks accessing shared resources.

We denote the ability to predict timing behaviors of MPSoC systems as *timing predictability*. Since many hardware resources are involved in the task execution, the execution time becomes more difficult to be predicted. We refer the task execution time on each hardware element as *elementary timing*. The characterization of these elementary timings is crucial for the timing predictability of an application made of multiple tasks and allocated to a multip-processor platform. In the scope of this thesis, we focus on capturing timing variability from hardware perspectives caused by shared resource effects. In the next section, we introduce the notion of *timing compositionality* which represents the ability to exhibit different shared resource effects of MPSoC systems.

1.1.3 Timing compositionality of MPSoC systems

Before defining the *timing compositionality*, Hahn et al. [8] provided a notion of decomposition of a system's timing into the timing contributions of its components. The system states are related with the corresponding states of each component by the decomposition. Furthermore, the decomposition provides a *combination function* that combines the timings of the individual components and captures the *type of composition*. The complexity of the combination function depends on the chosen decomposition. The *timing compositionality* is then defined that the timing behavior of a system can be inferred from the timing contributions of its constituent components and the type of composition. For example, given a MPSoC system consisting of multiple processors that access to a shared memory via a shared bus, the execution time of the program running on one processor can be predicted from its computation time, the memory access time and the bus blocking time.

The type of composition is classified based on two following notions *timing anomalies* or *domino effects* as defined in [9, 10, 11].

— A *timing anomaly* is a situation where the local worst-case does not contribute to the global worst-case. In [12], different potential sources can lead to timing anomalies, such as: scheduling, branch prediction, cache and cache replacement policy.

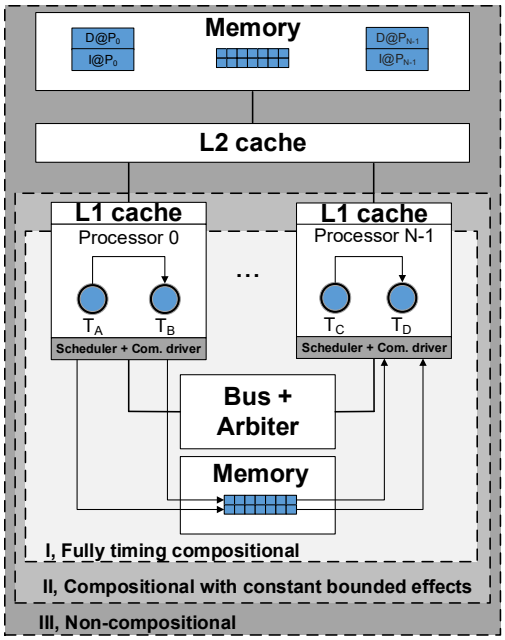


Figure 1.5 – Classification of architectures according to the level of resources compositionality

— A *domino effects* is a situation where the difference in timing between two states cannot always be bounded by a constant. For example, given a program loop, the difference in execution time between states of the pipeline (or caches, or other hardware resource) increases in each iteration.

In [13], Cullmann et al. classified three types of MPSoC architectures depending on whether they exhibit timing anomalies and/or domino effects. These architectures are illustrated in Fig. 1.5.

1. *Fully timing compositional architectures* does not exhibit timing anomalies or domino effects. In this case, a local worst-case path can be analyzed safely without considering other paths. In this thesis, we consider a hardware platform with predictable computation time due to un-cached local and exclusive data and instruction memory.
2. *Compositional with constant bounded effects architectures* exhibit timing anomalies but no domino effects. The analysis has to consider the effects of local data/instruction caches (L1 caches as illustrated in Fig. 1.5). Possible cache misses are caused by the difference between cache size and data in the main memory. In general, an analysis has to consider all execution paths. A trade-off between precision and efficiency is to safely discard local non-worst-case paths by adding a constant number of cycles to the local worst-case path. The Infineon TriCore [14] is an example of this class.

3. *Non-compositional architectures* exhibit domino effects and timing anomalies. Different level of caches are taken into account in the analysis. Possible cache misses are caused by the replacement of data on shared cache (L2 cache) by processors. For such architectures timing analyses always have to follow all paths since a local effect may influence the future execution arbitrarily. In [15], Schneider presented a domino effect in the pipeline of the PowerPC 755.

To evaluate such architectures, appropriate modeling and analysis approaches are needed to describe and analyze timing behaviors. In the next section, we aim to present the performance evaluation of MPSoC systems related to different timing analysis approaches.

1.2 Performance evaluation of MPSoC systems

1.2.1 Hardware/software codesign

Hardware/software codesign approaches aims to concurrently design hardware and software components of embedded systems. The main objective is first to meet design constraints such as cost, performance, and power of the product and secondly to reduce the time-to-market of the product [16].

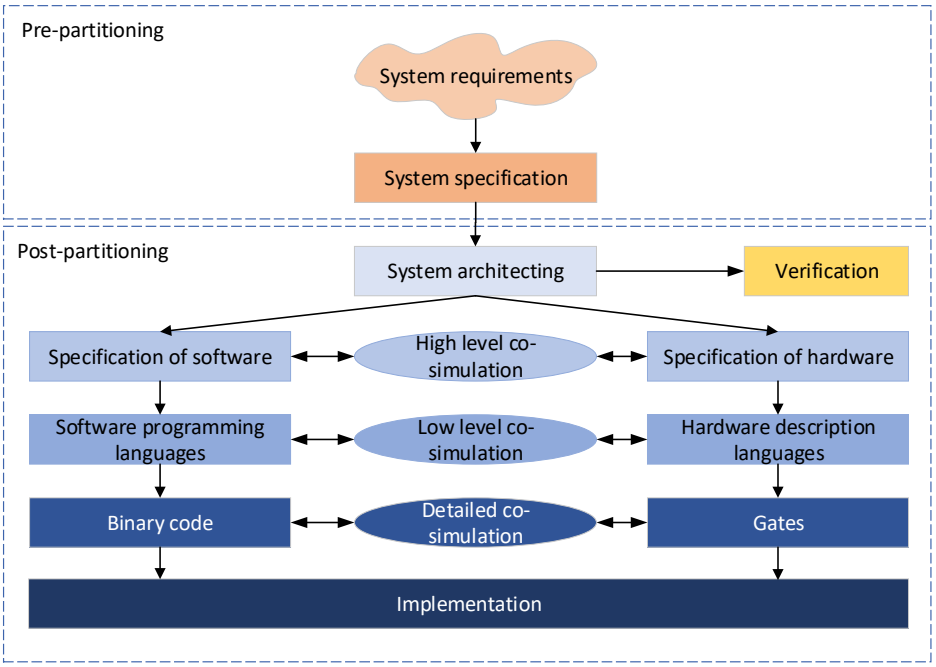


Figure 1.6 – Hardware/software codesign with pre-partitioning and post-partitioning phases. System level verification captures the system performance early in the design process.

In Fig. 1.6, the hardware and software codesign steps with the pre-partitioning and post-partitioning are illustrated. In the pre-partitioning, the codesign process first identifies the system requirements to be satisfied. The system architecting represent the considered system using system level design (SLD) tools. This high level system design avoids detailed consideration of the system. However, it requires availability of reliable estimations about component models to create an accurate system. As illustrated in [16], a 10% error is acceptable at this phase. At system level, a high level of modeling language can be used to describe the considered system, for example Unified Modeling Language (UML). In this thesis, we focus on the verification step which allows early exploration of the system performance by using the simulation-based or formal approaches.

Different abstraction levels of hardware/software codesign are then illustrated in the post-partitioning steps of Fig. 1.6. From the software side, the functional specification of software such as the computation and communication parts can be mapped and scheduled on one or multiple processing elements. This step can be done by using a target programming languages such as C, C++. At the binary code level, the software code such as a function, a method or a basic block is compiled and linked to the selected hardware platform to build an executable description [1]. From the hardware side, processes and tasks are implemented as hardware accelerators are synthesized down to RTL description by using hardware description languages such as VHDL, Verilog. At the gates level, the granularity of the objects considered during logic synthesis are implemented by using logic gates and flip flops. Finally, the implementation of an application executing on a hardware platform is for validation and verification purposes. The efficiency of the proposed system level model can be evaluated by comparing the simulation results with the implemented results.

Several languages have been proposed as intermediate solution for system design, for example SystemC [17] and SpecC [18], [19]. SystemC is a class library of C++, while SpecC is a superset of ANSI C. They provide useful data types and concurrent programming structures for describing both software and hardware parts.

System level design approaches have been proposed to allow estimation of HW/SW architecture performances early in the design process. In system level design approaches, workload models are used to capture the influence of application execution on platform resources. Timing properties of architectures and related power consumption can then be assessed under different working scenarios. However, the efficiency of system level design approaches strongly depends on the created HW/SW architecture models that should deliver both fast analysis time and good accuracy. Especially, captured workload models should correctly abstract low-level details of system components but still provide good estimations about the whole system performance. High-level models must capture the possible variability in multi-core platform resources usage caused by cache management, bus interleaving, and memory contention. The creation of efficient

architecture models represents thus a time-consuming effort that limits the efficiency of current system level approaches.

The different categories of system-level analysis approaches are discussed in the next sections.

1.2.2 Simulation-based analysis approaches

Simulation-based approaches are proposed to partially test system properties. Different simulation-based approaches have been compared in [20] to support evaluation of multi-core architecture performance early in the design process. In the existing approaches, models of hardware-software architectures are formed by combining an application model and a platform model. In the early design phase, full description of application functionalities is not mandatory and workload models of the application are used. A workload model expresses the computation and communication loads (e.g., time, power consumption, memory cost) that an application causes when executed on a hardware platform. The captured performance models are generated as executable descriptions and simulated. The execution time of each load primitive is approximated as a delay. Delays are typically estimated from measurements on real prototypes or analysis of low level simulations. SystemCoDesigner [21], Daedalus [22], SCE [19], and Koski [23] are good examples of academic approaches. Other existing academic approaches are presented by Kreku et al. in [24] and by Arpinen et al. in [25]. Simulation-based approaches require extensive architecture analysis under various possible working scenarios. Since the possible working scenarios of the created models are very huge, the created models can not be exhaustively simulated in a reasonable analysis duration and worst-case working scenarios are hardly identified and assessed. One other important issue concerns the accuracy of created models. As architecture components are modeled as abstractions of low level details, there is no guarantee that the created architecture model reflects with good accuracy the whole system performance. Finally, with the rising complexity of MPSoC platforms, execution of simulation models requires more simulation time. In the next section, formal analysis approaches are presented which provide a more exhaustive analysis of system state space.

1.2.3 Formal analysis approaches

Due to insufficient corner case coverage, simulation-based approaches are limited to determine guaranteed bounds of system properties. Different formal approaches have thus been proposed to analyze multi-core systems and provide time bounds. These formal approaches are commonly classified as *analytical* methods and *state-based* methods.

Most of the available static real-time methods are of analytical nature. An overview of such methods are given in [26]. Since analytical methods depend on solving closed-form equations, they have the advantage of being scalable to analyze large-scale systems. However, analytical methods abstract from state-based modus operandi of the system under analysis (SUA) leads

to pessimistic over-approximated results compared to state-based real-time methods [26]. For example, such systems can be complex state-based arbitration protocols or inter-processor communication task dependencies. Analytical methods that combine analysis of processor and bus scheduling for distributed embedded systems can be classified as *holistic* methods [27], [28], [29] and *compositional analytical* methods [30], [31].

State-based real-time methods are based on representing the SUA as a transition system (states and transitions) and since they reflect the real operation states of the actual system behavior, tighter results can be obtained compared to analytical methods. Many recent approaches for software timing analysis on many- and multi-core architectures are built on state-based analysis techniques. The two main considered application classes are streaming applications (modeled as synchronous data flow graphs) [32], [33] and generic real-time task-based applications [34], [35].

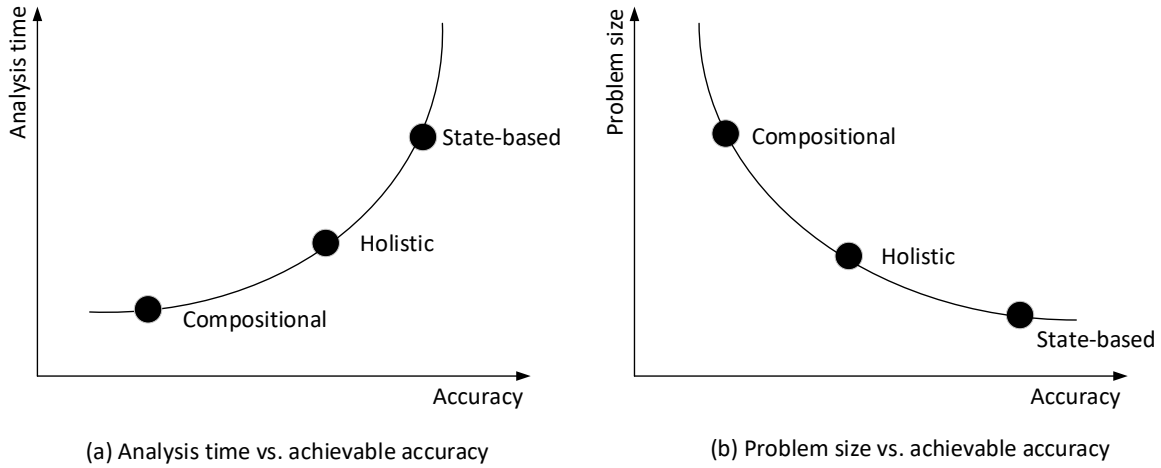


Figure 1.7 – Comparison different formal approaches according to the achievable accuracy and analysis time (a) and the obtained accuracy with the rising of problem size (b).

In Fig. 1.7, we compare different formal approaches according to the achievable accuracy and analysis time (a) and the potential gain in addressable problem size (b). As illustrated, state-based approaches provide tighter results compared to analytical (holistic and compositional) approaches. However, state-based approaches allow exhaustive analysis of system properties at the expense of time-consuming modeling and analysis effort. Analytical approaches can address more complex systems.

1.2.4 Probabilistic analysis approaches

Probabilistic approaches are a combination of probabilistic models and analysis techniques. In the context of embedded systems, they represent a means of capturing system variability. In this context, variability mainly comes from system sensitivity to environment and low level effects of hardware platforms. Probabilistic models can be used to appropriately capture this variability. Probabilistic models are extensions of labeled transition system and allow variations about execution times and state transitions to be considered, for example the discrete time Markov chains or Markov automata, etc. Analysis of probabilistic models allow quantitative measures to be obtained. As an illustration, the adoption of probabilistic model checking for evaluation of dynamic data-flow behaviors is presented in [32]. Markov automata is used as the fundamental probabilistic model to capture and analyze architectures. Characteristics as application buffer requirements, timing performance, and platform energy consumption are estimated. However, this approach is restricted to fully predictable platforms, with low influence of platform resources on timing variations.

Probabilistic approaches that combine simulation and formal approaches appear to be a good compromise to deliver both accuracy and limited exploration effort. Statistical Model Checking (SMC) [36] has been proposed as an alternative to formal approaches to avoid an exhaustive exploration of the state-space model. SMC refers to a series of techniques that are used to explore a sub-part of the state-space and provides an estimation. SMC designates a set of statistical techniques that present the following advantages:

— As classical model checking approach, SMC is based on a formal semantic of systems that allows inference of behavioral properties. SMC is used to answer qualitative questions (Is the probability for a model to satisfy a given property greater or equal to a certain threshold?) and quantitative questions (What is the probability for a model to satisfy a given property?).

— It only requires an executable model of the system that can be simulated and checked against state-based properties expressed in temporal logics. The observed executions are processed to decide with some confidence whether the system satisfies a given property.

— As a simulation-based approach, it is less memory and time intensive than exhaustive approaches. SMC can thus be seen as a trade-off between testing and formal verification.

This approach has been considered in various application domains [37]. First, SMC allows to approximate systems that can not be assessed with an exhaustive approach. Second, it only requires a simulation model of the system, that can be checked against state-based properties. Various existing probabilistic model checkers have been extended to support statistical approach. PRISM [38] is based on a symbolic model checker that relies on numerical techniques. UPPAAL-SMC [37] is a statistical model checking extension for the UPPAAL model checker. Plasma-Lab

[39] is a modular and extensible statistical model checker that can be extended with external simulator and checkers. The use of Plasma-Lab to carry out statistical analysis for systems modeled in SystemC is presented in [40]. SMC has been adopted in [41] to evaluate a many-core architecture. A statistical model checking tool called BIP-SMC was proposed in [42]. It was used to evaluate the probability that some timing aspects (e.g. execution time, variability of processing time) were bounded. A very recent work in [43] demonstrates the applicability of SMC for a quantitative evaluation of uncertainty-aware hybrid AADL designs [43] (of a train control system) against various performance queries. However, adoption of SMC techniques to analysis of multi-core systems have rarely been considered. This can be explained by the large amount of required effort to setup this approach. Creation of probabilistic models remains a difficult task and is not well supported for multi-core systems. This thesis aims at delivering some practical guidelines to facilitate the creation of probabilistic models and favour the adoption of probabilistic analysis approaches for multi-core systems.

1.3 Contributions and organization of the manuscript

In the scope of this thesis, we aim to study the way adoption of probabilistic modeling and timing analysis methods of MPSoC systems. The main contributions of this work are:

1. Since the lack of a systematic evaluation workflow leads to difficulties to use probabilistic approaches, a new evaluation workflow is proposed to evaluate the these approaches. This workflow is based on three main parts: (1) A measurement-based approach is first used to characterize timing behaviors of the execution of application on a real platform. This characterization is done in both computation and communication parts, (2) A probabilistic modeling approach implements both software and hardware parts on a specific programming language. This implementation captures the variation of the execution time by using the statistical timing models, and (3) A probabilistic analysis approach is based on statistical model checking (SMC) which estimates a probability that our probabilistic model can satisfy a timing property.
2. At transaction-level, the simulation process remains slow which causes difficulties to analyze complex systems. We propose a message-level communication model of a multiprocessor bus to deliver fast yet accurate simulation results.
3. A *probabilistic analysis approach* which considers statistical model checking (SMC) approach for MPSoCs is taken into account. This SMC approach partially explore system state-space, but still makes possible to bound the probability of making an error about predictions by controlling the number of simulation runs by using statistical algorithms. In this analysis, the effects of the statistical algorithms parameters to the analysis results are further investigated.

The remains of this manuscript is organized as follows:

— Chapter 2 gives an overview of the state of the art, comparing the existing timing analysis approaches and accuracy/simulation speed improvement approaches.

— Chapter 3 presents our workflow which is the working environment with the related the model of architecture (MoA), the model of computation (MoC), the measurement infrastructure and the simulation model. Some case-studies are presented and results achieved with the developed environment are given.

— Chapter 4 introduces the message level communication model. This high level model improves the simulation model in both accuracy and simulation speed. The proposed model showed a significant simulation speed-up comparing to the transaction-level model (TLM) without degrading the analysis accuracy.

— Chapter 5 presents the probabilistic timing analysis approach. We aim to demonstrate the efficiency of statistical model checking approach for timing analysis of MPSoCs. A further study about different statistical algorithms is provided in this chapter. An extension of the hardware architecture with private cache for each tile is presented. Analysis results of such system show a good level of accuracy.

— Chapter 6 summarizes the contributions and provides different perspectives and future work.

Chapter 2

State of the art

In the previous chapter, we introduced the context of our work. Different analysis approaches can be used to verify the performance of MPSOC systems. Probabilistic approaches are proposed to capture the variability of the execution time caused by contentions at shared resources. In this chapter, we first review the performance analysis approaches for MPSoC systems. These approaches are compared regarding their analysis effort, their accuracy and their scalability. Second, we specifically discuss some existing works that were proposed to improve the efficiency of system-level simulation-based approaches.

2.1 Performance analysis approaches

Analysis approaches are commonly classified as (1) simulation-based approaches, which partially test system properties based on a limited set of stimuli, (2) formal approaches, which check system properties in an exhaustive way, and (3) hybrid approaches, which combine simulation-based and formal approaches.

2.1.1 Simulation-based approaches

In this section, we present simulation-based approaches. We emphasize the way performance models are created. Then we present the considered use-cases and the obtained results. Several simulation-based approaches were compared in [44], such as SystemCoDesigner [21], Daedalus [45, 46], system-on-chip environment (SCE) [19, 47] and Koski [48] which are good examples of academic approaches.

SystemCoDesigner presented in [21] is an SystemC-based electronic system level (ESL) design framework for design space exploration (DSE). It allows to automatically map applications into a heterogeneous MPSoC platform. The SystemCoDesigner design flow starts by describing an application as an *actor-oriented model* using SystemC. Some or all actors are then generated using behavioral synthesis to create architecture templates. An architecture template contains all possible hardware modules, processors and the communication infrastructure. Design space exploration is used to select the best architecture solution that fulfills the user requirements, such as the overall throughput and required hardware resources. From a set of obtained solu-

tions, the designer selects promising implementations for rapid prototyping on FPGAs. In [21], Keinert et al. implemented a Motion-JPEG decoder case study. The DSE evaluated 7600 different solutions over 5.10^{33} possible solutions in 2 days 17 hours 46 minutes. The exploration time for each solution is about 30 seconds. In fact, it took a lot of time to explore a very small part of the solution space and good solutions could be missed. There were 366 solutions found and several of these solutions were implemented onto a Xilinx Virtex II FPGA (XC2V6000). Different objectives were evaluated such as: 1) throughput; 2) latency; 3) number of required flip-flops; 4) lookup tables (LUTs); and 5) block random access memories (BRAMs). The FPGA implementations results showed differences in latency and throughput up to 30% comparing to the performance estimations during DSE due to scheduling overhead. The estimated hardware resources during DSE required 15% more than in the implementation on FPGA. The accuracy of the performance model of this approach still needs to be improved.

Daedalus is a system-level design flow for MPSoC based embedded multimedia systems. It provides a high automation of design space exploration (DSE), system-level synthesis, application mapping, and system prototyping of MPSoCs. The Daedalus design flow is detailed in [45, 46], it consists of three main steps implemented by the KPNgen, Sesame and ESPAM tools. In the first step, a sequential application specification written in C or C++ is automatically converted into a parallel Kahn Process Network (KPN) specification using the KPNgen tool. In the second step, the generated KPNs are then used by the Sesame modeling and simulation environment to perform system-level architectural DSE. Sesame evaluates the performance of different applications to architecture mappings, hardware/software partitionings, and target platform architectures. More details about Sesame tool can be found in [49]. In the third step, the ESPAM tool takes as input the specifications of promising candidate system designs obtained from the previous step. It automatically generates synthesizable VHDL that implements the candidate MPSoC platform architecture. The target MPSoC of Daedalus is composable MPSoCs, in which the IP components are strictly composed. This prototyping allows to calibrate and validate the created performance models. Several case studies were used to demonstrate the feasibility of Daedalus approach. In [45], Thompson et al. explored different implementation option for a Motion-JPEG encoder case study running on a 4 heterogeneous processor MPSoC. The case study state space was exhaustively explored with 10148 solutions in 2.5 hours. The exploration time for each solution is less than 1 second which is much faster than in the System-CoDesigner approach. Several solutions were used for the prototyping on Xilinx Virtex II Pro FPGA (xc2cp20). The estimated simulations showed an average error of 12% and worst-case error of 19% to the FPGA implementations which still need to be improved. Another work about Daedalus approach presented by Nikolov et al. in [46] that performed a DSE study of a JPEG encoder application executing on both homogeneous (MicroBlaze processors) and heterogeneous (MicroBlaze processors and Discrete Cosine Trans-form (DCT) IP cores) MPSoCs. The number

of processing cores is up to 30. They achieved a performance speed-up of up to 20x compared to a single processor system. The DSE experiments and the real implementation of 25 different hardware solutions on Xilinx Virtex II-6000 FPGA were performed within 5 days with around 70% of this duration taken by the low-level commercial synthesis and place-and-route FPGA tools. They presented a very good accuracy with the error between the estimated models and real implementation is around 5%. However the experiment duration is still long while exploring a limited number of hardware solutions.

The system-on-chip environment (SCE) [19, 47] implements an automated system design flow from specification down to hardware/software implementation. The SCE design flow starts with an abstract model specifying the design functionality. The design flow then explores the design space consisting of the architecture, scheduling and network exploration to make different design decisions. A new lower level of abstraction model is refined by integrating design decisions into the previous model. In the next phase, hardware/software synthesis is performed. Hardware synthesis takes behavioral hardware models down to structural register-transfer level (RTL) descriptions. While in software synthesis, application written in SpecC [18], middleware, drivers and interrupt handlers are generated, cross-compiled, and targeted toward and linked against real time operating system (RTOS) to create a final target binaries. In SCE, all design steps are integrated under a common graphical user interface (GUI). The GUI provides interactive and visual design model and database browsing, decision entry, and design analysis. For the case study, they demonstrated the design space exploration of their approach by considering six industrial examples: the JPEG encoder, the GSM voice codec, floating- and fixed-point versions of an MP3 decoder and the GSM vocoder. For each example, they considered different architectures using Motorola DSP56600, Motorola ColdFire and ARM7TDMI processors together with custom hardware coprocessors and I/O units. They used various communication architectures with DSP, CF, ARM (AMBA AHB) and simple handshake busses. The exploration duration of the design space for each example is about few seconds. Timing errors range from 12.5% down to an average of 3% depending on back annotation of profiling or trace-based estimates.

In [48], Kangas et al. presented Koski design flow for MPSoC that covers the design phases from system-level modeling to FPGA prototyping. Koski design flow consists of five main steps. First, designers capture the requirements of an application and architecture, including design constraints. Second, the functionality of the system is described with an application model in the unified modeling language (UML) design environment and verified with functional simulations. The architecture model is designed from the application model and the given platform. A mapping model is used to describe the relationship between application and architecture models. Third, application and architecture models are transformed to an abstracted model for fast architecture exploration by using the UML interface. Fourth, the exploration step consists of static and dynamic exploration which analyzes an extensive set of architectures to find optimized

Related work	Fully timing compositional	Compositional with bounded effects	Non-compositional
SystemCoDesigner [21]	◑	○	○
Daedalus [45, 46]	●	○	○
SCE [19, 47, 44]	●	○	○
Koski [48]	◑	○	○

Table 2.1 – Classification of simulation-based approaches according to the level of compositionality of addressed platforms

mapping. The static exploration is used to optimize the allocation, mapping and scheduling, while the dynamic exploration is based on iterative simulation of application and architecture to optimize the cost function parameters. Designers can control the architecture exploration by specifying different constraints, such as the platform parts that can be used as well as the allowed mapping combinations or performance, area, and power constraints. In the last step, the physical implementation is performed by generating both software/hardware parts down to low level software code and RTL descriptions which are combined for the implementation of real hardware platform. As example, they studied a dynamic reservation time division multiple access (TDMA) based medium access control (MAC), so-called TUTMAC. The target platform is a part of a Wireless Local Area Network (WLAN) terminal (TUTWLAN). The architecture exploration was performed to optimize the initial allocation and mapping model. The static exploration time is very fast which is 7 minutes for 74935 mapping iterations. For the dynamic exploration, it took a very long time to explore the whole 74935 mapping iterations which is around 8 days. Therefore they explored only 56 mapping iterations within 9 minutes. For the FPGA prototyping, they used a Stratix FPGA from Altera. However, the comparison between the performance model and the real implementation is not presented in their work.

In Tab. 2.1, we summarize the presented simulation-based approaches which are classified according to the above identified three categories of platforms. We have estimated the efficiency of these approaches for each type of platforms according to the case studies presented in each related work (● well supported , ◑ partially supported and ○ not supported). However, these works support only fully timing compositional architectures, in which all the behaviors of components are well characterized. In these presented works, the performance models were executed in limited number of simulation runs that might led to insufficient corner case coverage in analyzing of such performance models. Furthermore, shared resource effects of MPSoC systems have been rarely considered. The variability of the execution time of such systems was not taken into account. In the next section, we present the formal approaches which are managed to exhaustively check the system properties.

2.1.2 Formal approaches

In this section, we discuss different formal approaches which have been proposed to analyze multi-core systems and provide real-time and performance bounds. The formal approaches are commonly classified as *analytical approaches* and *state-based approaches*. The analytical approach consists of *holistic* and *compositional* approaches. The holistic approaches create equations representing the system in a whole, while the compositional approaches combine the equations of system components. We first review some holistic approaches in the following.

In [27], Tendell et al. presented a holistic schedulability analysis for fixed-priority tasks with arbitrary deadlines which communicate by messages passing and shared data areas. This schedulability analysis was used to determine the worst-case response time (WCRT) of a distributed task set. They first introduced the equations to compute the WCRT of tasks for a single processor example. A communication model was then provided from the communications schedulability analysis of a multiprocessor system with a shared broadcast bus using the TDMA arbitration protocol. An integration of the computation and communication model was presented to produce analysis for a distributed hard real-time system architecture. The integration model consists of equations that represent the system behaviors. For the case study, they considered a simple system consisting of three processors with 32 tasks and 7 objects that share a broadcast bus. The computation of a hypothetical aircraft control system was considered as tasks which send 14 different messages in the system, of which 13 require transmission across the shared communication bus. They showed the WCRT of tasks based on their equations. However, their example remained very simple and they did not compare their results to any implementation on real hardware platform.

In [28], Yen et al. presented an holistic analysis algorithm to derive tight bounds on execution time for real-time distributed systems. Their analysis algorithm is based on two main techniques: *phase adjustment* and *separation analysis* to compute both upper and lower bounds on the worst-case delay of a task graph executed on multiple PEs. *Phase adjustment* uses a modified longest-path algorithm to model the constraints imposed by data dependencies in the task graph, while *separation analysis* uses a modified max-constraint algorithm to solve the preemtion between tasks. These two phases are repeated to get tighten bound. In one iteration, the results of one phase were used to improve the results of the other. A limit on the number of iterations can also be set if faster delay estimation is desirable. For the case study, they considered a three tasks example running on a 4 CPU system consisting of three Intel i960s and one Lucent DSP3210. They captured the execution time bound for all functions of tasks. Then they applied their algorithms using the lower bound for each period and the upper bound for each computation time on a Sun SS20 workstation. Their algorithms showed the simulation time with a factor of 5348x less than the simulation on 4 CPU system. They presented an interesting algorithm to get a tight bound on execution time which is also a main property of our work. However, they have

not yet consider exhaustive simulation taking into account possible values between lower bound and upper bound. Moreover, they did not consider the shared resource effects. No consideration of MPSoC was presented in their work. The consideration of preemption and task pipelining are interesting but they are not in the scope of our work.

In [29], Pop et al. presented a holistic scheduling and analysis approach of mixed time/event triggered distributed embedded systems which are communicating through mixed static/dynamic bus protocols. Given an application and a system architecture, a correct static schedule for the time triggered (TT) tasks and static messages has to be constructed. They first presented the schedulability analysis for the event triggered (ET) sub-system considering the influence of a given static schedule. Then they constructed the static schedule for TT tasks and static messages. They also discussed some specific optimization issues which can be solved by efficient design space exploration, such as the partitioning of the system functionality into time triggered and event triggered activities, determining the optimal structure of the bus access cycle. Then they proposed a bus access optimization to solve the problem of insufficient bandwidth allocated for the communication of messages between ET tasks. For the case study, they first generated 80 applications. Each application consisted of 80 tasks mapped on 10 processor nodes. They observed the bus optimization results by changing the percentage of event trigger tasks and the processor utilization. The percentage of event triggered tasks was 40% of the total number of tasks for half of the application set and 60% for the other half. Processor utilisation was 60% and 80%. The communication infrastructure was mixed equally dynamic and static protocol. All experiments were done on an AMD athlin 850 MHz PC. The obtained results showed an average improvement of the schedulability between 24% and 34% with an average optimization time is about 2 minutes. However, they did not provide any comparison to other existing scheduling and analysis approaches or real implementation. Since their target architecture is simple, further demonstration in more complex architectures is needed.

These holistic approaches are limited to the system configurations which simplify the equations, such as deterministic TDMA networks. The lack of procedure to solve the holistic equations for arbitrary systems leads to the avoidance of these approaches for real-time analysis of multiprocessor system. The following works represent the second type of the analytical approaches, called the compositional approaches.

In [30], Huang et al. presented formal performance analysis approach for real-time streaming applications on MPSoCs. This approach aims at integrating modular performance analysis into a distributed operation layer (DOL). Distributed operation layer is a platform-independent MPSoC programming environment used for real-time streaming and signal processing applications. In this DOL, they presented their contributions relating to the *generation* and *calibration* of formal performance analysis models. The model generation consists of two phases. First, a meta-model is generated representing the data dependencies of actors in the dataflow process

network and the mapping of the application onto an architecture. Second, this meta-model is refined using the method-specific abstractions and the corresponding code is generated. These two phase are described in the context of DOL design flow. For the analysis, they proposed the modular performance analysis (MPA) which provides hard upper and lower bounds for various performance criteria in a real-time system, such as end-to-end delays, buffer requirements, or resource utilization. For the experiments, they considered three different applications: a producer-consumer, a Motion-JPEG (MJPEG) decoder and a wave field synthesis (WFS) running on a multiprocessor ARM architecture (MPARM) cycle-accurate simulator. The duration of the single steps to generate, calibrate and evaluate the MPA models were minute in the producer-consumer to hours in the MJPEG. In terms of the accuracy, they compared the observed worst-case execution time between simulation and using MPA for two mappings. The results showed the inaccuracy of the performance models because of two main reasons: first, several components in the formal performance analysis do not yield tight bounds; second, the timed simulations only exhibit the worst-case and best-case behavior at component-level but not at system level.

In [31], Henia et al. presented SymTA/S which is a system-level performance and timing analysis approach based on formal scheduling analysis techniques and symbolic simulation. The core of SymTA/S is to couple local scheduling analysis algorithms using *event streams* describing the possible timing of input/output events of tasks. A SymTA/S application model consists of tasks which are mapped and executed on a computation or communication resource. A task is activated due to activating events. The possible timing of activating events is captured by *event models* which can be described by sets of parameters. Their compositional performance analysis approach can requires possible timing of output events for propagation to the next scheduling component. It has the ability to adapt the possible timing of events in an event stream. In order to get tighter analysis bounds, they proposed to use advanced performance analysis techniques, called *system contexts*. These techniques take into account the correlations between successive computation or communication requests. In SymTA/S, the design space exploration for system optimization is presented by defining the search space and the optimization objectives. They explored different parameters, such as the mapping , priority of tasks, time slot sizes, time slot order, scheduling policy and system clock frequency. They combined optimization algorithms with system sensitivity analysis for rapid design space exploration. A *sensitivity analysis* captures the bound of system performance due to the variation of the parameters. They considered two metrics for the sensitivity analysis: the variation of task execution/computation times and the variation of resource speed. For the case studies, they analyzed the execution of one task which issues actuator commands to the physical system and collects routine sensor reading on a system-on-chip example which consists of a micro-controller, a digital signal processor (DSP) and dedicated hardware connected via an on-chip bus. The analysis duration is around 10 seconds

including optimization. They have already applied their approach in case studies in co-operation with industrial partners in telecommunications, multimedia, and automobile manufacturing. Their sensitivity analysis results are promising to capture the variation of system performance from the variability of its components. However, further demonstrations of using their approach on complex architecture and application are not presented.

Besides the presented analytical approaches, many other recent approaches represent system under analysis as a transition system (states and transitions). These approaches belong to the class of *state-based approaches*. We first consider the *generic real-time state-based approaches* which uses timed automata to represent real time systems.

In [50], Norstrom et al. proposed a framework based on model checking techniques for schedulability analysis of event-driven systems. They used timed automata as formal model to describe their system. In their timed automata, a task consists of two parameters: the worst execution time and deadline. It is executed following the earliest deadline first (EDF) scheduling strategy. They assumed that the tasks are non-preemptive. The schedulability problem can be transformed to a reachability problem for ordinary timed automata and thus it is decidable. For the case study, they presented a speed control system. Then they analyzed the schedulability and safety properties of the created model. This approach proposed the idea of creating formal model that can be verified by using UPPAAL model checker tool. However, their case study remained simple and they have not yet considered any complex application running on MPSoCs which might show difficulties to create the performance model. Moreover, they did not provide any results related to the accuracy and simulation time of their approach.

In [51], Lv et al. presented a timing analysis approach which combines abstract interpretation and model checking for multicore system. They considered a multicore architecture where each core has a local L1 cache and share a memory bus. They presented the *abstract interpretation* technique to analyze the local cache behavior of a program running on a dedicated core. Then they constructed a timed automaton (TA) to model the timing behavior of the program accessing to the memory bus. The created TA models were explored using the UPPAAL model checker to find the WCETs of the program. Based on the presented techniques, they have developed a tool that allows automatic generation of the TA models from binary code and WCET estimation for any TA model of the shared bus. For the case study, they analyzed six benchmark programs running on a dual core system using the TDMA and FCFS shared bus. The analyzed results showed that the WCET bounds were tightened by up to 240% for the TDMA bus and 82% for the FCFS bus compared with the bounds estimated assuming worst-case delays for bus accesses. The longest experiment was executed in 3362 seconds. A consideration on a more complex hardware architecture is needed to demonstrate the scalability of such an approach.

We introduce in the following the second class of state-based approach which is the state-based synchronous dataflow graphs (SDFGs) approach.

In [14], Fakih et al. proposed a real-time analysis approach based on model-checking for synchronous dataflow (SDF) applications running on MPSoCs with shared communication resources. The analysis flow starts with the synthesis of a SDF model of computation (MoC) and a model of architecture (MoA) into an annotated parallel hardware/software model, called model of structure (MoS). A model of performance (MoP) which is described as a network of timed automata (TA) representing all actor worst case execution time (WCETs), communication delays, scheduling and communication resource access protocols of the platform are extracted from the synthesis process. The TA templates are configured and instantiated in the UPPAAL framework, taking into account the mapping, timing and platform configuration. Timing requirements are converted into UPPAAL Computation Tree Logic (CTL) queries. Then the TA model can be evaluated by using the UPPAAL model checker. They developed the *sdf2ta editor* which allows to generate automatically UPPAAL model from all needed parameters, such as SDFGs, mapping, hardware constraints. For the case study, a multi-phase electric motor control algorithm mapped to Infineon's TriCore-based Aurix multicore hardware platform with two mappings was analyzed. The performance analysis using UPPAAL model checker provided upper and lower bounds on the execution times which showed tighter bounds in the worst-case response time prediction, compared to an analytical approach. In [52], the simulation time of their approach for the 2 tile platform was from 0.5 second with 8 actors to 1050.1 seconds with 96 actors. In a 4 tile platform, the simulation time was 34.6 seconds with 8 actors and 1038 seconds with 36 actors. This approach is limited in a complex case study because of the exponentially increasing system state space. In our approach, we aim to provide the variation of the execution time instead of providing only the upper and lower bounds.

In [33], Stemmer et al. presented a model-checking based real-time analysis approach for the analysis of timing bounds for finite state machine scenario aware dataflow (FSM-SADF) graphs mapped on a multicore architecture with shared memory. In their work, the FSM-SADFGs are translated to timed automata (TA) semantic model to represent worst-case execution time of actors and shared communication resource. This was done by using the SDF2TA tool. For the case study, they analyzed an MPEG decoder running on dual core architecture with shared memory using UPPAAL model-checker. The best-case and worst-case execution times were captured from the analysis. However, they only presented a simple case study on both application and architecture. The simulation time was also not provided. Beside that they did not compare their work to real implementation results.

In Tab. 2.2, we summarized the presented formal approaches with the consideration of their supported architecture. We also estimated the efficiency of these approaches for each type of platforms according to the case studies presented in each related work (● well supported , ◑ partially supported and ○ not supported). Most of the approaches aimed to get tight execution bound of the program. Some approaches can support the compositional with bounded effects

Classes of estimation approaches	Related work	Fully timing compositional	Compositional with bounded effects	Non-compositional
Holistic approaches	Tendell et al. [27]	●	○	○
	Yen et al. [28]	●	○	○
	Pop et al. [29]	●	◑	○
Compositional approaches	MPA-RTC [30]	●	◑	○
	SymTA/S [31]	●	◑	○
State-based *SDFGs* approaches	Fakih et al. [14]	●	◑	○
	Stemmer et al. [33]	●	○	○
State-based *generic tasks* approaches	Norstrom et al. [50]	●	○	○
	Lv et al. [51]	●	◑	○

Table 2.2 – Classification of timing formal approaches according to the level of compositionality of the adressed platforms.

architectures. However, it is limited in considering the overall execution time including the cache effects. However the biggest problem of these approaches is their analysis time. This long time leads to difficulties to analyze complex systems. In the next section, we present the probabilistic approaches which provide a compromise between accuracy and analysis time compared to the simulation-based and formal approaches.

2.2 Probabilistic approaches

The simulation-based approaches are limited in the ability to explore the complete state space of the model which leads to inaccuracy of the analysis. The formal approaches need long analysis time to explore the whole state space. In this section, we present existing probabilistic approaches which combine simulation and formal approaches to deliver both accuracy and limited exploration time effort. Especially, we consider the use of statistical model checking (SMC) techniques that explore a sub-part of the state-space and provides an estimation to satisfy predefined properties.

PRISM [38] is a probabilistic model checker, a tool for formal modeling and analysis of systems that exhibit random or probabilistic behavior. It has been used to analyze systems from many different application domains, including communication and multimedia protocols, randomized distributed algorithms, security protocols, biological systems, etc., . PRISM can build and analyse several types of probabilistic models (Discrete-time Markov chains (DTMC), Continuous-time Markov chains (CTMC), Markov decision processes (MDPs), Probabilistic automata (PAs), Probabilistic timed automata (PTAs)) and extensions of these models with costs and rewards. Models are described using the PRISM language, a simple, state-based language. PRISM provides support for automated analysis of a wide range of quantitative properties. The property specification language incorporates different temporal logics, such as Probabilistic real-time Computation Tree Logic (PCTL), Linear Temporal Logic (LTL) [53], etc., as well as ex-

tensions for quantitative specifications and costs/rewards. PRISM also includes a discrete-event simulation engine, providing support for approximate/statistical model checking, and implementations of various different analysis techniques, such as quantitative abstraction refinement and symmetry reduction. In [54], Kwiatkowska et al. presented an overview of model checking for both discrete and continuous-time Markov chains (DTMCs and CTMCs) through three real-world case studies: a probabilistic contract signing, dynamic power management in devices and a biological pathway. They verified the probability evolution of one properties by changing some parameters, expected reachability properties, etc. The simulation time is from seconds to several minutes depending on the system complexity. Other case studies can be found on the PRISM website [55]. However, this approach have been rarely applied to analyze MPSoCs.

Bulychev et al. present in [37] a survey of UPPAAL-SMC which is a statistical model checking approach that can analyze performance properties for networks of Priced Timed Automata (NPTA). Priced Timed Automata have its clocks that can evolve with different rates, while being used with no restrictions in guards and invariants. NPTA is generated from different Priced Timed Automatas that communicate via broadcast channels and shared variables. In [37], they used weighted temporal properties expressed in the logic $\text{WMTL}_{\leq}$ (*Weighted Metric Temporal Logic*) to specify properties of NPTAs. Given an NPTA M and a property ψ to be satisfied, a statistical algorithm is used to answer three types of questions:

1. *Hypothesis testing*: Is the probability that M satisfies ψ greater or equal to a threshold $p \in [0,1]$?
2. *Probability evaluation*: What is the probability that M satisfies ψ?
3. *Comparison*: Is the probability that M satisfies ψ_1 greater than the probability that M satisfies ψ_2?

For the qualitative questions (1 and 3), they used sequential hypothesis testing, while for the quantitative question (2), they use an estimation algorithm that resemble the classical Monte Carlo simulation. They demonstrated the utilization of UPPAAL-SMC through some practical case studies (e.g robot control, firewire, bluetooth, etc.,). However, none of these applications relates to the analysis of MPSoCs.

In [41], Nouri et al. present high level modeling and performance evaluation of many core embedded systems. Their approach aims to build a stochastic abstract performance models using statistical inference and model calibration. They propose a formal model based on the *Behavior-Interaction-Priority* (BIP) formal framework to build complex systems by coordinating the behavior of a set of atomic components. Behavior is defined as a transition system extended with data and functions described in C/C++. The description of coordination between components consists of two layers. The first layer describes the interactions between components. The second layer describes dynamic priorities between interactions and is used to express scheduling policies. For the system-level verification, they use stochastic BIP (SBIP) statisti-

cal model checker [56] as performance evaluation technique. The proposed workflow consists of four main steps: (1) *Code generation* produces instrumented code of the implementation of application on target architecture, (2) *Statistical Inference* characterizes the execution traces obtained from the execution to provide probabilistic model of performance data, (3) *Calibrated model* uses the probabilistic model to calibrate the BIP application model, (4) *Statistical Model Checking* analyzes quantitatively the obtained model. They demonstrated their approach on an image recognition case study, called HMAX, this is a hierarchical computational model of object recognition which attempts to mimic the object recognition of human brain. Their target the STHORM architecture platform, a power efficient manycore architecture consisting of a host processor and a manycore fabric. The host processor is a dual-core ARM cortex A9 and the fabric comprises 4213 computing clusters, inter-connected via a NoC. For the performance evaluation, they focused on verifying bounded temporal properties for stochastic systems. They computed the probabilities that the overall execution time is always lower than a given time bound and the variability of processing time of successive lines is always bounded by a threshold. They observed that the time on the calibrated BIP model is about 20% lower than what they obtained on the test-board. For SMC, they use the Sequential Probability Ratio Test (SPRT) algorithm implemented within the SBIP statistical model checker. They observed the probability evolution of the overall execution time for different time bound. Then they checked the overall execution time with different pipelining rates. This method provides a good example of SMC in timing analysis of a manycore architecture. However, the computation and communication time were not separately captured in their approach which led to difficulties in analyzing shared resources effects to the execution time. Following this approach, when we change the application or the mapping of the application on target platform, we need to repeat the implementation and statistical inference steps. This may lead to the problem of a time-expensive modeling effort in this approach.

In [32], Katoen et al. presented a probabilistic model checking method to analyze uncertain Scenario-Aware Dataflow (SADF) models. The uncertainty is SADF comes from both the execution time of processes and the generation of scenarios. In their work, exponentially-timed SADF (eSADF) is considered which is based on asynchronously communicating actors, exponential firing durations and discrete-time Markov chains for sub-scenario selection. The compositional semantics of eSADF are thus captured by using *Markov automata* (MA). The eSADF graphs are then quantitatively analyzed by using algorithms and software tool for verifying Markov automata. However, the state space of the eSADF graphs is very large because of the highly concurrent character of typical data-flow computations. To solve the effect on the state space growth, they proposed a technique *confluence reduction* to reduce the state space. The key of confluence reduction is to detect independent concurrent transitions and to eliminate all non-determinism in the MA semantics of eSADF. They considered two case studies: the MPEG-4

decoder and a face recognition application. In the MPEG-4 decoder, they first showed the effect of confluence reduction, in which the number of states and transitions of the applications were reduced 2 or 3 times. Then they analyzed different properties such as buffer occupancy, throughput and probability to reach certain buffer occupancies with a given deadline. They compared the analysis results using two tools: MAMA and SDF^3. The two tools provided the similar analysis results. For the verification time, the MAMA tool took from minutes to several hours while the SDF^3 only took some seconds. The face recognition application is larger than the MPEG-4 decoder as it exhibits a higher degree of parallelism. They studied the auto-concurrency effect that relates to the simultaneous firing of an actor. This effect can be expressed as the parallel composition of multiple enabled copies of the actor process. They also compared the analysis results of buffer occupancy, expected time and response delay with and without auto-concurrency. They introduced an extension of eSADF with hardware platform for power consumption analysis. In their model of architecture, they adopted a communication assist for tile-based MPSoC to decouple the computation and communication tasks. This made the analysis easier and more predictable. They demonstrated their extension on a MPEG-4 decoder running on Samsung Exynos 4210. The maximal and minimal power consumption were evaluated that showed the feasibility of their extension.

Bao et al. proposed in [43] a statistical model checking based framework that can perform quantitative evaluation of uncertainty-aware hybrid architecture analysis and design language (AADL). AADL supports modeling of hardware and software components for the design and analysis of safety-critical real-time systems such as automotive, avionics, and railway systems. It can be extended to hybrid AADL to support continuous behavior modeling via the hybrid annex. In their approach, they extended the syntax and semantics of hybrid AADL specifications by proposing the uncertainty annex. The uncertainty annex specifies various performance variations such as network delays and sensor inputs and performance requirements. Then they proposed a set of mapping rules that can automatically transform uncertainty-aware hybrid AADL designs into corresponding uncertainty-aware Networks of Priced Timed Automata (NPTA) model. To evaluate the performance of generated NPTA models, they implemented a tool chain that integrates both UPPAAL-SMC and the open-source AADL tool environment OSATE. This tool chain enables the automated performance evaluation and comparison of uncertainty-aware hybrid AADL designs. Two kinds of queries were considered: *performance queries* to check the probability that an expected performance metric can be achieved under a given resource limit, and *safety queries* to check the probability that an unexpected scenario can happen eventually with a given resource constraint. For the case study, they demonstrated the efficiency of their approach via a train control system. In the analysis, they used UPPAAL-SMC with the statistical algorithms that provides a confidence of 98%. For the performance analysis, they obtained a probability interval within 132 seconds. For the safety analysis, UPPAAL-SMC took

17s to have the result. In this approach, they presented an interesting probabilistic analysis using UPPAAL-SMC. However, they have not introduced any case study relating to MPSoCs.

In [57], Chen *et al.* presented an UPPAAL-SMC based framework to evaluate the performance of Task Allocation Scheduling (TAS) strategies under time and power constraints variation for MPSoC. TAS strategies aim to maximize the utilization of available processing elements (PEs) while satisfying various design constraints, such as response time, power. Their approach adopts the Priced Timed Automata (PTAs) as the formal model. The workflow first defines the application task graph, the MPSoC architecture and design constraints. Then they define different mapping rules which can automatically convert the generated TAS instances with variation information into Networks of PTA (NPTA) models. In the meantime, design constraints can be translated into properties which enable queries for performance evaluation. The generated NPTA models are then analyzed and evaluated by UPPAAL-SMC model checker to find the satisfying TAS instances. If there is no satisfying TAS instance, their framework can automatically iteratively change the design architecture and constraint parameters, regenerate TAS instances and perform re-evaluation. If there are multiple satisfying TAS instances, the best one will be reported. For the case study, they evaluated a synthetic example of 22 node task graph. They used the gaussian distribution to represent the power distribution and the execution time of PEs. Evaluation results were done under different power and architectural constraints. They obtained a probability interval with a confidence of 95% with a simulation time of 2067 seconds. In this approach, they introduced an interesting utilization of UPPAAL-SMC for the evaluation of TAS strategies for MPSoC. However, their case study remains simple in both application and architecture. Moreover, the effects of shared resources have not been taken into account in this approach.

In [40], Ngo et al. presented an analysis approach based on Plasma Lab [59] statistical model checking techniques for SystemC models. This method allows both qualitative and quantitative analysis to estimate a probability to satisfy temporal properties of SystemC models. Their SMC-based verification tool implementation consists of two main components: a *monitor and aspect advice generator* (MAG) and a *statistical model checker* (SystemC Plugin). Running the verification framework consists of four steps: (1) A configuration file containing necessary information (e.g., temporal properties, libraries, etc.,...) is generated by MAG to get monitor files, an Aspect-Advice file and other necessary information, (2) The Aspect-Advice file, the monitor files and the SystemC model are then instrumented by using AspectC++ [60], (3) The instrumented SystemC models are then compiled and linked to the patched SystemC [61] to build an executable model, (4) The temporal properties are verified in Plasma Lab by observing the execution traces obtained from the simulation of the executable model. The statistical model checker is implemented as a plugin of Plasma Lab that establishes a communication, in which the generated monitor transmits execution traces of the model-under-verification (MUV). When

Approaches	**Dataflow**	**Formal model**	**Analysis tool**	**Application**	**Properties**	**Hardware platform**
PRISM [54]	X	DTMC, CTMC	PRISM	a probabilistic security protocol, dynamic power management and a biological pathway	PCTL	X
Nouri et al. [42, 56]	BIP flow	BIP model	SBIP	HMAX object recognition	Bounded temporal properties	STHORM with shared interconnect and shared memories
Katoen et al. [32]	eSADF	MA	Mama and SDF^3	MPEG-4 decoder and Face recognition	Expected time, long-run average and timed reachability	Samsung Exynos 4210 with a shared bus
Bao et al. [43]	AADL	NPTA	UPPAAL-SMC	Train control system	Performance and safety queries	X
Ngo et al. [40]	X	CTMC	Plasma Lab	FIFO channel and embedded control system	Bounded Linear Temporal logic (BLTL)	X
Chen et al. [57]	X	NPTA	UPPAAL-SMC	a synthetic example	Performance evaluation under constraints	8 PEs MPSoC with shared interconnect and shared memory
Bulychev et al. [37]	X	NPTA	UPPAAL-SMC	Robot control, firewire, bluetooth	WMTL	X
Proposed approach [58]	SDFG	X	PlasmaLab-SMC	Sobel filter and JPEG decoder	Bounded Linear Temporal logic (BLTL)	Xilinx ZC702 with private cache, shared bus and shared memory

Table 2.3 – Specifications of the presented probabilistic approaches

a new state is requested, the monitor reports the current state to the plugin. The length of the traces depends on the satisfaction of the formula to be verified, which is finite because the temporal operators are bounded. Similarly, the required number of execution traces depends on the hypothesis testing algorithms in use, such as sequential hypothesis testing or 2-sided Chernoff bound [62]. For the case study, they introduced two case studies: a simple case study with a FIFO channel and an embedded control system for demonstrating the use of their verification tool. They obtained a probability to satisfy their properties with a confidence of 98%. The simulation time for the FIFO channel example is about minutes. For the embedded control system is around 2 hours, in which 90 properties were verified. However, their case studies remain simple and they have not yet applied their approach on MPSoCs.

In Tab. 2.3, the specification of probabilistic approaches is presented. Most of them validated the feasibility of their approaches by presenting simple case studies without considering a real hardware platform. Only the approach in [42] presented a comparison between the analysis and real implementation results. However, the adoption of shared resources effects in the performance analysis have not yet been considered. This can be explained by the large amount of effort to setup this approach. Creation of probabilistic models remains a difficult task and is not well supported for multi-core systems. These presented works for performance evaluation of MPSoC lead to several conclusions:

1. A **systematic workflow** to apply the probabilistic approaches is needed to evaluate the performance of MPSoC systems.
2. Further consideration of **probabilistic modeling approaches** is needed to represent the shared resource effects of such MPSoC systems.
3. Further study on **Statistical Model Checking approach** for performance evaluation is needed to capture the effects of statistical algorithms to the performance evaluation.

In Tab. 2.3, we take into account these points in our work which will be described latter, especially in chapter 3 and chapter 5. In the next section, we discuss different methods that aim to accelerate the simulation speed and/or improve the accuracy.

2.3 Simulation speed/Accuracy improvement methods

Different criteria are needed to be considered in the performance evaluation, such as the accuracy, the simulation speed and the scalability. The simulation speed depends on the abstraction level of models. The accuracy of the whole system depends on the accuracy of the characterization of each component. In this section, we discuss some existing work which relates to the simulation speed/accuracy improvement.

In [63], Schiner et al. present a result-oriented modeling (ROM) technique for transaction level model (TLM). In their approach, ROM is considered as a "black box" which eliminates

intermediate states and produces only the end result of the process. At the starting moment of a process, ROM uses an optimistic approach to predict the termination time and final state of the process. However, in the execution of the process, a disturbing influence can occur that may change the system state. ROM checks at the end of the predicted time whether there is a disturbing influence. In that case, ROM retroactively adjusts to the new conditions and takes corrective measures. In their work, they present two communication busses applying ROM concept: an on-chip multiplexed bus system with a centralized arbitration scheme Advanced Microcontroller Bus Architecture (AMBA) and an off-chip serial bus with decentralized arbitration Controller Area Network (CAN). They define three classes of granularity applicable to any bus protocol, and match these granularity classes to three model types: the Transaction Level Model (TLM) that models user transactions, the Arbitrated Transaction Level Model (ATLM) at bus transaction granularity, and the bus cycle accurate Bus Functional Model (BFM). Further detail about these classes can be found in [64]. They implemented for both two busses using the SpecC system level design language (SLDL). Three aspects were evaluated: (1) the accuracy, (2) the simulation speed and (3) the number of prediction updates that are needed by ROM. They used a multi-node setup which consists of two senders and two receivers. For the accuracy, they obtained 100% accuracy in timing for both two busses. For the simulation speed, they measured the simulation time of the whole system comparing ROM with TLM, ATLM and BFM. In the AMBA bus, ROM is as fast as TLM, three orders of magnitude faster than the BFM and one order of magnitude faster than the ATLM. In the CAN bus, ROM is 2x slower than TLM, 24x faster than ATLM and 12700x faster than the BFM. For the prediction updates, the probability of updates drops exponentially with the number of updates in both cases. This approach proposed an impressive level of accuracy by predicting the termination time and final state of the process. The simulation time is as good as other TLM approaches. However, the presented case studies remain simple. Further demonstration of this approach for applications running on MPSoC is needed.

In [65], Bobrek et al. proposed a stochastic contention level (SCL) simulation for single chip heterogeneous multiprocessors (SCHM). SCL simulations capture contention effects of shared resources by using execution blocks. Each execution block represents thousands to millions clock cycles. This approach aims to create a Statistical Contention Model (SCM) which enables the high-level simulation to estimate the impact of contention for shared resource accesses without access to the clock-cycle-level information within the system. They proposed three *access attributes* that can summarize the access pattern behaviors observed at the shared resources: *Average Requested Utilization*, *Access Balance*, and *Thread Concurrency*. The SCL design flow starts with a brief cycle-accurate simulation which captures shared resource access patterns and collecting the resulting contention information. Then the access attributes and the access/contention information are extracted and used to train a non parametric regression model. The

trained regression contention model can be used to predict contention within a MESH simulation, gaining a significant simulation performance advantage. MESH simulator allows designers to answer questions about how the numbers and types of processors and communications resources, the scheduling decisions, and the software tasks affect the overall performance of SCHM systems. The statistical modeling approach allows MESH to simulate heterogeneous multiprocessor systems significantly faster than cycle-accurate simulators, while still accurately capturing contention. For the evaluation of their approach, they selected several multimedia, encryption, compression, and signal processing applications from SPEC2000 and MiBench benchmark suites: adpcm (adaptive differential pulse code modulation), FFT, jpeg, gzip, rijndael (encryption), rsynth (speech synthesis), and crc (cyclic redundancy check). The SCL approach results in speedups of 40x over cycle-accurate simulation, with average simulation errors of less than one percent with 95% confidence intervals of about ±3%, providing a unique combination of simulation capabilities, performance, and accuracy. This significant increase in simulation performance enables the system designers to explore more of the design space than possible with traditional simulation approaches. This approach proposed a very interesting example that delivers both accuracy and simulation speed. However, the key limitation of this approach relates to the dependence on cycle-accurate training. When the system changes, the contention model has to be retrained. This leads to time-expensive modeling effort of this approach.

In [66], Le Nours et al. presented a hybrid simulation approach for fast and accurate timing analysis of MPSoCs considering communication resources conflicts. They considered that some parts of a system model can be abstracted and replaced by an equivalent executable model. This executable model presents the same evolution as the abstracted elements from an external viewpoint, but the number of events managed by the simulation kernel is reduced. The equivalent model incorporates the expressions of the synchronization instants among the abstracted elements. They adopted the timed Petri net formalism to express the time dependencies among the abstracted elements and the related synchronization instants. This formal representation is used to compute the synchronization instants during simulation and to determine the execution order between the abstracted processes. This approach considers the data dependencies which can be formulated through the created timed Petri net and influence execution of the equivalent executable model. The key point of their approach is that the synchronization instants are instantaneously computed during simulation. In the case of contention on shared resources, a disturbing influence can cause the computed synchronization instants to become inaccurate because delays due to access conflicts at shared resources are not simulated. The computed synchronization instants potentially need to be updated to correctly reflect the influence of shared resources. Thus they proposed a correction technique to update computed synchronization instants influenced by shared resources. For the case study, they first validated their approach in a system model made of 20 functions, two processors and shared communication resources. The

simulation speed-up factor is 14.5 with no loss of the accuracy. For a communication receiver case study, they obtained a simulation speed-up by a factor of 4 with no loss on timing accuracy. In their work, they demonstrated their approach on the computation part. Further application of this approach on the communication part should be adopted which is strongly influenced by the contention on shared resources.

In summary, all of these approaches provided a very good level of accuracy. The approaches in [65] and [66] presented a good simulation speed up. However, their case studies are still simple or have not yet been applied for MPSoC sytems. In our work, we aim to provide a fast yet accurate message level communication bus model for timing prediction of applications on MPSoC systems which will be presented in Chapter 4.

Chapter 3

Working environment

In the previous chapter, we have presented different timing analysis approaches and simulation speed/accuracy improvement methods for MPSoC systems evaluation. In this chapter, we present the developed modeling and analysis workflow. This workflow is used to evaluate the efficiency of probabilistic models for timing property analysis of MPSoCs. We first present an overview of the proposed workflow. We give the definition of our system which is based on a model of computation (MoC), a model of architecture (MoA), a mapping model and a measurement infrastructure. The created probabilistic models use the separation between computation and communication aspects. Preliminary analysis results are presented to validate the feasibility of the proposed approach. At the end of this chapter, we propose the improvements corresponding to our contributions in Chapter 4 and Chapter 5.

3.1 Overview of the proposed workflow

The proposed workflow is presented in Figure 3.1. Its objectives are twofold: (1) It first provides means to create probabilistic models based on measured data. (2) It then allows to evaluate the accuracy and simulation speed of probabilistic models used for timing properties analysis of MPSoC systems.

The workflow consists of three main parts. First, an *implementation and measurement* phase is used to create hardware and software models from a model of architecture and a model of computation with respect to the predefined specifications. These hardware and software models get implemented on a real hardware FPGA platform. Computation and communication delays needed for the next part are measured by a measurement infrastructure.

Second, we consider a separation of the computation time and communication time modeling in the *probabilistic modeling* part. This separation is allowed due to the use of Synchonous Dataflow (SDF) model of computation (detailed in Section 3.2.1). For the computation modeling, the computation time can variate depending on either software or hardware resources influences. From the perspective of software resources, different input data can follow different branches of a given actor's algorithm that can lead to different computation durations. From the perspective of hardware resources, the interferences caused by concurrent accesses to shared resources can lead to variations in both computation and communication time. These variations

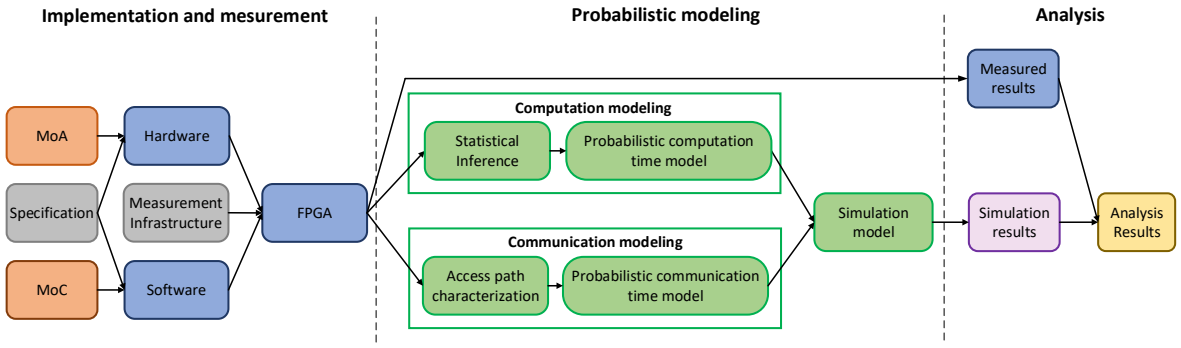


Figure 3.1 – The established workflow to evaluate the efficiency of the probabilistic approach for timing property analysis of MPSoCs.

can be captured by measurement. The measured computation and communication delays are used in the modeling process to create the computation and communication time models. For the probabilistic computation time model, we consider a probability distribution to represent the variation of measured computation delays. For the communication model, we also measure the delays to access shared resources. Due to the possible contentions at shared resources according to mapping and scheduling, the communication delays can also variate. These measurements are then used to calibrate a probabilistic communication time model. The created probabilistic computation and communication models get integrated into a specific language to have a probabilistic simulation model. This simulation model is then executed and analyzed in the next part.

Third, in the *analysis* part, the probabilistic simulation model gets executed within a defined number of iterations to have analysis results. These analysis results are compared with the measured results from the implementation on real hardware platform, for example a FPGA, to evaluate the efficiency of the probabilistic simulation model. Different criteria can be taken into account in this evaluation: the accuracy of the created simulation model, the simulation time and the scalability of the proposed approach.

3.2 System model

In our modeling workflow, the considered system model is made of descriptions of an application, a platform and a mapping. A measurement infrastructure is used to measure the timing behaviors of the system. The system model is illustrated in Figure 3.2.

3.2.1 Model of computation

We use synchronous data flow (SDF) as model of computation [67]. This model describes the application data flow between *actors* via communication *channels*.

Definition 1 *(SDFG) A synchronous dataflow graph is defined as* $SDFG = (\mathcal{A}, \mathcal{C})$*, which consist of:*

1. *a finite set* $\mathcal{A}$ *of actors* A*.*
2. *a finite set* $\mathcal{C}$ *of channels* C*. A channel is a tuple* $C = (R_i, R_o, B)$ *with*
   (a) *The input rate* R_i *defining the number of tokens that can be written into the channel during the write phase of an actor.*
   (b) *The output rate* R_o *defining the number of tokens that will be read from the channel during the read phase of an actor.*
   (c) *The buffer size* B *in number of tokens.*

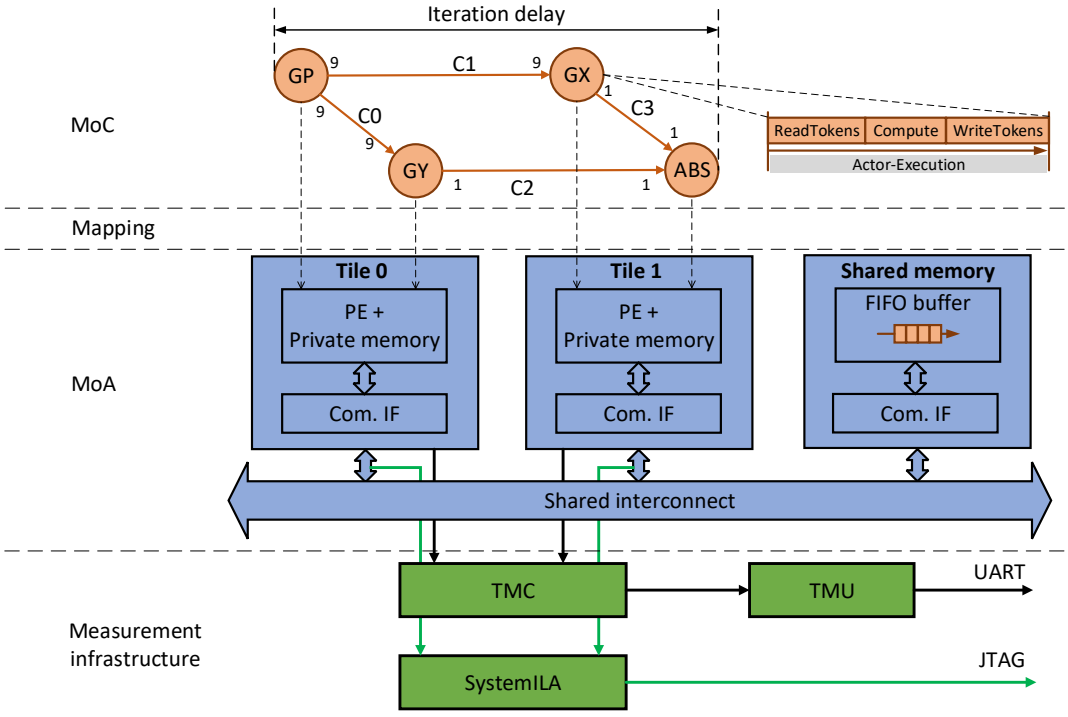


Figure 3.2 – Illustration of the system model that consists of (a) the model of computation (MoC), (b) the model of architecture (MoA), (c) the mapping model and (d) the measurement infrastructure. The delay of application iteration (iteration delay) is observed.

The SDF graph of the Sobel filter example (in brown) is given in Figure 3.2(a). It consists of four actors (*GP*, *GX*, *GY* and *ABS*) and four channels (*C0*, *C1*, *C2* and *C3*). The input

and output rates are presented as the integer values on each channel. For example, the channel *C0* has the input token rate $R_i = 9$, the output token rate $R_o = 9$ and the buffer size $B = 9$. Each actor has a strict separation of computation (*Compute*) and communication (*WriteTokens*, *ReadTokens*) statements. As illustrated in the example of the actor GX in Figure 3.2(a), these functions are executed consecutively. During the *ReadTokens* phase of an actor, tokens get read from a channel buffer. During the *Compute* phase, the sequence of processes are executed. During the *WriteTokens* phase, tokens get written into a channel. As FIFO access is blocking, an actor can only switch to its computation phase after it reads all tokens from all incoming channels. After the computation phase, the actor switches into write phase to write all tokens to the outgoing channels. In our work, all channels buffers are mapped onto shared memory, while the actors use the private memory of the processing element they get executed on. One important observation is that this setup is fully composable. The computation phases of any actor can be considered independent from communication phases.

Communication is done by two functions used in the implemented software: *WriteTokens* that writes tokens onto the shared memory, and *ReadTokens* that reads from shared memory. Figure 3.3 gives the observed state-transition diagram of the write process of n tokens. The states in grey denoted by *polling*, *writing* and *update buffer usage* correspond to the situations when the PE accesses to the shared resources. The states denoted by *initialization*, *inter-polling stage*, *pre-writing*, *inter-writing stage*, *post-writing* correspond to the execution on the PE without interfering with another resource. At the beginning of the tokens write process, once initialization

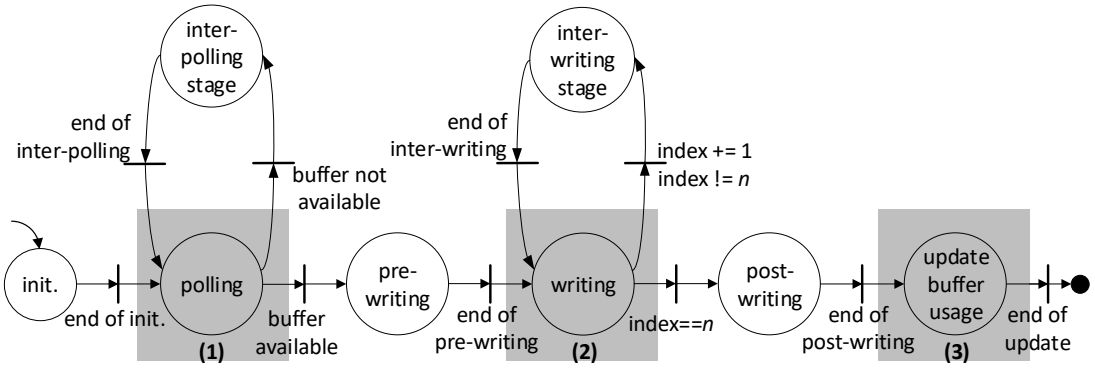


Figure 3.3 – State-transition diagram of the write process of n tokens to the shared memory.

done, the polling step aims to verify the availability of the buffer. If the buffer is ready to write, the actor prepares to copy the data onto the buffer. Otherwise, it continues polling until the condition is satisfied. A delay exists between two consecutive polling states, it corresponds to the duration of the inter-polling state. The tokens write phase finishes when all the data are copied onto the buffer. At the end of the tokens write process, the PE finally updates the usage

of the buffer to indicate that the data is ready for the read process. The tokens read process is implemented with a similar sequence of elementary states.

3.2.2 Model of architecture

The hardware platform follows the definition of a tile-based platform as given in [68, 69]. A simple example of this platform (in blue) is shown in Figure 3.2(b).

Definition 2 *(Tile) A tile is a tuple* $T = (PE, M_p, IF)$.

1. PE *is the processing element.*
2. M_p *is the private memory only accessible by the processing element.*
3. IF *is the communication interface that ensures the accesses to the communication resources (shared interconnect).*

A tile can execute software without interfering with other tiles as long as the software only accesses the private memory. While a tile can be connected to multiple interconnects, we assume that every memory is connected to only one interconnect. Furthermore we assume that tiles can only communicate via a shared memory. We adopt a tile-based platform organization where each tile consists of a processing element (PE) with private instruction and data memories, and a communication interface (IF). A tile can execute software without interfering with other tiles as long as the software does not explicitly access shared resources.

Definition 3 *(Execution Platform) An execution platform is defined as* $EP = (\mathcal{T}, \mathcal{M}_s, \mathcal{I})$ *where*

1. $\mathcal{T}$ *is a finite set* $\mathcal{T}$ *of tiles* T *as defined in Def.2.*
2. $\mathcal{M}_s$ *is a finite set of shared memory* M_s.
3. $\mathcal{I}$ *is a finite set of shared interconnect* IF.

An execution platform consists of a finite set of tiles, a finite set of shared memories and a shared communication bus. In the scope of our work, we consider a first-come-first-served (FCFS) bus arbitration protocol. Shared memory is used to temporary store channel buffers exchange between actors of the application.

3.2.3 Mapping model

We present in Figure 3.2(c) the mapping example of the Sobel filter application on two tile hardware platform. The two actors *GP* and *GX* are mapped to the *Tile 0* while the two actors *GY* and *ABS* are mapped to the *Tile 1*. All the channels of the modeled application, denoted by C_0 to C_3, are mapped to the shared memory. Then the tiles have to access to the shared memory to read or write data.

3.2.4 Measurement infrastructure

For the experiments, we considered the Xilinx ZC702 Evaluation Board as the real platform. A tile was implemented as a Xilinx Microblaze processor with local instruction and data memories. The MicroBlazes connect to the shared memory BRAM via an AXI4LITE interconnect.

For our performance models we need to characterize the duration of the computation phase of each actor as well as the duration of the communication between actors. In Figure 3.2(d), the measurement infrastructure (in green) is illustrated. For the computation phase, we measure computation time in clock cycles of actors. We use the same measurement infrastructure as presented in details in [70]. It consists of two main components: *Time Measurement Unit* (TMU) and *Time Measurement Controller* (TMC) . TMU is basically a counter with the same cycle rate of the processors. The counter can be started and stopped from any tile individually without interference. When the counter got stopped, it sends the counter value via an UART to a host computer. For this method, it was important that all components were clocked with the same frequency. The management of the individual start/stop signals from the tiles are managed by the TMC.

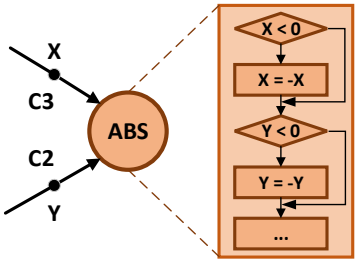
For the communication phases, the adopted measurement infrastructure called Xilinx System Integrated Logic Analyzer (SystemILA) is able to monitor and store the signals that are relevant to estimate the communication durations. It is used to measure the durations of each elementary states involved in communication, as for example the ones identified in Figure 3.3, and delays caused by contention at shared resources. SystemILA connects to the AXI4LITE bus and observes the signals that relate to the communication process. For the writing process, the write address valid signal (*AWVALID*) indicates that the channel is signaling valid write address and control information. The write response valid signal (*BVALID*) indicates that the channel is signaling a valid write response. For the reading process, we observed the signals *ARVALID* and *RVALID*. These signals are sent to the host computer via a JTAG cable. Further details about AXI4 protocol can be found in [71].

3.3 Probabilistic modeling of computation and communication times

In this section, we first present the computation and the communication time models. Then these models get integrated into a simulation model written in SystemC language. As presented earlier in Figure 3.2, the execution of an actor in SDFG model of computation consists of three phases: ReadTokens, Compute and WriteTokens. Since the computation and communication phases have no influence to each other, we can separately characterize these phases.

3.3.1 Computation time modeling approach

The computation time model is created by considering the actor's compute phase. We present in the following the software and hardware resources influences which cause the variation of the computation time. From the software perspectives, an actor's algorithm can have different branches. For each input data set, the actor is executed following a corresponding branch. Therefore, these branches can provide different execution delays. It leads to the variation of actor's computation time. An example is given in Figure 3.4, in which we present the computation algorithm of the *ABS* actor of Sobel filter application in (a). This actor computes the gradient magnitude of two gradient estimations GX and GY. Its execution depends on the combination of input values from the channel C3 (token X) and channel C2 (token Y). In Figure 3.4 (b), four different branches are listed with their measured execution delays by running the Sobel filter application on a real hardware platform FPGA. The computation time variation of the ABS actor is then implemented in a specific programming language in Figure 3.4(c).



(a) Computation of ABS actor

Branches	Delay (cycles)
X ≥ 0 ∧ Y ≥ 0	53
X < 0 ∧ Y ≥ 0	62
X ≥ 0 ∧ Y < 0	62
X < 0 ∧ Y < 0	71

(b) Measured computation time

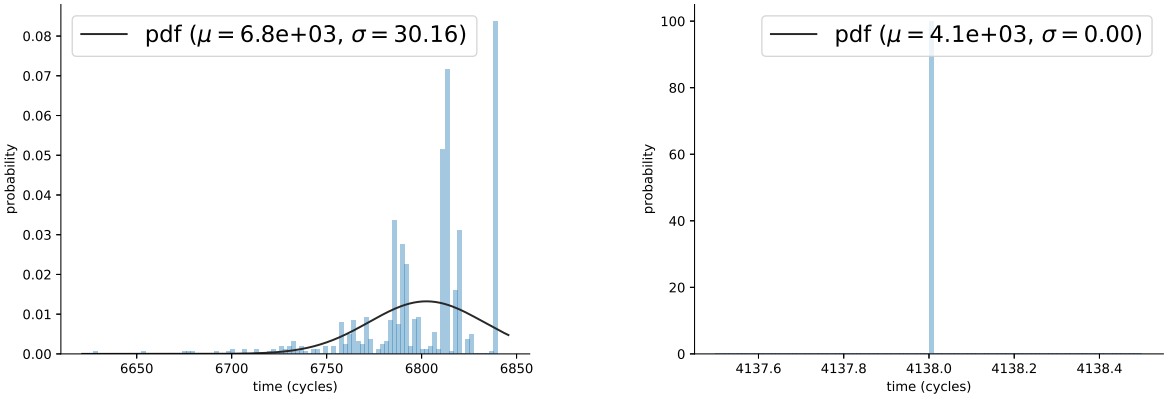
```
if (X < 0 and Y < 0)
    wait(71);
else if (X < 0 or Y < 0)
    wait(62);
else
    wait(53);
```

(c) Implementation of ABS actor

Figure 3.4 – Illustration of different execution branches of the ABS actor (a), the measured execution time (in cycles) (b) and (c) its implementation in specific programming language.

From the hardware perspectives, the micro-architecture can also provide different computation time. Specific hardware accelerator such as Floating Point Unit (FPU) or Multiplier (MUL) can change the computation time. For example in the Sobel filter application, the GX actor contains the convolution operation in their computation process. We illustrated the computation time variation of the GX actor depending on the hardware accelerator MUL. In Figure 3.5 (a), we do not apply MUL. Thus the computation time of GX strongly depends on the input data. This leads to a variation that can be represented using a gaussian distribution with *mean* $\mu = 6800$ cycles and *standard deviation* $\sigma = 30.16$ cycles. In Figure 3.5 (b), the MUL reduces the computation time of GX. In this case, we measured a constant value of this computation time which is of 4138 cycles.

In Figure 3.6, we present the workflow to create the probabilistic computation time model which is based on measurement and statistical inference technique. The workflow starts with



(a) GX without MUL

(b) GX with MUL

Figure 3.5 – Computation time of the GX actor depending on the hardware accelerators (a) without MUL and (b) with MUL.

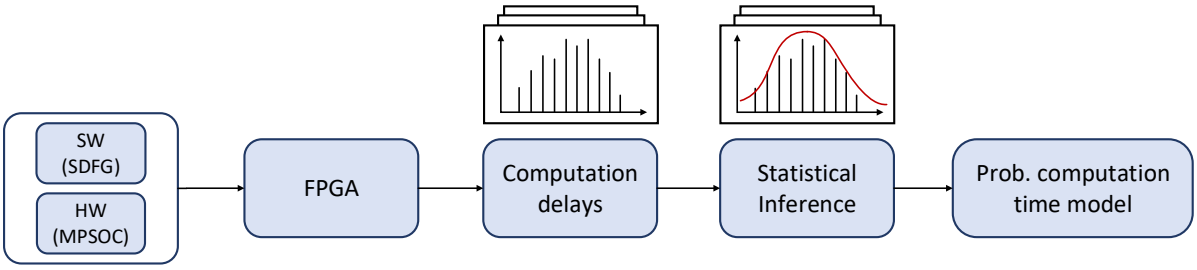


Figure 3.6 – Illustration of the probabilistic computation time modeling.

the implementation of software and hardware models on a real hardware platform. Then we measure the computation delays of all the actors within a defined number of iterations. The computation delay of actor does not change while mapped on different processors of a same type. This computation delay executed on a specific processor is measured once and then used in different mappings. A statistical inference technique is then used to capture the variation of the measured computation delays. This variation is then represented as a probabilistic distribution function (PDF) such as the gaussian or uniform distribution. Once the probabilistic computation time model is created, it can be used for different mappings of the considered application.

3.3.2 Communication time modeling approach

The communication time model is created from the two main functions: WriteTokens and ReadTokens. The variation of the communication time comes from the interferences of concurrent accesses to shared resources. At this stage of our work, we try to explain why the duration of

Delay	Signification	Delay	Signification
t_i^w	Write initialization delay	t_i^r	Read initialization delay
t_p^w	Write polling delay	t_p^r	Read polling delay
t_{pl}^w	Write inter-polling delay	t_{pl}^r	Read inter-polling delay
t_{pr}^w	Pre-writing delay	t_{pr}^r	Pre-reading delay
t_{wr}^w	Writing delay	t_{rd}^r	Reading delay
t_l^w	Inter-writing delay	t_l^r	Inter-reading delay
t_{po}^w	Post-writing delay	t_{po}^r	Post-reading delay
t_u^w	Write update buffer usage	t_u^r	Read update buffer usage

Table 3.1 – Signification of the elementary delays.

WriteTokens and ReadTokens are variable.

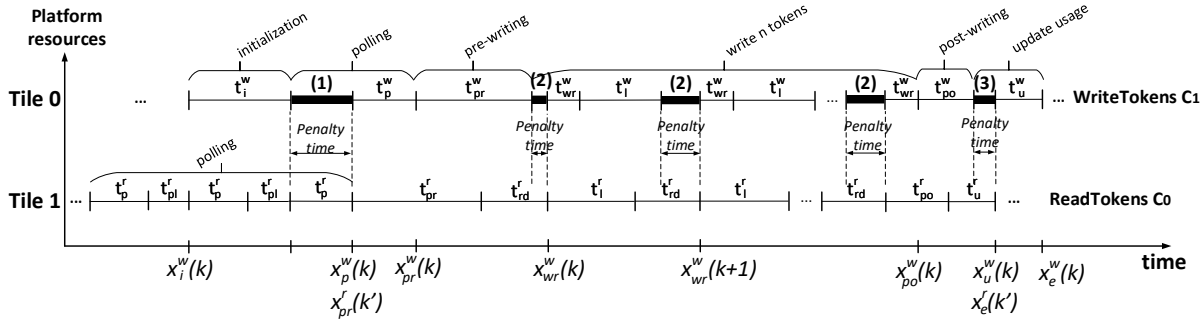


Figure 3.7 – Illustration of writing n tokens on channel C1 and reading n tokens from channel C0 at transaction level. The elementary delays and penalty delays caused by the contention are exhibited.

In Figure 3.7 we present two simultaneous communications to a shared memory through a shared communication bus. It illustrates the situation of writing n tokens on channel C_1 and reading n tokens from channel C_0 with the protocol described in Figure 3.3. In Figure 3.7, each elementary step of the communications is described with a specific duration called *elementary delay*. In the following, elementary delays with an upper index w are related to an elementary step in the tokens write process and ones with an upper index r are related to an elementary step the tokens read process. The same notation is used to denote transition instants between elementary steps. For example, the durations of the *polling*, *writing*, *inter-writing*, and *update buffer usage* steps are respectively denoted by t_p^w, t_{wr}^w, t_l^w and t_u^w. Each instant denoted by $x_j^w(k)$ (respectively, $x_j^r(k)$) corresponds to the k^{th} transition instant between two successive elementary steps of the tokens write process (respectively, the tokens read process). The instant denoted by $x_u^w(k)$ corresponds to the k^{th} transition instant between the *post-writing* state and the *update buffer usage* state. In Tab. 3.1, we summarize the signification of the elementary delays for both

writing and reading process.

Contention effects at shared resources are illustrated in Figure 3.7. The situations denoted by (1), (2), and (3) are related to the states highlighted in grey in Figure 3.3. The situation denoted by (1) corresponds to a contention due to two concurrent polling steps. The communication bus arbitration allows the polling step of the tokens read process to execute first, the polling step of the tokens write process has to wait until the end of the read polling phase. Situation (1) could also correspond to the situation where the polling step interferes with an elementary reading or update buffer step. The situation denoted (2) corresponds to the situation where the elementary writing phase is delayed due to a simultaneous elementary reading phase. The situation denoted by (3) corresponds to the situation where the C_1 update usage phase is delayed due to simultaneous C_0 update usage phase.

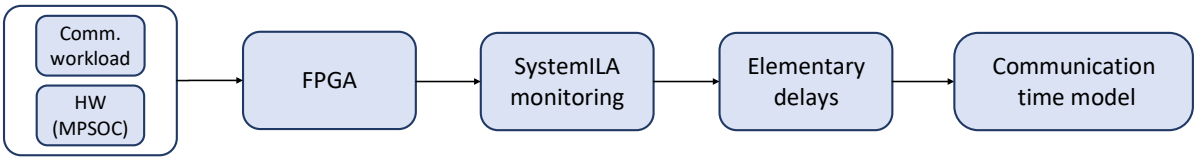


Figure 3.8 – Illustration of the communication time modeling.

In Figure 3.8, we present the workflow to create the communication time model. In Section 3.2.1, we argued that communication and computation do not interfere each other and they can be considered independently. We thus create a *communication workload* that consists only of the communication phases of an application. We then implement this *communication workload* on a hardware platform. The elementary delays are measured using the SystemILA [72] IP component provided by Xilinx which observes the communication process on the bus. In fact, these elementary delays do not depend on the application mappings, we only need to do the measurement once. In our communication time model, we implement a bus arbitration policy to manage the contention situations. Different mappings cause different contention situations. These contention situations cause additional delays called *penalty times*. Thus the variation of the communication time depends on mappings. Finally, our communication time model is built from the measured elementary delays and the implementation of bus arbitration policy.

3.3.3 Simulation model

The simulation model is implemented using the SystemC language [73]. It allows to model systems made of hardware and software resources at different abstraction levels such as the register transfer level (RTL) or transaction level (TL). The RTL gives the description of bit accurate hardware resources according to a reference clock signal, while the TL considers description of resources with a higher level of abstraction than the RTL. A discussion of possible

levels of abstraction can be found in [74]. SystemC is built as a C++ language library and can be compiled to produce an executable model. An event-driven simulation kernel is provided in SystemC to control this executable model.

A SystemC model is built from basic blocks called *modules* which can be described with the *SC_MODULE* macro [75]. The *modules* are classes in C++. A module contains *ports* that allows the module to communicate with its environment and a set of *processes* running concurrently to describe the functionality of the module. The communications between modules and between processes within modules are done through *channels* via their ports. A channel can be either a primitive channel or a hierarchical channel. Basically, channels are buffers that contains data and can generate events in the simulation kernel whenever the contained data changes.

Processes are basic units of functionality in SystemC which can be simulated concurrently. A process can be either a *method* or a *thread* depending on the different needs in expressiveness and simulation performance. Once a method process is triggered, it always executes from the beginning to the end, while a thread process can be suspended by calling the *wait()* function or any of its variant. The thread process remembers its point of suspension. When the execution is resumed, it will continue from that point. This shows a greater expressiveness for thread processes than method processes.

SystemC *sc_event* classes determine whether and when to trigger or resume the execution of a process. An event reports the change of state in a process by using *notification*. When the event fires, the scheduler is informed of which processes to trigger. Further details about SystemC can be found in [17].

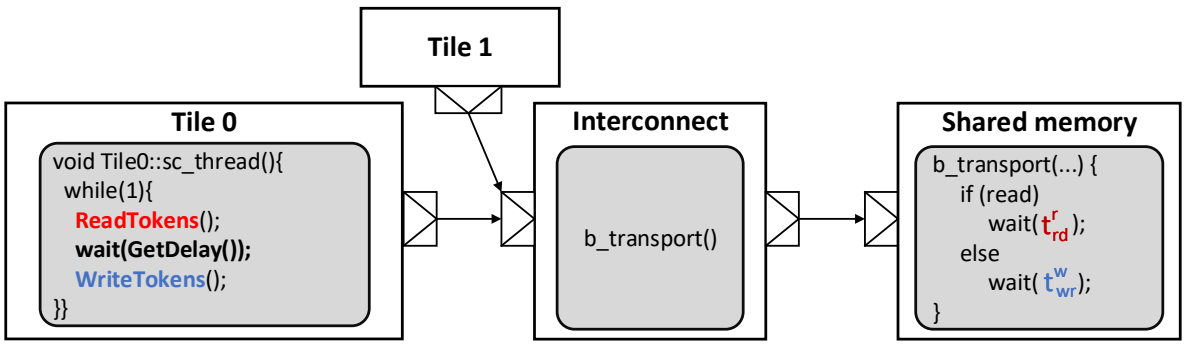


Figure 3.9 – Illustration of a SystemC model with two tiles.

In Figure 3.9, we present the organisation of our system model. It corresponds to the description of the mapped actors and channels on the resources of a two-tile execution platform. The SystemC model consists of three main parts. The *Tile* modules describe the execution of

the actors mapped on processing elements. The timing behaviors of the actors are described in an *SC_THREAD* of *Tile* modules. The computation phase of an actor is described with a *wait* statement calling the `GetDelay` function. In each simulated iteration of the system, a computation time of actor gets selected from a list of observed delays. This is realized by reading a text file that provides the raw measured computation time of this actor. We randomly select data from the measured delays, taking care that no element of this list gets selected twice. We refer to this random selection of recorded execution times as *injected data* in the following experiments. The `GetDelay` function can select the computation time of the actor following the distribution function (gaussian or uniform) representing the measured computation time. The use of the *uniform* and *gaussian* distribution can be found in the GNU Scientific Library (GSL) [76]. The parameters for these distributions are derived from the measured delays. The communication phase of actors is based on two functions `ReadTokens` or `WriteTokens`. These two functions call the *Interconnect* module.

The *Interconnect* module manages the connection of tiles to shared memories. This module arbitrates the accesses to the shared memories of the tiles. The `ReadTokens` or `WriteTokens` functions are implemented following the protocol, as described in Figure 3.3. Whenever the functions `ReadTokens` or `WriteTokens` are called, the function *b_transport* blocks the access to the shared memory for a duration to read or write token.

The *Shared Memory* module simply distinguishes the access time to memory to read/write one token or the reading delay t^r_{rd} and the writing delay t^w_{wr} as presented in Tab. 3.1. Depending on the access type, it proceeds the simulation time by the related time. For reproduction of our simulations, we provide git repositories with SystemC models [1]

3.4 Evaluation of the proposed framework

In this section, we present the preliminary results of the proposed workflow. We first describe our two image processing case studies: Sobel filter and JPEG decoder. Then we present a 7-tile heterogeneous platform with different mappings for each case study. Finally, we present and discuss the analysis results.

3.4.1 Case studies

Two different use-cases and various possible mappings were considered to evaluate the efficiency of the created models in terms of accuracy and analysis time. For every considered use-cases and mappings we predicted the duration for one iteration of the data flow graph to be executed. We refer this duration as the iteration delay. We executed all these experiments on a real platform and measured the actual execution times of each iteration. The obtained

1. Temporary Repository: `https://doi.org/10.5281/zenodo.4243071`

predictions were compared between simulation and measured results. Figure 3.10 represents the two data flow applications considered to validate the proposed approach.

Sobel filter

Sobel filter is a simple and popular application that is an edge-detection for image processing (see Figure 3.10 (a)). A pixel matrix gets read by the *GP* actor. The gradient component of each orientation is then measured by the *GX* and *GY* actors and returned to the *ABS* actor that calculates the resulting pixel. The communication part in this use-case takes most of the execution time. The tokens rate of the channels between the actors *GP* and *GX* or between the actors *GP* and *GY* depends on the size of the image matrix. One token is equal to one 32 bit data word on bus. In Figure 3.10 (a), an example of 3×3 pixels matrix is illustrated.

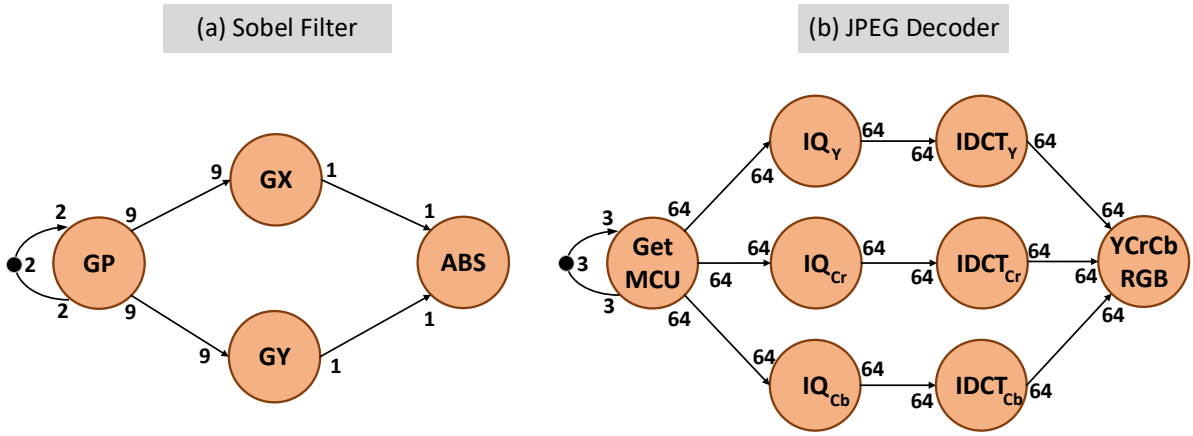


Figure 3.10 – Two SDF applications are used in our experiments: (a) Sobel filter and (b) JPEG decoder.

JPEG decoder

JPEG decoder is an image decompression application which is presented in Figure 3.10 (b). The *GetMCU* actor reads a Minimum Coded Unit of an JPEG encoded image. The inverse quantization (IQ) is then done for the input data unit. Then a rough classification of possible input data is obtained from Inverse Discrete Cosine Transformation(IDCT) actor. Each of the three color channels of an encoded JPEG MCU *(Y , Cr, Cb)* has specific characteristics that influence the execution time distribution of the IDCT actor. This leads to different possible execution times for each color channel. This use-case presents a huge computation part with an unmanageable amount of execution paths in most of the actors except for the IQ-actors that contain only one execution path. The communication part became more important when the

application is executed on a high number of tiles. In this use-case, we used as many tiles as possible for this application to demonstrate the scalability of the created models.

3.4.2 Hardware platform

In Figure 3.11, we present the heterogeneous multiprocessor system that was used for all the experiments. The platform in blue contains 7 tiles that are connected to a shared memory via a shared interconnect. On the lower part of Figure 3.11, the measurement infrastructure which was detailed in Section 3.2.4 is presented in green.

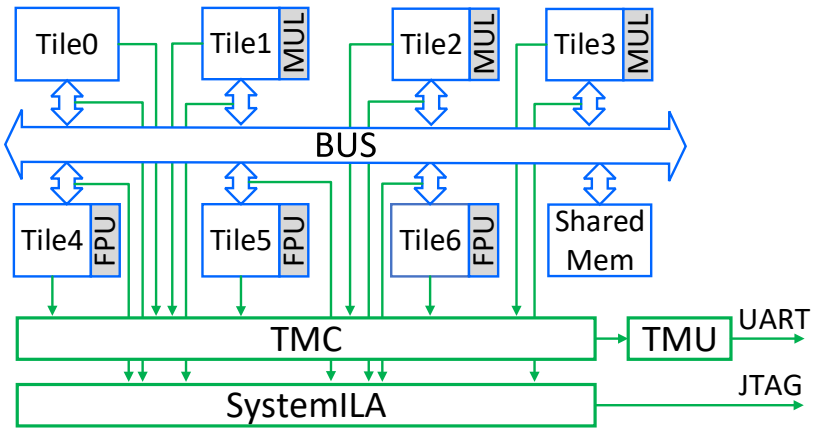


Figure 3.11 – Heterogeneous platform with 7 tiles connected to a shared memory via a shared interconnect. In green, the measurement infrastructure contains the System Integrated Logic Analyzer (SystemILA) that characterizes the elementary delays, while the Time Measurement Controller (TMC) that merges all control signals from the tile trigger the Time Measurement Unit (TMU). Input and output images are generated in library files and stored in shared memory.

The individual tile use a MicroBlaze soft core that was implemented with private instruction and data memories. The 7 tiles provide different hardware accelerators (MUL, FPU). These hardware accelerators can allow a better performance in computation depending on the actor's algorithm. The tile *Tile0* does not contain any hardware accelerator. *Tile1*, *Tile2*, *Tile3* tiles were extended by a hardware multiplication unit (MUL). *Tile4*, *Tile5*, *Tile6* tiles were extended with a floating point unit (FPU). Furthermore one shared memory was used for communication between the tiles.

Different mappings were applied for each use-case as shown in Tab. 3.2. The first column presents the different actors of the JPEG decoder experiment. In the second column, we show the 3 mappings of the JPEG decoder: *Jpeg1*, *Jpeg3* and *Jpeg7*. For each mapping, the number presents the tile where each actor was mapped. For example, the experiment denoted by *Jpeg1* corresponds to a complete mapping on *Tile0*. This means that all the actors were mapped on a

Experiment→ Actor ↓	**Jpeg1**	**Jpeg3**	**Jpeg7**	**Exp. → Actor ↓**	**Sobel1**	**Sobel2**	**Sobel4**
Get MCU	0	0	0	**GP**	1	1	1
IQ_Y	0	1	1	**GX**	1	2	2
IQ_{Cr}	0	1	2	**GY**	1	1	3
IQ_{Cb}	0	1	3	**ABS**	1	2	0
$IDCT_Y$	0	4	4				
$IDCT_{Cr}$	0	4	5				
$IDCT_{Cb}$	0	4	6				
YCrCb RGB	0	0	0				

Table 3.2 – Mapping of the Sobel filter and JPEG experiments on the tiles of the hardware platform. The number prese nts the tile where each actor was mapped.

single tile. In this experiment, all actors were executed in static order without any contention. In the *Jpeg3* experiment, we mapped the actors on 3 tiles to have parallel execution of the application. The MUL and FPU were used to accelerate the computation time of the actors. In the *Jpeg7* experiment, 7 tiles were considered to have the highest possible parallelization and to challenge the communication model in terms of accuracy and simulation time.

In the third and fourth columns of Tab. 3.2, the actors and mappings of the Sobel filter experiments are presented. We also simulated 3 mappings for Sobel filter: *Sobel1*, *Sobel2* and *Sobel4*. In the *Sobel1* experiment, all the actors were also mapped on a single tile (*Tile1*). This tile contains the MUL hardware accelerators. For the parallelization in the execution of this application, we considered two additional mappings *Sobel2* and *Sobel4*. For all the experiments the instruction and local data of an actor were mapped on the private memory of a tile.

3.4.3 Experiment setup

In the experiments, we considered the *injected data*, *gaussian* and *uniform* distribution for the computation time model. For the distribution of execution times of an actor, we use the observed execution times of 1 000 000 iterations. For the communication time model, we considered the communication model that was presented in Section 3.3.2. We measured elementary delays needed for building the communication model. For the 7-tile heterogeneous platform, we observed that the elementary delays were constant for all mappings and for every iterations of the applications. These elementary delays measured using SystemILA on the hardware platform are presented in Table 3.3.

For the simulation results, we ran 1 000 000 iterations and captured the iteration delay. We considered parallel simulation by splitting our simulations into 20 processes. Each process simulated 50 000 iterations on dedicated processor with 12288 Go de RAM (https://ccipl.univ-nantes.fr).

Elementary delay	**Value**	**Elementary delay**	**Value**
t_i^r	15	t_i^w	16
t_p^r	8	t_p^w	8
t_{pl}^r	7	t_{pl}^w	7
t_{pr}^r	15	t_{pr}^w	15
t_r^r	8	t_{wr}^w	5
t_l^r	14	t_l^w	13
t_{po}^r	11	t_{po}^w	9
t_u^r	5	t_u^w	5

Table 3.3 – Elementary delays (in cycles) for communication measured using the SystemILA. One cycle is equal to 10 ns.

3.4.4 Results

In this section, we present the preliminary results of the proposed workflow. In Table 3.4, we compare the results of the simulation model with the measured data. We present the average iteration delay in the upper part of Table 3.4 and the observed worst case iteration delay in the lower part. In our simulation model, we applied different representations of the computation time, such as the injected data, the uniform and gaussian distribution. In the first column *Experiment*, all the experiments are listed. Then in the second column *Measured*, we show the average measured iteration delay. In the next three columns, we presented in order the result of the simulation models using the injected data, the uniform and gaussian distributions.

For the average iteration delay, our simulation model using the injected data presents an over-estimation of 3.57 % for *Sobel1* experiment. This over-estimation decreases to 2 % in the *Sobel2* experiment. Then the *Sobel4* experiment presents an under-estimation of 7.7 %. Since we used the injected data for the computation model, the errors thus come from the communication model. In the JPEG decoder experiments, the application has a huge computation part comparing to the communication part. Therefore, the errors are very low compared to the Sobel filter experiments. For example, the *Jpeg7* experiment shows at the highest under-estimation of 1.6 %.

Our simulation model for the Sobel filter experiments using the uniform distribution shows the similar error of the model using the injected data. However the JPEG decoder experiments show much higher under-estimation compared to the injected data. This can be explained because the uniform distribution leads to a higher possibility to select the worst case computation time. Our simulation models using the uniform distribution show similar errors as the model using the injected data in all experiments. The results show that the gaussian distribution have a better performance to represent the variation of the measured computation time than the uniform distribution.

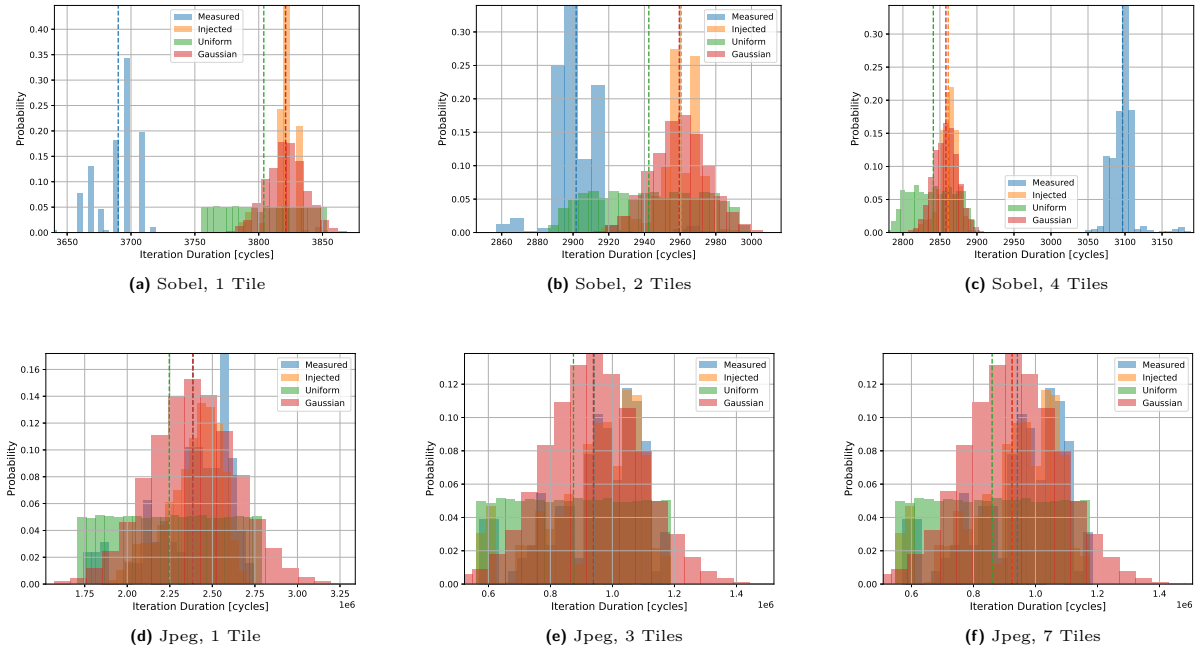
For the observed worst case iteration delay, the simulation results of the models are sim-

	Experiment	Measured	Injected data	Uniform	Gaussian
Average Case	Sobel1	3690	3821.6 (3.57 %)	3804 (3.1 %)	3821 (3.55 %)
	Sobel2	2902.5	2960.6 (2 %)	2942.5 (1.38 %)	2959.4 (1.96 %)
	Sobel4	3097.4	2859.6 (-7.7 %)	2841 (-8.3 %)	2858 (-7.7 %)
	Jpeg1	2385860	2387757.6 (0.08 %)	2246952.8 (-5.83 %)	2386251.3 (0.02 %)
	Jpeg3	940836	940174.7 (-0.07 %)	874531.1 (-6.73 %)	938843.9 (-0.21 %)
	Jpeg7	941059	925969.7 (-1.61 %)	860285.2 (-8.6 %)	924640.4 (-1.75 %)
Worst Case	Sobel1	3719	3854 (3.63 %)	3853 (3.6 %)	3882 (4.38 %)
	Sobel2	2994	2999 (0.17 %)	2999 (0.17 %)	3020 (0.87 %)
	Sobel4	3197	2905 (-9.13 %)	2904 (-9.16 %)	2923 (-8.57 %)
	Jpeg1	2746197	2749873 (0.13 %)	2789559 (1.58 %)	3387677 (23.36 %)
	Jpeg3	1185223	1188393 (0.27 %)	1188484 (0.27 %)	1547288 (30.55 %)
	Jpeg7	1185483	1174158 (-0.96 %)	1174174 (-0.95 %)	1533008 (29.31 %)

Table 3.4 – Comparison of the results of the simulation model using the injected data, the uniform and gaussian distribution with the measurement data. The table shows the average iteration delay (in cycles) in the upper part and the observed worst case iteration delay in the lower part. The error to the measurement data is computed for each mapping. The negative values mean the under-estimation, otherwise over-estimation. The experiments are done for 1 000 000 iterations.

ilar in the Sobel filter experiments. In the JPEG decoder experiments, the models using the injected data and the uniform distribution also provide similar results. This is because the selected computation time in the two models stays within the best-case and worst-case measured computation delays. However, the model using the gaussian distribution shows a much higher error up to 30 %. This can be explained because in the gaussian distribution, the selected value of a computation time can be much higher than the observed worst-case measured computation delay.

In Figure 3.12, we compare the iteration delay distribution of the measured data (blue) and the analyzed results using the injected data (orange), the uniform distribution (green) and the gaussian distribution (red) for the computation time. The main objective of using the injected data in the simulation model is to validate the created SystemC model. Eventually we aim to create a probabilistic model that represents the variation of the computation and communication times. In the upper part of Figure 3.12, we compare the distribution of the iteration delays of our simulation model to the measured data of the Sobel filter experiments while the comparison of the JPEG decoder experiments are presented in the lower part. In Figure 3.12 (a) and (b), the over-estimation of the simulation results compared to the measured data were illustrated, while the under-estimation was presented in Figure 3.12 (c). These distributions show the efficiency of the computation time representation approaches to the measured computation delay. Our simulation model using the injected data can provide a similar shape of distribution as the measured data.



(a) Sobel, 1 Tile **(b)** Sobel, 2 Tiles **(c)** Sobel, 4 Tiles

(d) Jpeg, 1 Tile **(e)** Jpeg, 3 Tiles **(f)** Jpeg, 7 Tiles

Figure 3.12 – Distribution of the measured data (blue) compared to the results of the models using the injected data (orange), the uniform distribution (green) and the gaussian distribution (red). Figures (a), (b) and (c) show the results of the Sobel filter . Figures (d), (e) and (f) present the results of the JPEG decoder. The colored dashed lines show median execution time of the corresponding models.

While the model using the gaussian distribution partly present the shape compared to the measured data and the uniform distribution showed a totally different shape. This remains to find better probabilistic distribution to represent the measured computation time.

In Tab. 3.5, we first show the duration to execute 1 000 000 iterations of the applications running on the real hardware platform. We then present the simulation time of models with different representations of the computation time. For each experiment, we simulated 1 000 000 iterations. We finally show in the speed up factor between the simulation time and measurement time. In Table 3.5, our simulation models using the different representations of the computation time present a similar simulation time for each experiment. In the Sobel filter experiments, the simulation time is around 10 s because of the simplicity of the application. The speed up factor decreases from 73.8 in the *Sobel1* experiment to 39.4 in the *Sobel4* experiment which is caused by the increase of polling states. For the JPEG Decoder, the simulation time variates from half a minute for *Jpeg1* to 2 h for the *Jpeg7* experiment. The speed up factor is 1765.6 in the *Jpeg1* experiment. It quickly decreases to 10.8 in the *Jpeg3* experiment and 2.6 in the *Jpeg7* experiment. This is because several actors in JPEG decoder have huge computation time. When one of these

Experiment	Measured	Injected data	Uniform	Gaussian	Speed up
Sobel1	0:07:23	0:00:06	0:00:06	0:00:06	73.8x
Sobel2	0:07:03	0:00:09	0:00:07	0:00:08	52.8x
Sobel4	0:07:13	0:00:11	0:00:11	0:00:12	39.4x
Jpeg1	13:14:31	0:00:32	0:00:27	0:00:27	1765.6x
Jpeg3	5:12:58	0:44:53	0:38:58	0:44:16	10.8x
Jpeg7	5:13:02	2:11:44	1:58:57	2:02:49	2.6x

Table 3.5 – Simulation time (HH:MM:SS) is done for 1 000 000 iterations on a Intel® Xeon® Broadwell-EP CPU E5-2630 v4 (2.20 GHz) at https://ccipl.univ-nantes.fr. Simulation split into 20 processes, each on a dedicated processor. Measured data is obtained from measurement of 1 000 000 iterations executed on real hardware platform Xilinx ZC702.

actors computes, the others poll the data in parallel. In the TL model we considered that actor *GetEncodedImageBlock* took an average computation time of 600 000 cycles and the polling time step took 20 cycles. The number of polling steps can be estimated to around 30 000 per iteration by the model. This caused the long simulation time in the *Jpeg3* and *Jpeg7* experiments which contain a huge number of polling statements in the execution. This remains to consider the system at higher level of abstraction to reduce the number of states that needs to be considered in the simulation process.

3.5 Conclusion

In this chapter, we have presented the working environment used in this thesis. A workflow was proposed to evaluate the efficiency of the measurement-based modeling approach for timing property analysis of MPSoC systems. A SystemC simulation model is built from the separation of computation and communication modeling at transaction level. We then presented our preliminary results on two image processing applications (Sobel filter and JPEG decoder) executing on a 7 tile heterogeneous platform. The results showed the validation of our proposed approach. However, different aspects should be improved in the remaining work of this thesis. For the accuracy aspect, the errors of the experiments are caused by the communication time model. Our communication time model did not well predict the multiprocessor mappings as presented in a communication intensive application as Sobel filter. This requires higher efforts to improve the communication model. For the simulation time aspect, the simulation time is up to 2 hours in the *Jpeg7* experiment which needs to be improved.

We position our contributions in Figure 3.13 that are going to be presented in the next chapters. In Chap. 4, we aim to improve the accuracy and reduce the simulation run-time of analyzing multiple FIFO communication channels mapped on a shared bus with a shared memory. We adopt an analytical model to formulate the time dependencies between the elementary

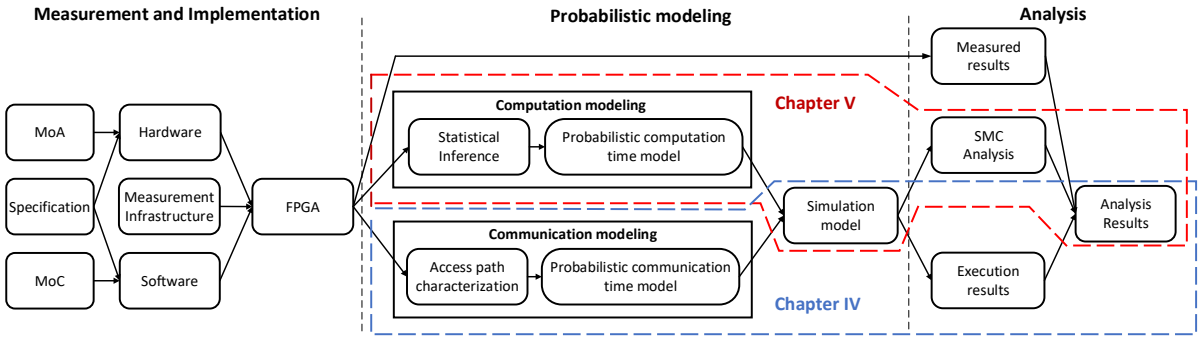


Figure 3.13 – Our contributions are presented in two parts: The first part (in blue dash line) is the message-level communication model and the second part (in red dash line) is the statistical model checking approach.

communication phases taking into account potential penalty delays due to contention at shared resources. It is used in a message-level simulation model of the communication infrastructure which demonstrates good scalability for performance prediction of different possible mappings.

In Chap. 5, we aim to use the statistical model checking (SMC) method [69, 77] for the analysis process. SMC refers to a series of techniques that are used to explore a sub-part of the state-space and provides an estimation about the probability that a given property is satisfied. SMC approaches reduce the required number of simulation runs by using statistical algorithms such as Monte-Carlo or Sequential Probability Ratio Test (SPRT). By controlling the number of simulation runs, a trade-off between high confidence and fast analysis time is possible. Furthermore, such an approach could be adopted to evaluate different properties of the created models such as the probability to miss a deadline.

Chapter 4

Definition of a fast yet accurate message-level communication model

In the previous chapter, we presented our proposed workflow to evaluate the efficiency of the created probabilistic models in analyzing timing properties of MPSoC systems. The preliminary results showed the needs to improve the created models in terms of accuracy and simulation time. In this chapter, we thus propose a message-level communication model that is based on a run-time prediction technique of the whole communication time of the application actors. This message-level model can be used for different mappings and applications. In this message-level model, the communication time prediction is done by using an analytical model which is defined from our observations of the application execution of the application on the hardware platform. This message-level model reduces significantly the number of simulation events considered in the execution process. Through our experiments, we observed that the simulation time is largely reduced without degrading the level of accuracy of the created model.

In this chapter, we first introduce the proposed message-level communication model. Secondly we present the implementation of this model in SystemC language. Afterward, simulation results are then presented with comparisons to the transaction-level of the previous chapter and the measured results. The discussion of our proposed communication model is finally given.

4.1 Proposal of a message level communication model

The proposed modeling approach allows accurate estimation of communication timings with a limited simulation run-time. This approach consists in associating a message-level simulation model of the mapped communication channels with an analytical expression of the communication durations. Without loss of generality, we present the application of this message level model to a First Come First Served bus arbitration policy. The definition of the analytical model is given by using the Petri net formalism to represent the contention situations between the actors. Finally, a workflow is proposed to calibrate the proposed message-level model.

4.1.1 Message-level model principles

In this work, communications are modeled at two different abstraction levels. At transaction level (TL), communications are described with bus transaction granularity and the bus arbitration is expressed for each transaction. The description of communication at transaction level is presented in Section 3.2.3, especially with Figure 3.7. An execution over time of a n tokens write function at transaction level is illustrated on the upper part of Figure 4.1. Different elementary steps with their duration have to be considered. These durations are classified in two types. The first type relates to the situation that the PE accesses a shared resource, such as: the polling time t_p^w, the writing time t_{wr}^w. The second type corresponds to the situation that the transaction is executed only on the PE, such as: the inter-polling time t_{pl}^w, the pre-writing time t_{pr}^w, the inter-writing time t_{wl}^w and the post-writing time t_{po}^w.

As presented in the previous chapter, the TL model can provide long simulation time due to the huge number of states considered during simulation process. We propose a message-level (ML) communication model to solve this problem. At the message-level, communications are described at the application data granularity. Timing is simulated by a single wait-for-time statement and arbitration is not explicitly modeled. The main idea of the proposed approach is illustrated in the lower part of Figure 4.1. Three significant instants are exhibited:

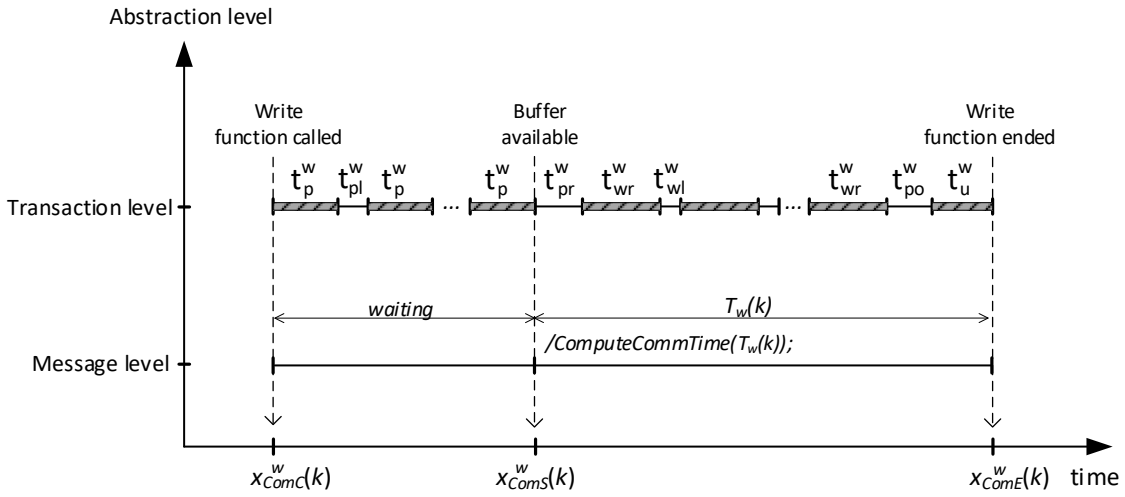


Figure 4.1 – Principle of the proposed message-level communication model with run-time computation of communication durations comparing with the transaction level model.

— Instant when tokens communication function is called, denoted by x_{ComC}^w in Figure 4.1. This is followed with a waiting state which corresponds to the duration until buffer is available for tokens write.

— Instant when buffer is available, denoted by x^w_{ComS}. This would correspond to the instant when data is available in the case of a read tokens function.

— Instant when tokens communication ends, denoted by x^w_{ComE}.

Communication durations and instants are computed during simulation. The computation is based on an analytical model which determines instants when shared resources are used. The computation is denoted by the `ComputeCommTime()`. At message level, the communication duration is denoted T_w. When multiple tokens communications are simulated, instants when shared resources are used are considered to determine if contention situations occur. In that case, instants are adapted accordingly to appropriately set the communication duration. It is thus possible to limit the number of simulation kernel calls but with still accuracy about communication resource usage. In previous work [78], Le Nours et al. presented a similar approach to model the computation resources. In the scope of this chapter, we focus on communication resources modeling. In the following, we illustrate the application of this approach through a didactic example and we detail the setup of the analytical model used for communication duration computation.

4.1.2 Application of the proposed modeling approach to a FCFS bus arbitration policy

We present in Figure 4.2 two simultaneous communications through a shared communication bus and a shared memory at the two different abstraction levels. It illustrates the situation of writing n tokens on channel C_1 and reading n tokens from channel C_0 with the protocol described in Figure 3.3. In part (a) of Figure 4.2, we present the communication model at transaction level which was already detailed in Section 3.3.2. Considering a shared communication bus, there are different contention situations that can occur depending on the number of concurrent accesses and their types. For example, possible contention situations between two tiles was presented in Figure 4.2 (a), such as: (1) contention between two polling steps, (2) contention between one writing step and one reading step and (3) contention between an update buffer usage step and a reading step. In part (b) of Figure 4.2, we adopt the message-level communication model. At the message-level, the number of simulation states is commonly reduced at the expense of accuracy. To deliver still accurate results, it is needed to appropriately set the durations of the communications, especially in the situations when contentions at shared resources occur. In part (b) of Figure 4.2, the successive polling steps of WriteTokens and ReadTokens are replaced by waiting two synchronization events, denoted *read_C1* and *write_C0*, that indicate instants when the buffer is available to write or the data is available to read. At message level, an `UpdateStatus` function is called at three considered instants of the WriteTokens or ReadTokens. This function identifies the contention situation which is needed for the computation of the communication time. In the proposed approach, the created simulation model uses computation of the whole

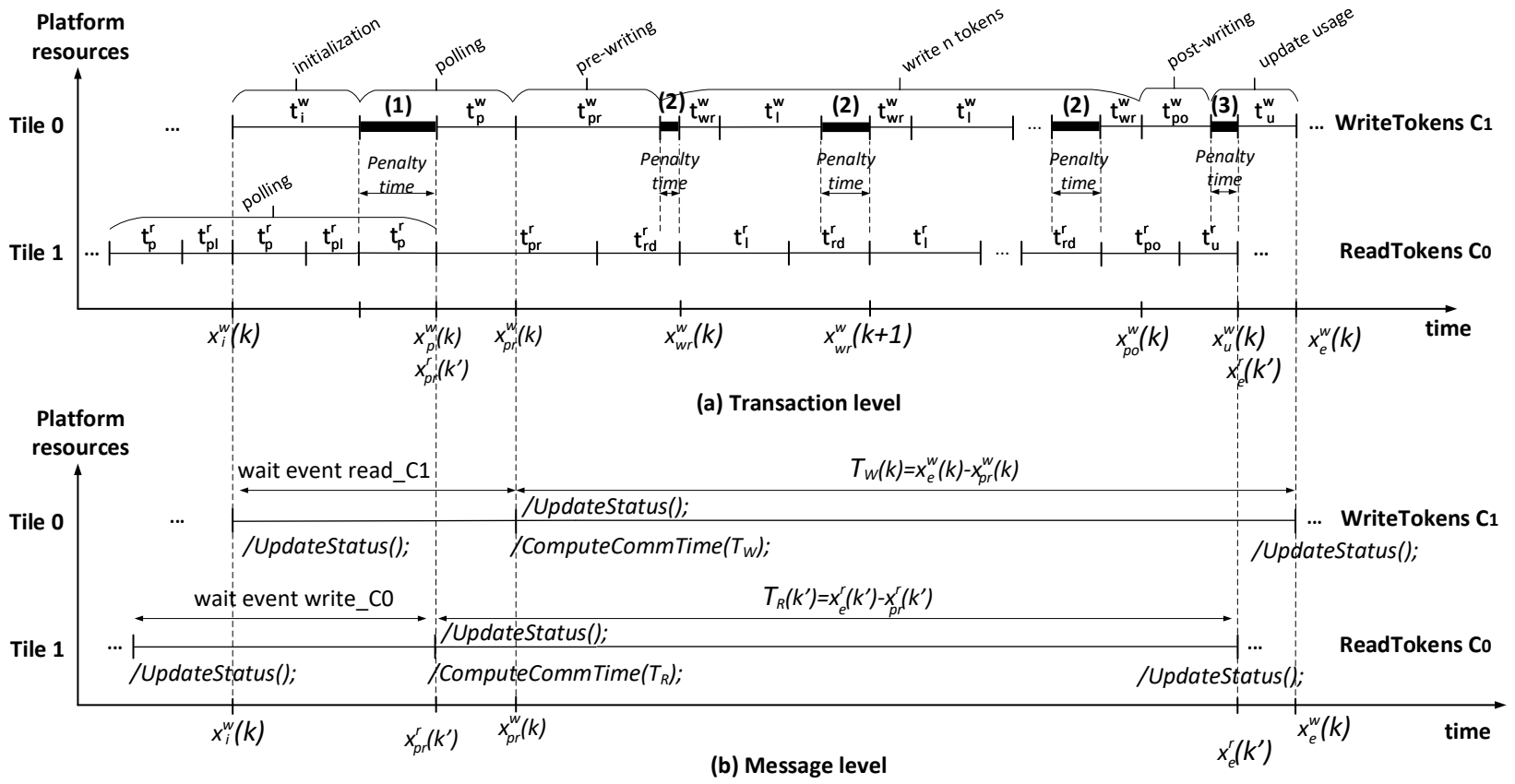


Figure 4.2 – Modeling and simulation of writing n tokens on channel C1 and reading n tokens from channel C0 (see Figure. 3.2 in the previous chapter for details). (a) At transaction level, the elementary delays and penalty delays caused by the contention are exhibited. (b) At the proposed message level, the required communication durations (denoted by T_W and T_R) are computed locally.

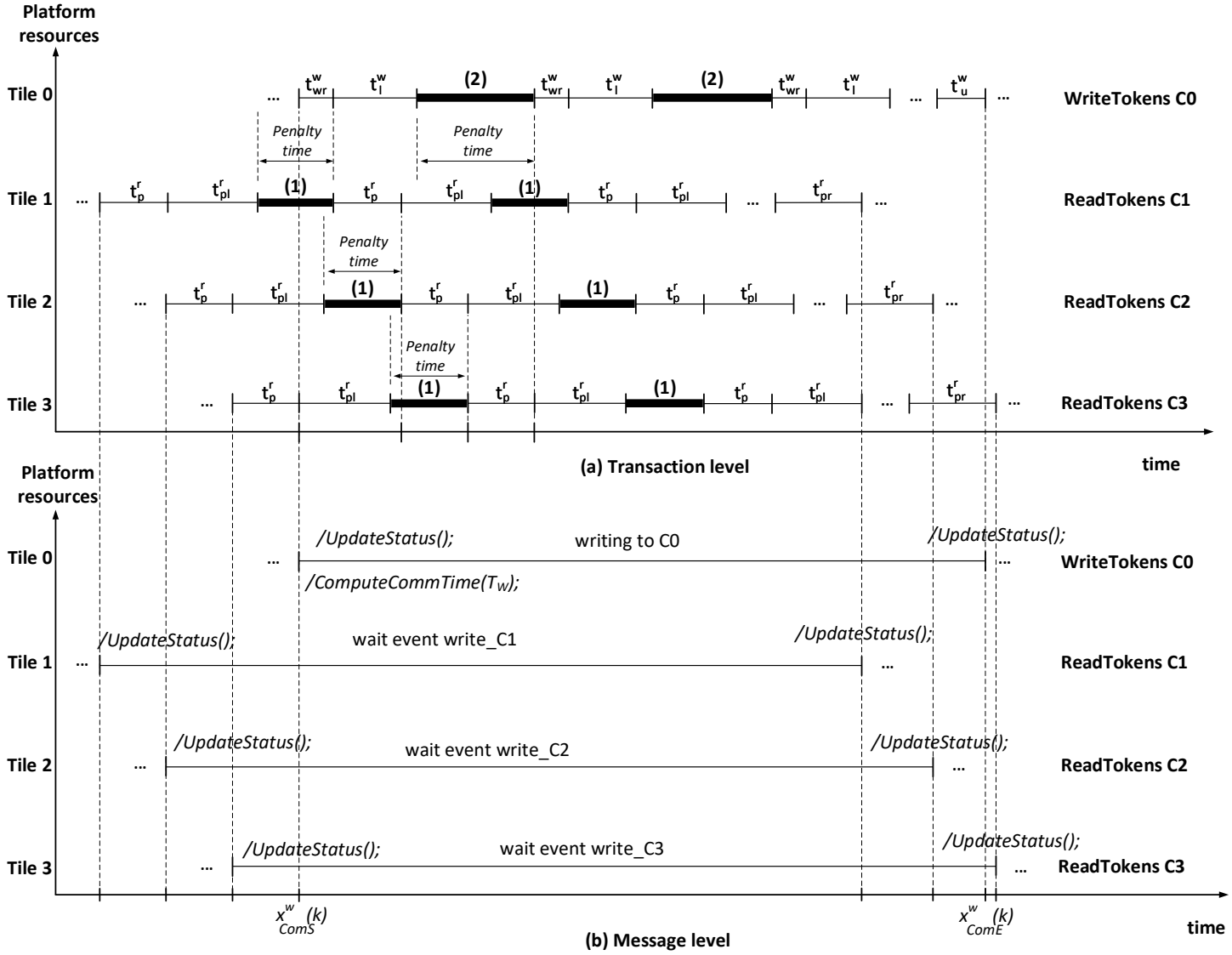


Figure 4.3 – Penalty delays caused by the contention situation of 4 tiles at (a) transaction level and (b) message level.

communication durations. The computation of *ReadTokens* and *WriteTokens* durations is denoted by action `ComputeCommTime`. It is performed at the instants denoted $x^w_{pr}(k)$ and $x^r_{pr}(k)$ in Figure 4.2.

In Figure 4.3, we present another possible contention situations between 4 tiles accessing a FCFS bus arbitration protocol at (a) transaction level and (b) message level. The contention situation is between the writing phase (situation (2)) of one WriteTokens and the polling phases (situation(1)) of 3 ReadTokens. At transaction level, the elementary delays with their duration and the penalty time are illustrated in Figure 4.3 (a). In each contention situation, the penalty time can be computed from the elementary delays of the ReadTokens and WriteTokens. For example, the communication duration for the WriteTokens function is computed at the instant x^w_{ComS} by using the functions `UpdateStatus` and `ComputeCommTime`. The `UpdateStatus` function first identifies the contention situation at the instant x^w_{ComS} which consists of 1 writing step and 3 polling steps. The `ComputeCommTime` function then computes the communication duration from the elementary and the penalty delays caused by this contention situation. The penalty delays are computed by using an equation obtained from the analysis the contention situations of the considered bus. Further details about the equation creation are given in the following sections. In the next section, we have established an analytical model to express the time dependencies among transition instants between elementary communication steps based on the implemented communication functions and the FCFS arbitration bus protocol.

4.1.3 Definition of the analytical model

We create the analytical model which is built from the knowledge of communication phases and the bus arbitration policy. In our approach, the timed Petri net (TPN) formalism is adopted to formulate the relationships between elementary communication steps. It represents a timed extension of Petri nets for which time is expressed as minimal durations on the sojourn of tokens on places. In the Petri net notation, circles correspond to places and thick lines to transitions between places.

In Figure 4.4, we first present the TPN of writing n tokens into buffer at transaction level. In this example, a transition q is enabled if each upstream place p contains at least one token. The k^{th} instant when transition q_i is enabled is denoted by $x_i(k)$. The holding time t in a place is the time a token must spend in the place before contributing to the enabling of the downstream transition. Different states as illustrated in Figure 3.3 are also presented. Each holding time corresponds to the duration of the elementary states. For example, the initialization state starts at the instant $x^w_i(k)$ when the transition q^w_i is enabled. The duration of this state is $t^w_i(k)$.

In this TPN, there are two possible loops: the polling loop and the writing loop. In the polling loop, the instant at which transition q^w_p is fired for the k^{th} time (denoted by $x^w_p(k)$) is when the polling step checks the availability of the buffer. The duration for this polling step is

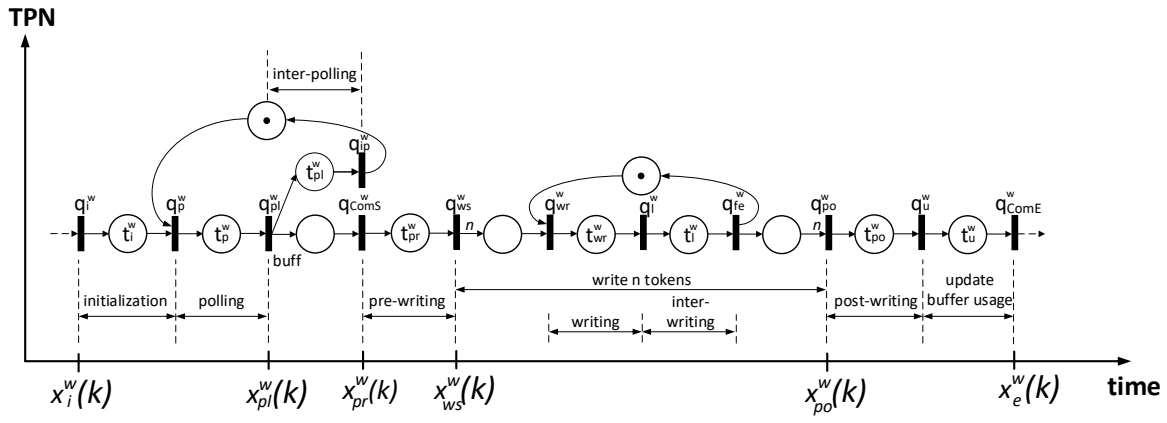


Figure 4.4 – Timed Petri net model of writing n tokens at transaction level.

$t_p^w(k)$. At the instant $x_{pl}^w(k)$ if the buffer is not available, the polling step continues until the buffer is ready. The duration between two successive polling is denoted by $t_{pl}^w(k)$. In Figure 4.4, when the condition denoted by *buff* is satisfied, it indicates that the buffer is available to write data. Then the pre-writing state starts at when the transition q_{ComS}^w is fired (instant $x_{ComS}^w(k)$). The writing loop starts when transition q_{ws}^w is fired at the instant $x_{ws}^w(k)$. The duration to write a token is $t_{wr}^w(k)$. The duration between two successive writing tokens is denoted by $t_{pl}^w(k)$. The writing loop ends at the instant $x_{po}^w(k)$ when all the tokens are written into the buffer.

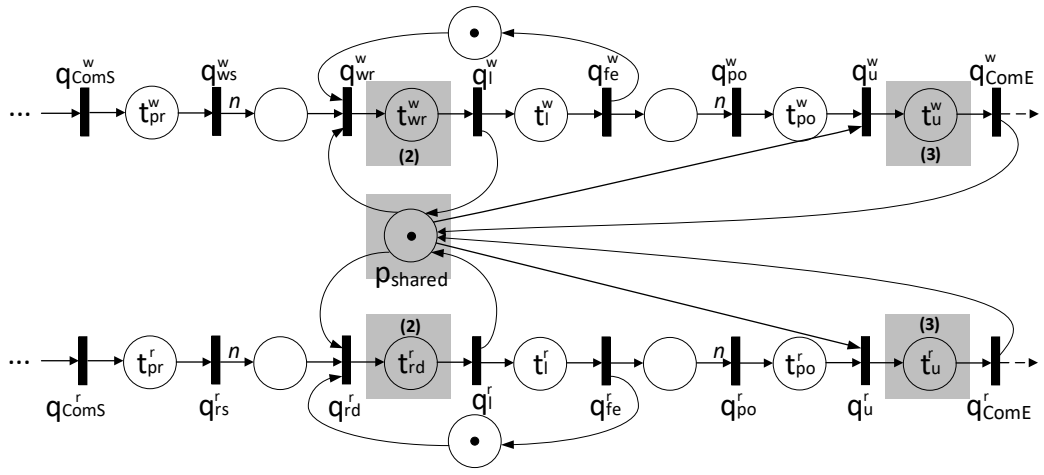


Figure 4.5 – Timed Petri net model of writing n tokens on channel C1 and reading n tokens from channel C0 at transaction level with FCFS arbitration policy. Rectangles in grey emphasize the situations where shared resources are used.

We present in Figure 4.5 the TPN description for C_1 *WriteTokens* and C_0 *ReadTokens* through the studied shared communication bus and memory. In Figure 4.5, the place denoted by p_{shared} represents the limited availability of the shared resources and the fact that only some phases of the communication can occur at a time. In the message-level communication model, we replace the successive polling steps by using synchronization events. From the instant $x^w_{ComS}(k)$ that the transition $q^w_{ComS}(k)$ is fired, the relationships between transition instants can basically be expressed using two operators: addition and maximization [79]. For the situations that cause accesses to shared resources (*i.e.*, situations (2) and (3) in Figure 4.5) operator maximization is used to express the effect of mutual exclusion on transition instants values. The transition instants from Figure 4.5 are given as follows:

$$x^w_{ws}(k)=x^w_{ComS}(k)+t^w_{pr}(k) \tag{4.1}$$

$$x^w_{wr}(k)=max(x^w_{ws}(\left\lfloor\frac{k}{n}\right\rfloor), x^w_{fe}(k-1), x^r_l(k'), x^r_{ComE}(k')) \tag{4.2}$$

$$x^w_l(k)=x^w_{wr}(k)+t^w_{wr}(k) \tag{4.3}$$

$$x^w_{fe}(k)=x^w_l(k)+t^w_l(k) \tag{4.4}$$

$$x^w_{po}(k)=x^w_{fe}(nk) \tag{4.5}$$

$$x^w_u(k)=max(x^w_{po}(k)+t^w_{po}(k), x^r_{ComE}(k'), x^r_l(k')) \tag{4.6}$$

In the situation (3), the instants when the step *update buffer usage* can start is expressed as follows:

$$x^w_{ComE}(k)=x^w_u(k)+t^w_u(k) \tag{4.7}$$

where x^w_{po} is the instant when post-writing starts, t^w_{po} is the duration of the post-writing step, x^r_l is the instant when elementary read finishes, and x^r_e is the instant when ongoing tokens read process finishes. The equations for the ReadTokens function can be created similarly.

These expressions are used to compute the communication durations in the case of one tokens write and one tokens read for a two-tile platform. The adoption of the TPN formalism allows to describe different communication situations with different numbers of tiles and simultaneous read and write processes. It is thus possible to systematize the obtaining of the equations that give the instants when shared resources are accessed. Figure 4.6 illustrates a more complex situation with four simultaneous communications which was presented in Figure 4.3 (b) (one write tokens process and three read tokens processes).

This example emphasizes the situation (2) where the elementary write step competes with the elementary read step. The instant x^w_{wr} when a token is written depends on the end of each elementary read step (x^{r1}_l, x^{r2}_l, x^{r3}_l). The instant when a token is read can also be computed as

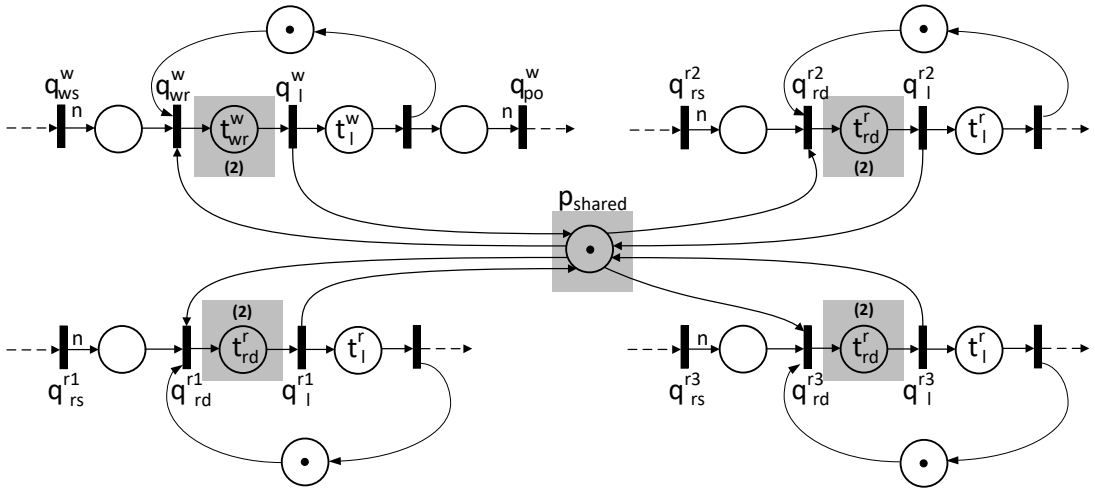


Figure 4.6 – Timed Petri net model of the contention between one n tokens writing process and three n tokens reading processes.

follows:

$$x^w_{wr}(k)=max(x^w_{ws}(\left\lfloor\frac{k}{n}\right\rfloor), x^w_{fe}(k-1), x^{r1}_l(k'), x^{r2}_l(k''), x^{r3}_l(k''')) \tag{4.8}$$

$$x^{r1}_{rd}(k)=max(x^{r1}_{rs}(\left\lfloor\frac{k}{n}\right\rfloor), x^{r1}_{fe}(k-1), x^w_l(k'), x^{r2}_l(k''), x^{r3}_l(k''')) \tag{4.9}$$

$$x^{r2}_{rd}(k)=max(x^{r2}_{rs}(\left\lfloor\frac{k}{n}\right\rfloor), x^{r2}_{fe}(k-1), x^w_l(k'), x^{r1}_l(k''), x^{r3}_l(k''')) \tag{4.10}$$

$$x^{r3}_{rd}(k)=max(x^{r3}_{rs}(\left\lfloor\frac{k}{n}\right\rfloor), x^{r3}_{fe}(k-1), x^w_l(k'), x^{r1}_l(k''), x^{r2}_l(k''')) \tag{4.11}$$

The TPN formalism is suitable in the case of the studied FCFS bus arbitration policy. As it makes possible to express synchronization and mutual exclusion, it could be adopted for different bus protocols and arbitration policies. An example of using this formalism can be found in [80] to study communication latency for a network on chip infrastructure. In our approach, the content of the `ComputeCommTime` method would be adapted to describe the influence on the communication duration.

4.1.4 Message-level model creation

The proposed message level communication model is built from the elementary delays introduced at transaction level and the defined analytical model for a bus arbitration policy. In Figure 4.7 we present the workflow to create the message level communication model. The idea is to characterize the elementary delays from the implementation of a communication workload

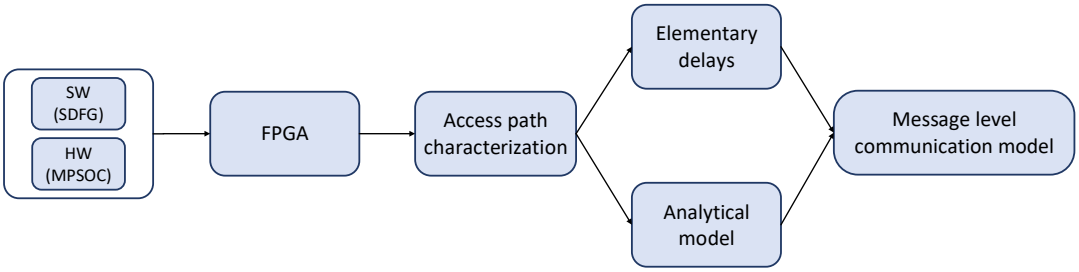


Figure 4.7 – The workflow to establish our proposed ML communication model is made of a measured elementary delays and our defined analytical model.

on real hardware platform. This communication workload model has the same communication phase as an application model but the computation phase is ignored. This is done based on the assumption that the computation phase has no influence on the communication phase in the SDF representation. The elementary delays are measured and the contention situation is characterized using the IP SystemILA.Based on our observations, the elementary delays depend on both hardware platform specification and software compilation option. In the scope of this thesis, we characterized the 7 tile heterogeneous platform which is described in Section 3.4.2. The compilation is optimized using the option $-O3$. The elementary delays are constants as presented in Tab. 3.3 and can be used for different applications and mappings.

The created analytical model depends on the bus arbitration policy. Once the analytical model is created for the considered bus, it can be used for different applications and mappings. When we consider another bus, a new analytical model should be created. The analytical model and the elementary delays are then combined to create the equations that can be used to compute the communication time of actors in contention. These equations are then implemented in the communication time model of the SystemC simulation model as the message level communication model.

4.2 ML description of the communication model

In this section, we present the implementation of our communication model in SystemC language.

4.2.1 Message-level evolution of the communication model

In Figure 4.8, we present the Petri net of our message-level communication model for a one place FIFO buffer through a shared bus and memory. For each WriteTokens or ReadTokens communication, we consider 3 instants: (1) The instant when the function is called, (2) the

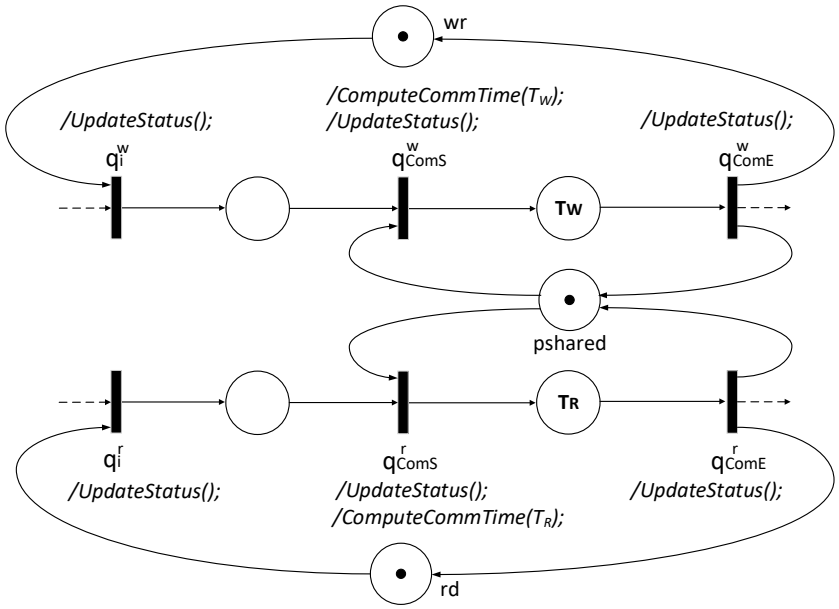


Figure 4.8 – Illustration of the message-level communication model for a 1-place FIFO buffer mapped on a shared memory.

instant when the writing/reading data process starts and (3) the instant when the function finishes. The `UpdateStatus` function which identifies the contention situation (i.e., number of polling, writing and reading states) is called at each of these instants. For WriteTokens, the place *wr* indicates that the next transition q_i^w can not be fired before firing the previous transition q_{ComE}^w. The same indication is made for the place *rd* of ReadTokens. The place *pshared* indicates the availability of the FIFO buffer. In this example, two communication access an one-place FIFO buffer which allows one communication to access the buffer and the other one has to wait until the FIFO buffer is released. When the FIFO buffer is available, the waiting communication is notified and triggered to access to the buffer. At this instant, the function `ComputeCommTime` is called to compute the communication duration.

4.2.2 Simulation model

The created communication model has been implemented in the SystemC language [73]. We present in Figure 4.9 the organization of the system model at message-level. It corresponds to the description of the mapped actors and channels on the resources of a two-tile execution platform. The SystemC model consists of two main parts. The *Tile* module describes the execution of the actors mapped on processing elements. The timing behavior of the actors is described in an SC_THREAD for each *Tile* module. The computation phase of an actor is described with

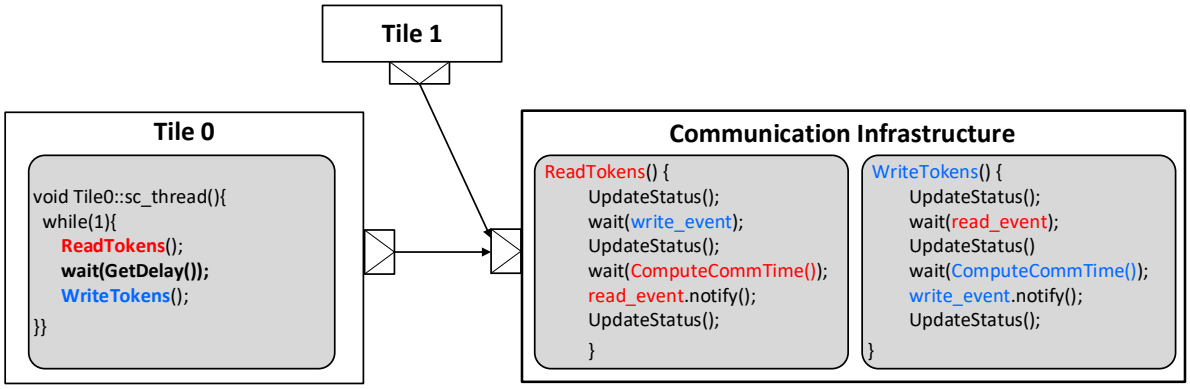


Figure 4.9 – Illustration of the SystemC model with two tiles at message level

a wait statement by calling the `GetDelay` function to get the computation time of the actors. The `ReadTokens` and `WriteTokens` functions are used to describe the communication processes between actors. The *Communication infrastructure* module uses synchronization events to trigger the accesses to shared resources of the actors. For example, the `ReadTokens` function waits for the availability of data in the buffer until the *write_event* is notified. The communication duration is then computed by the `ComputeCommTime` function for a wait statement. Afterward, it notifies the end of the access to shared resources via the *read_event*. The `UpdateStatus` function is used to update the contention situation.

Figure 4.10 illustrates the evolution of the model created for two actors *GetPixel* mapped on *Tile 0* and *GY* mapped on *Tile 1*. They share a FIFO buffer C_0. In this example, the *GetPixel* actor starts with its computation which finishes at the instant $x^w_{ComC}(k)$ and then the function `WriteTokens(C_0)` is called. The function `UpdateStatus` updates the communication situation by setting the ongoing number of polling, reading, or writing phases. The `WriteTokens(C_0)` function then waits for the availability of the channel C_0. The example in Figure 4.10 corresponds to the situation that the channel C_0 is not ready to write. Thus the writing process has to wait until the event *read_C0* notifies that the channel C_0 is available. The function `ComputeCommTime` computes then the communication duration of the write process. At the end of the writing phase, the `WriteTokens(C_0)` function notifies the event *write_C0* to trigger the `ReadTokens(C_0)` function to read data from the channel C_0.

In our experiment, we used the raw measured computation time of actors to validate the proposed communication model independently from the computation modeling approach. We denote them as *injected data*. During simulation, they get chosen randomly from a file in each simulated iteration.

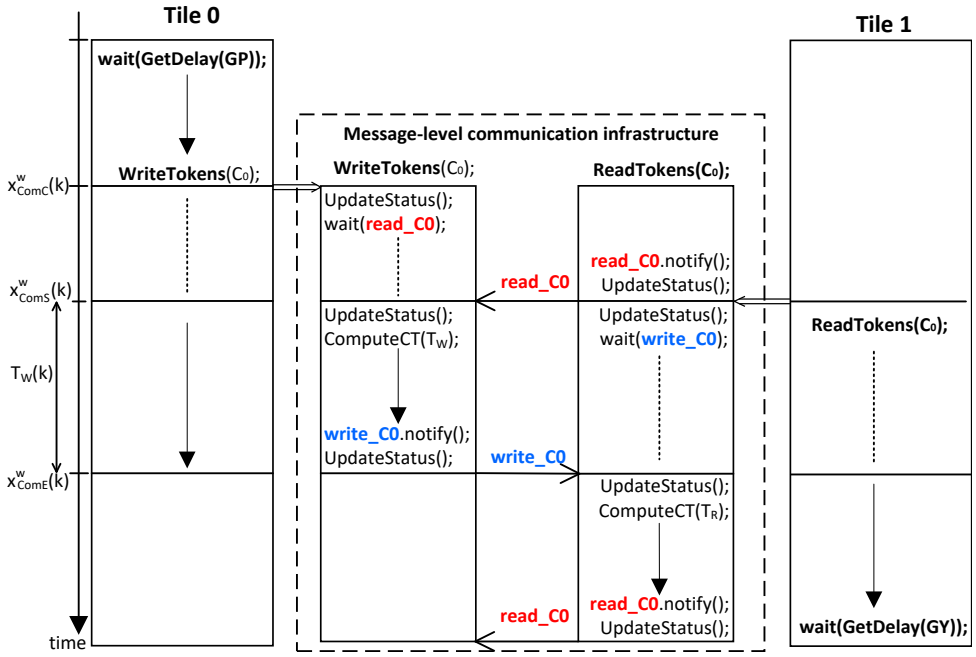


Figure 4.10 – Execution of created message-level SystemC model following the proposed communication and computation modeling processes.

4.3 Experiments

In this section, we first present the communication characterization phase for the AXI4LITE bus. Then we demonstrate the simulation results of our message level communication model. From the application part, we considered two image processing SDF applications described in Chap. 3: a Sobel filter and a JPEG decoder. For the hardware platform, we also used the 7 tile heterogeneous platform introduced in Section 3.4.2. For the mapping model, the following mappings were considered: *Sobel1*, *Sobel2* and *Sobel4* for Sobel filter and *Jpeg1*, *Jpeg3* and *Jpeg7* for JPEG decoder. The details of the experiment setup are detailed in Section 3.4.3.

4.3.1 Communication characterization phase for the AXI4LITE bus

In this section, we aim to present how the equations to compute the penalty delays were built for an AXI4LITE FCFS bus arbitration policy. We used two familiar examples which were already presented in this chapter.

We present in Figure 4.11 the computation of penalty times of contention situations that were observed on the bus AXI4LITE by using SystemILA. The contention situation is between WriteTokens n tokens to the buffer C_1 and ReadTokens m tokens from the buffer C_0. We denote different instants that relate to the computation of the penalty delay of writing one token and

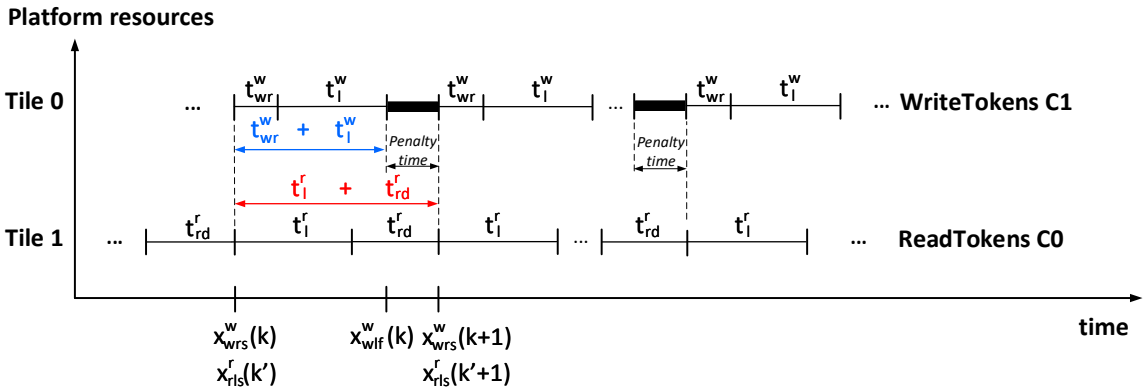


Figure 4.11 – Example of the penalty delays computation on the AXI4LITE bus based on the elementary delays and the contention situations of 2 tiles.

reading one token as follows:

— $x^w_{wrs}(k)$ and $x^w_{wrs}(k+1)$ denote the instants to start accessing the buffer C_1 to write the k^{th} and $(k+1)^{th}$ tokens, respectively.

— $x^w_{wlf}(k)$ denotes the instant when the inter-writing step between the tokens k^{th} and $(k+1)^{th}$ finishes.

— $x^r_{rls}(k')$ and $x^r_{rls}(k'+1)$ denote the instants when the inter-writing steps between the tokens k'^{th} and $(k'+1)^{th}$ start.

The transition instants in Figure 4.11 are given as follows:

$$x^w_{wlf}(k) = x^w_{wrs}(k) + t^w_{wr}(k) + t^w_l(k) \tag{4.12}$$

$$x^r_{rls}(k') = x^w_{wrf}(k) \tag{4.13}$$

$$x^r_{rls}(k'+1) = x^r_{rls}(k') + t^r_l(k') + t^r_{rd}(k'+1) \tag{4.14}$$

$$x^w_{wrs}(k+1) = max(x^w_{wlf}(k), x^r_{rls}(k'+1)) \tag{4.15}$$

The penalty delay for writing the k^{th} token $t^w_{pen}(k)$ can be computed as follows:

$$t^w_{pen}(k) = x^w_{wrs}(k+1) - x^w_{wlf}(k)) \tag{4.16}$$

$$= t^r_l(k') + t^r_{rd}(k'+1) - (t^w_{wr}(k) + t^w_l(k)) \tag{4.17}$$

$$\tag{4.18}$$

Similarly, the penalty delay of reading the k'^{th} token $t^r_{pen}(k')$ can also be computed as follows:

$$t^r_{pen}(k') = x^w_{wrs}(k+1) - x^r_{rls}(k'+1)) \tag{4.19}$$
$$= x^r_{rls}(k'+1) - x^r_{rls}(k'+1) = 0 \tag{4.20}$$

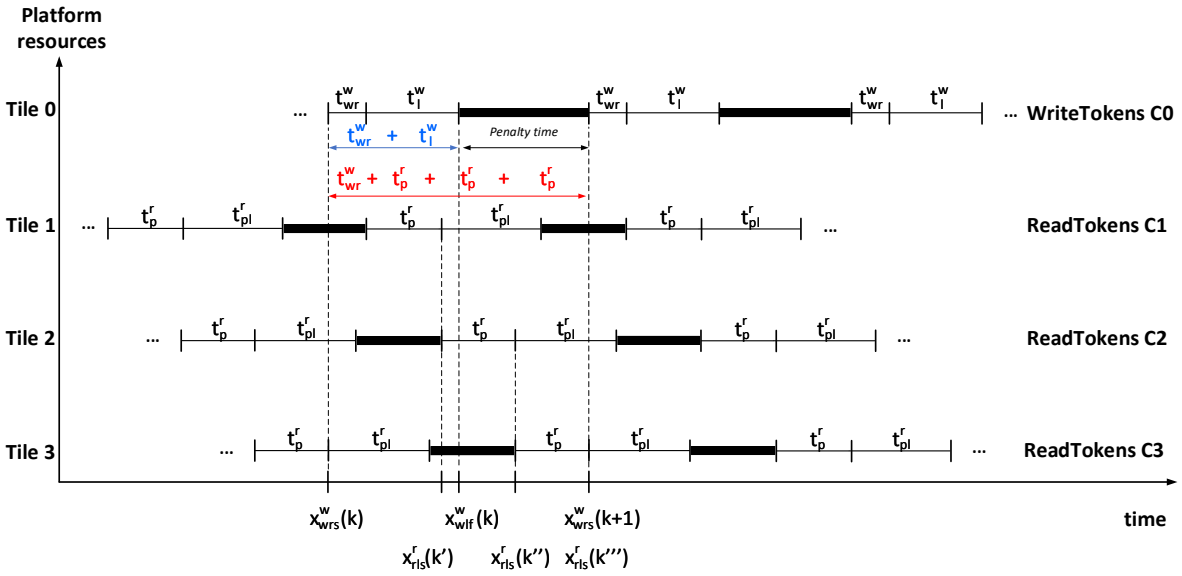


Figure 4.12 – Example of the penalty delays computation on the AXI4LITE bus based on the elementary delays and the contention situations of 4 tiles.

In Figure 4.12, the contention situation between WriteTokens to the buffer C_0 and ReadTokens from the buffers C_1, C_2 and C_3. The instants $x^w_{wrs}(k)$ and $x^w_{wrs}(k+1)$ and $x^w_{wlf}(k)$ are denoted similarly as in Figure 4.11. The instant $x^r_{rls}(k')$ is when the $(k')^{th}$ inter-polling step on the buffer C_1 starts. Similarly, the instants $x^r_{rls}(k'')$ and $x^r_{rls}(k''')$ are when the $(k'')^{th}$ inter-polling step on the buffer C_2 and the $(k''')^{th}$ inter-polling step on the buffer C_3 start, respectively. Then the penalty delay of writing the k^{th} token $t^w_{pen}(k)$ can be computed as follows:

$$t^w_{pen}(k) = x^w_{wrs}(k+1) - x^w_{wlf}(k) \tag{4.21}$$
$$= max(x^w_{wlf}(k), x^r_{rls}(k'), x^r_{rls}(k''), x^r_{rls}(k''')) - x^w_{wlf}(k) \tag{4.22}$$
$$= t^r_p(k') + t^r_p(k'') + t^r_p(k''') - t^w_l(k) \tag{4.23}$$

Since the penalty delay of a contention situation for transmitting one token is computed, we can approximate the penalty delay of the whole WriteTokens or ReadTokens functions by

multiplying this unary penalty delay with the number of transmitted tokens. Afterward we have built the function `ComputeCommTime` to compute the communication duration in SystemC.

In the next section, we present the experimental results to evaluate the efficiency of the proposed approach.

4.3.2 Results

We aim to demonstrate the validation of our proposed communication model by comparing the average iteration delay of our simulation model with the transaction level model and the measured data. We used the same use cases as in Chap. 3. For the purpose of validation, we used the injected data for the computation time.

In Tab. 4.1, we compare the analyzed results of the TL model and the ML model against the measured data. In the first column (*Experiment*), the experiments are listed. The second column (*Measured*) presents the average execution time of 1 000 000 iterations that are the actual measured execution time of the application on the real hardware platform. The next column shows the results of the TL model. The next column shows the results of the ML model without considering the penalty delays caused by the contention situations. The last column shows the results of the ML model that takes into account the penalty delays. We used a noise image as input for these experiments. The noise image uses 1 Pixel Gaussian noise to provide a source with many edges to detect.

In Tab. 4.1, the results of TL model show an over-estimation of 3.57 % for *Sobel1* experiment, 2 % for *Sobel2* experiment, while *Sobel4* presents an under-estimation of 7.7 %. The errors in experiments with the JPEG decoder remain lower because this application has a huge computation part in the overall execution of the program. These errors were discussed in Section 3.4.4 of Chap. 3.

For the *ML model No-Comp.* experiments, the *Sobel1* and *Sobel2* experiments show a better estimation compared to the TL model. However the *Sobel4* experiment presents a higher under-estimation up to 8.1 %. This under-estimation shows the need of considering the penalty delays for the accuracy of the simulation model. Since the communication part is minor part in the JPEG decoder application, the experiments of the ML model show the same level of accuracy as the TL model.

All the experiments of our *ML model Comp.* show a high accuracy result that slightly over-approximates the measured data. In a communication intensive application as Sobel-Filter, the highest error is upto 1.81 % in *Sobel4* experiment. In the JPEG decoder application with a huge computation part, the errors stay even lower.

In Figure 4.13, we compare the iteration delay distribution of the measured data (in blue) and the analyzed results (in orange) of experiments. The results of the Sobel filter experiments are shown in the upper part of Figure 4.13. While the results of the JPEG decoder experiments are

Experiment	Measured	TL model	ML model No-Comp.	ML model Comp.
Sobel1	3690	3821.6 (3.57 %)	3759.8 (1.8 %)	3756.6 (1.8 %)
Sobel2	2902.5	2960.6 (2 %)	2923.8 (0.73 %)	2936.6 (1.17 %)
Sobel4	3097.4	2859.6 (-7.7 %)	2847.8 (-8.1 %)	3153.6 (1.81 %)
Jpeg1	2385860	2387757.6 (0.08 %)	2387489.7 (0.07 %)	2387489.6 (0.07 %)
Jpeg3	940836	940174.7 (-0.07 %)	939597.3 (-0.13 %)	941445.3 (0.07 %)
Jpeg7	941059	925969.7 (-1.61 %)	925419.7 (-1.67 %)	941518.1 (0.05 %)

Table 4.1 – Comparison of the message level model with and without the computation of the communication time and the TL model results with the measurement data. The table show the average execution time (in cycles).

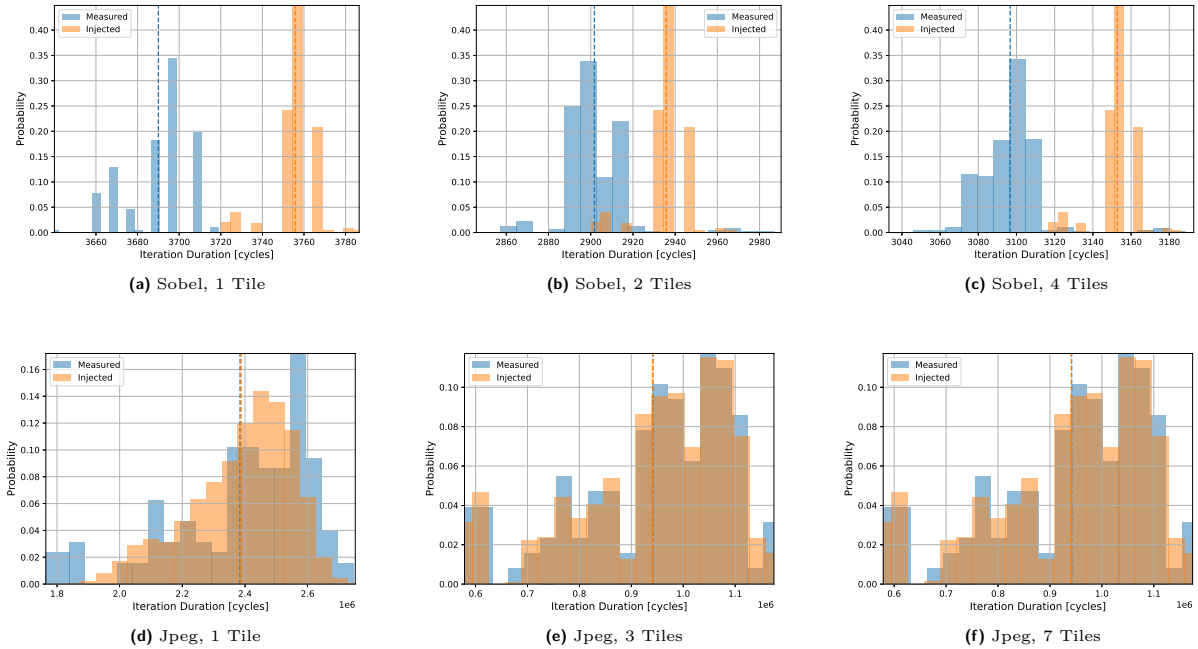
Experiment	Measured	TL model	ML model	Speed up
Sobel1	0:07:23	0:00:06	0:00:03	2x
Sobel2	0:07:03	0:00:09	0:00:03	3x
Sobel4	0:07:13	0:00:11	0:00:02	5.5x
Jpeg1	13:14:31	0:00:32	0:00:06	5.3x
Jpeg3	5:12:58	0:44:53	0:00:04	673.25x
Jpeg7	5:13:02	2:11:44	0:00:04	1976x

Table 4.2 – Simulation time (HH:MM:SS) is done for 1 000 000 iterations on a Intel® Xeon® Broadwell-EP CPU E5-2630 v4 (2.20 GHz) at https://ccipl.univ-nantes.fr. Simulation split into 20 processes, each on a dedicated processor. Measured data is obtained from measurement of 1 000 000 iterations executed on real hardware platform Xilinx ZC702.

in the lower part. We used the noise image as input of the applications to get these distributions. The distributions of simulated results show a similar shape between simulation and measured data. This means that our ML model was well created.

In Tab. 4.3.2, we first show measurement time of 1 000 000 iterations to run the applications on the real hardware platform. We then compare the simulation time of our ML model with the TL model. Both models apply the injected data for the computation part. For each experiment, we simulated 1 000 000 iterations. The column *Speed up* presents the reduction ratio between the two models. For the Sobel filter experiments, the simulation time is around 10 s because the short computation time of the actors caused a few polling states. In the JPEG decoder experiments, the simulation time variates from 32 s for *Jpeg1* to 2 h 11 min 44 s for the *Jpeg7* experiment. This is due to the huge number of polling statements considered by the system.

In our ML model, the simulation time stays less than 10 s for both Sobel-Filter and JPEG decoder. The simulation time of the ML model reduced compared to the TL model simulation. The reduction ratio is 673.25 for the *Jpeg3* experiment and upto 1976 for the *Jpeg7* experiment.



(a) Sobel, 1 Tile **(b)** Sobel, 2 Tiles **(c)** Sobel, 4 Tiles

(d) Jpeg, 1 Tile **(e)** Jpeg, 3 Tiles **(f)** Jpeg, 7 Tiles

Figure 4.13 – Distribution of the measured data (blue) compared to the results of the ML model using the injected data (orange). Figures (a), (b) and (c) show the results of the Sobel filter . Figures (d) , (e) and (f) present the results of the JPEG decoder. The dashed lines show the median execution time.

4.3.3 Discussion

For the consideration of shared memory access over a bus, we demonstrated the effectiveness of a message-level simulation approach on the AXI4LITE bus using a FCFS bus arbitration policy. This simple bus facilitated the characterization of the elementary delays and the communication behaviors to create an analytical model. For another bus arbitration policy, a new characterization phase needs to be done that consists of the measurement of new elementary delays and the creation of new analytical model. This one-time phase should take some overhead duration. In the column *ML model* of Tab.4.1, the JPEG decoder experiments demonstrated the benefit of our approach for design space exploration. The average iteration delay of the *Jpeg3* experiment decreased by a factor 2.5 compared to the *Jpeg1* experiment. This result come from the use of more tiles and the effects of the hardware accelerators which reduce the actor's computation time. However, the *Jpeg7* experiment present a higher degree of parallel execution but did not show the improvement in the average iteration delay compared to the *Jpeg3* experiment. It demonstrates well the boundedness of the application speed-up due to communication overhead on the shared memory.

4.4 Conclusion

In this chapter, we presented a message-level communication model that is based on an analytical model of the concurrent accesses to shared resources and the characterization of elementary delays. We applied this ML model to a FCFS bus arbitration policy AXI4LITE. Then we implemented this ML model in SystemC language. The simulation results of this ML model were compared to the measured data that showed an improvement in both accuracy and the simulation time compared to the TL model in Chap. 3. Further adoption of this proposed approach on other bus arbitration policies can be done by creating a new analytical model and measuring the corresponding elementary delays.

Chapter 5

Probabilistic timing analysis approach

In previous chapters, we have presented our proposed probabilistic modeling approach for MPSoC systems. The analysis results showed good accuracy and fast simulation speed of the proposed approach even considering a large number of simulation runs. However, we can not ensure that this number of simulation runs is actually sufficient. If it is less than needed, we could miss some corner cases. Otherwise, we could waste simulation efforts without having better results. In this chapter, we aim to present a statistical model checking (SMC) approach that gives further control on the simulation runs of the created probabilistic models. We then study the efficiency of this SMC approach with different statistical algorithms, such as Monte Carlo or Sequential Probability Ratio Test (SPRT). Furthermore, we also extend the complexity of our hardware architecture considering private cache for processing elements to show the scalability of the approach.

5.1 Statistical model checking

5.1.1 Overview of statistical model checking

In previous chapters we have illustrated the way that probabilistic models represent a possible solution to capture variability caused by shared resources on parallel software execution. Quantitative analysis of probabilistic models can be used to quantify the probability that a given time property is satisfied. Numerical approaches exist compute the exact measure of the probability at the expense of a time-consuming analysis effort. Another approach to evaluate probabilistic models is to simulate the model for many runs and monitor simulations to approximate the probability that time properties are met. This approach, which is also called Statistical Model Checking (SMC), is far less memory and time intensive than probabilistic numerical methods and it has been successfully adopted in different application domains [81]. Statistical Model Checking (SMC) refers to a serie of techniques that are used to explore a sub-part of the state-space and provides an estimation. Given a probabilistic system $\mathcal{S}$ and a property φ, SMC can be used to answer two types of questions:

Qualitative: Is the probability for a model to satisfy a given property φ greater or equal to a certain threshold θ ?

Quantitative: What is the probability for a model to satisfy a given property?

In the following sections, we further detail the methods used to perform quantitative and/or qualitative analysis of probabilistic models.

5.1.2 Qualitative analysis

In [82], Younes presents an approach to answer qualitative question which is based on *hypothesis testing*. To answer the qualitative question, he considers the Bernoulli distribution. Let B_i be a discrete random variable with a Bernoulli distribution of parameter p that only takes two values: 0 and 1. If the probability $Pr[B_i = 1] = p$ then $Pr[B_i = 0] = 1 - p$. In our case, if each variable B_i is associated with one simulation of the system then the outcome for B_i, denoted b_i, is 1 if the given system $\mathcal{S}$ satisfies φ and 0 otherwise. To determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A simulation-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The strength of a test is determined by two parameters (α, β), such that the probability of accepting K when H holds, called a Type-I error (false positive) is less or equal to α and the probability of accepting H when K holds, called a Type-II error (false negative) is less or equal to β. However, it is impossible to ensure a low probability for both types of errors simultaneously. A solution is to relax the test using an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. A value δ is chosen such that $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$. Two solutions were proposed by Younes in [83, 82] to test the requirements above: the Single Sampling Plan (SSP) and the Sequential probability ratio test (SPRT). In this thesis, we are going to use SPRT in the analysis process.

Sequential probability ratio test (SPRT) [84] is defined to reduce the expected number of observations required to achieve a desired test strength. In SPRT, one has to choose two values A and B, with $A > B$. These two values should be chosen to ensure that the strength of the test is respected. Let n be the number of observations that have been made so far. The test is based on the following quotient:

$$\frac{p_{1n}}{p_{0n}} = \prod_{i=1}^{n} \frac{Pr(B_i = b_i \mid p = p_1)}{Pr(B_i = b_i \mid p = p_0)} = \frac{p_1^{d_n}(1-p_1)^{n-d_n}}{p_0^{d_n}(1-p_0)^{n-d_n}}$$

Where $d_n = \sum_{i=1}^{n} b_i$. The idea behind the test is to accept H_0 if $\frac{p_{1n}}{p_{0n}} \geq A$, and H_1 if $\frac{p_{1n}}{p_{0n}} \leq B$. An algorithm for sequential ratio testing consists of computing $\frac{p_{1n}}{p_{0n}}$ for successive values of n until either H_0 or H_1 is satisfied. In [85], a logarithmic based SPRT algorithm was proposed that given p_0, p_1, α and β implements the sequential ratio testing procedure.

5.1.3 Quantitative analysis

For the *quantitative* analysis, an estimation procedure was presented in [86] by Herault et al. to compute the probability p for a model to satisfy a given property φ. Given a *precision* δ, their procedure computes an approximation p' such that $|p' - p| \leq \delta$ with *confidence* α, i.e., $Pr(|p' - p| \leq \delta) \geq 1 - \alpha$.

Let $Y_1 ... Y_n$ be n discrete random variables with a Bernoulli distribution of parameter p associated with n simulations of the system. Recall that the outcome for each of the Y_i, denoted y_i, is 1 if the simulation satisfies φ and 0 otherwise. Let $p' = (\sum_{i=1}^{n} b_i)/n$. According to the *Chernoff-Hoeffding bound* [62]:

$$Pr(|p' - p| > \delta) < 2e^{-\frac{n\delta^2}{4}}$$

If we consider a number of simulations $n \geq \frac{4}{\delta^2} log(\frac{2}{\alpha})$, then we are guaranteed that $P(|p' - p| \leq \delta) \geq 1 - \alpha$. This estimation procedure is latter applied in this chapter with the Monte-Carlo statistical algorithm.

5.1.4 Bounded Linear Temporal Logic

To apply the above approach, we have to ensure that the simulation result is obtained in a finite time. This means that the considered properties are bounded. In the scope of this thesis, we consider the Bounded Linear Temporal Logic (BLTL) [87] to express timing properties. BLTL is an extension of Linear Temporal Logic (LTL) with time bounds and temporal operators. The semantics of BLTL logic are the semantics of LTL logic restricted to a time interval. A BLTL formula φ is defined over a set of atomic propositions AP, the logical operators (e.g., $true$, $false$, $\neg$, $\rightarrow$ and $\wedge$) and the temporal modal operators (e.g., U for until, X for next, F for eventually, G for always, M for strong release and W for weak until). The φ is defined by the grammar as follows (| is denoted as or):

$$\varphi := true \mid false \mid ap \in AP \mid \neg\, \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1\, U_{\leq T}\, \varphi_2$$

The time bounds T is the duration of one simulation run during which we analyze the property. Temporal modality F can be derived from the "*until*" U as $F_{\leq T}\ \varphi = true\ U_{\leq T}\ \varphi$. It means that the property φ is eventually satisfied within T. Similarly, temporal modality G can be derived from F as $G_{\leq T}\ \varphi = \neg F_{\leq T}\ \neg\varphi$. This equation can be explained as: the hypothesis that the property φ is not satisfied within T will not occur or the property φ is always satisfied within T.

The semantics of BLTL are defined w.r.t execution traces of the model. Let

$$\omega = (s_0, t_0), (s_1, t_1), ..., (s_{N-1}, t_{N-1}), N \in \mathbb{N}$$

be an execution trace of the model where each state (s_i, t_i) comprises a discrete state s_i and a time $t_i \in \mathbb{R}_{\geq 0}$. We denote $\omega^k = (s_i, t_i), ..., (s_{N-1}, t_{N-1})$ be the suffix of ω starting at step i. We denote the BLTL formula $\omega \models \varphi$ is that ω satisfies the property φ.

We present latter in this chapter the way to express our timing properties in BLTL language.

5.2 SMC for SystemC model of MPSoC systems

5.2.1 Proposed workflow

In the field of embedded system design, executable specifications built with the use of the SystemC language are now widely adopted [73]. SystemC models are used for the purpose of timing analysis that typically capture workload models of the application mapped on shared resources of the considered platform. Timing annotations are commonly expressed as average values or intervals with estimated best case and worst case execution times. SMC techniques controls the number of simulation runs and provide a confidence level of analysis results that could deliver a good compromise between accuracy and analysis time. Thus the adoption of these techniques to analyze SystemC models for multi-processor systems is promising. However, it requires a more sophisticated timing model based on probability density functions, inferred from measurements on a real prototype. Thus, the creation of trustful probabilistic SystemC models is challenging. Since SMC methods have rarely been considered to analyze timing properties of applications mapped on multi-processor systems with complex hierarchy of shared resources, exploring their application on trustful probabilistic SystemC models remains a significant research topic.

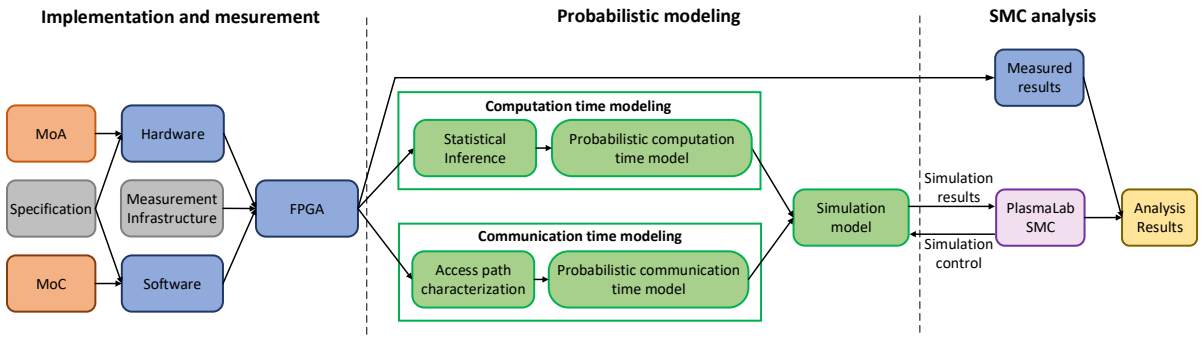


Figure 5.1 – The established workflow to evaluate the efficiency of the statistical model checking approach for timing property analysis of probabilistic models of MPSoC systems.

We present an experimental modeling workflow that is used to evaluate the efficiency of SMC methods for MPSoC systems, as illustrated in Fig. 5.1. We propose a probabilistic modeling process for both computation and communication time models which is based on a measurement-

based approach to appropriately prepare timing annotations and calibrate the simulation model. Then the created model is simulated by using Plasma Lab SMC approach which controls the number of simulation runs and analyzes the simulated results. We evaluated the SMC methods efficiency with respect to accuracy and analysis time. Evaluation was done by comparing a real multi-processor implementation with related estimation results. In [69], we evaluated this workflow on a Sobel filter case study running on a two tile homogeneous hardware architecture. However, we considered the communication bus model at transaction level. In this chapter, we consider the message level communication model as presented in Chap. 4. For the evaluation, we demonstrate the feasibility of the proposed approach using the two SDFGs image processing applications: a Sobel filter and a JPEG decoder running on the 7-tile heterogeneous hardware architecture. We also take a deeper analysis on the effects of statistical algorithms to the accuracy and analysis time of the simulation process.

5.2.2 PLASMA Statistical Model Checker

Platform for Learning and Advanced Statistical Model checking Algorithms (PLASMA) [88] is an efficient self-contained SMC tool and software library written in Java. In Fig. 5.2, we present the architecture of PLASMA which consists of three main parts: an API, a controller and plugins (*statistical algorithms*, *checkers* and *simulators*) [89]. An SMC algorithm collects samples obtained from a checker component. The checker asks the simulator to initialize a new trace. Then, it controls the simulation by requesting new states, with a state on demand approach: new states are generated only when needed to decide the property. Depending on the property language, the checker either returns Boolean or numerical values. Finally, the algorithm notifies the progress and sends the results through the controller API.

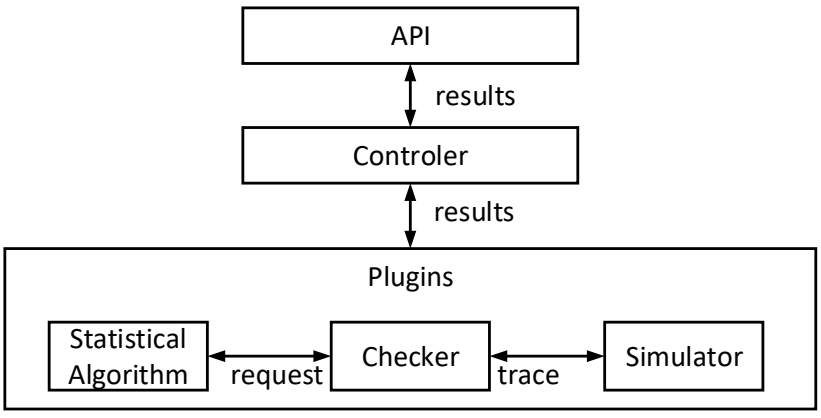


Figure 5.2 – Illustration of the PLASMA architecture.

PLASMA supports different simulators, such as Reactive Module Language (RML) which is the input language of the tool PRISM for Markov chains models, an extension of RML for adaptive systems (RML Adaptive), a biological language for writing chemical reactions, a simulator of Matlab/Simulink and a SystemC plugin for the simulation of SystemC models. For the checkers, PLASMA accepts the Bounded Linear Temporal Logic (BLTL), Adaptive Linear Temporal Logic (ALTL), Goal and Contract Specification Language (GCSL), BLTL checker enhanced with nested probability operator (Nested) and RML Observer.

PLASMA supports different statistical algorithms [90], such as: simple Monte-Carlo, Monte-Carlo using a Chernoff confidence bound and sequential hypothesis testing. Users define the parameters before running the simulation. PLASMA supports multi-threaded mode that implements parallel SMC algorithms to distribute the simulations. In distributed mode, PLASMA acts like a server that controls the experiment from PLASMA clients. The clients perform simulations and send the results to the server. Distributed results are aggregated and shown in the GUI as final result.

5.2.3 Monitor and aspect-advice generator and SystemC plugin

The PLASMA statistical model-checker workflow is illustrated in Fig. 5.3. Users first define a *configuration file* which contains the *observed variables*, the *BLTL properties* and the *temporal resolution*. This configuration file is then used by a *Monitor and aspect-advice generator* (MAG) tool to generate a *monitor model* and an *aspect-advice file* [40]. The monitor model captures the observed variables to verify the BLTL properties. It contains a monitor class and a *local_observer* class. The *local_observer* class has a callback function that invokes a *step*() function of the appropriate monitor class at a given sampling point during the simulation [91]. The *step*() function captures the value of the observed variables and their instances to produce execution trace [92]. The aspect-advice file declares the monitor class as a *friend* class of the SystemC model. Thus the monitor can access the private variables of the observed SystemC model [87]. The BLTL properties are verified in the analysis process. The temporal resolution specifies the granularity of simulation time.

Afterward, the generated monitor model and the probabilistic model are instrumented using AspectC++ with the help of the aspect-advice file. The instrumentation exposes the user model's states and syntax to the monitors [91]. The instrumented models are compiled and linked to the libraries of a patched version of SystemC [93, 94, 95] to build an executable model. The patched version of SystemC facilitates the communication between the simulation kernel and the monitor and implement a random scheduler for the kernel.

In the simulation phase, PLASMA iteratively triggers the executable model to run simulations. The generated monitor observes and delivers the execution traces to PLASMA. An execution trace contains the observed variables and their simulation instances. The BLTL prop-

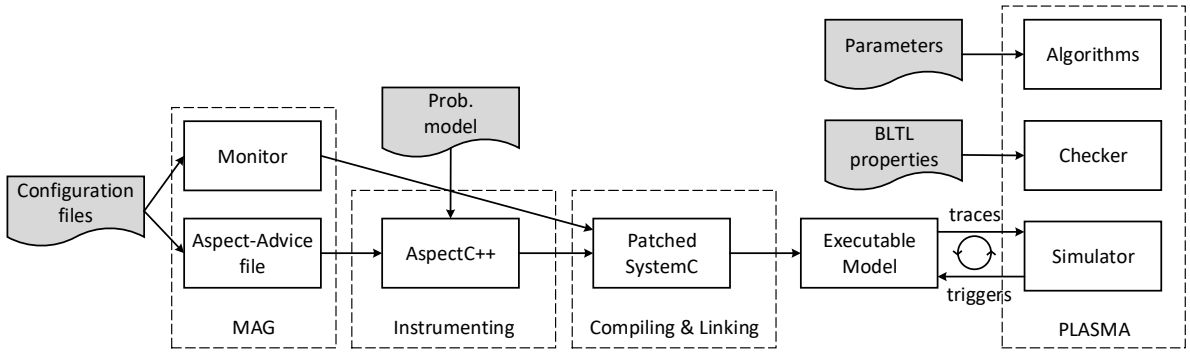


Figure 5.3 – Creation of the SystemC plugin for PLASMA workflow.

erties are used by the checker to verify the execution traces. The required number of execution traces is computed by the supported statistical algorithms in PLASMA (e.g., Monte Carlo with Chernoff-Hoeffding bounds or SPRT). Users set the parameters of the statistical algorithms to control the confidence level of the analyzed results. The obtained results are the probabilities that the model satisfies the defined properties.

In the next section, we present our case studies and the experiment results.

5.3 Experiments

5.3.1 Case studies

From the application part, we considered the two image processing SDF applications described in Chap. 3: the Sobel filter and the JPEG decoder. For the hardware platform, we used the 7 tile heterogeneous platform with two different setups. The first setup is the same as in Sec. 3.4.3. For the mapping model, the following mappings were considered: *Sobel1a*, *Sobel2a* and *Sobel4a* for Sobel filter and *Jpeg1a*, *Jpeg3a* and *Jpeg7a* for JPEG decoder.

The second setup is illustrated in Fig. 5.4. In this configuration, a private cache is activated for each tile. Two shared memories are used: a DDR and a BRAM memory. The computation code of actors is placed on the DDR memory where the code is accessed via a data/instruction bus (BUS 1) with the help of a cache controller. The inter-tile communication takes place via the data bus (BUS 0) and the shared BRAM memory. The interest of this setup is that it allows more complex instruction sections to be implemented due to larger DDR capacity. The JPËG decoder experiments have a very long execution time. We thus consider only the Sobel filter experiments: *Sobel1b*, *Sobel2b* and *Sobel4b*. Detail of the mappings can be found in Tab. 5.1.

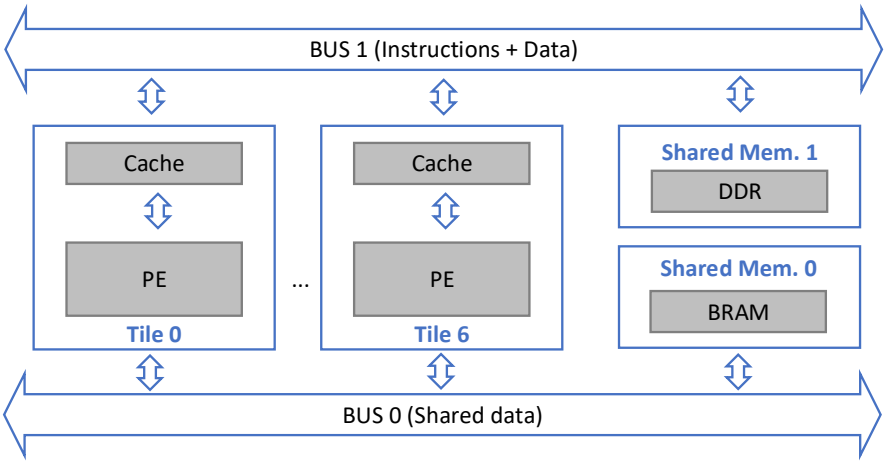


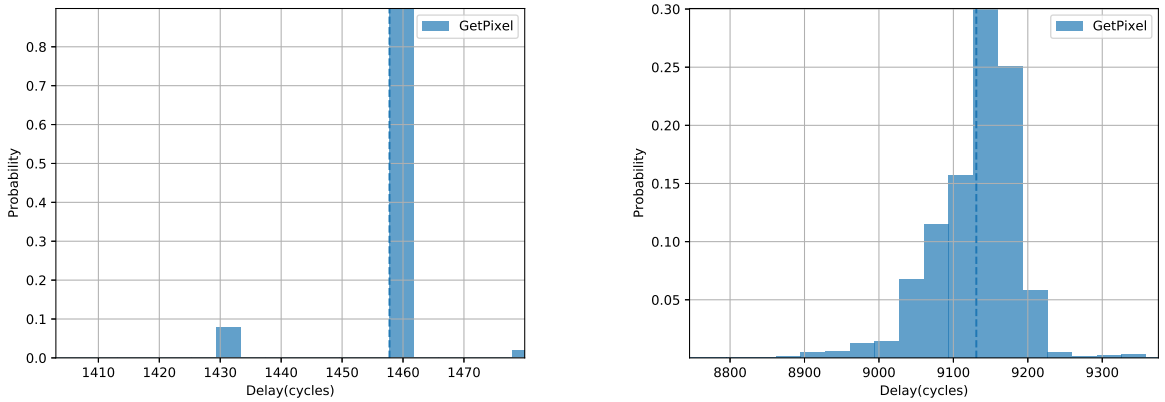
Figure 5.4 – Hardware platform with cache activated. The instructions and input data are mapped to the DDR memory and the shared data between PEs are in the BRAM memory.

Experiment→ Actor ↓	**Jpeg1a**	**Jpeg3a**	**Jpeg7a**	**Exp. → Actor ↓**	**Sobel1a**	**Sobel2a**	**Sobel4a**	**Sobel1b**	**Sobel2b**	**Sobel4b**
Get MCU	0	0	0	**GP**	1	1	1	1	1	1
IQ_Y	0	1	1	**GX**	1	2	2	1	2	2
IQ_{Cr}	0	1	2	**GY**	1	1	3	1	1	3
IQ_{Cb}	0	1	3	**ABS**	1	2	0	1	2	0
$IDCT_Y$	0	4	4							
$IDCT_{Cr}$	0	4	5							
$IDCT_{Cb}$	0	4	6							
YCrCb RGB	0	0	0							

Table 5.1 – Mapping of the Sobel filter and JPEG experiments on the tiles of the two hardware platforms.

5.3.2 Definition of used computation and communication time models

The probabilistic model written in SystemC language is created based on the computation and communication time models. In Fig. 5.5, we show the variation of the measured GetPixel's computation time in the first hardware setup (a) with the *Sobel1a* experiment and the second hardware setup (b) with the *Sobel1b* experiment. In Fig. 5.5 (a), the computation time of GetPixel actor variated within a few values around 1450 cycles, while in the Fig. 5.5 (b), this computation time increases upto around 9100 cycles with a clear variation as illustrated. The increase of the computation time in the second setup can be explained because of the longer access time to the instructions and data which are stored in the DDR memory via private caches. Beside that, the high variation come from the possible interferences between PEs to access to the DDR memory and the possible cache misses. For the computation time model of the first hardware setup, we considered gaussian and uniform distributions to model the variation of



(a) First hardware setup

(b) Second hardware setup

Figure 5.5 – The computation time of the GetPixel actor in the first hardware setup (a) and in the second hardware setup (b).

Elementary delay	**Value [BC,WC]**	**Elementary delay**	**Value [BC,WC]**
t_i^r	[146, 157]	t_i^w	[146, 157]
t_p^r	8	t_p^w	8
t_{pl}^r	7	t_{pl}^w	7
t_{pr}^r	[13, 66]	t_{pr}^w	[43, 86]
t_{rd}^r	8	t_{wr}^w	5
t_l^r	[43, 86]	t_l^w	[12, 65]
t_{po}^r	[12, 27]	t_{po}^w	[9, 36]
t_u^r	5	t_u^w	5

Table 5.2 – Elementary delays (in cycles) measured using the SystemILA.

the measured computation time, as presented in Sec. 3.3.1. However, in the second hardware setup, we only considered the gaussian distribution since the uniform distribution could not well present the variation of the measured computation delays.

For the communication time model, the proposed message-level communication model is used to capture the communication duration taking into account contention. We then built the analytical model and measured the elementary delays. The same analytical model was used for both hardware setups, as presented in Sec. 4.2.1, while the elementary delays were different for each setup. For the first hardware setup, the constant elementary delays were measured, as indicated in Tab. 3.3. For the second hardware setup, the elementary delays are shown in Tab. 5.2. We observed a limited number of iterations (10 different iterations) to capture the elementary delays because there is currently no automation tool to observe the execution process. Some constant elementary delays were observed, such as: the polling delay, the inter-polling delay, the

update buffer usage delay, the read delay and the write delay. The other delays are variable because of two reasons. First the possible cache miss situations cause penalty delays. Second, since the instructions were mapped on the DDR memory, the concurrent accesses of the PEs to read these instructions could provide interferences on the shared bus. In Tab. 5.2, we report the observed Best-Case and Worst-Case [BC, WC] of the elementary delays. We then used the uniform distribution to represent the variation of these delays in the SystemC model. Further analysis on the variation of the elementary delays should be done in the future.

These two computation and communication models are then integrated into a probabilistic simulation model which is the input of the presented PLASMA workflow.

5.3.3 Analysis results for the first hardware setup

In this section, we use PLASMA to estimate the iteration delay for different use-cases. We declared *t_latency* as a variable to observe the iteration delay in the SystemC model. Then we defined different timing properties to be verified by PLASMA.

Quantitative analysis

We first did the quantitative analysis by verifying the probability that the iteration delay stays in a given time interval [BC, WC]. We divided this time interval into several smaller interval and then observed the *probability distribution* which presents the distribution of the probability in the given time bound. This property can be expressed in BLTL as follows:

$$declare\ var := [min; max; inc]\ end$$
$$F \leq T(t\_latency \geq var)\&(t\_latency < var + inc)$$

In this BLTL expression, we declared a variable *var* that stays in a range of values [min, max]. This generates a set of BLTL formulas that can be checked simultaneously. The variable *var* is assigned a minimum value *min* and a maximum value *max*. In each instantiation, PLASMA increments an constant value *inc*. Then PLASMA quantified the probability that the iteration delay *t_latency* stay in the interval [var, var+inc]. PLASMA verifies the property in a temporal bound T which is either the number of simulation states or a real-time bound. The modal operator F means that the property is eventually satisfied. We used the Monte Carlo algorithm with Chernoff-Hoeffding bound with absolute error $\delta = 0.02$ and confidence $1 - \alpha = 0.98$. The experiments were done on an Ubuntu PC core i7 2.50 GHz with 8 Gb of RAM.

In Tab. 5.3, we show in the second column, the average iteration delay obtained from the measurement. In the next two columns, the average iteration delays obtained from the SMC analysis for the simulation model using the gaussian and uniform distributions are computed from the obtained probabilities and their time intervals. We compared the SMC analysis results

Exp.	Measured	SMC gau.	SMC unif.	1M gau.	1M unif.
Sobel1a	3690	3799.2 (2.96 %)	3743.1 (1.44 %)	3797.5 (2.91 %)	3738.5 (1.31 %)
Sobel2a	2902.5	2977.8 (2.58 %)	2924.1 (0.74 %)	2977.5 (2.58 %)	2918.5 (0.55 %)
Sobel4a	3097.4	3189.0 (2.96 %)	3133.2 (1.16 %)	3194.5 (3.13 %)	3135.5 (1.23 %)
Jpeg1a	2385860	2376010.1 (-0.42 %)	2281862.7 (-4.36 %)	2382722.5 (-0.13 %)	2247578.5 (-5.8 %)
Jpeg3a	940836	911764.7 (-3.1 %)	817079.2 (-13.16 %)	912219.5 (-3.05 %)	848620.7 (-9.81 %)
Jpeg7a	941059	886538.5 (-5.8 %)	807857.1 (-14.43 %)	896367.7 (-4.75 %)	829775.0 (-11.83 %)

Table 5.3 – Comparison of the SMC analysis results with the simulation results of 1000000 iterations using the gaussian and uniform distribution and the measured data. The table shows the average iteration delay (in cycles). The error to the measurement data is next to the results. The SMC experiments are done using the Monte-Carlo algorithm with Chernoff bound (the absolute error $\delta = 0.02$ and the confidence $1 - \alpha = 0.98$, 5757 simulation runs).

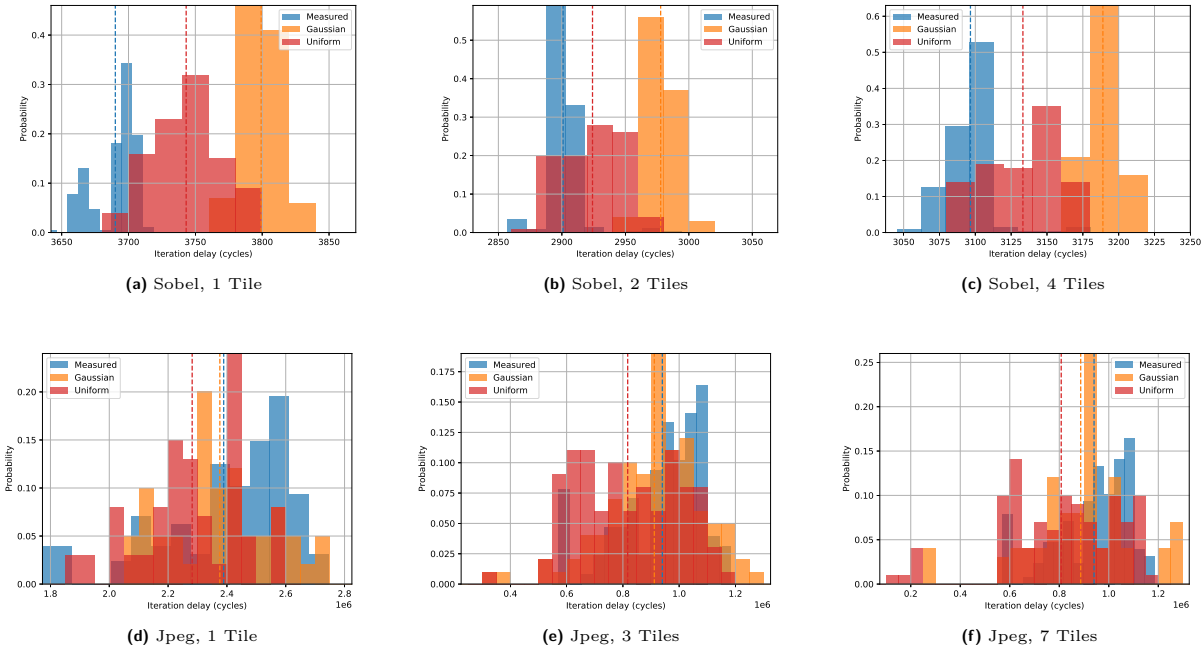
to the measured data. In PLASMA, 5757 simulation runs were done for each experiment. In the last two columns, the average iteration delay of 1000000 simulation runs for the simulation model using the gaussian and uniform distribution are then presented with the error related to the measured results.

In the Sobel filter experiments, the SMC results show a good accuracy for the model using both gaussian and uniform distribution. The simulation models using the gaussian distribution presents an over-approximation of around 3%. The simulation models using the uniform distribution shows better results with the highest error is 1.44% for the *Sobel1a* experiment. The results of 1000000 simulation runs present a same level of accuracy compared to the measured results.

For the JPEG decoder experiments, the SMC results show under-estimations compared to the measured data. The simulation models using the uniform distribution presents the highest error of 5.8% for the *Jpeg7a* experiment. The simulation models using the gaussian distribution shows higher errors upto 14.43% for the *Jpeg7a* experiment. The results of 1000000 simulation runs show better accuracy but not significant compared to the SMC results except for the *Jpeg1a* experiment with the uniform distribution.

The higher under-estimation results of the model using the gaussian distribution is due to the fact that the selected computation delays can be much higher (lower) than the worst-case (best-case) measured computation delay. The model using the uniform distribution have lower error because the selected computation delays always stay in the [BC, WC] measured computation delays.

In Fig. 5.6, we compare the iteration delay distribution of the measured data (blue) and the analyzed results using the uniform distribution (red) and the gaussian distribution (orange) of the experiments. The results of the Sobel filter experiments are shown in the upper part of Fig. 5.6. The distributions of the simulation results using gaussian distribution show a similar shape composed to the measured data. For the uniform distribution, the shape of the distributions is



(a) Sobel, 1 Tile **(b)** Sobel, 2 Tiles **(c)** Sobel, 4 Tiles

(d) Jpeg, 1 Tile **(e)** Jpeg, 3 Tiles **(f)** Jpeg, 7 Tiles

Figure 5.6 – Distribution of the measured data (blue) compared to the SMC results of the ML model using the gaussian distribution (orange) and the uniform distribution (red). The iteration delay is in cycle. Figures (a), (b) and (c) show the results of the Sobel filter . Figures (d) , (e) and (f) present the results of the JPEG decoder. The dashed lines show the median execution time.

different. The results of the JPEG decoder experiments are in the lower part. The distributions of simulated results using the gaussian distribution partly show similar shape according to the measured data. For the uniform distribution, the distributions also present different shapes. Since the JPEG decoder application consists of more actors than in the Sobel filter, the influence of the distribution representing the computation times to the iteration delay might be higher in the JPEG decoder experiments.

Simulation time

In Tab. 5.4, we present the simulation time of the experiments. In the first column, the measured duration of 1 000 000 iterations of the applications running on the real hardware platform is presented. We then show the simulation time of the SMC analysis for the simulation model using the uniform and gaussian distribution. For each SMC experiment, 5757 simulations were done with the confidence of the analysis is 98%. All the experiments took from around 30 seconds to 2 minutes. The simulation time of 1 000 000 iterations without using PLASMA are showed in the two last columns. It took from 4 seconds to around 30 seconds. The higher simulation

Experiment	Measured	SMC gau.	SMC unif.	1M gau.	1M unif.
Sobel1a	0:07:23	0:00:29	0:00:30	0:00:04	0:00:04
Sobel2a	0:07:03	0:01:39	0:01:39	0:00:05	0:00:04
Sobel4a	0:07:13	0:01:44	0:01:43	0:00:05	0:00:05
Jpeg1a	13:14:31	0:00:39	0:00:39	0:00:07	0:00:06
Jpeg3a	5:12:58	0:01:52	0:01:52	0:00:12	0:00:11
Jpeg7a	5:13:02	0:02:16	0:02:16	0:00:29	0:00:27

Table 5.4 – Simulation time (HH:MM:SS) with PLASMA using Monte-Carlo algorithm with Chernoff bound (the absolute error $\delta = 0.02$ and the confidence $1 - \alpha = 0.98$, 5755 simulation runs) and without PLASMA (1000000 simulation runs).

Absolute error δ	Confidence α	Simulation runs	Simulation time(s)	Prob.
0.01	0.01	26492	123.9	0.141
0.01	0.02	23026	105.5	0.135
0.01	0.03	20999	96.4	0.114
0.01	0.04	19561	90.3	0.062
0.01	0.05	18445	87.3	0.179
0.02	0.01	6623	33.1	0.101
0.03	0.01	2944	18.7	0.083
0.04	0.01	1656	9.6	0.118
0.05	0.01	1060	9.6	0.131

Table 5.5 – Analysis of the *Jpeg7a* experiment with the gaussian distribution using different parameters of Monte-Carlo.

time using PLASMA can be explained that PLASMA needs to capture the observed variables at every simulation state of the SystemC model. This caused the additional simulation time compared to the simulation without PLASMA.

Influences of the parameters

We aim to study the influence of the statistical algorithm to the simulation results by changing their parameters. In Tab. 5.5, we present the analysis of the *Jpeg7* experiment with the gaussian distribution using different sets of parameters (the absolute error δ and the confidence α) of the Monte-Carlo algorithm. We defined a property to express the probability that the iteration delay of the *Jpeg7a* experiment stays in the interval [950000, 1000000] cycles. For each set of parameters, we captured the number of simulation runs, the simulation time in second and the probability to satisfy this property. As we can see in Tab. 5.5, we first used the same absolute error $\delta = 0.01$ and increased the confidence α from 0.01 to 0.05. This means that the confidence of the analysis decreases from 99% to 95%. In these cases, the number of simulation

runs slightly decreased. The simulation time also decreased as well as the number of simulation runs. The obtained probability to satisfy the defined property also decreased from 0.141 to 0.062. However, it increased to 0.179 in the case of $\delta = 0.01$ and $\alpha = 0.05$. Afterward, we used the same confidence $\alpha = 0.01$ and increased the absolute error. The number of simulation runs and the simulation time decreased quickly when the absolute error increased. The confidence has much smaller influence to the number of simulation run compared to the absolute error. The obtained probability had small differences between the set of parameters. However these differences were not huge and were between 0.083 to 0.141.

Qualitative analysis

We considered the qualitative analysis by verifying another property (*cumulative probability*) to bound the worst-case iteration delay (WCID). It is the probability that the iteration delay is less or equal to a time bound. This property can be expressed in BLTL as follows:

$$declare\ var := [min; max; inc]\ end$$
$$G \leq T(t\_latency \leq var)$$

This BLTL property is used to test if all the iteration delays is less or equal to a given time bound *var* during the simulation time T. The variable *var* increments in the range [min, max]. With a temporal bound T, we quantify the probability that the property is always satisfied by using the modal operator G. In Fig. 5.7, we present the cumulative probability for the *Jpeg7a* experiment using the uniform distribution. In this experiment, we considered the temporal bound T as the duration to finish 100 iterations. It means that the property is satisfied if all 100 iteration delays are less or equal to the time bound. We used two statistical algorithms: Monte-Carlo with Chernoff bound ($\delta = \alpha = 0.02$) and SPRT ($\alpha = \beta = 0.001, \delta = 0.01, \theta = 0.9$). The evolution of the cumulative probability is illustrated for both statistical algorithms. We can identify the WCID when the cumulative probability is 1. Therefore, we bound a WCID is 1 170 000 cycles for the *Jpeg7a* experiment. PLASMA supports parallel simulation with multi-threaded execution. We used 4 threads for this experiment to reduce the simulation time. The analysis time for this experiment is 10 minutes for *Monte-Carlo* and 3 minutes for *SPRT*.

In these presented experiments on the first hardware setup, the SMC analysis presented the same level of accuracy as the simulation of 1 000 000 iterations but with much less simulation iteration (5757 iterations to have a confidence level of 98%). This validated our proposed approach to use SMC analysis to control the simulation runs and provide a quantification of the analysis confidence. Different BLTL properties can be analyzed by the proposed SMC analysis approach. Users can control the analysis process by configuring the parameters of the supported

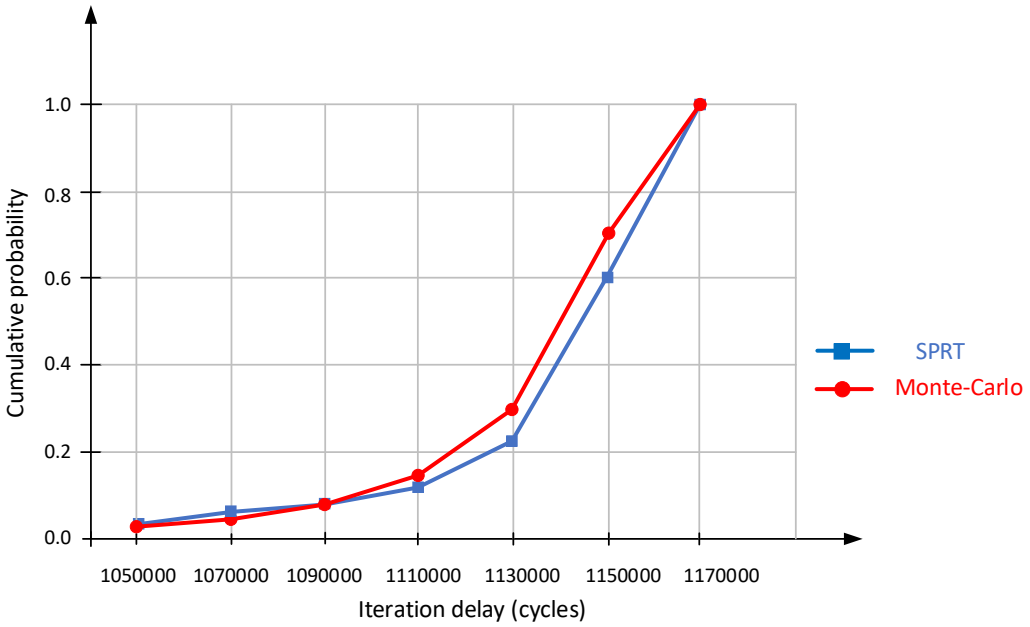


Figure 5.7 – Cumulative probability of the *Jpeg7a* experiment using the uniform distribution. The simulation is done using Monte Carlo and SPRT.

statistical algorithms. However, the simulation time of SMC analysis remains much longer than the simulation executed without SMC. This can be explained by the fact that PLASMA causes undesired delays in the analysis process. Further optimization process is needed to provide faster analysis speed of this SMC tool.

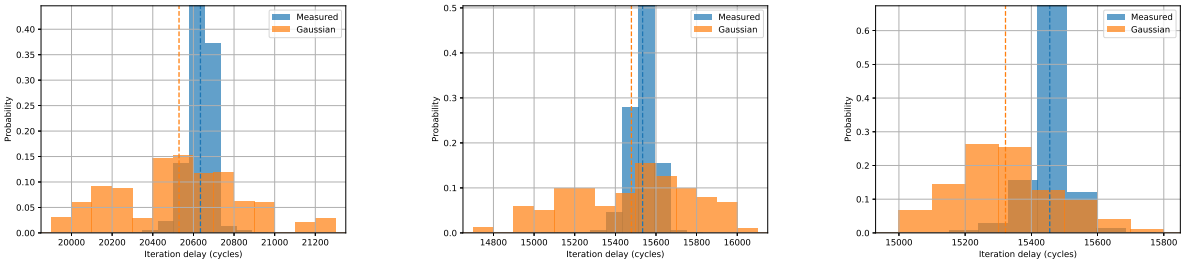
5.3.4 Analysis results for the second hardware setup

In this section, we aim to demonstrate the scalability of the proposed approach by running Sobel filter application on the second hardware setup. In this experiment, the use of cache caused higher variability of the communication time. For the computation time, we only considered the gaussian distribution since the uniform distribution do not well represent the variation of the measured computation time. For the SMC analysis, we analyzed the *distribution property* using the Monte Carlo algorithm with Chernoff-Hoeffding bound (the absolute error $\delta = 0.02$ and the confidence $1 - \alpha = 0.98$). The experiments were done on an Ubuntu PC core i7 2.50 GHz.

In Tab. 5.6, we compare the average iteration delay obtained from the SMC analysis with the simulation results of 1 000 000 iterations measured results. The iteration delays in these experiments are much higher than in the first hardware setup because of the long access time to data on the DDR memory. For the *Sobel1b* experiment, the SMC results for the models using the

Experiment	Measured	SMC gau.	1M gau.
Sobel1b	20634.2	20529.2 (-0.51 %)	20547.5 (-0.42 %)
Sobel2b	15533.8	15479.0 (-0.35 %)	15459.7 (-0.48 %)
Sobel4b	15455.1	15321.9 (-0.87 %)	14907.7 (-3.55 %)

Table 5.6 – Comparison of the results of the simulation model using the gaussian distribution with the measurement data. The table shows the average iteration delay (in cycles). The error to the measurement data is next to the results. The experiments are done using the Monte-Carlo algorithm with Chernoff bound (the absolute error $\delta = 0.02$ and the confidence $1 - \alpha = 0.98$).



(a) Sobel, 1 Tile **(b)** Sobel, 2 Tiles **(c)** Sobel, 4 Tiles

Figure 5.8 – Distribution of the measured data (blue) compared to the SMC results of the ML model using the gaussian distribution (orange) and the uniform distribution (red). The dashed lines show the median execution time.

uniform distribution showed a very good accuracy. It over-estimated around 0.04% the measured data.

The simulation models using the gaussian distribution present an under-estimation around 0.51%. For the other experiments, the simulation models using gaussian distribution showed under-approximation to the measured data. However, the errors are very low with the highest error of 1.57% in the *Sobel4b* experiment. The simulation results of 1 000 000 iterations show a similar level of accuracy compared to the SMC analysis results in the *Sobel1b* and *Sobel2b* experiments. In the *Sobel4b* experiment, the level of accuracy was decreased. This can be explained that the SMC analysis did not consider some worst-case computation delays of actors since the number of simulation runs is only 5757.

In Fig. 5.8, we compare the iteration delay distribution of the measured data (blue) and the analyzed results using the gaussian distribution (orange) of the experiments. The shape of the simulation result distributions are different from the measured data. Therefore, other distribution function is needed to better represent the measured computation time.

We show in the *Measured* column of Tab. 5.7 the measurement time of executing 1 000 000 iterations to run applications on the real hardware platform. We then show in the next two columns the simulation time of the SMC analysis for the simulation models using the gaussian

Experiment	Measured	SMC gau.	1M gau.
Sobel1b	0:07:23	0:01:43	0:00:05
Sobel2b	0:07:03	0:01:43	0:00:05
Sobel4b	0:07:13	0:01:43	0:00:05

Table 5.7 – Simulation time (HH:MM:SS) of 5757 simulation runs using Monte-Carlo algorithm with Chernoff bound (the absolute error $\delta = 0.02$ and the confidence $1 - \alpha = 0.98$). 5757 simulation runs were done in each experiment.

and uniform distribution. For each SMC experiment, 5757 simulations were done with the confidence of the analysis of 98%. All the experiments took the same simulation time of less than 2 minutes. In the last two columns, the simulation time of 1 000 000 iterations for the simulation models using the gaussian and uniform distribution is presented. The simulation time for each experiment is around 5 seconds.

In these presented experiments of the second hardware setup, the message-level communication model is still applicable to capture the variability of the communication time caused by such architecture. However, the gaussian distribution do not represent very well the variation of the measured computation time. Further investigation in inference techniques is needed to find better capture of such variation.

5.4 Conclusion

In this chapter, we have presented the use of SMC analysis approach for MPSoC systems. We validated this approach by running the two image processing applications (Sobel filter and JPEG decoder) on two different hardware platform setups. The SMC analysis showed good results compared to the measured data and the same level of accuracy as the simulation of 1 000 000 iterations. Two BLTL timing properties were analyzed in this chapter, the *probability distribution* and the *cumulative probability*. This showed the possibility to apply such SMC approach for verifying other timing properties, such as the probability to miss a deadline of task. The errors to the measured data are caused by the gaussian and uniform distributions which did not well represent the variation of the measured computation time. The establishment of appropriate distribution function is needed to improve the SMC analysis results. The statistical inference technique, called *Kernel Density Estimation* (KDE) [96, 97] could be considered, it is a method to estimate the probability density function based on an incomplete set of samples. The KDE processed data represents the sum of all individual distribution functions. This method allows us to not only cover observed execution time during our simulation, but also other delays that are likely to appear when using the actor with different stimuli. In our experiments, the simulation time of PLASMA SMC tool is from seconds to minutes to finish the analysis which is

still much longer than the traditional simulation run. Thus, the optimization of this SMC tool is needed to reduce its analysis time. Furthermore, we can also run the analysis on cluster to reduce the analysis time because this SMC tool supports distributed simulation.

Chapter 6

Conclusions and perspectives

In this chapter, we summarize our contributions presented in this manuscript. Some perspectives are also proposed to further develop our work in the future.

6.1 Conclusions

In the context the complexity increases of multi-processor system-on-chip (MPSoC) systems in both software and hardware, the performance evaluation of such systems is very important in the early phases of the design process. Such performance evaluation verifies that the system constraints are met, especially timing. In MPSoCs systems, concurrent accesses to shared resources cause interferences which lead to additional delays and thus create high variations in software execution time. However, in the context of software performance prediction on multi-processor platforms, creation of high abstraction level models with good prediction accuracy at affordable analysis times represents a key challenge. One issue that makes creation of efficient performance models a time-consuming effort is due to complex interactions among shared resources. Thus the timing prediction of such systems is a challenging task. Moreover, an appropriate analysis approach must be chosen which not only provides a good compromise between precision and analysis time, but also good scalability.

In this thesis, we aim to study the adoption of probabilistic modeling and analysis methods to improve the efficiency of timing analysis approaches for MPSoC systems. Several contributions are presented to achieve this goal:

First, we proposed a systematic evaluation workflow in Chap. 3 that considers probabilistic modeling and analysis approaches for MPSoC systems. This workflow is based on three main parts:

— A measurement-based approach is first used to characterize timing behaviors of actors that constitute the application on a real platform. This characterization is done for both computation and communication parts

— A probabilistic modeling approach describes both software and hardware parts on a specific programming language. This description captures the variation of the execution time by using statistical timing models.

— A probabilistic analysis approach is based on statistical model checking (SMC) which estimates a probability that the created model can satisfy a timing property.

In this workflow, created performance models are separated between computation and communication modeling. For the computation time model, we use the injected data and the gaussian/uniform distributions to represent the variation of the measured computation delays. For the communication time model, we consider the communication model that is presented in Section 3.3.2. We then presented our preliminary results of two image processing applications: the Sobel filter and the JPEG decoder executed on a 7 tile heterogeneous platform. The simulation results are compared to the measured results to show the validation of the proposed approach. For the average iteration delay, our simulation model using the injected data presented an under-approximation of 7.7% for the *Sobel4* experiment and only 1.6% for the *Jpeg7* experiment. Different aspects should be improved in the proposed approach, such as the pessimistic communication time model that caused the errors of the analysis and the long simulation time (around 2 hours for the *Jpeg7* experiment).

Second, we proposed a message-level communication model of a shared bus in Chapter 4 to deliver fast yet accurate simulation results compared to simulation results at transaction level and the measured results obtained from real implementation on FPGA. This message-level communication model is based on a run-time prediction technique of the whole communication time of the application actors. In this message-level model, the communications time predictions are done by using an analytical model which is defined from our observations of the execution of the application on the hardware platform. We applied this ML model to a FCFS bus arbitration policy AXI4LITE. The injected data and the gaussian/uniform distributions are considered to build the computation time model. The simulation results of this ML model showed a significant improvement in both accuracy and simulation time compared to the TL model. For example, the average execution time of the ML model using the injected data presented an over-approximation of 1.81% for the *Sobel4* experiment and 0.05% for the *Jpeg7* experiment. For the simulation time, the ML model showed a reduction factor of up to 1976 compared to the TL model.

Finally, we presented a *probabilistic analysis approach* in Chap. 5 which considers statistical model checking (SMC) method to analyze MPSoC systems. This SMC approach partially explore system state-space, but still makes possible to bound the probability of making an error about predictions by controlling the number of simulation runs by using statistical algorithms. In this analysis, the effects of the statistical algorithms parameters to the analysis results are further investigated. We also used two case studies: Sobel filter and JPEG decoder to validate this approach. We considered the uniform/gaussian distributions to represent the variation of the measured computation time. Two different hardware platform setups are studied which are the 7 tile heterogeneous platform without private cache for the first setup and with private cache for the second setup. The SMC analysis results are obtained from 5757 iteration runs that show

good accuracy compared to the measured data and provide a same level of accuracy as the SystemC execution analysis of 1 000 000 iterations. For example, the SMC results of using the gaussian distribution presented an over-approximation of 2.96% for the *Sobel4* experiment and an under-approximation of 5.8% for the *Jpeg7* experiment. The errors to the measured data are caused by the gaussian and uniform distributions which did not well represent the variation of the measured computation time. In our experiments, the simulation time is from seconds to minutes to finish the analysis which is still much longer than the SystemC execution analysis.

6.2 Perspectives

Different aspects should be considered to improve the proposed approach. We propose to extend our work in the future as follows:

1. From the communication part, we have considered a FCFS bus arbitration protocol to demonstrate the efficiency of the message level communication model. The use of another bus arbitration policy (e.g., TDMA) is needed to demonstrate the adaptability of the proposed approach. In that case, we have to characterize a new message-level communication model for the considered bus which consists of its analytical model and elementary delays. This one-time characterization phase takes overhead duration.
2. For the characterization of the elementary delays, we executed communication workload on a hardware platform and characterized the elementary delays by observing a limited number of iterations. However, this could not guarantee that the worst-case elementary delays have been captured in the case of using private cache for the processing elements. The variation of some elementary delays were thus not well captured. Therefore, a measurement tool is needed to observe more iterations to capture such variations of the elementary delays.
3. From the computation part, we have considered the gaussian/uniform distributions which did not very well represent the variation of the measured computation time. Thus, further investigation in the inference techniques is needed to improve the simulation results, such as the Kernel Density Estimation (KDE). This is a method to estimate the probability density function based on an incomplete set of samples. The KDE processed data represents the sum of all individual distribution functions. This method allows to not only cover observed execution time during the simulation, but also other delays that are likely to appear when using the actor with different stimuli.
4. For the SMC analysis using PLASMA, it requires several intermediate steps to create the SystemC plugins used by this tool, such as the generating step using the MAG tool, the instrumenting step using the AspectC++ or the compiling and linking steps. These steps causes overhead time which should be reduced by creating an automation script that

runs these steps sequentially. Beside that, PLASMA observes all the simulation states of the performance model that requires additional time in simulation process. This leads to much longer analysis time compared to SystemC execution analysis. Therefore, further optimization of this tool is needed to reduce the analysis time.

5. In terms of case studies, we have considered two simple image processing applications: Sobel filter and JPEG decoder executed on a 7 tile heterogeneous platform (with optional L1 caches) in this thesis. More complex case studies should be considered to demonstrate the scalability of the proposed approach. This means more complex industrial applications and COST platform extended with L2 cache should be taken into account.
6. Different timing properties should be taken into account to better analyze the MPSoC systems, such as the probability to miss a deadline of tasks. The influences of statistical algorithms to the analysis results should be further investigated to find the best trade-off between the accuracy and analysis efforts.
7. Extend the performance evaluation approach for other properties of MPSoC systems, such as energy, power consumption, temperature, etc. This requires an appropriate way to create the performance model for such properties.

BIBLIOGRAPHY

[1] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, vol. 100, no. Special Centennial Issue, pp. 1411–1430, 2012.

[2] A. Nouri, "Rigorous system-level modeling and performance evaluation for embedded system design," Ph.D. dissertation, 2015.

[3] M. Radetzki, *Languages for Embedded Systems and Their Applications: Selected Contributions on Specification, Design, and Verification from FDL'08.* Springer Science & Business Media, 2009.

[4] O. Kotaba, J. Nowotsch, M. Paulitsch, S. M. Petters, and H. Theiling, "Multicore in real-time systems – temporal isolation challenges due to shared resources," *Workshop on Industry-Driven Approaches for Cost-effective Certification of Safety-Critical, Mixed-Criticality Systems*, 2014.

[5] A. Abel, F. Benz, J. Doerfert, B. Dörr, S. Hahn, F. Haupenthal, M. Jacobs, A. H. Moin, J. Reineke, B. Schommer *et al.*, "Impact of resource sharing on performance and performance prediction: A survey," *International Conference on Concurrency Theorys*, pp. 25–43, 2013.

[6] D. Monniaux and V. Touzeau, "On the complexity of cache analysis for different replacement policies," *Journal of the ACM (JACM)*, vol. 66, no. 6, pp. 1–22, 2019.

[7] T. Mitra, J. Teich, and L. Thiele, "Time-critical systems design: A survey," *IEEE Design & Test*, vol. 35, no. 2, pp. 8–26, 2018.

[8] S. Hahn, J. Reineke, and R. Wilhelm, "Towards compositionality in execution time analysis: definition and challenges," *ACM SIGBED Review*, vol. 12, no. 1, pp. 28–36, 2015.

[9] T. Lundqvist and P. Stenstrom, "Timing anomalies in dynamically scheduled microprocessors," in *Proceedings 20th IEEE Real-Time Systems Symposium (Cat. No. 99CB37054)*. IEEE, 1999, pp. 12–21.

[10] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand, "Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 7, pp. 966–978, 2009.

[11] J. Reineke and R. Sen, "Sound and efficient wcet analysis in the presence of timing anomalies," in *9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

[12] J. Reineke, B. Wachter, S. Thesing, R. Wilhelm, I. Polian, J. Eisinger, and B. Becker, "A definition and classification of timing anomalies," in *6th International Workshop on Worst-Case Execution Time Analysis (WCET'06)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[13] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," *Proceedings of Embedded Real Time Software and Systems*, vol. 36, p. 42, 2010.

[14] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg, "State-based real-time analysis of sdf applications on mpsocs with shared communication resources," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 486–509, 2015.

[15] J. Schneider, *Combined schedulability and WCET analysis for real-time operating systems*. Shaker, 2003.

[16] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded system design: modeling, synthesis and verification*. Springer Science & Business Media, 2009.

[17] T. Grötker, S. Liao, G. Martin, and S. Swan, *System Design with SystemCTM*. Springer Science & Business Media, 2007.

[18] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao, *SpecC: Specification language and methodology*. Springer Science & Business Media, 2012.

[19] R. Dömer, A. Gerstlauer, J. Peng, D. Shin, L. Cai, H. Yu, S. Abdi, and D. D. Gajski, "System-on-chip environment: A specc-based framework for heterogeneous mpsoc design," *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, p. 647953, 2008.

[20] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. P. Stefanov, D. D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, 2009.

[21] J. Keinert, M. Streubūhr, T. Schlichter, J. Falk, J. Gladigau, C. Haubelt, J. Teich, and M. Meredith, "Systemcodesigner—an automatic esl synthesis approach by design space exploration and behavioral synthesis for streaming applications," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 14, no. 1, pp. 1–23, 2009.

[22] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP Journal on Embedded Systems*, vol. 2007, no. 1, p. 082123, 2007.

[23] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna, "Uml-based multiprocessor soc design framework," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 5, no. 2, pp. 281–320, 2006.

[24] J. Kreku, M. Hoppari, T. Kestilä, Y. Qu, J.-P. Soininen, P. Andersson, and K. Tiensyrjä, "Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 1–18, 2008.

[25] T. Arpinen, E. Salminen, T. D. Hämäläinen, and M. Hännikäinen, "Performance evaluation of uml2-modeled embedded streaming applications with system-level simulation," *EURASIP Journal on Embedded Systems*, vol. 2009, no. 1, p. 826296, 2009.

[26] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. González Harbour, "Influence of different abstractions on the performance analysis of distributed hard real-time systems," *Des. Autom. Embedded Syst.*, vol. 13, no. 1-2, pp. 27–49, Jun. 2009. [Online]. Available: http://dx.doi.org/10.1007/s10617-008-9015-1

[27] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming*, vol. 40, no. 2-3, pp. 117–134, 1994.

[28] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 11, pp. 1125–1136, 1998.

[29] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," in *Proceedings of the tenth international symposium on Hardware/software codesign*, 2002, pp. 187–192.

[30] K. Huang, W. Haid, I. Bacivarov, M. Keller, and L. Thiele, "Embedding formal performance analysis into the design cycle of mpsocs for real-time streaming applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 11, no. 1, pp. 1–23, 2012.

[31] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the symta/s approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[32] J.-P. Katoen and H. Wu, "Probabilistic model checking for uncertain scenario-aware data flow," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, pp. 1–27, 2016.

[33] R. Stemmer, M. Fakih, K. Grüttner, and W. Nebel, "Towards state-based rt analysis of fsm-sadfgs on mpsocs with shared memory communication," in *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. ACM, 2017, p. 6.

[34] C. Norstrom, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No. PR00306)*. IEEE, 1999, pp. 182–189.

[35] M. Lv, W. Yi, N. Guan, and G. Yu, "Combining abstract interpretation with model checking for timing analysis of multicore software," in *2010 31st IEEE Real-Time Systems Symposium*, Nov 2010, pp. 339–349.

[36] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *International conference on runtime verification*. Springer, 2010, pp. 122–135.

[37] P. Bulychev, A. David, K. G. Larsen, M. Mikučionis, D. B. Poulsen, A. Legay, and Z. Wang, "UPPAAL-SMC: Statistical model checking for priced timed automata," *arXiv preprint arXiv:1207.1272*, 2012.

[38] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *International conference on computer aided verification*. Springer, 2011, pp. 585–591.

[39] C. Jegourel, A. Legay, and S. Sedwards, "A platform for high performance statistical model checking–plasma," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2012, pp. 498–503.

[40] C. Ngo Van, A. Legay, and J. Quilbeuf, "Statistical model checking for systemc models," in *High Assurance Systems Engineering Symposium*, 2016.

[41] A. Nouri, M. Bozga, A. Molnos, A. Legay, and S. Bensalem, "Building faithful high-level models and performance evaluation of manycore embedded systems," in *2014 Twelfth ACM/IEEE Conference on Formal Methods and Models for Codesign (MEMOCODE)*. IEEE, 2014, pp. 209–218.

[42] A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, and A. Legay, "Statistical model checking qos properties of systems with sbip," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 2, pp. 171–185, 2015.

[43] Y. Bao, M. Chen, Q. Zhu, T. Wei, F. Mallet, and T. Zhou, "Quantitative performance evaluation of uncertainty-aware hybrid aadl designs using statistical model checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 12, pp. 1989–2002, 2017.

[44] A. Gerstlauer, C. Haubelt, A. D. Pimentel, T. Stefanov, D. D. Gajski, and J. Teich, "Electronic system-level synthesis methodologies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1517–1530, 2009.

[45] M. Thompson, H. Nikolov, T. Stefanov, A. D. Pimentel, C. Erbas, S. Polstra, and E. F. Deprettere, "A framework for rapid system-level exploration, synthesis, and programming of multimedia mp-socs," in *2007 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 9–14.

[46] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. Deprettere, "Daedalus: Toward composable multimedia mp-soc design," in *2008 45th ACM/IEEE Design Automation Conference*, 2008, pp. 574–579.

[47] A. Gerstlauer, J. Peng, D. Shin, D. Gajski, A. Nakamura, D. Araki, and Y. Nishihara, "Specify-explore-refine (ser): from specification to implementation," in *2008 45th ACM/IEEE Design Automation Conference*. IEEE, 2008, pp. 586–591.

[48] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, and H. T.D., "Uml-based multiprocessor soc design framework," *ACM Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 281–320, 5 2006.

[49] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP Journal on Embedded Systems*, 2007.

[50] C. Norstrom, A. Wall, and Wang Yi, "Timed automata as task models for event-driven systems," in *Proceedings Sixth International Conference on Real-Time Computing Systems and Applications. RTCSA'99 (Cat. No.PR00306)*, 1999, pp. 182–189.

[51] M. Lv, W. Yi, N. Guan, and G. Yu, "Combining abstract interpretation with model checking for timing analysis of multicore software," in *2010 31st IEEE Real-Time Systems Symposium*, 2010, pp. 339–349.

[52] M. Fakih, K. Grüttner, M. Fränzle, and A. Rettberg, "Towards performance analysis of sdfgs mapped to shared-bus architectures using model-checking," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '13.

San Jose, CA, USA: EDA Consortium, 2013, pp. 1167–1172. [Online]. Available: http://dl.acm.org/citation.cfm?id=2485288.2485570

[53] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal aspects of computing*, vol. 6, no. 5, pp. 512–535, 1994.

[54] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2007, pp. 220–270.

[55] "http://www.prismmodelchecker.org/."

[56] A. Nouri, S. Bensalem, M. Bozga, B. Delahaye, C. Jegourel, and A. Legay, "Statistical model checking qos properties of systems with sbip," *International Journal on Software Tools for Technology Transfer*, vol. 17, no. 2, pp. 171–185, 2014.

[57] M. Chen, D. Yue, X. Qin, X. Fu, and P. Mishra, "Variation-aware evaluation of mpsoc task allocation and scheduling strategies using statistical model checking," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 199–204.

[58] R. Stemmer, H.-D. Vu, M. Fakih, K. Grüttner, S. Le Nours, and S. Pillement, "Feasibility Study of Probabilistic Timing Analysis Methods for SDF Applications on Multi-Core Processors," IETR ; OFFIS, Research Report, Mar. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02071362

[59] A. Legay and S. Sedwards, "On statistical model checking with plasma," in *2014 Theoretical Aspects of Software Engineering Conference*, 2014, pp. 139–145.

[60] O. Spinczyk, A. Gal, and W. Schröder-Preikschat, "Aspectc++ an aspect-oriented extension to the c++ programming language," in *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications*, 2002, pp. 53–60.

[61] S. Dutta, D. Tabakov, and M. Y. Vardi, "Chimp: a tool for assertion-based dynamic verification of systemc models," *Program Proceedings*, p. 38, 2013.

[62] M. Okamoto, "Some inequalities relating to the partial sum of binomial probabilities," *Annals of the institute of Statistical Mathematics*, vol. 10, no. 1, pp. 29–35, 1959.

[63] G. Schirner and R. Domer, "Result-oriented modeling—a novel technique for fast and accurate tlm," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 26, no. 9, pp. 1688–1699, 2007.

[64] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 8, no. 1, pp. 1–29, 2009.

[65] A. Bobrek, J. M. Paul, and D. E. Thomas, "Stochastic contention level simulation for single-chip heterogeneous multiprocessors," *IEEE Transactions on Computers*, vol. 59, no. 10, pp. 1402–1418, 2010.

[66] S. Le Nours and A. Postula, "A hybrid simulation approach for fast and accurate timing analysis of multi-processor platforms considering communication resources conflicts," *Journal of Signal Processing Systems*, vol. 90, no. 12, pp. 1667–1685, 2018.

[67] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[68] R. Stemmer, H. Schlender, M. Fakih, K. Grüttner, and W. Nebel, "Probabilistic state-based RT-analysis of SDFGs on MPSoCs with shared memory communication," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 3 2019.

[69] R. Stemmer, H.-D. Vu, K. Grüttner, S. Le Nours, W. Nebel, and S. Pillement, "Experimental evaluation of probabilistic execution-time modeling and analysis methods for SDF applications on MPSoCs," in *2019 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. Springer, 7 2019, pp. 241–254.

[70] C. Schlaak, M. Fakih, and R. Stemmer, "Power and execution time measurement methodology for sdf applications on fpga-based mpsocs," *arXiv preprint arXiv:1701.03709*, 2017.

[71] A. Limited, "Amba® axi™ and ace™ protocol specification," aXI3, AXI4, and AXI4-Lite ACE and ACE-Lite.

[72] "https://www.xilinx.com/support/documentation/ip_documentation/system_ila/v1_0/pg261-system-ila.pdf."

[73] I. S. Association *et al.*, "Ieee standard for standard systemc language reference manual," *IEEE Computer Society*, 2012.

[74] L. Cai and D. Gajski, "Transaction level modeling: An overview," in *Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES+ISSS '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 19–24. [Online]. Available: https://doi.org/10.1145/944645.944651

[75] V. C. Ngo, A. Legay, and J. Quilbeuf, "Statistical model checking for systemc models," *2016 IEEE 17th International Symposium on High Assurance Systems Engineering*, pp. 197–204, 2016.

[76] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, "The gnu scientific library reference manual, 2007," 2007. [Online]. Available: http://www.gnu.org/software/gsl

[77] A. Nouri, M. Bozga, A. Moinos, A. Legay, and S. Bensalem, "Building faithful high-level models and performance evaluation of manycore embedded systems," in *ACM/IEEE International conference on Formal methods and models for codesign*, 2014.

[78] S. Le Nours and A. Postula, "A hybrid simulation approach for fast and accurate timing analysis of multi-processor platforms considering communication resources conflicts," *Journal of Signal Processing Systems*, vol. 90, no. 12, pp. 1667–1685, 2018.

[79] F. Baccelli, G. Cohen, G. Olsder, and J. Quadrat, *Synchronization and linearity, an algebra for discrete event systems.* New York: Wiley & Sons Ltd, 1992.

[80] B. Li, J. Zhao, J. Wang, and W. Dou, "A max-plus algebra approach for network-on-chip end-to-end delay estimation," in *2012 Eighth International Conference on Semantics, Knowledge and Grids*, 2012, pp. 217–220.

[81] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," *International conference on runtime verification*, pp. 122–135, 2010.

[82] H. L. Younes, "Verification and planning for stochastic processes with asynchronous events," CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, Tech. Rep., 2005.

[83] H. L. Younes and R. G. Simmons, "Statistical probabilistic model checking with a focus on time-bounded properties," *Information and Computation*, vol. 204, no. 9, pp. 1368–1409, 2006.

[84] A. Wald, "Sequential tests of statistical hypotheses," *The annals of mathematical statistics*, vol. 16, no. 2, pp. 117–186, 1945.

[85] H. L. Younes, "Verification and planning for stochastic processes with asynchronous events," Ph.D. dissertation, Carnegie Mellon University, 2005.

[86] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet, *Approximate probabilistic model checking.* Springer, 2004, no. 73–84.

[87] V. C. Ngo, A. Legay, and J. Quilbeuf, "Dynamic verification of systemc with statistical model checking," *arXiv preprint arXiv:1412.0885*, 2014.

[88] "Plasma project page. https://project.inria.fr/plasma-lab/."

[89] A. Legay and S. Sedwards, "On statistical model checking with plasma," 2014.

[90] B. Boyer, K. Corre, A. Legay, and S. Sedwards, "Plasma-lab: A flexible, distributable statistical model checking library," in *International Conference on Quantitative Evaluation of Systems*. Springer, 2013, pp. 160–164.

[91] S. Dutta, D. Tabakov, and M. Y. Vardi, "Chimp: a tool for assertion-based dynamic verification of systemc models," *Program Proceedings*, p. 38, 2013.

[92] D. Tabakov, K. Y. Rozier, and M. Y. Vardi, "Optimized temporal monitors for systemc," *Formal Methods in System Design*, vol. 41, no. 3, pp. 236–268, 2012.

[93] D. Tabakov and M. Y. Vardi, "Monitoring temporal systemc properties," in *Eighth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE 2010)*. IEEE, 2010, pp. 123–132.

[94] ——, "Automatic aspectization of systemc," in *Proceedings of the 2012 workshop on Modularity in Systems Software*, 2012, pp. 9–14.

[95] D. Tabakov, G. Kamhi, M. Y. Vardi, and E. Singerman, "A temporal language for systemc," in *2008 Formal Methods in Computer-Aided Design*. IEEE, 2008, pp. 1–9.

[96] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *Ann. Math. Statist.*, vol. 27, no. 3, pp. 832–837, 09 1956. [Online]. Available: https://doi.org/10.1214/aoms/1177728190

[97] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, no. 3, pp. 1065–1076, 09 1962. [Online]. Available: https://doi.org/10.1214/aoms/1177704472

List of publications

International publications:

1. Hai-Dang Vu, Sébastien Le Nours, Sébastien Pillement, Ralf Stemmer, Kim Grüttner. A Fast Yet Accurate Message-level Communication Bus Model for Timing Prediction of SDFGs on MPSoC. Asia and South Pacific Design Automation Conference_ASP-DAC 2021 (Virtual Conference), Jan 2021, Tokyo, Japan. pp.1183.
2. Ralf Stemmer, Hai-Dang Vu, Kim Grüttner, Sébastien Le Nours, Wolfgang Nebel, Sébastien Pillement. Towards Probabilistic Timing Analysis for SDFGs on Tile Based Heterogeneous MPSoCs. 10th European Congress on Embedded Real Time Software and Systems (ERTS 2020), Jan 2020, Toulouse, France. #paper 59.
3. Ralf Stemmer, Hai-Dang Vu, Kim Grüttner, Sébastien Le Nours, Wolfgang Nebel, Sébastien Pillement. Experimental Evaluation of Probabilistic Execution-Time Modeling and Analysis Methods for SDF Applications on MPSoCs. International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIX), Jul 2019, Samos, Greece. paper #28.

Technical reports:

1. Ralf Stemmer, Hai-Dang Vu, Maher Fakih, Kim Grüttner, Sébastien Le Nours, Sébastien Pillement. Feasibility Study of Probabilistic Timing Analysis Methods for SDF Applications on Multi-Core Processors. [Research Report] IETR; OFFIS. 2019.

National presentation:

1. Poster in Ecole d'hiver Francophone sur les Technologies de Conception des Systèmes embarqués Hétérogènes (FETCH 2018), Saint-Malo, France.
2. Poster in 2nd Scientific day – Hardware interference and temporal determinism for modern SoC 2018 , IRT Saint Exupéry, Toulouse, France.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE NANTES

Titre : Modèles de performance précis et rapides pour l'analyse probabiliste des propriétés temporelles de SDFGs sur MPSoCs.

Mot clés : Modélisation probabiliste haut niveau, Analyse temporelle, Modèle Checking Statistique, MPSoC

Résumé : L'analyse temporelle est une étape très importante dans la conception d'un système multiprocesseur sur puce (MPSoC) pour garantir que les contraintes de temps sont pleinement respectées avec une durée d'analyse acceptable. Cependant, les interférences sur l'accès aux ressources partagées des MPSoC entraînent la variabilité de l'exécution du programme qui conduit à des difficultés pour l'analyse temporelle. Cette thèse vise à étudier l'adoption de méthodes de modélisation et d'analyse probabilistes pour améliorer l'efficacité du processus d'analyse temporelle des systèmes MPSoC.
Nous avons contribué à une approche basée sur la mesure pour caractériser les temps de calcul et de communication des applications SDFG fonctionnant sur une plate-forme MPSoC basée sur des tuiles. Dans cette approache, les effets des ressources partagées sont saisis et représentés comme des fonctions de distribution. Nous proposons un modèle de communication au niveau message d'un bus multiprocesseur pour fournir des résultats de simulation rapides mais précis. Le modèle proposé a montré une accélération significative de la simulation par rapport au modèle au niveau transactionnel (TLM) sans dégrader la précision de l'analyse. Nous évaluons certaines méthodes de modèle checking statistique (SMC) pour démontrer l'efficacité de l'analyse temporelle probabiliste des systèmes MPSoC. Dans cette analyse, différents algorithmes statistiques sont étudiés plus en détail. Enfin, l'efficacité de l'approche proposée est évaluée en exécutant différentes applications de traitement d'images sur différentes configurations d'une architecture matérielle hétérogène. Les résultats de la simulation ont montré un temps de simulation rapide avec des résultats précis par rapport aux résultats mis en œuvre sur une plate-forme matérielle réelle FPGA.

Title: Fast and Accurate Performance Models for Probabilistic Timing Analysis of SDFGs on MPSoCs

Keywords: High-level probabilistic modeling, Timing Analysis, Statistical Model Checking, MPSoC

Abstract: Timing analysis is a very important step in the design phase of multiprocessor system-on-chip (MPSoC) to guarantee that timing constraints are fully met with acceptable analysis duration. However, interferences on accessing to shared resources of MPSoCs cause variability of the program execution which leads to difficulties for timing analysis. This thesis aims to study the adoption of probabilistic modeling and analysis methods to improve the efficiency of the timing analysis process of MPSoC systems.
We have contributed to a measurement-based approach for characterizing computation and communication times of SDFG applications running on a tile-based MPSoC platform. In this approach, shared resource effects are captured and represented as distribution functions. We propose a message-level communication model of a multiprocessor bus to deliver fast yet accurate simulation results. The proposed model showed a significant simulation speed-up comparing to the transaction-level model (TLM) without degrading the analysis accuracy. We evaluate some statistical model checking (SMC) methods to demonstrate the efficiency of probabilistic timing analysis of MPSoC systems. In this analysis, different statistical algorithms and their parameters are further investigated. Finally, the efficiency of the proposed approach is evaluated by running different image processing applications on different configurations of a heterogeneous hardware architecture. Simulation results showed a fast simulation time with accurate results comparing to the measured results from the implementation of the applications on real hardware platform FPGA.