INSTITUTO UNIVERSITARIO DE SISTEMAS INTELIGENTES Y APLICACIONES NUMÉRICAS EN INGENIERÍA

Programa de doctorado
Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería

TESIS DOCTORAL

System Level Design Space Exploration for MPSoC: Methods, Algorithms and New Infrastructure

Firma de los Directores de la Tesis

Dr. D. Tomás Bautista Delgado **Dr. D. Antonio Núñez Ordónez**

Firma del Doctorando

D. Jia Li Zai Jian

Jia Li Zai Jian

Las Palmas de Gran Canaria, 2010

Para mi padre, mi madre y mi hermana.

Sois todo lo que necesito.

Agradecimientos

“El maestro y el pupilo caminan juntos por el sendero del saber hasta que sus destinos les separen.”

– Proverbio chino –

Este proverbio chino refleja con bastante acierto el transcurrir de mi vida en estos últimos años. Lo cierto es que desde que terminé la carrera de Ingeniería de Telecomunicación, me propuse como meta personal encontrar mi sitio en el mundo. Y no me refiero solamente a un sitio donde asentar mi vida, sino también un sitio espiritual e intelectual en el que me sintiera a gusto, y en el que me pudiera dedicar profesionalmente en el día de mañana. Sin lugar a duda, mis genes y orígenes orientales han contribuido en gran medida a que tenga este tipo de reflexiones metafísicas a lo largo de mi vida.

Y para encontrar este sitio que tanto ansío, empecé con este Doctorado con el fin de seguir aprendiendo y profundizando en aquellos campos que más me atrajeron durante la carrera. En este punto, doy las gracias al grupo del profesor Antonio Falcón, Cayetano Guerra y Mario Hernández por recibirme con los brazos abiertos y transmitirme sus conocimientos sobre el mundo de la visión por computador, unos conocimientos que me ha permitido conocer la complejidad del área de los Sistemas Inteligentes Autónomos en los primeros años de mi doctorado.

Sin embargo, lo que yo no sabía por entonces es que mi viaje por el sendero del conocimiento me tenía preparado años más tarde una parada en la Universidad de Amsterdam. Esta estancia no sólo ha sido enriquecedora a nivel intelectual, sino también a nivel afectivo. He de reconocer que aparte de un ambiente de trabajo inmejorable, también tuve la suerte de conocer a unos buenos amigos como Andy Pimentel y Mark Thompson. De hecho, parte del mérito de esta tesis se debe en gran medida a sus dedicaciones desinteresadas y a sus enseñanzas sobre el diseño a nivel de sistema, para las cuales no encuentro palabras para expresar mi gratitud hacia ellos.

Por otro lado, debo insistir en que navegar por los mares del conocimiento sin una brújula que te guía es como dar palos de ciego. Y por ello, quiero agradecer en este punto a mis directores de tesis, Tomás Bautista y Antonio Núñez, por su dedicación y su siempre disponibilidad a lo largo de estos años. Doy las gracias a Antonio por

transmitirme su optimismo y alegría, así como los sabios consejos que me ha dado en estos últimos años, los cuales me han orientado en todo momento hacia la dirección que debo avanzar. Asimismo, también quiero agradecer a Tomás, quien aparte de ser uno de mis directores de tesis, ha sido un bueno amigo, y que **siempre** ha estado a mi lado para lo bueno y para lo malo, y especialmente le doy las gracias por ser esa farola que necesitaba en los días de oscuridad (que a propósito, no fueron pocos).

Finalmente, estoy convencido de que hoy no estaría aquí si no fuese por mis seres más queridos: mi familia y mis amigos. A mis amigos les doy las gracias por sus ánimos y las palmadas en la espalda en los momentos que más lo necesitaba, los cuales también me ayudaron a ver la luz del final del túnel. Asimismo, doy las gracias a mi familia por su apoyo incondicional. Especialmente, agradezco a mi padre por su paciencia y sabiduría, a mi madre por su amor y lucha, y a mi hermana por su sacrificio y comprensión.

Por último, me despido diciendo que creo firmemente en que el destino de cada persona no es algo que está fijado de antemano, sino que somos nosotros los que vamos escribiendo con nuestras acciones del día a día las páginas de la historia de nuestra vida. Y la pregunta es: ¿qué voy a escribir en la mía?

"Ladies and gentlemen, Carlos has left the building".

Esta tesis ha sido financiada por el Ministerio de Ciencia y Tecnología del Gobierno de España, beca FPU AP2006-02986.

Summary

The increasing level of on-chip integration is leading to more complex embedded systems, which typically contain multiple storage elements, networks, I/O components, and a number of heterogeneous programmable processors for flexible application support as well as dedicated processing elements for achieving specific design goals. As a result, the system designers have to explore an exponentially growing design space in order to reach an optimal design solution considering multiple design objectives. In this context, it is widely believed that traditional design methodologies and tools are not appropriate enough to explore efficiently such a large design space, therefore coming short for designing modern SoC. In order to cope with the design complexity of such embedded systems, the working abstraction level is raised from RTL to a system level, where *design space exploration* (DSE) is becoming a key task of such *system-level design*.

At the same time that the abstraction level of system design is raised to system level, an important number of system-level modelling and simulation tools emerged. However, although these tools facilitate the designers to explore a wide range of design decisions during the early design stages, these tools only provide a partial solution for the DSE process, since an overall framework and new methodologies are still needed to explore the design space in a systematic and time-efficient way. In fact, without a disciplined and renewed design methodology, system designers will still have to resort to *ad-hoc* techniques to perform DSE experiments. This latter is a doubtable proposition, since it severely limits the designer's productivity and the amount of the design space that can be explored in a reasonable time.

This thesis provides some novel techniques and methodologies that help addressing the above challenges, being our ultimate goals to reduce the efforts of the system designers in the design process as well as to improve the efficiency of the DSE in the early design stages. To this end, we first propose a *dimension-oriented DSE methodology* for co-exploring multi-dimensional design spaces. The idea underlying this approach is that a large design space can be decomposed/separated in design space dimensions, such that the system designers can select different combinations of (tailored) search strategies for exploring simultaneously all dimensions of the design space, or to fix one or more of these dimensions and to focus the exploration within other dimensions. Second, we

implemented a tool to automatically generate a large variety of architecture models. Thus, it frees designers from the efforts to manually create such models, and as a result, a wide range of architectural alternatives (and consequently, a larger design space) can be easily explored. Third, we also have developed NASA infrastructure, which is a generic and modular framework to facilitate and support system-level DSE experiments. Moreover, NASA is a unified framework that allows for integrating our generator of architectural models, enables designers to incorporate different system-level simulation tools, as well as to couple different combinations of search methods (i.e., it supports our dimension-oriented DSE methodology) by means of a simple plug-in mechanism. As a result, NASA provides a flexible and re-usable environment to explore the multi-dimensional design spaces in a systematic and automatic way. And last, we have also proposed a *hierarchical DSE methodology* to address the problem of mapping a real-time application onto a target MPSoC platform. Our hierarchical approach enables designers to combine the benefits of both analytical estimation methods and system-level simulations to address the aforementioned design problem. As a result, and compared to traditional DSE methodologies using only analytical models or system-level simulators, our hierarchical DSE approach not only can explore and prune rapidly a large design space, but also can reach higher quality design solutions considering multiple design objectives in a time-efficient and accurate way.

Finally, in order to demonstrate and validate the capabilities and different key properties of our approaches, several sets of DSE experiments have been illustrated in this thesis as a *proof-of-concept*. Apart from providing valuable feedbacks and establishing the overall directions for our future work, the results obtained in these experiments reveal that the methodologies and techniques such as the ones presented in this thesis can effectively improve the designer's productivity as well as the efficiency of DSE at system-level by two orders of magnitude. Our further goal is to perform more experiments to prove and consolidate progressively the capabilities of our methodology for different application domains and an extensive range of MPSoC platforms.

Index

Index of figures

Index of Tables

Glossary

ADS	ARM Developer Suite
BTU	Basic Topology Unit
CBD	Component-based Design
ccd	Contention delay
cpd	Protocol delay
CHAT	Channel Administration Table
CHB	Channel Buffer
CMM	Total Communication time
CMP	Total Computation time
CVS	Computer Vision System
DSE	Design Space Exploration
EPE	Efficiency of Processing Elements usage
ESM	Executable System Model
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GA	Genetic Algorithm
HCE	Host Code Emulation
HDL	Hardware Description Language
HW	Hardware
ICCP	Inter-Component Communication Protocol
idVP	Identifier of the Virtual Processor
IP	Intellectual Property
IPT	Inter Processing Elements Traffic
ISS	Instruction Set Simulator
KPN	Kahn Process Networks
LuB	Load unbalancing
MoC	Models of Computation
MPSoC	Multiprocessor System on Chip
NASA	Non Ad-hoc Search Algorithm
NPE	Number of Processing Elements actually used

NRE	Non-Recurrent Engineering
NSE	Number of Storage Elements actually used
NE	Network Elements
OSCI	Open SystemC Initiative
PBD	Platform-based Design
PE	Processing Elements
RTC	Real Time Constraint
RTL	Register Transfer Level
rwd	SE access dealy
SA	Search Algorithm
SAD	Sum Absolute Difference
SCs	Scenarios
SE	Storage Elements
SLD	System Level Design
SLS	System-level Synthesis
SoC	Systems-on-Chip
SW	Software
TG	Task Graph
TLM	Transaction Level Modelling
TTL	Task Transaction Level
tVP	Virtual Processor type
VP	Virtual Processor
VPL	Inter-Clusters Link
WCET	Worst Case Execution Time
YML	Y-Chart Modelling Language

Chapter 1
Introduction

In order to cope with the design complexity of modern embedded systems, system level design has been proposed as a complement to the traditional design process. In this context, design space exploration is becoming a key task of such system-level design. In this chapter, we first introduce the goals of the system-level design space exploration, and subsequently, we survey the state of the art and main aspects that the system designers are addressing for achieving an efficient exploration process. Finally, we introduce the main contributions of this thesis in the domain of system-level MPSoC design space exploration. These contributions will be dealt with in detail in the following chapters.

1.1 Systems-on-Chip design challenges

The International Roadmap for Semiconductors (ITRS) predicts that a single integrated circuit (IC) will be able to contain a billion of transistors by the end of the decade [1]. This continuous technology progress toward a major capacity of integration on chip is leading to more complex embedded *Systems-on-Chip* (SoC) by adding more heterogeneous components and a big variety of functionalities into them. While offering high scale integration, high computing power and low power consumption, SoC designs also face several challenges:

- **Shorter Time-to-Market**. Nowadays electronic consumer market competitiveness and time-to-market imposes that no matter how complex the design problem is, the design time budget has to be kept as short as six months from the initial specification to a final (and correct) implementation. However, shortening the time-to-market and at the same time meeting the requirements of the new application domains require tremendous changes in current design methodologies. This is imposing an enormous pressure on development teams, which have to cope with more complex systems in less time. According to the Embedded Market Forecast, more than 50% of the new embedded system developments run late, whereas 20% do not achieve the initial specification requirements, or even are completely cancelled [2].

- **Increasing Non-Recurrent Engineering cost**. Another key factor in SoC design is the rapid increase of the *Non-Recurrent Engineering* (NRE) and production costs. These increasing costs are moving manufactures towards parts that have high-volume production guaranteed (in the order of millions of units from a single mask set). However, the greater cost of redesign (i.e., re-spin) is the increase of development time and the risk of missing the market window.

  Extensive verification is one way to reduce the number of refinements of a design. Typically, verification is accomplished with a whole suite of tools and teams of people at the end of the design cycle. Verification teams must have a complete knowledge of the system behaviour at the cycle/bit level, and define a great number of scenarios and stimuli that reproduces such behaviour. Usually, a written specification containing these scenarios and verification stimuli is created, which can take up to 50% of the design cycle. The cost and time spent in verification is expected to increase as SoCs integrate more components and functionality into hem. Moreover, some studies show

that up to 70% of the re-spins are due to functional bugs (i.e., differences between the specified functionality and the implemented functionality) [3].

- **Toward heterogeneous Multiprocessors System-on-Chip**. As the performance demanded by new applications continues growing, SoC are becoming heterogeneous *Multiprocessors Systems-on-Chip* (MPSoC). This heterogeneity consists in including multiple storage elements (SEs), network elements (NEs), I/O components, and a large number of processing elements (PEs) such as diverse programmable processors for flexible application support, as well as dedicated processing elements for achieving specific performance and power goals.

  The underlying idea is that these MPSoCs could provide a flexible and re-usable platform for different product versions (or family of products), and could be more easily modified in response to market needs, user requirements, and product updates during the design and product life cycle.

- **Large design space exploration**. The modern SoCs are characterized by a large design space, since there is a large amount of *design decisions* (or options) that should be taken into account by the system designers during SoCs design process. Such design choices include the partitioning of application in several tasks, assignment of the application functionality into the resources of the platform, and the choice of appropriate resource allocation schemes. It is well acknowledged that the design of such complex systems requires performance evaluation and validation techniques during the whole design trajectory. Because of the overall system complexity, fast evaluation methods in an early design stage are critical for making profound design decisions.

- **Efficiency in the resources usage**. In this new technology era, SoCs are very often real-time systems in which timeliness of the task execution is as important as its functionality. Usually, timeliness can be easily achieved by *over-dimensioning* the hardware resources or reducing the application performance. But the challenge is to deliver timing guarantees without such forms of *over-provisioning*. On the other hand, the intrinsic computational power of a SoC should not only be used efficiently, but the time and effort to design a system containing both hardware and software must also remain acceptable. System design methods implementing applications with the latest multiprocessor architecture must harness exponential hardware resource growth in a scalable and modular way.

In this point, we wonder whether the established methodologies and tools (and/or current ad-hoc approaches) for systems design are appropriate enough to meet these challenges. Without a disciplined design methodology, however, system designers will have to resort to ad-hoc techniques to implement concurrent applications on complex MPSoC platforms, which is doubtful proposition. We believe that new design methodologies and engineering practices are necessary for the design of complex MP-SoC to achieve the productivity and efficiency that has already been reached for single processor systems.

1.2 Limitations of conventional design methodologies

It is now widely believed that traditional design methodologies come short for designing the modern SoCs due to following reasons [4, 12]:

- A classical design methodology often starts from a single application specification, making it inflexible for broader exercises. HW/SW co-design could represent an example of such a classical design methodology. Typically, HW/SW co-design methods start from a single system specification that is gradually refined and synthesized into an architecture implementation. This architecture implementation is usually built with programmable processors and/or dedicated hardware components. However, the major disadvantage of this approach is that it forces the system designer to make early decisions on the HW/SW partitioning of the system – that is to identify, at a very early stage, parts of the system which will be implemented in hardware and software, which only makes sense if the right HW/SW partition is known beforehand.

- The co-simulation frameworks that model the classical HW/SW co-design approach, generally combine low (low-level) simulators, one for simulating the programmable components running the software and another for the dedicated hardware. The common practice is to employ instruction-level simulators for the software part, while the hardware part is usually simulated at RTL level using a hardware description language like VHDL or Verilog. Building these detailed simulation models for the hardware part requires significant effort, making them impractical in the early design stages. Moreover, these low level simulators usually have low simulation speeds what hinders fast exploration of the design space.

On the other hand, such tools often limit the size and complexity of the systems due to the enormous amount of details the designer has to handle, which obscure the system-wide view and slows the simulation of the design model considerably. Using this low productivity approach, designers occasionally settle for *whatever works* rather than designing *what works best* — and risk launching an uncompetitive product.

- Most of the design decisions are made in a very early design phase, and since they cannot be validated until very late, wrong decisions have an enormous impact on the success or failure of the final design. For example, architecture decisions as well as the best partitioning of the application functionality are taken during this phase using static techniques (e.g., spreadsheets) or just based on the expertise of the system designer. Unfortunately, these techniques are not suitable enough to deal with the complexity of new designs. Static techniques cannot capture the complex use cases and interactions of the applications running on the designs nor their dynamic behaviour.

- The HW/SW integration cannot be performed until the HW platform is available, which happens too late in the classical design flow. The main problem is that functional and non-functional mismatches with the specification cannot be detected until the simulation of the complete system is not executed. Moreover, as new SoCs become heterogeneous multiprocessors with an important amount of SW running on them, the HW/SW interactions are also more complex, and the chance for a first-time working design is very low.

1.3 Electronic System Level

In order to overcome the aforementioned shortcomings of the classical design methodology, *System Level Design* (SLD) methods have been proposed by the embedded system design community as a complement to the traditional methods, as well as to improve the productivity of the system designers. A complete overview of the SLD field can be seen in [5, 6]. In the literature SLD is also referred as ESL “Electronic System Level”, being design included in this expression.

1.3.1 Reasons for adopting the System Level Design

Roughly speaking, SLD has been focused mainly in three key aspects: (i) higher abstraction level: moving designers towards higher abstraction levels above RTL, (ii) separation of concerns: separating the various aspects of the design to allow a more effective exploration of alternative solutions, and (iii) refinement: allowing the progressive refinement of the system from abstract descriptions down to implementation.

Higher abstraction levels

Like the designer's abstraction level moved from standard-cells design (using full-custom techniques) to the RTL (using hardware description languages, that is, HDLs) in the 90's, nowadays new design methods propose to elevate the designer's abstraction level from RTL to system level. At the system level, the basic elements are of a bigger granularity such as *Intellectual Property* (IP) blocks or complete subsystems, which are described using high-level modelling languages such as SystemC [7, 8], SpecC [9], SystemVerilog [10] and ArchC [11]. This means that SLD develops SoCs and verifies its behaviour in terms of high-level block input and output events, and inter-block data transfers at the transaction level. The elimination of so many implementation details:

- simplifies the design effort, since new (or derivative) design can be obtained by maintaining the SoC platform and enhancing and/or adding specific features in an easier way than if using a lower abstraction level.
- speeds up the simulation time, allowing system-level simulations to execute more than 1,000 times faster than RTL [12].
- allows the system designer a direct and clear view of system behaviour and design attributes relevant to the system design.

Separation of concerns

Separating the various aspects of the design problem beyond hardware and software is a key issue in SLD in order to overcome the limitations of conventional design methods. The concepts of *orthogonalization of concerns* [13], *interface-based design* [14], and the *Y-Chart* approach [15] have established the basis to provide such separation in SoC design. All these approaches facilitate a more effective exploration of alternative designs, as will be explained later in Section 1.4.

Refinement

Once the system designer had found an optimal system design at system level, the model of this design can be used as an "executable specification" that drives the entire

subsequent RTL implementation, i.e., replacing each system-level component with another one at RTL, for example. This way, progressively refining a design from a higher abstraction level down to a lower abstraction level is a clear strategy to reduce the complexity of the design process [16]. Although such refinement process can be performed manually, real benefit comes when the process is automated via tools. Examples of this latter are the high-level synthesis tools that automatically transform a C description into an RTL implementation [17], or the code generation tools that create C or C++ code from UML descriptions.

1.3.2 System Level Design

In the last decades, several SLD approaches have been proposed taking into account the above mentioned aspects. These approaches can be broadly categorized into the following three groups:

- **Component-based design (CBD)**. It follows a *bottom-up* design approach, where complete architectures are built up assembling together pre-designed components. These components are interconnected and communicate each other by automatically inserting wrappers among them [18], or by using standard interfaces and bus protocols.

- **System-level synthesis (SLS)**. This approach moves the designers towards working at a higher abstraction level, where the starting point is a behavioural description of the system [19, 20]. SLS follows a *top-down* design flow, where the architecture is generated from that behavioural description by gradually adding implementation details until the final implementation is reached.

- **Platform-based design (PBD)**. PBD [13] is a *meet-in-the-middle* design approach, i.e., it combines a bottom-up approach for creating a predefined architectural platform, and a top-down approach to map the system behaviour onto the components of the architecture. This way, a common architectural platform can be specified and shared by multiple applications (inside a particular application domain). Moreover, PBD promotes the reuse of IP blocks for the purpose of increasing productivity and reducing manufacturing costs for high production volumes.

1.4 Design space exploration at system level

1.4.1 The goals of the design space exploration

Harnessing the benefits of working at system level, *Design Space Exploration* (DSE) is becoming a key task of SLD. In this context, a design space is composed of a set of *design points*, where each of them represents a system design with different design decisions, concerning HW/SW partitioning, platform topology, resources allocation in the architecture, number and type of architectural components, mapping of the application functionality on the architecture, and so on. This way, the ultimate goal of the system-level DSE is to explore such design space (i.e., search and select among somehow different design points from the design space) in a time and effort efficient way during the early design stages, and analyse each candidate in terms of its system-wide metrics (e.g., performance, cost/area and power consumption) in order to achieve the best design solution[1] (or a set of optimal solutions) that satisfy a set of design constraints or objectives imposed by the designer. Evidently, it is worth to mention that the more design decisions (or options), the larger the resulting design space (or the more possible design points) is, and the more effort and time is needed to carry out the DSE.

On the other hand, besides exploring a wide range of design options, DSE also facilitates the system designers to detect the impact of different design decisions on the global system behaviour, and make optimization decision prior to completion of prototype hardware. Therefore, such early DSE is of paramount importance, as early design choices heavily influence the success or failure of the final product, and can avoid wasting time and effort in further design steps without the possibility of meeting design requirements because of the inappropriate choices of design decisions.

1.4.2 Components of the design space exploration

The process of system-level DSE logically consists of three interdependent components: (i) the search method to systematically travel through the design space, (ii) the evaluation technique to assess the quality (in terms of system-wide metrics) of each design point selected by the search method, and (iii) a mechanism to generate the system description that is used by the evaluation technique to provide the needed system-wide metrics. The resulting relationship between these three components is shown in Fig. 1.1.

[1] In this document, the terms “design solution”, “candidate”, “design point” and “system design” are used interchangeably.

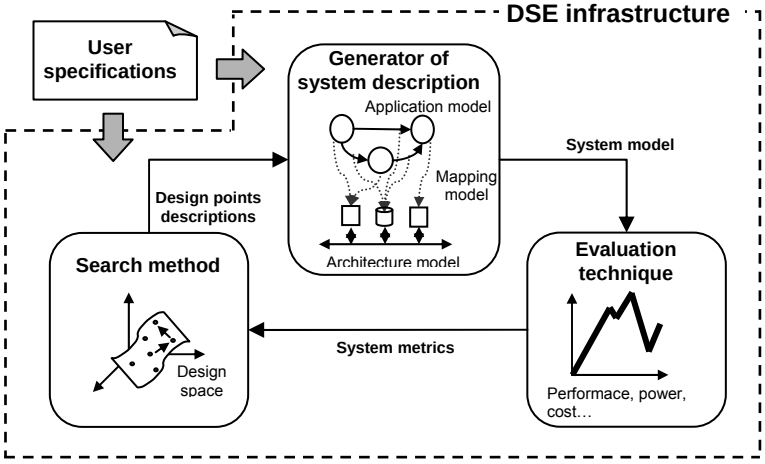


Fig. 1. 1 Relationship between the components of the system-level DSE.

1.4.2.1 The search methods

Two categories of search methods for DSE can be distinguished [21]: strategies for covering the design space and methods for pruning the design space. It should be noticed that both categories are not strictly orthogonal to each other. In fact, Chapter 4 presents a methodology for system-level DSE which combines techniques belonging to both categories in order to improve the efficiency of DSE (in terms of run time), as will be explained later.

Strategies for covering the design space

Three kinds of strategies can be mentioned here: exhaustive, random, and guided search. While the two first techniques are often implemented in a control script customized for every DSE experiment, which makes them inflexible and hardly reusable, the guided search relies typically on heuristics.

- **Exhaustively evaluating every possible design point**. This straightforward approach evaluates every possible combination of design parameters, so it is prohibitive for large design spaces. Although the design space can be reduced by limiting the range of design parameters, the search process is completely unguided and unbiased with respect to the preferences of the designer. Examples of design systems and case studies based on exhaustive search can be widely found in the literature [22-32].

- **Randomly sampling the design space**. Evaluating only random samples is the obvious choice to cope with large design spaces. It also has the advantage of

revealing an unbiased view of the characteristics of the design space. Monte-Carlo algorithm [33] and simulated annealing [34] are some techniques that can be clustered in this group.

- **Incorporating knowledge of the design space**. Search strategies in this category try to improve the convergence behaviour towards optimal solutions by incorporating knowledge of characteristics of the design space into the search process. The knowledge may be updated with every iteration of the search process or may be an inherent characteristic of the search algorithm itself. Hill climbing algorithms [35] and evolutionary algorithms [36-45] are the most representative examples for this group of search strategy.

Methods for pruning the design space

All mentioned exploration methods can be employed with the techniques presented in this subsection to reduce the complexity of the search by pruning the design space.

- **Hierarchical exploration**. Starting with a coarse problem statement, interesting regions of the design space are identified and ranked. Typically, this last process is focused on eliminating rapidly design points that cannot satisfy design specifications, rather than a detailed evaluation of each design alternative. Subsequently, a set of candidate solutions (selected by the previous process) is evaluated in more detail [23, 46-48].

- **Subsampling of the design space**. Subsampling the design space is a reasonable choice if the designer is interested in an unbiased exploration, where an exhaustive search would be prohibitive. The subsampling pattern could be completely random, based on some regular grid, or biased by some expected shape of the design space and/or objective function. The Monte-Carlo technique, the simulated annealing algorithm and evolutionary algorithms can be considered as examples for this kind of techniques.

1.4.2.2 The evaluation techniques

During the system-level DSE, two kinds of techniques can be used to evaluate a single design point or system design: system-level simulation-based evaluation and analytical estimation approaches.

System-level simulation-based evaluation

System-level simulations are particularly well suited to investigate dynamic, sporadic, and/or unforeseeable effects in the system, such as the contention delays due to simultaneous access requested by multiple competing PEs. While an analytical model can be too pessimistic since these models often consider the worst-case estimation time (WCET) only, simulations may reveal more realistic results for average-case optimization. A review of these tools can be found in Chapter 2 of this thesis.

Analytical estimation approaches

Analytical estimation methods come into play if deterministic behaviour or WCET is a reasonable assumption for the system under evaluation. While simulation-based evaluation might be too time-consuming (even working at system-level), analytical models ease early design decisions by identifying corner cases of potential designs, which often are difficult to detect due to the overall complexity of modern SoCs. Nevertheless, the main drawback of this analytical approach in our point of view, is still the lack of accuracy in the estimation/evaluation. A review of the analytical estimation approaches can be found in Chapter 4 of this thesis.

1.4.2.3 The generator of the system description

Independently on the nature of the evaluation techniques (i.e., simulation or analytical model), a complete system model should be created first in order to evaluate appropriately each design point of the design space. This latter becomes a requirement for the case of simulation-based evaluation. In these cases, in order to perform the simulation and to obtain system-wide metrics, these tools typically require an *executable system model* (ESM), which can be considered as a *virtual-model* or *meta-model* of the system implementation that include not only information about the application functionality and details about the architecture, but also specify how both domains are related together.

The separation of concerns and the Y-Chart principle facilitate enormously the conception and creation of such ESM. According to these approaches, a system (and in our case, an ESM) can be specified with the combination of three models: an application model, an architecture model and a mapping model. The latter means that Y-Chart principle decouples application from architecture by recognizing distinct models for them. An application model – derived from a target application domain – describes the functional behaviour of the application (using, e.g., Kahn Process Networks (KPN) or task-graphs) in an architecture-independent manner. Simultaneously, an architecture model –defined

with the application in mind– specifies the architecture implementation (by means of highly configurable predefined components provided typically by the tool library) and captures their performance constraints. Finally, an explicit step (or model) maps the application model onto an architecture model for co-simulation, after which distinct system metrics can be quantitatively evaluated.

An ESM created in this manner presents several benefits when compared with other traditional solutions (e.g., emulators, FPGA or RTL simulations):

- **An ESM can be earlier available**. This is an implicit consequence of working at a higher abstraction level than RTL, which implies fewer implementation details, and therefore can be created with less effort.

- **An ESM is a more flexible solution**. Since these meta-models of system implementation contain fewer details and are built in a compositional way, changes can be done quicker and new models can be derived in a shorter time than with other traditional solutions.

Finally, we would like to notice that such ESMs can be created manually [49] or automatically [50]. However, this manual creation process (and set up of this model) can be a very error-prone, time-consuming and labour-intensive task for the system designer, while the automation of this process could improve significantly the design productivity and the DSE efficiency. An approach for generating automatically an ESM will be discussed in Chapter 3.

1.5 Contributions of the thesis

This thesis is focused on the methodologies and techniques for performing efficient MPSoC design space exploration at system level. More specifically, we have developed new methods, algorithms and infrastructure/support to deal with the design's complexity of modern SoCs that the system designers have to face in DSE during the early design stages. The main contributions of this thesis are:

- We propose a new and generic system-level MPSoC design space exploration infrastructure, called NASA (*Non Ad-hoc Search Algorithm*). This highly modular framework uses a set of well-defined interfaces to easily integrate different search

strategies as well as existing system-level simulation tools in a single environment in a simple *plug-and-play fashion*. As a result, the potentials for reuse of the framework are significantly increased since each DSE experiment can be performed without the need of preparing experiment-customized scripts, but it only requires a simple change of the user's input constraints values.

- We have implemented a new approach to gradually and automatically generate an ESM that is simulated for obtaining system metrics to evaluate design decisions. Thus, the entire DSE process (composed of search, ESM generation and design point evaluation) is performed in an automatic and systematic fashion. This improves design productivity and decreases the designer's efforts.

- We have developed a novel methodology for system-level MPSoC DSE called *dimension-oriented DSE* approach. According to our approach, a large design space is explicitly separated into dimensions, which could represent design decisions that are orthogonal to each other such as mapping, architectural components, and platform. Thus, the designer can choose to simultaneously explore all dimensions, or to fix one or more of these dimensions and to focus the exploration within other dimensions. This means that the designers should have to configure the appropriate number of (possibly different) search algorithms to simultaneously co-explore the various design space dimensions.

- We have proposed a new heuristic-based method, which is capable of taking into account multiple design objectives that handles the problem of the real-time application mapping onto a flexible MPSoC platform. Moreover, this analytical approach can evaluate a design point considering both static and dynamic system behaviours. This analytical approach forms part of a hierarchical DSE approach, where our analytical approach relies on a rough estimation to eliminate rapidly a large number of design points in the pruning phase, while only a small number of candidate mappings are evaluated accurately by a system-level simulator in the exploration phase. Our experimental results reveal that this hierarchical DSE approach can improve significantly the DSE efficiency as well as achieve good quality solutions.

1.6 Thesis structure

Chapter 2 introduces CASSE, a SystemC-based system-level modelling and simulation environment developed at Research Institute for Applied Microelectronics (IUMA) and used in this work. Since CASSE plays a key role in our proposed approach as well as in our DSE experiments, we first introduce some key concepts employed within the CASSE framework such as the modelling methods that allow separating and mapping application and architecture models, and the simulation technique. Then, we give a conceptual view of the CASSE framework, where we explain the aspects and the requirements of the CASSE interfaces that a system designer should handle in order to create and configure an appropriate ESM. Finally, in order to prove the CASSE exploration capabilities, we conclude this chapter by presenting a case study, the DSE for the modelling of a real-time visual objects tracking algorithm in a heterogeneous MPSoC.

Chapter 3 is dedicated to the NASA framework and the dimension-oriented DSE methodology. We first summarize the main weak points of both our current DSE methodology and other system-level DSE frameworks that can be found in the literature. Then, we present an overview of NASA and its key properties to overcome the weaknesses detected in other approaches. Subsequently, we provide detailed implementation information about the different components (or modules) inside NASA, and specially we focus on explaining the concept of *dimension-oriented approach* as well as how it is implemented in our work. Finally, in order to demonstrate the capabilities of the NASA framework and to illustrate the benefits of our dimension-oriented DSE methodology, we also present a wide range of DSE experiments results and comparative analyses between our approach and other traditional DSE strategies.

Chapter 4 focuses on the hierarchical DSE. We first discuss the strength and weakness of the DSE using an evaluation technique based on either simulation or analytical model. Then, we present an overview of our hierarchical DSE approach as well as the problem statement and concept definitions used throughout this chapter. Subsequently, we explain how the distinct intermediate steps in both design space pruning and design space exploration phases are implemented in our work. Moreover, we also provide a formal definition of our analytical estimation model and illustrative examples for a better understanding of the heuristic-based algorithms used in the pruning phase. Finally, we conclude this chapter with a set of experiments, where we compare the efficiency and quality of the results reached by our hierarchical DSE approach with the results obtained by other traditional DSE strategies.

In Chapter 5, we present the main conclusions of this work and some possible works that can be accomplished in the future.

Chapter 6 is a wide summary in Spanish of this document.

Chapter 2
CASSE: a SystemC-based modelling and simulation tool

A SystemC-based modelling and simulation tool, called CASSE, is introduced in this chapter. CASSE is a key piece in the methodologies/frameworks presented in this thesis, since it is used in our system-level DSE experiments to evaluate each design alternative, and providing thus the needed metrics to guide the search processes. Given its relevance in our approaches, this chapter aims to highlight the most important properties and implementation details of CASSE tool, as well as to explain the distinct description files required by CASSE to create/configure a simulatable system model. Finally, a set of DSE experiments carried out with CASSE is also presented in order to illustrate the capacities/strengths and weaknesses of CASSE tools.

2.1 System-level simulation tools and Open SystemC Initiative standards

As mentioned above, in Section 1.4.2.2, a system instance or design point can be evaluated by either simulation tools or analytical estimation models, and the main difference between both techniques is in the grade of accuracy achieved in their evaluations. It is well-known that simulation tools allow system designers to fully control the DSE process in each experiment [12, 51]. For example, these tools can guarantee that two scheduling strategies were run exactly in the same system conditions. Moreover, simulators are becoming more and more realistic, allowing to model real network topologies and their associated resource characteristics, such as operating frequency, platform with several clock domains, network bandwidth, the memory access latency, and so on. In addition, it can also take into account the effects of contentions generated by multiple PEs competing to access simultaneously to the same shared system resource. To summarize, all these reasons justify greatly why the simulation tools have been widely used for the DSE experiments in the literature.

On the other hand, at the same time that the abstraction level of system design is raised to the system level, an important number of system-level modelling and simulation tools have been developed in both academic and industrial communities. Although these tools can be implemented with different high-level modelling languages (e.g., SystemC [7, 8], SpecC [9], or SystemVerilog [10]), it has been seen that most of these tools are based and/or supported by the continuously deployment of SystemC [52] and TLM [53, 54]. In this context, the Open SystemC Initiative (OSCI), controlled by a steering group composed of thirteen major companies in the electronic design automation and electronic industry, promotes the development and standardization of both SystemC and TLM.

2.1.1 IEEE 1666 SystemC standard

SystemC is a modelling language that is intended to enable system-level design and IP exchange (at multiple abstraction levels) for systems containing both hardware and software components. SystemC is based on the C++ programming language and extends its capabilities to facilitate hardware description. These extensions are achieved via a class library that provides powerful new mechanisms to model hardware elements, concurrency, time and reactive behaviour. SystemC also provides a simulation kernel that allows simulating an executable specification of the design, i.e., an ESM.

2.1.2 OSCI TLM 2.0 standard

In TLM, the details of communication among computation components are separated from the details of their implementation. Communication is modelled as channels and transaction requests take place by calling interface functions of these channel models. The primary goal of TLM is to dramatically increase simulation speeds over traditional RTL simulations, while offering enough accuracy for exploring and validating implementation alternatives at higher levels of abstraction. The increase in speed is achieved by abstracting the number of events and amount of information (that have to be processed during simulation) to the minimum required. Besides this increase in speed, TLM reduces the amount of detail that the designer must handle, making modelling easier. TLM 2.0 proposes a standardized set of Interface Method Calls (IMC) to create transaction level models, a standard generic payload and protocol (GP) for modelling memory-mapped bus communication and a mechanism to extend the standard payload with protocol specific information [55].

2.2 Related work

Before presenting CASSE in the next section, we first survey in this section a number of existing system-level modelling and simulation frameworks that can be found in the literature. We should note that the following frameworks only represent some selective samples rather than an exhaustive review.

As mentioned in Section 1.4.2.3, the Y-Chart scheme is a typical example of a methodology that eases the DSE process by modelling independently functionality and architecture, and later on combining them in a separated mapping phase. CASSE follows a Y-Chart principle (as will be explained in the next section), and other similar frameworks based on Y-Chart scheme are Spade [56], Sesame [57, 58] and Artemis [4, 60]. Spade is an environment for system-level design that is based on trace-driven simulations. That is, Spade starts with a functional simulation of the application that is described in the form of a Kahn Process Network [66]. Then, during the simulation of the application, it generates a set of execution and communication traces, which works as stimuli to the architecture model. Actually, this means that the application is not executed in the target architecture, i.e., Spade does not simulate the application running on the target architecture, but it uses trace driven simulation or co-simulation between the application and the architecture.

Spade was the base that inspired both Sesame and Artemis frameworks. In addition to Y-Chart based modelling and trace-driven system-level simulation inherited from Spade, both Sesame and Artemis frameworks have additional capabilities such as pruning the design space by multi-objective search, gradual model refinement, and mixed-level modelling and simulation by coupling low-level simulators [59]. Moreover, an additional layer of virtual processors is introduced between applications and architectures in order to resolve possible deadlocks due to mapping decisions. Like CASSE, these tools use a set of text-based files to create/configure the system models. Their simulation outputs are limited to the system performance metric.

Several efforts have also been developed in the last years to extend the capabilities of Sesame framework. These efforts include an interactive visualization tool to understand/analyse the results obtained in the DSE experiments [70, 71], and a high-level microprocessor power-modelling technique based on event signatures [72, 73]. The Daedalus framework [68] also extends the work from Sesame by adding automatic application parallelization from sequential C/C++ description and automated synthesis and RTL integration towards FPGA technology.

Other frameworks enabling designers to model and simulate system models at multiple levels of abstraction are Ptolemy, MILAN, and Metropolis. Ptolemy [62] is an environment for simulation and prototyping of heterogeneous systems. It supports multiple models of computation (MoC) within a single system simulation. This is achieved by supporting domains to build subsystems each conforming to a different MoC. Using techniques such as hierarchical composition and refinement, the designer can specify heterogeneous systems consisting of various MoCs to model and simulate applications and architectures at multiple levels of abstraction. Ptolemy supports an increasing set of MoCs, including all data-flow MoCs [63]: Synchronous data-flow [64], Dynamic data-flow [65], Kahn Process Networks [66] as well as finite state machines, discrete-event and continuous-time domains.

MILAN [46] is a hierarchical design space exploration framework that facilitates integration of different design tools by using a set of tool-dependent interpreters or adapters. At the highest level, it makes use of a performance estimator which prunes the design space by constraints satisfaction. Subsequently, an analytical model is applied to estimate system-wide performance in the second step. Finally, low-level simulation is used to identify the optimal design points. This way, MILAN trades off accuracy of the results for simulation speed by choosing from a range of simulators at multiple

abstraction levels. These simulators range from high-level system simulators (e.g., MatLab and SystemC to verify the application behaviour) to cycle-accurate instruction set simulators (ISS) such as SimpleScalar [69].

On the other hand, Metropolis [67] targets at integrating modelling, simulation, synthesis, and verification tools within a single framework. It makes use of the concept of *metamodel*, which offers syntactic and semantics mechanisms to support functionality capture and analysis, as well as architecture description and mapping of functionality to architectural elements. Besides the function/architecture separation, Metropolis also proposes the separation between the capabilities of an architectural model versus the cost it bears when it implements a given behaviour. That is, architectural components are driven by events which are annotated with the costs of interest such as the energy or time for execution. Although quite valuable and innovative from the methodological point of view, this framework is not based on SystemC. This is an important drawback for their interoperability and integration with typical system-level design flows adopted in the industry.

An existing SystemC-based framework that applies similar design methodology than CASSE is presented in [61]. Kogel et al. [61] presents a Virtual Architecture Mapping methodology that enables the quantitative evaluation of an application-to-architecture mapping by means of an executable performance model. Shared processing resources are modelled via the Virtual Processing Unit that allows the spatial mapping and execution of multiple tasks on a single element. Similar to CASSE, this framework accelerates the exploration of a large design space by means of description files where individual timing annotations as well as the mapping are specified. Unlike CASSE, on the other hand, they use communication channels to capture timing aspect. That is, the channels provide methods to annotate processing delay and initiation interval, while in CASSE, the timing information is annotated into the tasks and the communication channels are used to exchange shared data/tokens and to synchronize status of the channel, as will be explained in Section 2.3.1.

2.3 CASSE design flow and internal structure

CASSE [12, 79, 91] is a SystemC-based modelling and simulation environment, which aims to help the system designers to explore and analyse application models running on an early available architecture model under a unified environment. CASSE has been

developed at Research Institute for Applied Microelectronics (IUMA) of ULPGC and the main key properties taken into account during its development are:

- **Application modelling and mapping**. During the specification phase, it is very unlikely that the application is available in its final stage (e.g., embedded software and/or hardware implementation). Therefore, other techniques should be applied that allow modelling and setting the application requirements in a more abstract way. Moreover, to allow an extensive DSE, separation of concerns has also to be applied. This means that the behaviour, interfaces and cost (e.g., timing information) of the applications are modelled in an orthogonal way and merged once the application is mapped on the targeted architecture.

- **A library of highly configurable generic architecture components**. During the specification phase, distinct architectures with different properties need to be explored. In order to have an ESM early during the specification phase, IP models have to be quickly available and therefore the effort to create and assemble them has to be minimal. As a solution to this limitation, a library of highly configurable generic component models, emulating the common functionality and timing of several IP types, has to be put in place. Such generic IP models can be used when the model for the specific IP is not available yet. Moreover, these generic component models have to support the mapping of application models onto them.

- **Advanced performance analysis**. During system-level simulations, tons of different performance data has to be interpreted and analysed before deciding whether a system instance fulfil the imposed requirements or not. Pruning this big amount of information in order to find performance bottlenecks and possible optimizations is not a trivial task in the design of modern SoCs. Therefore, new performance analysis techniques allow identifying hot spots and bottlenecks in a complex design and easily correlate them with the application model. This enables an easier and faster way to isolate and hence optimize the designs.

- **Short set-up time, fast simulation and relative accuracy**. For evaluating a single design point, an ESM needs to be created, configured, simulated, and the obtained results analysed in a short time. The shorter this process, the bigger number of different design alternatives that can be explored. More specifically, the requirements are:

- *Small effort to configure/set up the ESM.* User-friendly front-end tools and possibly some scripting capabilities are required to enable quick modifications in the ESM.

- *Fast simulations*. At least an improvement factor of x1000 compared to RTL simulations is required to perform meaningful simulations during the DSE. This leads us to simulation speeds in the range of several MHz.

- *Relative accuracy*. Although the highest accuracy (compared with RTL simulations) is always desired, very often certain level of accuracy can be traded in order to reduce the modelling effort and improve the simulation speed. For exploration purposes, the concept of relative accuracy or fidelity is introduced, so that the ESM can be used to evaluate, for instance, whether a system instance in particular is better or worse than another one for certain aspects.

2.3.1 CASSE methodology and design flow

Considering the aforementioned properties, CASSE is implemented following the Y-Chart principle, where the application functionality and the architecture are independently modelled and combined in a separate mapping phase (see Fig. 2.1). Quantitative information about the system execution is then obtained by means of simulations. The tool can be used to perform functional simulations of the application model only or performance simulations of the application mapped and executing on the architectural model. After simulations, the obtained information can be visualized and analysed in order to guide further optimizations in architecture, application and/or mapping structure.

CASSE controls all stages in the design flow by means of textual description files. These description files are read and parsed by the tool during elaboration time in order to create and properly configure the desired system model. The result is a model (i.e., ESM) that is executed using the SystemC kernel. Hence, the tool simplifies the exploration of several design points by means of these description files that can be easily modified in order to create a new system instance. Since changing the description files does not require recompiling the existing models, extensive parameters sweeps can be performed easily using scripts.

Application modelling. CASSE applies a parallel programming model based on the *Task Transaction Level* (TTL) interface [74]. According to the TTL specification, an application is described as a process network where parallel tasks communicate with

each other by means of unidirectional channels. A task is an entity that performs computations. Tasks are connected to channels via ports, and they communicate and synchronize with each other by calling TTL interface primitives on their ports. Hence, TTL provides a fair separation between computation and communication at the application level. TTL can be used both for developing parallel application models and as a platform interface for integrating hardware and software modules on a platform infrastructure. A good point of TTL is that it only defines the interface primitives and their functionality, but leaves their implementation open to the designer.

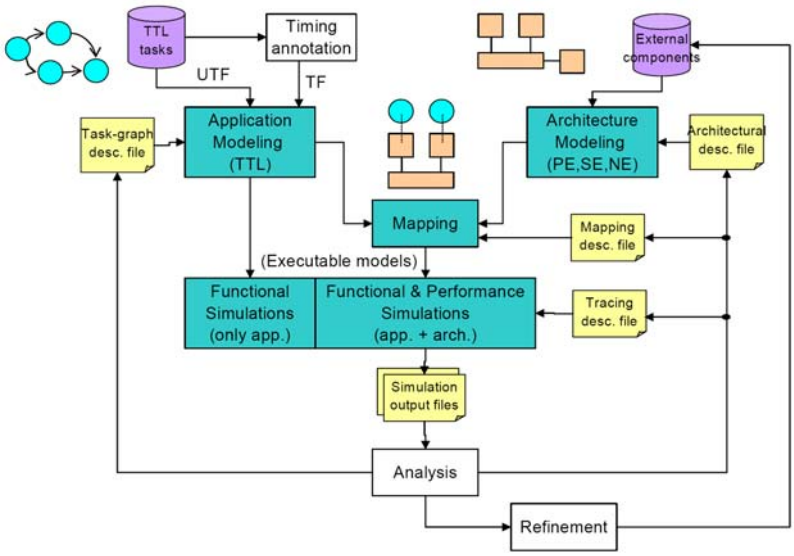


Fig. 2. 1 CASSE design flow.

In the tool, tasks containing the application functionality are written in C/C++ (i.e. the functionality per task is fixed at compile time). However, the network structure (i.e. port to channel connections) and its configuration (e.g. data/token size) are described in a separate text file. The tool uses this description file to instantiate and bind together tasks and channels creating an executable model of the network. This architecture-independent executable model can be simulated using CASSE in order to validate the functional correctness of the application. An example of a task file and another of a task-graph description file are shown in Fig. 2.2 and Fig. 2.3, respectively.

During this stage, we however would like to point out that the main drawback of CASSE is the parallelization process of the application source code in a task graph. In fact, as illustrated in Fig. 2.4, current CASSE version provides no mechanism to carry out automatically such process, on the contrary, the system designers have to perform

manually this task. Basically, this latter means that the designers should have to rely on their intuitions and/or experiences to adapt the source code, i.e., split the application in a set of tasks and incorporate appropriately the corresponding TTL interface primitive calls inside each task.

```
// task file
#include “task.h"
…
// constructor
task_name::constructor(const char* nm)
:task_if(nm)
{
}


int task_name::mapPorts()
{
          CASSE_BIND_PORT_ON;
          CASSE_BIND_PORT(OUTPORT,sizeof(token_size));
          CASSE_BIND_PORT(INPORT,sizeof(token_size));
          CASSE_BIND_PORT_OFF;
}


void task_name::main()
{
…
// Source code in C++
// including TTL interface primitive calls (i.e., read/write channels)
…
}
```

Fig. 2. 2 Example of task file.

```
// Task-graph description file


# Tasks creation #
.CREATE -TASK task_1 -N_PORT number_of_ports ;
.CREATE -TASK task_2 -N_PORT number_of_ports ;
…
.CREATE -TASK task_id -N_PORT number_of_ports ;


# Channels creation #
.CREATE –CHANNEL channel_1 -SIZE number_of_tokens -TSIZE token_size ;
.CREATE -CHANNEL channel_2 -SIZE number_of_tokens -TSIZE token_size ;
…
.CREATE -CHANNEL channel_id -SIZE number_of_tokens -TSIZE token_size ;


# binding part #
.BIND -TASK task_id -PORT port_id TO -CHANNEL channel_id -PRODUCER ;
…
.BIND -TASK task_id -PORT port_id TO -CHANNEL channel_id -CONSUMER ;
…
```

Fig. 2. 3 Example of task-graph description file.

The correct decomposition of the application is very important due to several reasons. On one hand, this way we can determine the order of the execution of tasks, their degree of dependence and the exact data that must be communicated between different tasks. This latter determines the exact communication data granularity and influences further

performance communication analysis. On the other hand, with this decomposition the number of communication points and channels on the system can be determined, and the sending points for producers and the receiving points for consumers in each task entity can be more clearly identified.

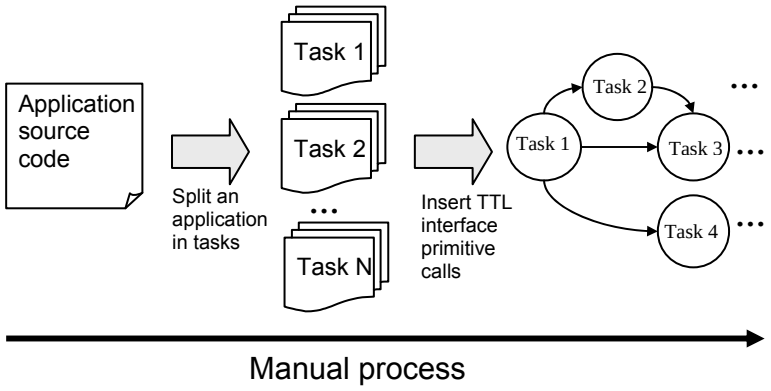


Fig. 2. 4 Parallelization process of the application in the task graph.

From our point of view, such manual code transformation process is not efficient neither viable for large and complex applications, besides it can be an error-prone and time-consuming task. Although some techniques/algorithms can be developed and integrated in CASSE to automate this process (e.g., generator of KPN used in [75, 76]), this latter is not addressed in this thesis, but it is proposed as future work to extend CASSE capabilities.

Architectural modelling. CASSE provides easy and fast architectural modelling by describing a system as a modular composition of highly configurable predefined elements (provided by the tool libraries). The library of predefined elements is composed of *processing elements* (PEs), *storage elements* (SEs), and *network elements* (NEs). PEs model generic multitasking computational units, which include an abstract task scheduler model that supports different arbitration schemes and advanced features like interrupts and pre-emption. SEs model generic multi-port memory elements. Finally, NEs model generic shared bus interconnections including programmable arbiters, address decoders, and optional input buffers. All predefined elements are connected together in a *plug-and-play* fashion by means of the *Inter-Component Communication Protocol* (ICCP) interface. ICCP is an abstract communication protocol, which defines a point-to-point interface and a group of communication primitives between two entities named *Initiator* and *Target*.

Both the ICCP interface and the library of predefined elements have been developed using SystemC and the TLM Standard library.

A separate description file is used in order to specify the architectural composition of the system (i.e., number of elements of each type, number of interfaces per element, and their interconnections), and its configuration (e.g. memory map, memory sizes, communication latencies per interface, task scheduler policy, etc.). An example of the architectural description file is shown in Fig. 2.5.

```
// Architectural description file


.CREATE -CLOCK clock_domain_1 -PERIOD 5 -UNIT SC_NS ;
.CREATE -CLOCK clock_domain_2 -PERIOD 15 -UNIT SC_PS ;
…
.CREATE -PROCESSING PE_name -N_INIT number_of_initiators ;
 // latency parameters
.CONFIGURE -PROCESSING PE_name -INIT init_id -WIDTH 32 -LAT 1 1 1 0 -CONNID PE_id ;
 // Tasks Scheduler parameters
.CONFIGURE -PROCESSING PE_name -TS MLQ 100 50 0 ;
.CONFIGURE -PROCESSING PE_name -PTW YIELD 20 20 10 20 ;
…
.CREATE -STORAGE SE_name -N_TARGET number_of_targets ;
.CONFIGURE -STORAGE SE_name –SIZE SE_size ;
.CONFIGURE -STORAGE SE_name -TARGET 0 -WIDTH 32 -LAT 0 1 1 ;
…
.CREATE -NETWORK bridge_id -N_INPUT number_of_inputs -N_OUTPUT number_of_outputs ;
.CONFIGURE -NETWORK bridge_id -WIDTH 32 -BUFFERED y -I_LAT 1 1 1 -O_LAT 1 1 1 0 ;
.CONFIGURE -NETWORK bridge_id -OUTPUT_B 0 -RANGE 0x00000000 0xffffffff ;
…
.CREATE -NETWORK NE_id -N_INPUT number_of_inputs -N_OUTPUT number_of_outputs ;
.CONFIGURE -NETWORK NE_id -WIDTH 32 -BUFFERED n -I_LAT 0 0 0 -O_LAT 0 0 0 0 ;
//Arbiter policy
.CONFIGURE -NETWORK NE_id -ARBITER ROUNDROBIN ;
.CONFIGURE -NETWORK NE_id -OUTPUT 0 -RANGE 0x00000000 0x0003ffff ;
…
.CREATE -LINK link_id -WIDTH 32 ;
…
.BIND -CLOCK clock_domain_1 TO -PROCESSING PE_name ;
.BIND -CLOCK clock_domain_1 TO -STORAGE SE_name -TARGET target_id ;
…
.BIND -CLOCK clock_domain_2 TO -NETWORK NE_id -INPUT ;
.BIND -CLOCK clock_domain_2 TO -NETWORK bridge_id -OUTPUT ;
…
.BIND -LINK link_id TO -PROCESSING PE_name -INIT PE_id ;
…
// Configuration of memory map
.CONFIGURE -MEMAREA map_id -SIZE SE_size INTO -STORAGE SE_name -BASE map_range ADDRESS 0 ;
```

Fig. 2. 5 Example of architectural description file.

Mapping and execution. One of the main advantages of the tool as a unified environment is the straightforward mapping support of the application functionality onto the modelled architecture. That is, CASSE supports the direct mapping of the TTL applications (i.e., tasks and channels) onto the architectural models, and the execution of the resulting system with no need for source code changes (i.e., the original source code of the tasks is executed directly in the architectural model). This technique is called *Host Code Emulation* (HCE). HCE avoids the usage of accurate hardware models and ISS models and, therefore, reduces the modelling effort and allows faster simulations.

Nonetheless, timing delays reflecting the computational costs of the functionality has to be annotated into the tasks. This can be performed by either automatic methods like those described in [77, 78] or the timing information has to be extracted and annotated by hand.

```
// Mapping description file

// Map tasks
.MAP –TASK task_1 INTO –PROCESSING PE_1 ;
.MAP -TASK task_id INTO -PROCESSING PE_name ;
.MAP -TASK task_id -PORT port_id INTO -PROCESSING PE_name -INTF init_id -LOCAL ;
…
// Map channels
.MAP –CHANNEL channel_1 INTO –MMAREA map_1 ;
.MAP -CHANNEL channel_id INTO -MEMAREA map_id ;
.MAP -CHAT channel_id -PORT port_id INTO -MEMAREA map_id ;
…
```

Fig. 2. 6 Example of mapping description file.

Like for the previous steps, another description file is used for the tool in order to control the mapping procedure. An example of such mapping description file can be seen in Fig. 2.6. The outcome of the mapping stage is an executable model containing the selected application/architecture instance. This executable model can be executed using the SystemC kernel in order to validate both the functional correctness and the performance of the system modelled.

At this point we would like to stress that, the current version of CASSE provides unfortunately no automated mechanism to facilitate the mapping of the application models (i.e., tasks and channels) onto the architecture models, but such a mapping process is carried out manually by designers. That means that the designers have to make explicitly the correct mapping decisions in order to achieve a feasible mapping. To this end, CASSE can ensure a feasible and dead-lock free mapping following the next two steps. First, TTL tasks should be mapped onto available PEs of the architectural model. Second, if two communicating TTL tasks are mapped onto different PEs, the TTL channel associated to those TTL tasks should be mapped onto available SEs of the architectural model such that the aforementioned PEs can access to these SEs. This latter is an important feature to warrant a feasible mapping, since all the exchanged data (or token) and synchronization information are contained in these channels.

In fact, TTL channels are composed of two parts: the channel buffer (CHB) and the channel administration table (CHAT). The CHB, where the data/token is stored, is unique for the producer and for all the consumer tasks connected to it. On the other hand, each channel keeps associated a producer CHAT for the port producing tokens into the

channel and a consumer CHAT for every port consuming tokens from the channel. Moreover, once all channel parts are mapped, CASSE automatically fill out all CHATs with information about how to access the token in the CHB as well as the location of the CHATs. This way, a feasible mapping is obtained whenever the producer CHAT and the consumer CHAT are “visible” each to other as well as the CHB is “visible” for both CHATs. Evidently, such a manual mapping process is error-prone and time-consuming, and particularly when these mappings decisions are made during the design of complex SoC. In order to overcome this drawback, Chapter 3 will present an approach that is capable to generate automatically feasible mappings, therefore improving significantly the productivity of the system designers as well as the efficiency of DSE process.

Performance analysis. During performance simulations the tool can obtain and record information about the system execution. This information allows the user to analyse and identify architectural bottlenecks and possible system optimizations at different levels. Based on this analysis, further iterations might be carried out allowing the designer to decide whether to further investigate on modifying both the application and the architecture models, or setting a new mapping. CASSE predefined elements are able to monitor and record two different kinds of information: performance metrics (statistics) and transactions [79]. Such tracing support is possible since all predefined elements incorporate built-in monitors that can be individually enabled to automatically gather statistics and record transactions during their execution.

However, performance monitoring can have a significant impact on the simulation speed of the architectural models. Thereby, instead of monitoring and recording all possible information regarding the system execution, the tool provides a fine grain controllability of what information to trace and where to trace it. This fine-grain monitoring is controlled via a separate CASSE description file. This *trace* description file has to be fed to the tool together with the *task-graph*, *architectural* and *mapping* description files.

Finally, it is worth to notice that despite of providing plenty of information about performance, traffic load in the system, and metrics about the resource usage efficiency, etc., other important metrics in SoC design such as power consumption and cost/area are not included in the current analysis of CASSE tool yet. Since the development of CASSE tool is behind of the scope of this thesis, this aspect is encouraged to be addressed as future work.

Refinement. Finally, once the expected requirements are fulfilled with a specific application/architecture/mapping instance, the system is ready for implementation. Hardware modules can be progressively refined from more abstract to more accurate (even synthesizable) descriptions in SystemC, and verified within the architectural model just by replacing predefined elements of the tool libraries with accurate models of external custom components. Likewise, software modules might be directly taken into an embedded compiler, and later integrated again in the system by means of an external component that integrates an ISS.

2.3.2 CASSE tool structure

As depicted in Fig 2.7, CASSE is structured in three layers:

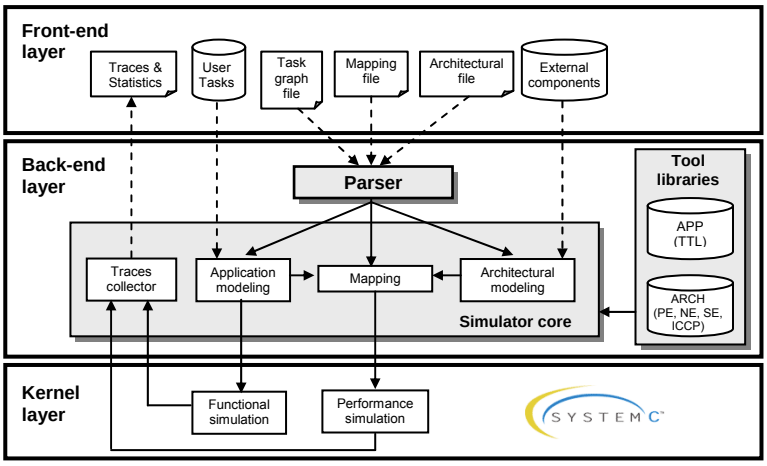


Fig. 2. 7 CASSE internal structure.

Front-end layer. The front-end layer serves as a user interface that controls the tool. As mentioned before, there are four groups of description files that allow the system designers to fully control the creation and configuration of each ESM: the task-graph (or application) files, the architectural file, the mapping file, and the trace file.

Back-end layer. This layer implements the core functionality of the tool. Besides a parser that reads and interprets the description files (specified by the designers), this layer contains also two specific libraries: the application library (APP) and the architecture library (ARCH). On the other hand, CASSE is able to carry out two kinds of simulations: functional simulations and performance simulations. While functional simulations only require the task-graph file, CASSE needs to read and parse the task-graph, the

architectural and the mapping files during performance simulations. The outcome of this process is an ESM, i.e., an executable model of the system instance.

Kernel layer. These ESMs are then run using the SystemC kernel, which constitutes the third layer of the tool. During SystemC simulations execution, traces and statistics can be recorded and dumped to output files for later inspection and analysis.

2.4 Real-time visual tracking system case study

In order to provide a more detailed view of the tool capabilities, we present the modelling and performance analysis of a real-time visual tracking system mapped on a target MPSoC architecture as a case study in this section. Our goal is to facilitate a better understanding about the output information generated by CASSE during simulations, since such information will be used to guide the design decisions and the search algorithms during DSE (as will be explained in Chapter 3). On the other hand, as the tracking system will be also used as the reference application in all DSE experiments in other chapters of this thesis, we will introduce a brief overview of this real-time application before presenting our experimental results.

2.4.1 Computer vision system and the real-time tracking algorithm

Real-time computer vision systems (CVS) have gained an increasing importance in many fields, including applications such as robotics, multimedia, industrial inspection, medical imaging, and autonomous navigation. Many of these CVS have been developed with notable contributions in these application domains [80-83].

CVS is usually composed of different types of applications, which may include algorithms for detection, recognition and tracking. Although our final objective is to obtain a CVS composed of these applications, at this moment a visual tracking algorithm based on correlation or block matching is being used as the main reference application. More specifically, we have used a visual objects tracking algorithm developed by researchers at ULPGC [89].

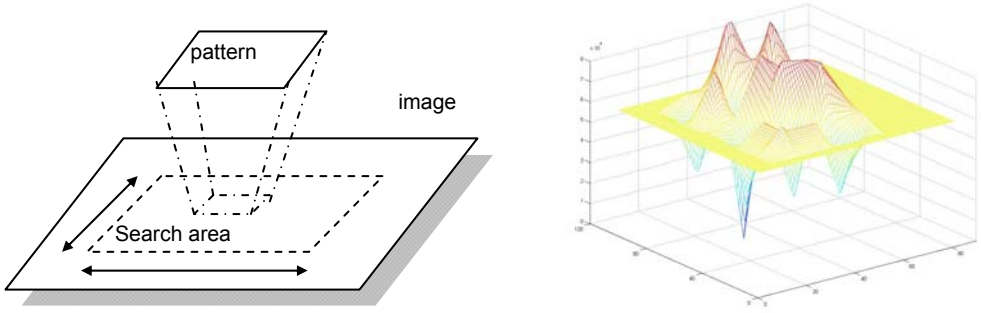
Several methods have been proposed for tracking, such as optical flow [82], Fast Fourier Transform (FFT) [84], extracting several features from the target [85] and correlation computation [86]. Correlation or block matching is widely used to detect a target object in static images [87, 88] and video [89], and in this case, it is the technique used to

implement the tracking algorithm presented in this chapter. This tracking algorithm implies an initial image format conversion. Thus, a small piece of the original image area with the target is extracted, and it is used like a pattern or reference image in the search. The next step is to perform a block matching process, i.e., to compare the pattern to the current frame (within a search area) in order to measure how well a candidate block matches the reference block. The matching criterion used is known as the sum-absolute difference (SAD). It requires the execution of simple operations (sum, subtraction and absolute value), and its expression is the following:

$$S_{ij} = \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} \left| image_{\,i+x,j+y} - pattern_{\,x,y} \right| \tag{2.1}$$

where S_{ij} is called a distortion value and a smaller value implies a higher correlation.

After displacing the pattern throughout the whole search area, a matrix with all distortion values is obtained (as depicted in Fig. 2.8b). If the minimum distortion is lower than a specific threshold (middle value between the two smallest distortion values), then the displacements (*i* and *j*) correspond to the position *(x,y)* of the target object in the current image, and the same pattern can be used to match the next coming image. In this other case, a new threshold and pattern need to be determined before working with a new image.



a. Pattern extraction process. **b. Matrix with correlation values**

Fig. 2. 8 Pattern extraction operation and resulting distortion surface.

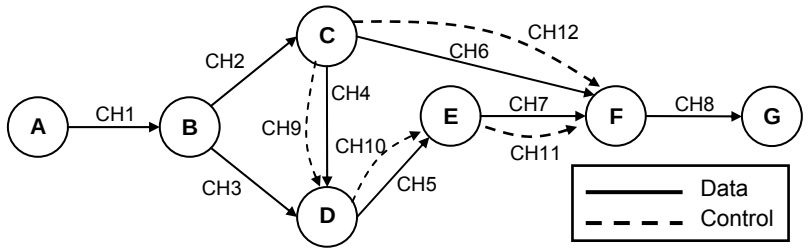
Unlike other traditional tracking algorithms based on the correlation method, the dynamic threshold update approach is the main difference of this tracking algorithm [89]. The pattern is updated only when there is a probability to lose the target object, which is

indicated by the minimum distortion value with respect to the threshold. This issue avoids unnecessary matching operations and also reduces substantially the number of execution operations and accesses to the memory.

2.4.2 Experimental results

2.4.2.1 Application modelling and a single processor solution

According to the CASSE methodology explained in Section 2.3.1, a task graph has been created manually from the initial application source code written in C++, where the corresponding ports, channels and TTL primitive calls have been included in each task according to the CASSE interface requirements. This application works with the real-time requirement of 40 ms (i.e., equivalent to process 25 frames/s), and its corresponding task graph is shown in Fig. 2.9.



Channels (in Bytes)			
CH1	19024	CH7	12024
CH2	7000	CH8	2012
CH3	7000	CH9	32
CH4	514	CH10	32
CH5	8224	CH11	32
CH6	15082	CH12	32

Task name		Computation time per frame (in ms)*	% usage time per frame*
A	Filter	28,63	6,774
B	Conversion	17,02	4,027
C	Pattern	15,99	3,784
D	SAD	355,95	84,229
E	Matching	3,41	0,807
F	Threshold	1,10	0,26
G	Vector	0,50	0,118

* Profiled for a reference ARM922T processor at 200 MHz

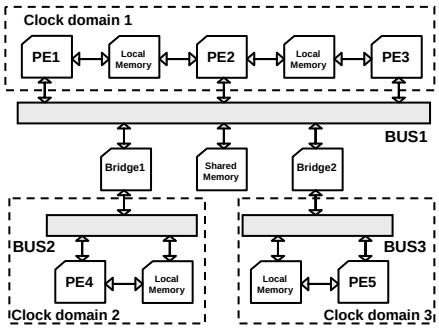
Fig. 2. 9 Tasks graph description of our tracking system.

The resulting system receives images data from a video source, which was captured in uncompressed RGB format with a resolution of 320×240 pixels. The search area in the image is of 160×96 pixels, being 24×24 pixels the pattern size. Later, this code was compiled with ADS (the ARM Developer Suite) Version 1.2 [90] for a reference processor, ARM922T at 200 MHz, and the processor execution time per frame was examined in order to identify the percentages of time spent in the different sections of the source

program. The computation time obtained for each task and the token size used in each channel are shown on Fig. 2.9. Examining this information, we can see that the system performance obtained in a single processor is too far to achieve real-time behaviour.

2.4.2.2 Multiprocessor solution analysis

In order to achieve the real-time constraint, we decided to map or distribute the application functionality (i.e., tasks and channels) onto a multiprocessor architecture. To this end, we used CASSE to specify/create manually a MPSoC description model, which is depicted in Fig. 2.10. Again, such architectural template can be built easily as a modular composition of configurable predefined elements provided by the CASSE libraries. For instance, a RISC ARM-9 processor can be obtained configuring the parameters of a PE, and also a NE can be configured as an AMBA bus with an arbiter policy based on round-robin. Bridges can be also incorporated for inter-buses communication in order to adapt different clock domains. These architecture parameters are configured with values shown in Fig. 2.10.



Architecture element	Value
PE 1	ARM-9
PE 2, PE 3	Dedicated hardware or hardware accelerator
PE 4, PE 5	Video I/O processor at 125 Mhz
Shared memory	Single port ram memory (256 KB)
Local memory	Double port ram memory (128 KB)
Buses	AMBA bus, 32 bit width, Round-Robin arbiter, Input/Output buffered

Fig. 2. 10 Architecture model used for the multiprocessor solution analysis.

In our initial mapping solution, only three PEs (PE 1, PE 2 and PE 4) were used. We would like to stress that we have made such mapping decisions based on the profiling information shown in Fig. 2.9 as well as on our intuition/experiences. This means that no search algorithms were used for the mapping in this experiment. Basically, we assigned the task with the highest computational cost (i.e., SAD) to a hardware dedicated block (PE 2), while the format conversion task and the rest of the tasks were mapped onto a video I/O processor (PE 4) and ARM processor (PE 1), respectively. Finally, once all the channels were mapped on the shared memory, an ESM was delivered by CASSE for performance simulation.

Using the ESM generated by CASSE, the design space was explored and analysed keeping fixed the bus, memory, video I/O processor and topology configurations, while the dedicated hardware acceleration factor and its operating frequency were altered. That is, the clock frequency of PE 2 and the processing timing annotated in SAD task file are changed and pondered during the DSE, in order to find the optimal configuration of the hardware dedicated block that ensures the system to achieve the real time requirement. On the other hand, the computation time annotated for the rest of tasks is kept fixed during the DSE. The resulting design space is shown in Fig. 2.11, where the green area represents the possible achievable tracking system performance with different combinations of dedicated hardware frequencies (from 200 to 800 MHz) and accelerations (from ×1 to ×10). Certainly, the number of possible solutions is very large, but given this system specification, for instance, working at 400 MHz with a ×10 hardware acceleration can achieve the real-time system specification, while at 600 MHz and with a ×5 acceleration just 17 frames per second can be processed. In our case, the criterion of minimum system bus load is used to determine the implementation solution, because this allows a bigger resource time sharing, and then more than one application (running concurrently with the tracking system) can be mapped in the same MPSoC. According to the above, a solution at 400 MHz with ×10 acceleration should be initially chosen.

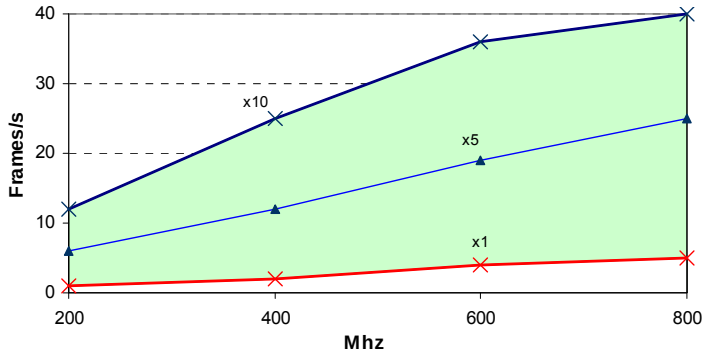


Fig. 2. 11 Achievable performance with different parameter configurations for PE 2.

In this point, it is important to highlight that the total bus load is composed, on the one hand, of the data from read and write operations, and, on the other hand, of all PEs synchronization information in order to access the shared memory via the main system bus, which becomes a bottleneck. If the processing time of the different PEs is not well balanced, then PEs have to wait for the availability of shared resources to access them. Moreover, when a task wants to read an input from a channel, it first tests the channel for the presence of data. If a data item is available, it is consumed and processed; if no data

is present, the process may decide to take some other action or continue testing the channel. This latter is a common cause of *oversynchronization*, which produces excessive unnecessary traffic in the system and even could affect the system performance. Fig. 2.12 shows different synchronization loads produced by several ports per each PE in this architecture. If all PEs are balanced perfectly, minimum synchronization load is obtained. However, this "ideally balanced" system is not always achieved due to communication delays or transmission effects of the network elements, although the tasks be balanced computationally. Finally, the synchronization loads of the initial solution are also shown in Fig. 2.12.

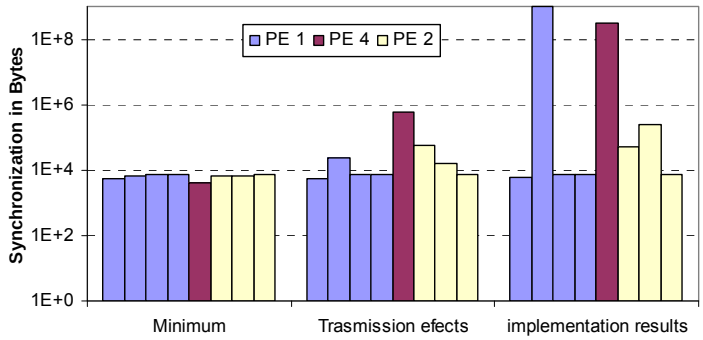


Fig. 2. 12 Synchronization bytes for each port of PE to process 125 frames.

2.4.2.3 Optimized solutions analysis

Examining this detailed information about the system traffic load (provided by CASSE and shown in Fig. 2.12), we can detect that the initial mapping solution has a clear symptom of oversynchronization in spite of being the solution with the lowest bus load among all possible candidates. This effect is relevant especially in port 2 of PE 1 and PE 4. This information shows that these ports are blocked too often waiting for data in their channels, and such high load is produced while polling the channels status.

The first optimized solution implies to map the most oversynchronized channel of PE 1 in a local memory. On the other hand, such optimized solution could still be improved using a local memory for PE 4 synchronization operations, and as a result, a second optimized solution is obtained. This way, both PE 1 and PE 4 can access the main bus and the shared memory to allocate data, while accessing their local memory for synchronization operations. Fig. 2.13 shows the bus load results due to synchronization for these three options.

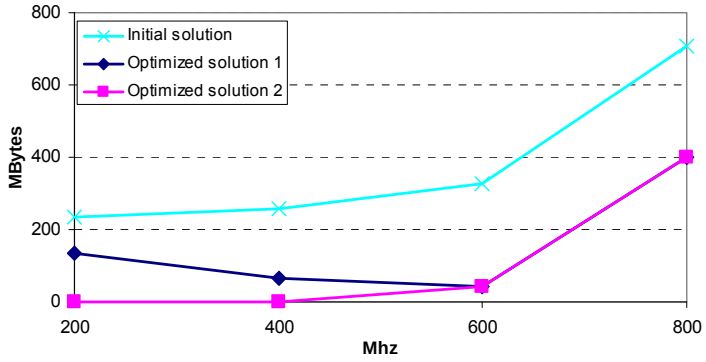


Fig. 2. 13 Total synchronization load generated by different mapping solutions.

2.4.2.4 Multi-applications analysis

Following our purpose of building a complete CVS, another specific objective of our analysis is to determine the possibility of mapping other applications in this platform and still complying with the real-time tracking system requirements. The typical characteristics of image-processing, audio-processing, and signal-processing applications, for which regular streams of data are processed at a constant rate or periodically, were taken into account to develop these new applications. More specifically, our tracking application was analysed in different scenarios, where it was coexisting with four different types of traffic generated by an additional producer-consumer application in the same system. That is, the producer-consumer application works like a generator of synthetic traffic load in the system, and it can generate four different kinds of traffics. First, constant traffic, where the new application shares the same shared memory with tracking application. Second, periodic traffic was taken into account, where the new application accesses the main memory at a specific rate, in this case with periods of 10 ms and 20 ms. Finally, random traffic also is included in this analysis. This analysis allows the designer to know how the target system performance is with regard to additional load produced by another application in the system. The results of this analysis are shown in Fig. 2.14.

In this case, tasks of the new application (i.e., the producer and the consumer) are mapped manually onto PE 3 and PE 5. Moreover, the channels of the new tasks graph were also mapped onto the shared memory. Some experiments were performed varying the data size processed on the new application. Following with the last example, the tracking system performance running at 400 MHz and with a ×10 acceleration and in the presence of the other application is shown in Fig. 2.14. The results indicate that the tracking system can work at 25 frames/s when an additional application produces a

constant traffic load with data of size lower than 100 Kbytes, or another one which produces traffic load with period of 10 ms, being its data size lower than 2 Mbytes.

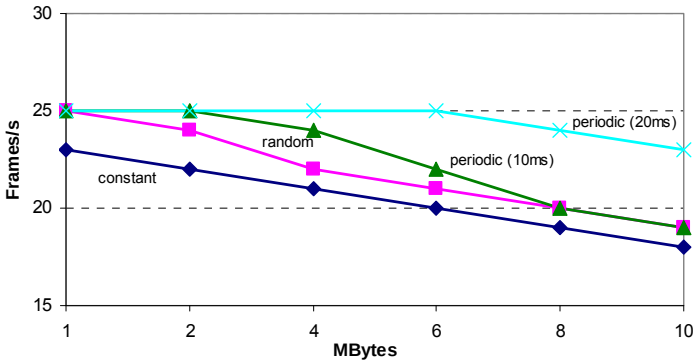


Fig. 2. 14 Tracking system performance with other applications running on the target MPSoC platform.

2.5 Conclusions

At the same time that the design abstraction level is raised to system level, an important number of modelling and simulation tools emerged to support system designers to make design decisions and DSE in an early design stage. In our case, a SystemC-based modelling and simulation tool called CASSE, developed at Research Institute of Applied Microelectronics (IUMA) of ULPGC, is used in this thesis. CASSE is based on the Y-Chart principle, and it allows the system designers to specify an application model, architectural model and a mapping model by means of the textual description files. As a result, CASSE generates an ESM that is used for performance simulation.

Although a wide range of system metrics (such as performance and system traffic load) can be obtained from CASSE after each simulation, other key design metrics (like power consumption and cost/area) are not provided by the current version of CASSE yet. On the other hand, all the aforementioned models (i.e., task-graph, architectural and mapping) need to be created manually by the system designers prior to each DSE experiment. For instance, the system designers have to change/configure manually such description files each time a new design decision is taken (e.g., different mappings, distinct combinations of component configurations, platform topology, etc). From our point of view, such manual design process is time-consuming, error-prone, and overall, inefficient for exploring large design spaces. Therefore, some kind of new infrastructure is needed to support the system-level DSE. Such infrastructure should (i) include some

mechanisms to generate automatically each ESM, (ii) allow the system designers to select the most appropriate technique to lead the search process, and (ii) be capable to integrate the simulation tools in order to evaluate different alternatives of design space, as well as provide the system metrics to guide the search algorithms. All these challenges will be addressed in Chapter 3 of this thesis, where a new DSE methodology and developments of the NASA framework are explained. Chapter 4 will explain new concepts and strategies developed in this thesis for application mapping and searching in DSE of heterogeneous MPSoCs, using NASA as our prime framework and showing ways to incorporate other frameworks.

Chapter 3
Methodology and infrastructure for multidimensional DSE

This chapter presents several contributions of this thesis: NASA infrastructure, dimension-oriented DSE methodology, and a generator of architectural platforms. Moreover, we also illustrate how NASA can couple the aforementioned approaches and CASSE in a unified environment, allowing thus the system designers to explore large design spaces in an efficient and automatic way. Finally, we configure NASA with genetic algorithms in order to perform several sets of DSE experiments. The goals of these experiments are to demonstrate the capability of NASA framework and to prove the benefits of our dimension-oriented DSE methodology.

3.1.1 Design space dimension

System-level design space exploration (DSE) consists of exploring a wide range of design choices during the early design stages, allowing system designers to have a rapid and complete vision about the impact of different design options on the system performance and behaviour. As a consequence, such early DSE is becoming a key element in system-level design as it influences heavily the success or failure of the final product, and can avoid wasting time and effort in further design steps without the possibility of meeting design requirements because of an inappropriate system design decision.

In modern SoC design, a wide variety of system parameters and design choices should be explored in order to find an optimal system design or design point. Such design decisions include the number and type of PEs, SEs and NEs in the MPSoC platform, the architectural topology, the mapping of tasks and communications onto architecture resources, and scheduling policies. This way, the design points are usually the result of specific combinations of choices related to some features of the design.

	Component parameters		
	PE	**SE**	**NE**
Topology	General organization, relations among components, resources allocation, restriction on number and type of instantiated components		
Architectural Components	Voltage, operating frequency, scheduler, context switch overhead	Latency, word size, storage capacity	I/O buffered, bandwidth, latency, energy
Mapping	HW/SW, clustering, migration	Resource sharing, cache structure, data transfer size	Arbitration policy, communication protocol overhead

Fig. 3. 1 An example of the classification of design options in design space dimensions.

On the other hand, the design decisions can be clustered in dimensions[2] of design space, as illustrated in Fig. 3.1. Typically, a dimension could represent design decisions that are

[2] It should be noticed that design decisions (or options) clustered in each dimension depend on the system designers, so different designers can use distinct criteria to fix the definition/classification of each design space dimension.

orthogonal to each other. In the example shown in Fig. 3.1, three dimensions can be distinguished: mapping, architectural components, and platform. Here, the platform dimension explores the topology or platform structure, defining the number of architectural elements and their topological interconnection; the architectural components dimension explores design decisions about the types of architectural components (PE types, SE types, etc.) inside a platform architecture; finally, the mapping dimension explores different mappings of application tasks and communications onto the underlying architecture. Evidently, the more details (or dimensions) taken into account, the larger the design space that needs to be searched, and therefore the more costly the analysis.

3.1.2 Generic infrastructure for multidimensional DSE

In order to assess such aforementioned large amount of design options (and consequently, large number of system design alternatives), the system designers can use different evaluation techniques: analytical model and/or simulations, as mentioned in Section 1.4.2.2. Although both techniques have received significant research attention during the last decades [58, 91-93], we will focus on simulations-based DSE in this chapter, while DSE based on analytical methods will be addressed widely in Chapter 4.

However, taking into account our experimental results shown in Chapter 2, we can see that creating manually each ESM and/or simulating exhaustively all possible design points of the design space is prohibitive for designing modern SoCs. Thus, the simulation tools only provide a partial solution for DSE, since an overall framework is still needed to systematically explore the design space. In fact, as pointed out in Section 1.4.2, the process of system-level DSE logically consists of three interdependent components: search mechanisms, ESM generator, and evaluation methods, such that a DSE framework should not only allow for exploring large design spaces in a time-efficient and automatic way, but also should be flexible and reusable to carry out different DSE experiments. Although many DSE approaches have been proposed, three common factors can be identified in all of them:

1) DSE efforts are usually targeted to specific system-level simulation tools (or analytical evaluation method), where each effort typically uses a different kind of simulator. Consequently, it is hard to re-use these DSE frameworks and the elements available in them.

2) Setting up the DSE experiments can be very labour intensive. It is often the case that for every experiment, control scripts need to be (re-)written to manipulate the simulation

parameters and configuration files (specifying the design instance to evaluate) according to the algorithm that searches through the design space. These scripts are often inflexible and hard to re-use for different types of DSE experiments, i.e., assessing different parameters or parameter ranges.

3) In spite of the wide variety of eligible architectures for implementing embedded systems applications, many DSE experiments are focused on a particular class of MPSoC architectures only. This mainly happens because no tools are currently available to automatically and generically generate different architecture models from abstract input descriptions and, consequently, designers have to write such models manually. This latter is an error-prone task and one of the bottlenecks in improving designer's productivity, and severely limits the size of the design space that can be explored in a reasonable time.

In summary, to the best of our knowledge, there does not exist a generic infrastructure to facilitate and support system-level MPSoC DSE experiments, and to foster the re-use of software in the context of system-level MPSoC DSE. This calls for a unified framework that integrates and couples both simulation and search mechanisms to efficiently and systematically explore design spaces, as well as a fast tool to automatically generate a wide range of architecture models, so that a large variety of architectures can be easily explored and evaluated.

In this chapter, we present a new methodology, techniques, and an infrastructure to address the above challenge. On one hand, we introduce a new generic system-level DSE infrastructure implemented in C++, called NASA (*Non Ad-hoc Search Algorithm*) [94]. Its main goal is to provide a single, common, and modular framework for system-level DSE experiments. It allows for incorporating different (existing) system-level simulation tools as well as different combinations of search strategies by means of a simple *plug-in* mechanism. On the other hand, an architectural platform generator has also been integrated in NASA to free designers from the efforts to manually create architecture models. Thus, this automation improves the design productivity and enables the designer to focus on the more valuable issue of making design decisions. Finally, we propose a new methodology for system-level DSE called *dimension-oriented DSE approach*, which allows designers to configure the appropriate number of search algorithms to simultaneously co-explore the various design space dimensions. As a consequence, our framework provides a flexible and re-usable environment to

systematically explore the multidimensional MPSoC design space, starting from a set of relatively simple user specifications.

3.2 Related work

Performing DSE in a time-efficient and accurate way is not a new problem and there exists a large body of related work in this area. Most of the approaches in the embedded systems domain are targeted to the system-level exploration of the optimal distribution of application tasks on a selected set of PEs such as generic RISC processors, ASIPs, or dedicated hardware IP blocks considering time constraints, power consumption, and/or hardware area/cost [91, 95-99]. To meet these constraints, numerous mapping approaches can be applied such as genetic algorithms [100, 101], combining analytical models and simulation tools [92, 102], and/or iterative heuristics methods [103].

Although these approaches are fairly efficient to explore various alternatives for mapping a specific application onto a target MP-SoC architecture, they typically still require significant effort to (re-)write scripts that control the evaluation mechanism (analytical model or simulator) during the search throughout the design space. In fact, this often means that there exists a repetitive effort in building customized scripts and/or architecture models for every different kind of DSE experiment. Thus, automating such a process becomes a key element in terms of reusability and flexibility for larger design space explorations in the design of an heterogeneous multiprocessor architecture.

Unfortunately, for architecture model generation, only a few approaches have been developed [93, 104, 105], and for most of them, manual interventions are still needed, which is time-consuming and error-prone. Moreover, these useful generators still need to be integrated in a single environment and coupled to some kind of evaluation tool(s) (analytical model or simulator) and search algorithms to completely automate the entire DSE process.

Several proposals to integrate external design-point evaluation tools into a DSE environment can also be found in literature. In [46], a hierarchical and three-phase DSE methodology is presented. It facilitates the integration of simulators by using a set of tool-dependent interpreters or adapters. Angiolini et al. [106] present a framework that integrates an ASIP tool-chain within a virtual platform to explore a number of axes of the MPSoC configuration space. However, unlike our work, this framework does not allow the

integration of external search methods. Moreover, it still requires human intervention in the feedback loop of the searching and optimization process.

The MultiCube project [102] has similar objectives as the framework presented in this chapter, but it targets the exploration of the configuration space of homogeneous chip multiprocessors rather than system-level MPSoC platform DSE. This implies that it has limited or no capabilities to explore different application-to-architecture mappings, heterogeneous processing elements or different interconnection schemes.

Other works have also developed a modular interface-based system-level MPSoC DSE framework [107, 108]. In these cases, different search algorithms can be plugged in, but the resulting DSE is limited in terms of the target MPSoC platforms that can be explored. Consequently, the different architecture instances that can be derived from those fixed MPSoC platforms, may not be flexible enough to meet the requirements of different application domains, and not scalable enough to meet the computational needs of a large variety of applications.

Künzli et al. [109] proposed a generic and modular framework based on PISA [110] for DSE of embedded systems. The PISA interface separates the problem-dependent variation and estimation part from the generic search and selection. The resulting two parts are implemented as independent processes that communicate via text files. As a result, it resolves the problem that existing optimisation methods cannot be coupled easily to the problem-specific part of a design exploration tool. But, unlike our work and to the best of our knowledge, they have only coupled analytical models to evaluate design points. This means that, e.g., the problem of incorporating a system model generator and external simulation tools has not been addressed.

Using pre-compiled and ready-to-use search algorithms available at [111] of the PISA framework, Madsen et al. [112] have created a multi-objective DSE framework. Different mapping alternatives can be evaluated (by means of analytical models) for a fixed or flexible platform during the exploration process. Moreover, the chosen representation formats for internal interfaces in [112] are problem specific, which means that they should be modified for each particular problem. In our case, these are dynamically and automatically updated according to an input constraints file. Finally, the kind of platforms generated in [112] is limited to hierarchical bus topologies, while our approach is not restricted to analyse a particular architecture.

3.3 Preliminaries and definitions

Before explaining the NASA framework in detail, the terms and notations used along this chapter are briefly presented.

Definition 1. A design point is a system design specification defined by a set of specific design options. An example of such a design point would be an architecture consisting of three ARM processors and two memories connected to a single AMBA bus, where the application functionalities are mapped onto a single ARM processor, and the communication channels are assigned to both memories. In this context, a feasible design point is a specific design of the system that meets the user constraints in terms of such design options. This means that, for example, only the available number and types of architectural components (PEs, SEs, NEs, etc.) can be used to create a feasible architecture. Moreover, the functionalities or tasks of an application have to be mapped onto processing resources actually instantiated in the architecture. Further, communications between application tasks must be mapped onto storage resources which are actually accessible by the PEs onto which the communicating tasks are mapped. If at least one of above conditions is not satisfied, then the resulting design is classified as an infeasible one.

Definition 2. A design space, D, can be formally defined as:

$$D = d_1 \times d_2 \times ... \times d_k \tag{3.1}$$

Here, d_i refers to the design decisions or options in a particular dimension i, and the operator $\times$ refers to the Cartesian product. Thus, a design point dp in D can be expressed by linking k available design option values ($d_{map} \times d_{arc} \times d_{pla}$) corresponding to each of k design space dimensions, where d_{map} represents a design decision in the mapping dimension, while d_{arc} and d_{pla} express a particular design decision for the architectural component and the platform dimensions, respectively. This way, finding the optimal or near-to optimal design point consists of a multidimensional exploration process, searching for the best combination of values in all dimensions of D that optimizes all the imposed objectives (e.g., performance, power, cost, etc.).

Definition 3. The size of a design space is equal to the product of the cardinalities of the set d_i, $\forall$i=1..k:

$$| D | = | d_1 | \times | d_2 | \times ... \times | d_k | \quad (3.2)$$

Consequently, the more dimensions or the larger each d_i, the larger the resulting design space is. To illustrate how large such a design space can become, we provide an approximation of the size of the design space we address in the results section of this chapter. To this end, we use the expression presented in [21] to calculate the size of one dimension, while the size of a multidimensional design space would be the product of all dimensions. Hence, the expression of the size of a 3D design space which considers mapping, architectural components and platform topologies, can be roughly approximated as follows:

$$| D | = w \times (nt^n * st^s * pt^p) \times p^t \quad (3.3)$$

Here t is the number of tasks in the application, p is the number of PEs, s is the number of SEs, n is the number of NEs, pt, st, nt are the number of types of PEs, SEs and NEs, respectively, and w represents the number of different platforms or topologies the designer desires to explore. In order to simplify the estimation, the mapping of the communication channels of the application onto storages elements has not been taken into account. Using equation (3.3), $1.3*10^{12}$ design points would have to be evaluated for the following (fairly moderate) set of values: $t=7$, $p=6$, $s=3$, $n=4$, $pt=3$, $st=3$, $nt=3$ and $w=3$. This result highlights the difficulty of exhaustively evaluating all these design points, even when the evaluation operates at a high abstraction level.

Definition 4. A *Processing Element* (PE) is an architectural resource onto which the application's functionalities can be mapped. We assume that there is a library of candidates PEs such as generic RISC processors, ASIPs, dedicated hardware IP blocks, and alike. Thus, the timing information for every task (i.e., how long it takes for each PE to execute each application's task) should be first determined. To obtain the execution time of an application's task on a processor, we measure execution time using instruction set simulators. Note that the execution time for a specific application's task on a dedicated hardware implementation is assumed to be already given by the designer or IP provider. To capture the impossibility to map application tasks onto a dedicated component that does not implement these tasks, we model an infinite amount of execution time for these tasks on this particular component. As a result, a table with t rows and pt columns is obtained.

3.4 Overview of the NASA framework

With our methodology and the NASA framework, we aim to provide a generic infrastructure for performing system-level MPSoC DSE experiments in an easy and fast manner. To this end, four key properties have been taken into account in the design of NASA:

- ***Modularity*.** NASA is a highly modular framework in which the interaction between its modules is established by well-defined interfaces, allowing each module to act like an independent black box inside the framework. As a result, different search algorithms, feasibility checkers, and system-level simulators can be easily integrated in a plug-and-play fashion.

- ***Flexibility*.** According to the designer's needs, different experiments to explore different aspects of the design space can be performed using NASA. For example, as it will be explained in more detail later, a key element in NASA is its *hierarchical* DSE approach in which the exploration dimensions are explicitly separated into three levels: platform exploration, architecture exploration, and mapping exploration. Subsequently, the designer can choose to simultaneously explore at all of these levels, or to fix one or more of these levels (e.g., to settle a fixed platform) and to focus the exploration on one or two levels (e.g., mapping exploration only).

- ***Re-usability*.** For a given set of user constraints, NASA is capable of exploring the design space in a systematic way, automatically generating the ESM of selected design points that need to be evaluated by the system-level simulator. Hence, there is no need to prepare experiment-customized scripts. To perform a new DSE experiment, a designer only requires changing the constraint values.

- ***Extensibility*.** Due to the modularity and the well-defined interfaces, new modules or functionalities can easily be plugged into the NASA framework. These new modules could, for instance, handle additional dimensions in the design space without needing to modify other modules.

The infrastructure of NASA is shown in Fig. 3.2. Essentially, six main modules can be distinguished in the framework: the Search module, Feasibility Checker, Architectural Platform Generator, Translator, Simulator and Evaluator. These modules work as independent black boxes, using well-defined interfaces based on text files to interact with

each other. The remainder of this section provides a high-level overview of the NASA framework. In the next section, each of the modules within NASA will be discussed in more detail.

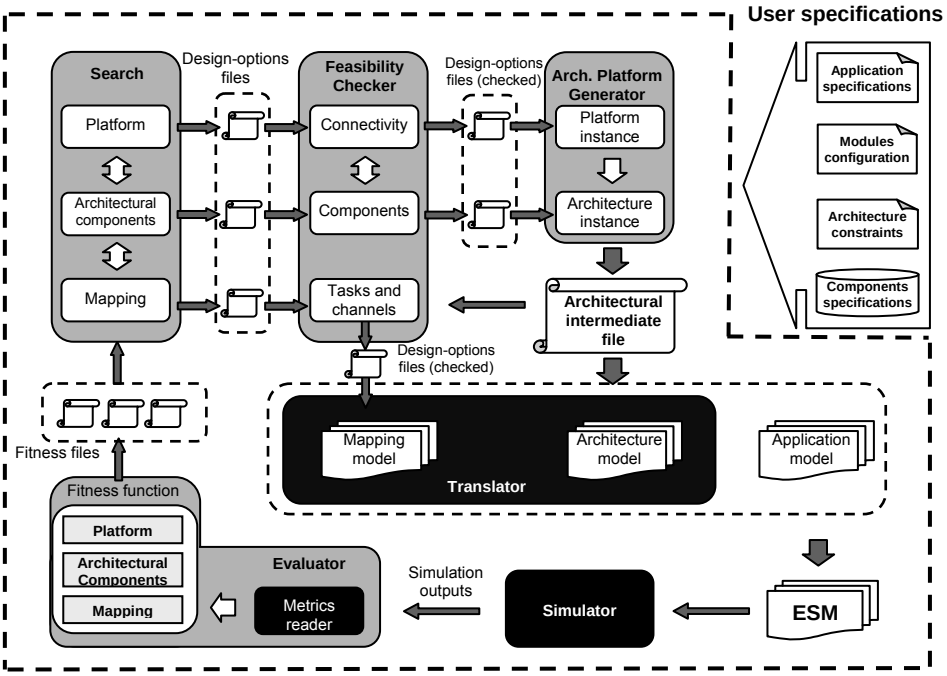


Fig. 3. 2 The NASA infrastructure.

In the Search module, the design space is explored in an iterative fashion. By means of plug-ins, such exploration can be done exhaustively, randomly, or using a heuristic search algorithm such as, e.g., evolutionary algorithms, simulated annealing, or ant colony algorithms. Moreover, as mentioned before, the DSE approach is hierarchical and currently distinguishes three levels or dimensions: platform, architectural components and mapping levels. These levels can be explored by using a single search algorithm or can be co-explored simultaneously, possibly using different search algorithms. This means that different (tailored) search algorithms can be used at each level. Moreover, our DSE methodology based on dimension-oriented co-exploration provides flexibility as it allows DSE experiments to fix one or more levels of exploration if needed. For example, to find an optimal platform configuration, one could fix the platform level and perform exploration at the architectural components and mapping levels.

Because the search algorithms may try to assess infeasible design points during the DSE process, a feasibility checking module is necessary to analyse the feasibility of design

points according to the user constraints files. If infeasible design points are detected, then NASA's repair mechanisms can be applied to convert them to feasible ones with only a minimum influence on the run-time of the framework. Note that this checking process should also be performed in a hierarchical fashion, because in order to map the tasks of an application onto different architectural components, a feasible architectural platform is needed first to reflect that decision. Thereby this implies that the checking of mapping feasibility should be only performed after ensuring that the corresponding design point consists of a feasible topology or platform with a feasible number and correct types of architectural components.

Notice that in order to integrate a system-level simulator in NASA, it is required that the simulator allows for explicitly modelling the design points that need to be simulated using some kind of file format. This means that a file-based ESM (composed of an application model file, architecture model file and mapping model file) is required by the system-level simulator to evaluate the specified design point in order to obtain different system-wide metrics.

In our framework, such an ESM can be generated in an automatic and gradual way. First, the Architectural Platform Generator combines the information about the platform and architectural components (in the architectural intermediate file) to describe the architectural design decisions for each selected design point. For example, it describes the number and types of NEs, PEs, SEs and the connectivity between components. Subsequently, the Translator translates the architectural platform representation of each selected design point and its associated mapping description into detailed and simulator-specific architecture and mapping models. These two file-based models form, together with the application model, the ESM which is required by the (external) system-level simulator. So, to integrate a new system-level simulator simply requires a new Translator module that generates the ESM that specifies a design point and the simulator-specific configuration file(s). This is why two kinds of module colours can be identified in Fig. 3.2: simulation-tool-dependent (black) and simulation-tool-independent (grey) modules.

Using the ESM generated by the Translator module, the system-level simulator is used to obtain different system metrics, like performance and power consumption. The simulation results are used by NASA's Evaluator module to evaluate the *fitness* of design points, providing feedback to the Search module and guiding it in a systematic way through the design space. Finally, after running a user specified number of search iterations, NASA provides as output information about all explored design points as well as a set of optimal

design solutions within the explored design space, identifying those design points that best meet the user constraints such as real-time application constraints, costs/area and power consumption. In the next section, we provide more details about the implementation of each of the aforementioned modules.

3.5 Implementation of NASA

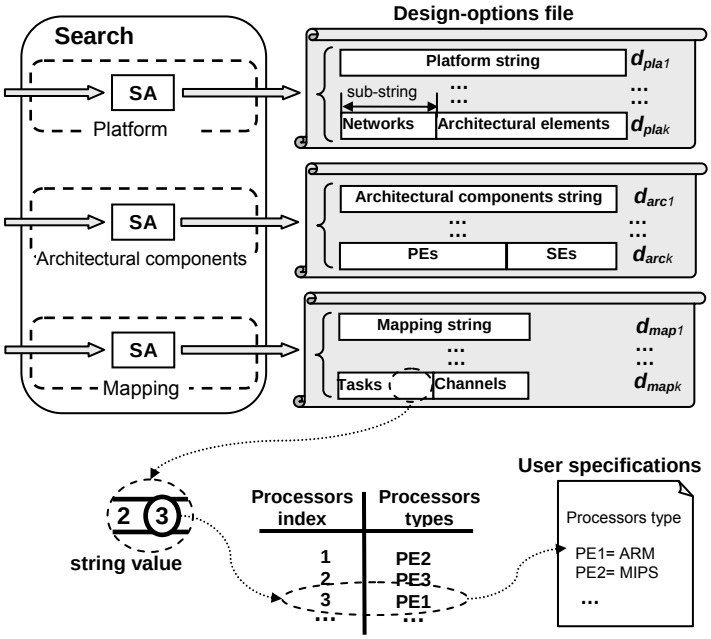
3.5.1 Interfaces

The files-based interfaces in NASA, which allow its modules to operate as black boxes, are an essential element to yield a flexible and extensible framework. Three kinds of interfaces are used inside NASA: the *architectural intermediate file* is used for communication between the Architectural Platform Generator and Translator, the *fitness file* links the Evaluator with the Search module, and the *design-options file* is used in all sub-modules of both the Search module and the Feasibility Checker. Note that these files are dynamically and automatically created (and updated) according to the user input files.

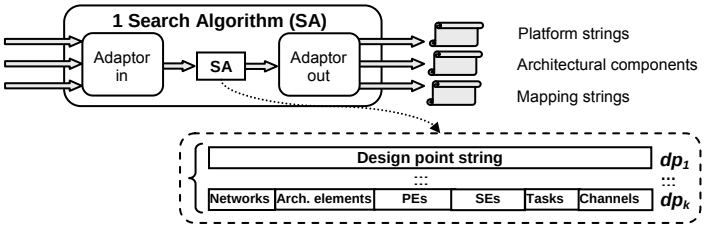
In our approach, both the design-options and fitness files share the same format, in which *design decisions are encoded in numeric strings*. Moreover, each explored dimension uses a separate design-options and fitness file. For example, in the 3-level design space exploration shown in Fig. 3.3, the platform dimension uses a design-options file to describe design decisions about the topology, NE type(s) and the connectivity properties for the rest of architectural elements of a design point; the architectural components dimension uses its corresponding design-options file to specify the type information of different components, while the decisions about the mapping of an application onto the different PEs and SEs are described in a third design-options file. If the designer decides to use less than one search algorithm per dimension, then adapter modules will automatically translate the input and output of the Search module to match the one design-option file per search algorithm interface. Note that the number of strings contained in any design-options file is equal to the number of design points explored by the Search module in each iteration, as will also be explained in the next section. Examples of design-decision strings are shown in Fig. 3.3, for three (Fig. 3.3a) and one (Figure 3.3b) search algorithm (SA) in the Search module.

The length of a string description for each dimension may vary. Using the example shown in Fig. 3.3, it is evident that the length of the string describing the mapping depends on the number of tasks and communication channels in the application. Similarly, the length

of the string describing the architecture instance is dependent on the number of PEs and SEs in the platform.



a. Search module configured with three search algorithms.



b. Single search algorithm for the DSE.

Fig. 3. 3 Search Algorithms (SA) and search strings in NASA.

Finally, the values inside the design-decision strings do not hard-code absolute values but are indirections to table entries (also illustrated in Fig. 3.3a). This means that, for example, in the case of the mapping dimension, the string elements do not directly hard-code the PEs (including their exact type) onto which application tasks are mapped. Instead, the string elements point to entries in a PEs table. Hence, this allows the designer to, e.g., change the type of PE or add a new type without the need to adapt any

module implementation. Clearly, this makes our framework more re-usable and extensible.

The last important interface in NASA is the architectural intermediate file. It describes the architectural platform design of each design point in a single file and, as will be explained in more detail later, it is gradually constructed using the platform and architectural components strings: the platform string produces a topological template instance of the design point, while the architectural components string specifies the types of the architecture components in this template. The architectural intermediate file is used by the Translator to generate an architecture model of the design point in question. Moreover, it is also used to check the mapping feasibility. Note that platforms are not fixed entities in NASA but are often also part of the exploration. Therefore, the Feasibility Checker requires, e.g., connectivity information specifying which and how PEs are connected, and which SEs are shared by which PEs. This information is needed to detect and repair infeasible mappings, as will be explained in Section 3.5.3.

3.5.2 Search module and dimension-oriented co-exploration

This module performs the actual search through the design space, iteratively pinpointing (a set of) design points that need to be evaluated by means of system-level simulation. As mentioned before, NASA applies a dimension-oriented design space exploration approach and currently distinguishes three dimensions (or levels): platform, architectural components and mapping level. This way, each dimension can be co-explored simultaneously using a single search algorithm, or using multiple and possibly different search algorithms for the various dimensions. The designer simply configures the number of search algorithms to be used in the exploration process. In this context, co-exploration means that, in spite of using one search algorithm per dimension, we do *not* perform the system-level design space exploration as multiple independent explorations, but instead, the results from *all* dimensions are simultaneously taken into account.

It should be noted that, independently of the number of search algorithms used in NASA, the Search module always provides *x* sets-of-strings (or design-options files) to the Feasibility Checker, where *x* is equal to the number of dimensions of the explored design space (*x=3* for our platform, architectural components and mapping dimensions). For example, when a single search algorithm is used in the Search module, adapter modules will be automatically plugged in to translate the inputs and outputs of the Search module to comply to this *x* set-of-strings interface, as illustrated in Fig. 3.3b.

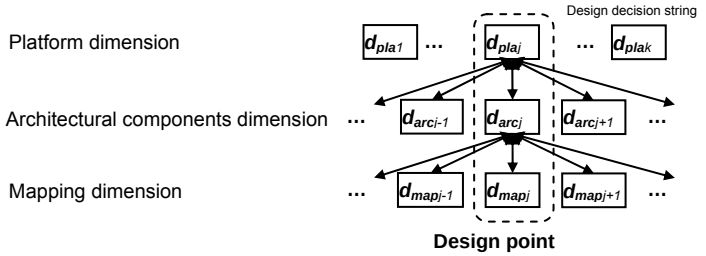
If multiple search algorithms are used to explore the design space, then there are many ways of linking the design decisions of each dimension to form a design point specification. For example, using a pyramidal technique (as shown in Fig. 3.4a), all (or some) of the design decisions in the mapping dimension are linked with each of the decision decisions in the architectural components dimension, while the latter are again all linked with each of the design decisions in the platform dimension. However, this means that the number of design points to be evaluated in each search iteration grows exponentially with the number of design decisions (or strings) of each dimension.

The other extreme is a pure one-to-one linking technique (as shown in Fig. 3.4b). This means that each design decision in each dimension is linked to only one design decision in the other dimensions. Thus, the number of design points explored per iteration by the Search module is equal to the number of design decisions (or strings) contained in any design-options file, assuming that all design-options files have the same number of strings. Clearly, this significantly reduces the number of required evaluations because of the linear relationship between the number of design decisions and design points. However, this approach may suffer from a possible convergence problem due to *under-exploration*, i.e., discarding a design decision (e.g., a specific platform instance) too soon based on the results of a premature evaluation.

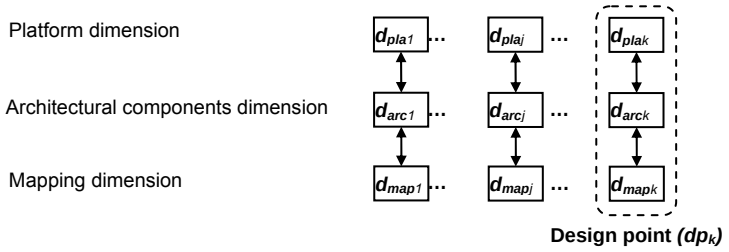
Example 3.1. Let $A = \{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ be a design point built with platform d^A_{pla}, architectural components d^A_{arc} and mapping d^A_{map}, while $B = \{d^B_{pla}, d^B_{arc}, d^B_{map}\}$ is another design point formed by platform d^B_{pla}, architectural components d^B_{arc} and mapping d^B_{map}. If it turns out after a single simulation that the fitness value of *A* is better than that of *B*, then this does not mean that platform d^A_{pla} or architectural components d^A_{arc} are always a better choice than d^B_{pla} and d^B_{arc}, but we can affirm indeed that the combination of design options $A = \{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ is better than $B = \{d^B_{pla}, d^B_{arc}, d^B_{map}\}$. For instance, this latter does not guarantee that $\{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ can provide a better fitness value than $\{d^B_{pla}, d^B_{arc}, d^C_{map}\}$, where d^C_{map} is another feasible mapping for *B*.

To address this under-exploration problem, we use a variant of one-to-one linking of design decisions. In this technique, unlike the pure one-to-one technique, only design decisions from the dimension of the lowest abstraction level (i.e., the mapping dimension in our case) are evaluated *and updated* during each search iteration. The search

algorithms for the higher-level dimensions (i.e., the platform and architectural components dimensions) keep collecting the fitness values (for different mappings) without actually changing their design decisions during a specified number of iterations, referred to as the *collecting iterations* (δ). Only when the search has reached δ iterations, design decisions are updated, after which the process starts again. Obviously, the higher the abstraction level, the more design alternatives can be derived for a single design option (e.g., a multitude of architecture instances can be obtained from a single platform) and, consequently, the higher the value of δ should be. Note that the above mentioned feedback information, i.e., the fitness values, needed to guide this search through the design space, are iteratively provided by the Evaluator module, which will be explained in Section 3.5.7.



a. Piramidal linking technique.



b. One-to-one linking technique.

Fig. 3. 4 Different techniques to link design decisions in a single design point.

3.5.3 Feasibility checker

The main task of the Feasibility Checker is to detect infeasible design points and repair those design points if possible. During this checking process, all sets-of-strings (or design-options files) are checked in a hierarchical fashion. For example, for our 3-level exploration as shown in Fig. 3.3a, in order to distribute or map the functionalities of the application onto different architectural components, a feasible architectural platform is

first required to be able to reflect that decision. Thus, the platform string is first checked to determine whether or not the specified platform template contains a valid topology and, e.g., whether it does not contain isolated islands of components (to be discussed in more detail in the Example 3.2). Next, the architectural components string is checked to determine whether or not the number and types of selected architectural components in the platform template comply with the constraints provided by the user. For example, if a design point deploys 4 ARM processors while the user has specified that only 2 ARM processors can be instantiated, then we have an infeasible design point. Finally, the mapping string is checked for infeasibility, e.g., when application tasks are mapped onto PEs that have not been allocated in the platform, or in the case there is no shared memory to map a logical communication channel between two tasks that have been assigned to different PEs. So, each design point is globally checked, i.e., taking all dimensions of the design point into account.

If an infeasible design point is detected, then different kinds of repair mechanisms can be applied, depending on the dimension where the problem occurs. Note that different repair techniques can also produce different feasible solutions from the same infeasible design decision. In our current implementation, we use heuristic minimum-distance repair techniques, which introduce a minimum number of modifications to an infeasible design string in order to obtain a feasible one. As a consequence, our repair techniques only have a minimal effect on the run-time of the framework. In the aforementioned infeasible mapping example (i.e., no reachable memory for two communicating tasks), only the communication channel of those two application tasks should be relocated into an reachable memory if a feasible mapping can be derived from such a repair[3].

The impact of these repair mechanisms on the number of explored feasible design points will be discussed in Section 3.6.2. Specifically, our experimental results reveal that these repair techniques can warrant the repair of a high percentage of infeasible design points in the DSE experiments.

Apart from separating the Feasibility Checker in three parts (platform checking, architectural components checking and mapping checking), specific sub-modules can also be distinguished inside each part. For example, there is a sub-module for checking the connectivity properties of PEs, one for checking SEs, one that specializes in task

[3] Although it is also possible to repair by mapping one of those two application tasks onto another available PE (or even both application tasks onto the same PE), this would require the resulting mapping to re-enter for a new mapping feasibility check as it may cause additional infeasibilities for other communication channels. In the worst case, this may even cause an infinite loop.

mapping checking, and so on. Since all these checking sub-modules are based on heuristics, the user can freely replace them by other implementations, which again illustrates the flexibility aspect of our framework, as mentioned in previous sections.

3.5.4 Architectural platform generator

The main mission of this module is to provide the architectural description for each design point by means of combining both feasible platform and architectural components information, which are contained in the strings of their respective design-option files. The resulting architectural description file is used later for (i) feasibility checking of mapping strings, and (ii) as input (to the Translator) to generate the architectural model. Thus, the Architectural Platform Generator can be considered as the first stage of the ESM generation process.

Basic Topology Unit

An architectural description is created in two steps: platform or topological template generation and architecture instance generation. The basic building block of these descriptions is the so-called *Basic Topology Unit* (BTU). As shown in Fig. 3.5, the BTU is a *logical pattern* consisting of a *network container* (the gray component) and a variable number of *element containers* (the white blocks). These element containers are labelled inside each BTU and can, in a later stage, be instantiated as architectural components such as PEs and SEs. The number of element containers in a BTU depends on the user specifications, like the maximum number of PEs and SEs in a platform. Note that network containers cannot directly connect to each other, while element containers can connect to both element and network containers.

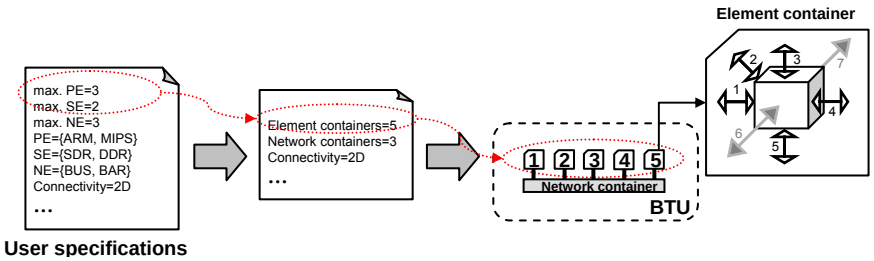


Fig. 3. 5 Example of a BTU generation and element container.

Meta-platform

The BTU is labelled and replicated a number of times to form a *meta-platform*, which is used later in topological template generation. In principle, the meta-platform is used as a

basis from which all feasible platform instance descriptions can be (gradually) derived and generated. The number of BTU replications in the meta-platform depends on the maximum number of NEs and connections allowed among element containers, as specified by the user. The latter is referred to as connectivity, which defines for each element container both the available links and the directions (represented with numbered arrows in box of element container of Fig. 3.5). Thus, a BTU can be replicated through two or three directions and, as consequence, different kind of meta-platforms can be generated according to the user specifications. A 2D meta-platform generation process is shown in Fig. 3.6, although a 3D meta-platform can be also generated if the gray links of an element container (see Fig. 3.5) are also used during this process. It should be noticed that the generation of the BTUs as well as the meta-platform is performed statically (but automatically) before the actual DSE process.

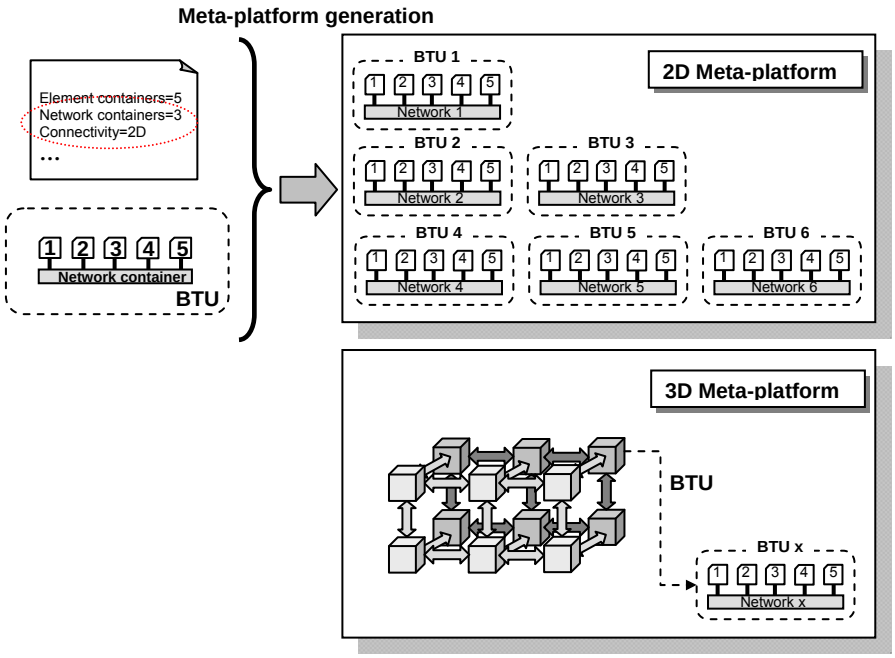


Fig. 3. 6 Example of 2D and 3D meta-platform generation.

Platform template generation

Driven by the exploration at platform level (in Search module), the meta-platform is used to generate topological template instances (as depicted in Fig. 3.7). To this end, the set of strings of feasible platforms is used to instantiate the topological templates from such a meta-platform: each string sets (for one design point) the type(s) and number of NEs in the platform. Moreover, the number of element containers in the platform as well as their

connectivity properties are also determined. Finally, a type classification of the element containers is made. This latter means that for each allocated element container in the BTUs, it is indicated whether it contains a PE or a SE. Note that, as explained in Sections 3.5.2 and 3.5.3, these platforms have been selected by the Search module and checked by Feasibility Checker. The latter repairs strings describing any infeasible topological templates such as, for example, isolated BTUs that do not connect to any other BTU, architectural elements with incorrect connectivity links, and other inconsistencies.

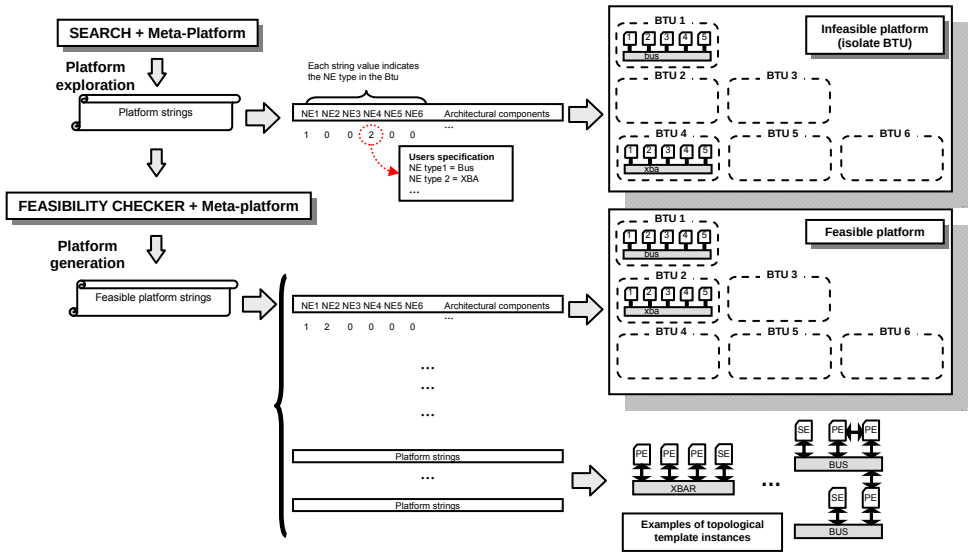


Fig. 3. 7 Platform instance generation and platform string checking.

Example 3.2. During the platform exploration process, the search algorithm could assess infeasible platform strings with isolate BTUs, as shown in Fig. 3.7. In this context, isolate BTU means that a BTU is not connected/linked to other BTUs, and as a result, an incoherent topological template is generated. When these cases are detected by the Feasibility Checker Module, our minimum-distance repair algorithm tries to correct them if possible. In the example shown in Fig. 3.7, both either BTU 4 or BTU 1 can be reallocated in BTU 2 in order to generate a feasible platform template.

Architecture instance generation

Finally, in order to obtain the complete specification of the architecture platform for each design point, the topological templates are further refined. In this process, which is driven

by the exploration at architecture component level, the same topological template can be reused to derive different architecture templates. For this propose, the actual component types of the element containers in a template are added. In the example of Fig. 3.8, this means that, e.g., a PE allocated in an element container either becomes an ARM or MIPS processor, and the SEs either SDRAM or DDRAM. Evidently, all this information is also provided by the strings of feasible architectural components.

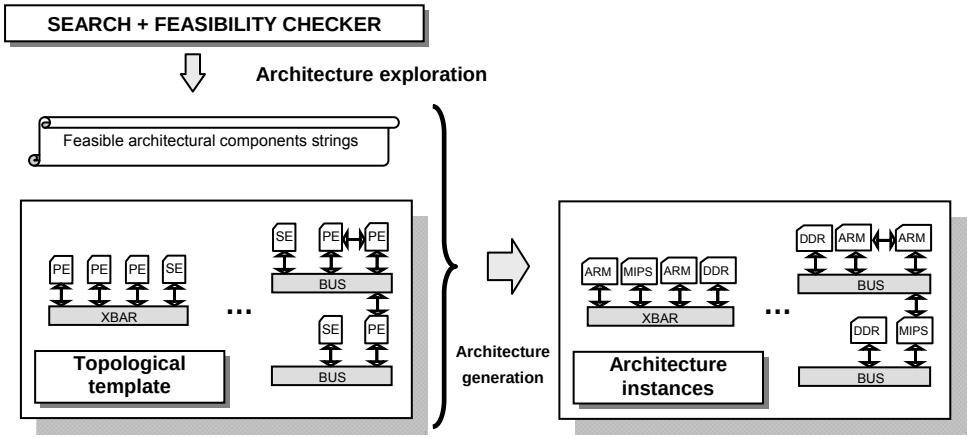


Fig. 3. 8 Example of architecture instance generation process.

3.5.5 Translator

As mentioned before, in order to evaluate selected design points using a system-level simulator that is plugged into the framework, an ESM for each design alternative should be generated first. Such an ESM, composed of an architecture model, an application model and a mapping model, can be provided by the Translator module in an automatic way. To this end, it uses as input the architectural intermediate file (output of the Architectural Platform Generator), the application specifications and the strings that describe feasible mappings, respectively. Thus, the Translator can be considered as the second (and last) stage in the generation process of the simulatable system model.

There exist three relevant benefits in including this module in our framework. First, the Translator converts NASA's internal format of a design point to a file-based format that is specific for the target system-level simulator. Note that in order to integrate a system-level simulator in NASA, it is required that the simulator allows for explicitly describing the design points that need to be simulated using some kind of file format. Second, the resulting ESM should be simulator-specific, i.e., only allowed syntaxes and command lines (for each simulation tool) have to be used. Fig. 3.9 illustrates the conversion to

different architectural models (from a single internal specification) for two system-level simulation tools, CASSE [91] and Sesame [58]. Both tools can be integrated in NASA since they comply with the condition of using a file-based format. However, a particular design point is described in different ways in both simulators: CASSE uses command-line expressions, while Sesame is based on an XML-based format called YML (Y-Chart Modelling Language). Finally, last but not least, the Translator module constitutes itself as an interface (or layer) which separates simulators-dependent and simulators-independent modules, as shown in Fig. 3.2. This implies that the integration of a new system-level simulator in NASA only requires the adaptation of the Translator module, i.e., tailoring the Translator for each different simulator, while all other modules remain unaffected. This adapter function of the Translator again highlights the flexibility and re-usability aspect of our framework.

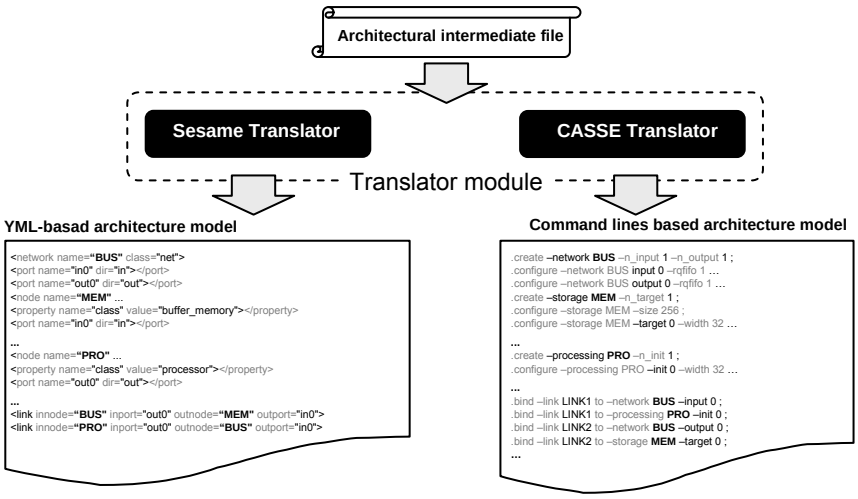


Fig. 3. 9 Plug-in examples for generating architectural model in Sesame and CASSE.

3.5.6 Simulator

At this moment, we have integrated CASSE in NASA, and another system-level simulator called Sesame is in the process of being integrated. Both tools follow a Y-Chart methodology, covering application and architecture modelling, as well as mapping and analysis within a unified simulation environment. Since the implementation of CASE tool has been widely discussed in previous chapters in this thesis, the interested readers are referred to Chapter 2 for more detailed information about CASSE.

3.5.7 Evaluator

During simulations, quantitative information about the system execution (e.g., data about performance, cost/area, and power consumption) can be gathered and dumped into files for later inspection. All these metrics can be used in system-level DSE to find a set of Pareto optimal design points, which then yields a multi-objective optimization problem. The essence of the Evaluator module is to provide this feedback about the quality of a set of evaluated design points to the Search module, influencing the search decisions taken in the exploration process.

Separating the Evaluator from the Search module again provides flexibility and enhanced reusability of the components in NASA. It allows for easily changing the optimization objectives or the function that quantifies the quality of a design point – using the various metrics such as performance, power and cost – without affecting the other components. Such a function is typically referred to as the *fitness function*. The Evaluator also provides the flexibility to, e.g., use a single fitness function for all search algorithms in the Search component (performing exploration at platform, architectural components and mapping levels), or to deploy a different, and possibly tailored, fitness function per search algorithm.

However, when multiple search algorithms and fitness functions are used together, these should be defined in a coherent way with respect to each other in order to avoid conflicting fitness functions and safeguard convergence. This is because there exists a tight connection between the different search algorithms and their respective fitness functions. This connection should be made explicit. In our current implementation, these relations can be defined by a set of hierarchical fitness functions, which can be used with a variant of the one-to-one linking technique (already explained in Section 3.5.2) to address the under-exploration problem in hierarchical design space explorations with multiple search algorithms. Formally, these hierarchical fitness functions are formulated as follows:

$$y_{L_i} = f_L(x_1, x_2, ..., x_k); \quad \forall i = 1..I;$$
$$y_{j_i} = f_j(x_1, x_2, ..., x_k) = \sum_{q=1}^{\delta_j} y_{L_q}; \quad \forall i = 1, \delta_j, 2\delta_j, ..., I \quad \text{and} \quad \forall j \neq L;$$
$$\delta_z > \delta_w; \quad \forall z, w = 1..\beta \quad \text{and} \quad z \supset w;$$

where y_{L_i} is the fitness value of a design point of the lowest-level dimension (the mapping dimension in our case) in the search iteration *i*, *I* is the total number of search iterations,

x_k represents the value of the metric k used in the fitness function f, y_{j_i} is the fitness value of a design point in any dimension other than the lowest one, and δ_j represents the collecting iterations for the individuals of dimension j. Moreover, for a given range of dimensions β, the number of the search iterations needed for collecting fitness information for dimension z (e.g., platform) should be bigger than the number of iterations needed for dimension w (e.g., architecture) if z has a higher abstraction level than w (denoted by the $\supset$ operator).

3.6 Experimental results

In this section, we present a number of DSE experiments to demonstrate the capabilities of our methodology and NASA framework, as well as to illustrate its distinct aspects.

3.6.1 NASA configurations for experiments and parameter settings

The first set of experiments aims at comparing the more traditional approach of using a single search algorithm for all design space dimensions to our dimension-oriented approach (using a separate search algorithm per dimension, i.e., 3 search algorithms (SAs) in total). To properly evaluate and compare the quality of the DSE between different search strategies, we can define three criteria:

- ***Diversity***. A large number of different design points should be explored in each DSE experiment to cover a wide range of design decisions for each dimension.

- ***Convergence***. The strategies should provide approximations to global (or near-to) optimal solutions without being trapped at local optima.

- ***Coverage***. The explored design points should be well-distributed in the design space for a complete view of the trade-off curve or landscape of the design space as well as catching boundary values.

Assessing the quality of the exploration is not equal to assessing the *quality of the obtained design points*. However, an exploration meeting all three criteria should lead to good design points in terms of fitness values (such as good performance). Different parameter settings for the experiments, i.e., different NASA configurations, lead to

different results in DSE quality and in the fitness values obtained. Next, we introduce several different NASA configurations together with the results obtained.

Parameter	Number	Types	Values
PE	≤ 6	3	ARM, PPC, MIPS
SE	≤ 3	2	DDR, SDR
NE	≤ 4	3	Bus, Fully-connected, Customized network
App. Tasks	7	-	(Chapter 2)
App. Channels	12	-	(Chapter 2)
Dimensions (β)	3	-	Platform, architectural components and mapping
Search algorithms (*SA*)	1 or 3	1	Genetic algorithms
GA Selection (*S*)	1	1	Proportional with elitism
GA Crossover (*C*)	1	2	1-point and 2-point
C probability *(pc)*	5	-	[0.1, 0.3, 0.5, 0.8, 1.0]
GA Mutation (*M*)	1	2	Simultaneous (M=1) and Independent (M=6)
M probability *(pm)*	5	-	[0.1, 0.3, 0.5, 0.8, 1.0]
Collecting iterations (δ_{arc})	1	-	2, architectural components dimension
Collecting iterations (δ_{pla})	1	-	4, platform dimension
Search iterations (*I*)	41	-	-
Population size (*N*)	10	-	Number of individuals per iteration
Simulation tool	1	-	CASSE

Table 3. 1 Parameter settings in our experiments.

In Table 3.1, the most important user specifications and parameters for the first set of experiments are listed. The studied MP-SoCs may consist of up to 6 PEs of the types ARM, PowerPC (PPC), or MIPS, up to 3 SE of either single (SDR) or double data-rate (DDR) type, and up to 4 NEs of three types (bus, fully connected, or a customized network consisting of a bus and point-to-point links). The application that is mapped onto the MPSoC is the computer vision algorithm presented in Chapter 2.

3.6.1.1 Search Algorithms settings

With respect to the search algorithm(s) we use for exploration, a multitude of them can be used (via a simple plug-in mechanism): from exhaustive search or random search, to heuristic search methods. We focus on implementations based on genetic algorithms

(GAs), since GA-based DSE has been widely studied in the domain of system-level design [59, 97, 101, 109, 112], and it has been demonstrated to yield good results. In this case, we use a proprietary implementation of the GAs, but any existing GA such as SPEA2 or NSGA-II [59] could also have been used.

3.6.1.2 Population and iteration settings

Typically, a GA works on populations of individuals, J_i, $i=1..N$, where N is called the population size. Each individual encodes a specific design point or design decision of a particular dimension, depending on whether a GA per dimension or a single GA for all dimensions is used in the DSE experiment, respectively. Thus, the GA performs an optimization process that applies the selection, crossover and mutation operators to our string representations that represent individuals. The purpose is to iteratively improve the population by means of combining random walk with a survival-of-the-fittest idea: each individual in the current population is ranked by evaluation of a fitness function that gives a measure how good an individual is in terms of performance, cost and/or power consumption, etc. Better individuals are more likely to survive from generation to generation and new individuals can be either created by mutation (random walk) or crossover of existing individuals. Finally, the GA terminates this optimization process after a certain number (I) of generated populations and outputs the individuals with the best fitness values. In these experiments, N is fixed to 10 individuals, and I varies from 1 to 41 iterations or "generations".

3.6.1.3 Crossover and mutation type settings

The crossover and mutation operators in our GAs are performed at the granularity of entire sub-strings (see Fig. 3.3) in a string that describes the topological platform, architectural components or mapping. These operators are applied according to their associated probabilities (*pc*: probability of crossover, and *pm*: probability of mutation). Further, the GA can perform either a 1-point or a 2-point crossover, and supports two types of mutation. In *simultaneous* mutation (*M=1*), a single random position is simultaneously changed in every sub-string. In *independent* mutation (*M=6*), the mutation probability is used for each of the six sub-strings to determine whether it is mutated or not. In the case that three GAs are used for exploration, different and customized values for the probabilities *pc* and *pm* can be used within each GA.

If all the GA parameters in Table 3.1 are taken into account, a large number of experimental combinations can be performed. From this set of experiments, we present a selection of four NASA configurations. The nomenclature used to denote these configurations is "SAgaCxM", where the meaning of each capital letter is defined in Table

3.1. For example, “3ga1x6” refers to the configuration with 3 GAs that simultaneously explore the platform, architectural components and mapping dimensions, a 1-point crossover, and *independent* mutation (*M=6*).

3.6.1.4 Group of experiments and run-times per simulation

All possible combinations of the *pc* and *pm* values (as listed in Table 3.1) have been evaluated. This results in 25 groups of experiments (5 *pc* probabilities x 5 *pm* probabilities) for each of the four mentioned NASA configurations. Note that, independent of the number of search algorithms used in these experiments (i.e., 1 GA or 3 GAs), a maximum of 410 simulations (41 iterations x 10 individuals per iteration) have been performed for each experiment, where each experiment has also been executed 20 times using different initial populations. As a result, a maximum of 205.000 simulations (25 groups of experiments x 20 different initial populations x 410 simulations) have been performed for each NASA configuration. The CASSE tool – which dominates the run-time of our DSE experiments – requires on average 40 seconds to simulate a single design point on a PC with a Pentium IV processor at 1,6 GHz and 2 GB main memory, running Linux.

3.6.1.5 Fitness functions for evaluation and optimization

In order to simplify the graphic representation of the results and the explanation of the examples in this section, without loss of generality, the fitness value in our experiments only takes a single system metric into account, namely performance. Although we focus on a single system metric in our DSE experiments (due to the limitation of CASSE tool, as mentioned in Chapter 2), we would like to stress, however, that multi-objective optimization can also be perfectly addressed with NASA.

3.6.2 DSE behaviour and sensitivity to various parameter settings

3.6.2.1 Impact of the number of search algorithms on DSE quality

The results of the above experiments are shown in the four scatter-plots of Fig. 3.10, which compare the behaviour of DSE experiments based on a single and multiple GA approach after 10, 20, 30 and 40 iterations, respectively. Each scatter-plot shows the average total of *different* explored design points (i.e., accumulated diversity) on the x-axis and the average of the best fitness values, in terms of processed data packets/s, on the y-axis for each of the experiments.

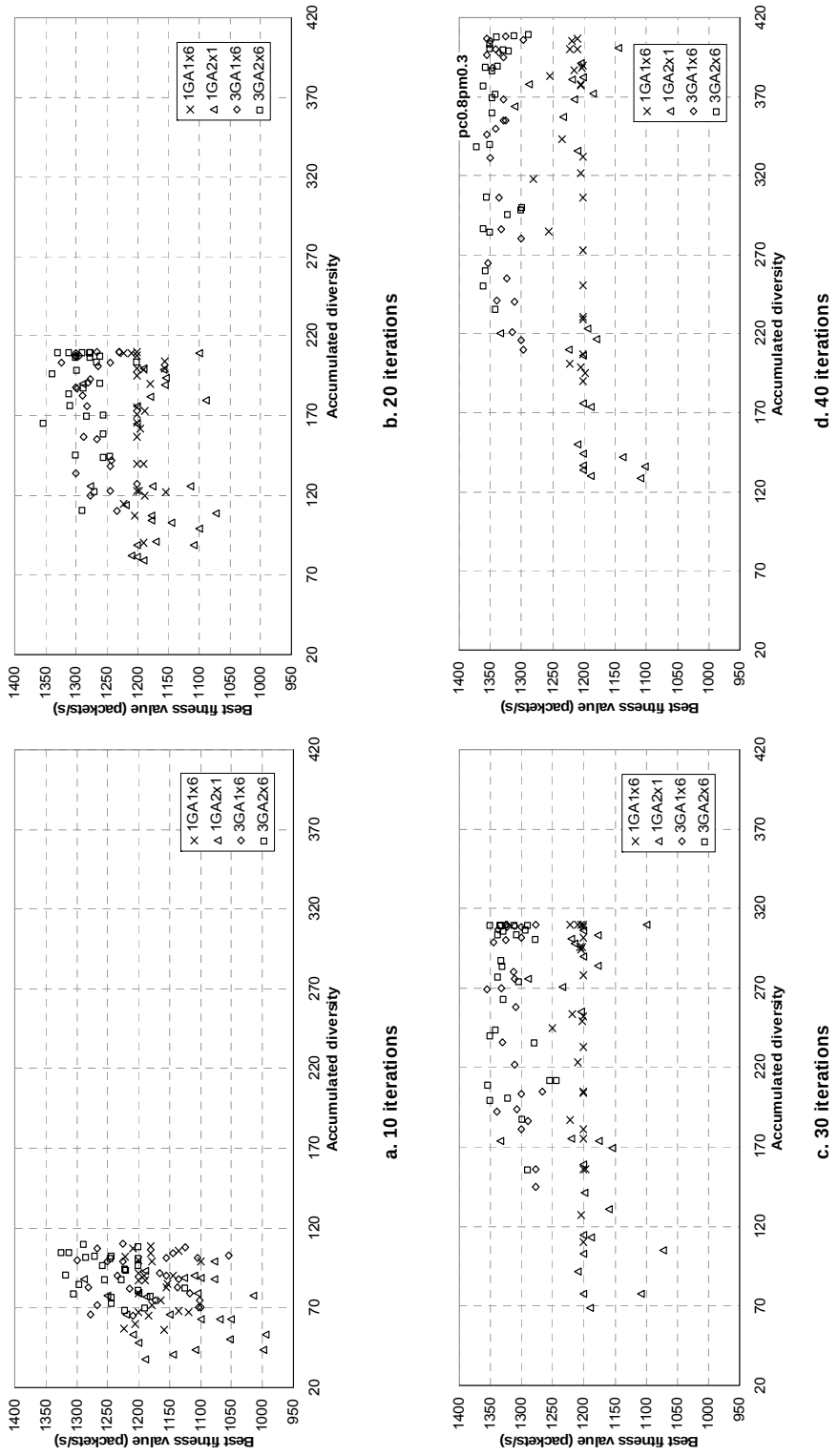


Fig. 3. 10 DSE results for four NASA configurations.

If the input arrival frame rate is 1450 packets/s and a minimum of 1250 packets/s has to be processed to satisfy the minimum real-time requirements of the studied application (which is equivalent to processing 25 frames/s), then using a 3 GA-based search approach in NASA not only provides the design alternatives with the best fitness values (in the upper right corner for each scatter-plot of Fig. 3.10) but the accumulated diversity of the explored design points is also largest. Notice that exploring the same design space with a traditional, single GA approach, optimal and near-to-optimal architectures are less often found. This is mainly due to a smaller accumulated diversity of explored design points. Moreover, it can also be seen that the larger the number of iterations, the larger the gap between traditional single GA-based DSE and our 3GA-based DSE in terms of accumulated diversity and best design points reached.

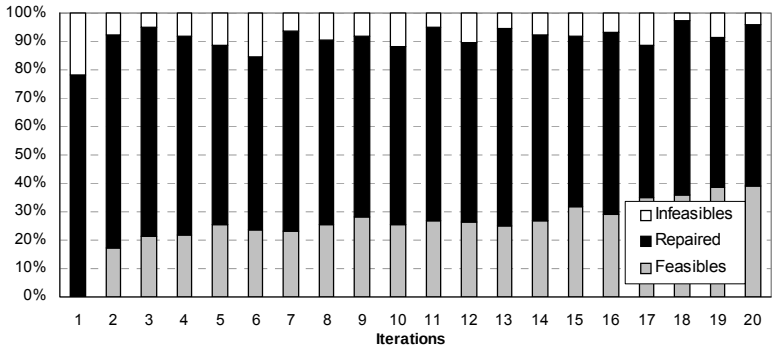
From this, it appears that the multiple GA search has a positive impact on the DSE quality. In other words, while all parameter settings affect the quality criteria of the exploration performed, and consequently the best design points obtained, the multiple GA-based searching seems to be an important factor for achieving quality.

For a detailed comparison between both approaches (single GA and multiple GA search), the three proposed criteria – diversity, convergence and coverage – are separately analyzed in Fig. 3.12, Fig. 3.13 and Fig. 3.14. To this end, we have selected one group of experiments for each of the four mentioned configurations, where all configurations use the values *pc*=0.8 and *pm*=0.3. With these probabilities, the “3ga2x6” configuration finds the design point with the best overall fitness value after 40 iterations (see upper right corner of Fig. 3.10d).

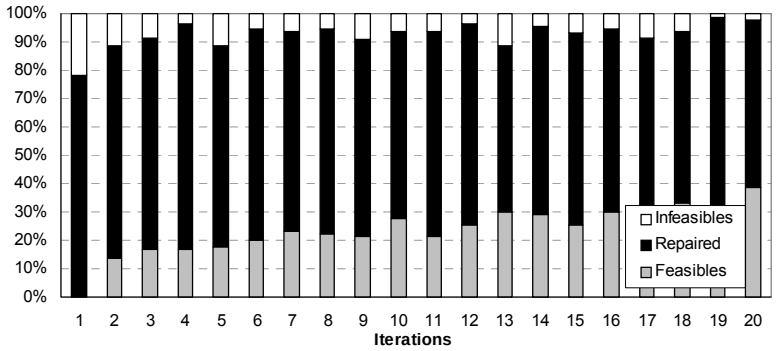
3.6.2.2 Impact of repair mechanisms on the efficiency of DSE process

At this point, it is important to highlight that, although the explored design points shown in our experimental results include both feasible and infeasible design points, only feasible design solutions are evaluated by the CASSE tool. However, due to the repair techniques applied in the Feasibility Checker module, most of the infeasible design points can be detected and converted to feasible ones. As a consequence, the feasible alternatives actually evaluated represent an important percentage of the total number of explored design points. This can be illustrated in Fig. 3.11a and 3.11b, which depict the average percentage of feasible, repaired and infeasible design points per iteration for the DSE experiments based on both approaches (single and multiple GA search). Note that the gray part of each bar (in Fig. 3.11) represents feasible design points without any repair, the dark part refers to repaired design points (i.e., infeasible design points repaired by the

Feasibility Checker module and converted to feasible ones), and the white part indicates the infeasible design points that cannot be repaired by our heuristic minimum-distance repair techniques.



a. 1GA: average percentage of repair: 84.21%.



b. 3GA: average percentage of repair: 86.68%.

Fig. 3. 11 Average percentage of feasible, repaired and infeasible design points per iteration in our DSE experiments.

From these data, it can be seen that our repair techniques can repair more than 84% of detected infeasible design points in each iteration, and as a result, more than 91% of explored design points can be actually evaluated by the CASSE tool. Thus, it seems that the repair mechanisms significantly affect and improve the efficiency of the DSE experiments.

3.6.2.3 Convergence rate and number of iterations

The convergence is illustrated in Fig. 3.12, where the horizontal axis indicates the number of explored design points (and iterations) and the vertical axis represents the fitness values in terms of processed data packets/s. Fitness values come from the used simulator, which is in this case the CASSE tool. Investigating these data, it can be seen

that 1 GA-based experiments have a higher convergence rate (i.e., a steeper slope) than 3 GA-based experiments in the first iterations. This phenomenon is the implicit effect of using the hierarchical fitness functions (explained in Section 3.5.7) and the variant of the one-to-one individual linking technique (presented in Section 3.5.2) in 3 GA-based experiments.

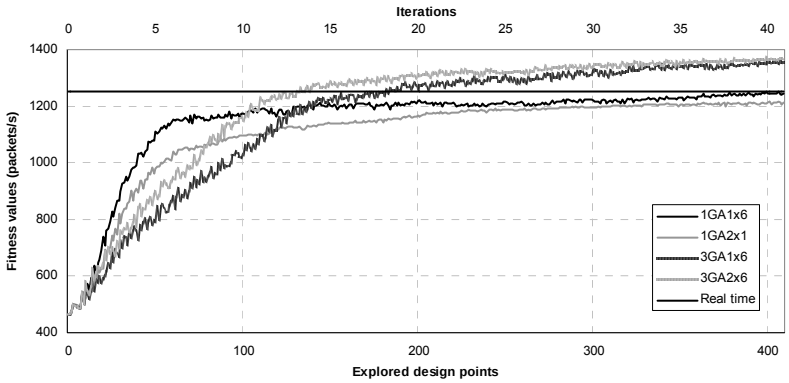


Fig. 3. 12 Average fitness values per iterations.

However, when the number of iterations increases, 3 GA-based experiments do not only gradually and progressively reach higher fitness values than 1 GA-based experiments, but they can also ensure that most of the individuals in each iteration satisfy the real-time restriction (1250 packets/s). In the 1 GA-based experiments, on the other hand, mostly design solutions with fitness values lower than the real-time restriction are reached. Moreover, the 1 GA-based experiment hardly improves or provides better design alternatives with the evolution of iterations. The latter could indicate that the GA is trapped in a local optimum, which occurs when design points explored in each experiment are not sufficiently different or well-distributed (i.e., partial coverage) to properly capture the design space in its entirety (e.g., covering only partially or some regions of the design space), caused by an insufficient variety of new individuals introduced in each iteration (i.e., a low incremental diversity) that prevents the populations to escape from such local optima. These aspects can be demonstrated in both Fig. 3.13 and Fig. 3.14.

3.6.2.4 Diversity and search approach

Each curve in Fig. 3.13 represents the percentage of new and different design points introduced in each iteration that have not been explored in any of previous iterations, i.e., the incremental diversity per iteration. These results highlight that 3 GA-based experiments clearly yield a higher incremental diversity per iteration than 1 GA-based

experiments, and especially in the case of 1 GA with *simultaneous* mutation (M=1). A direct consequence of the latter result thus explains the resulting gap of the accumulated diversity between both approaches, as already shown in Fig. 3.10.

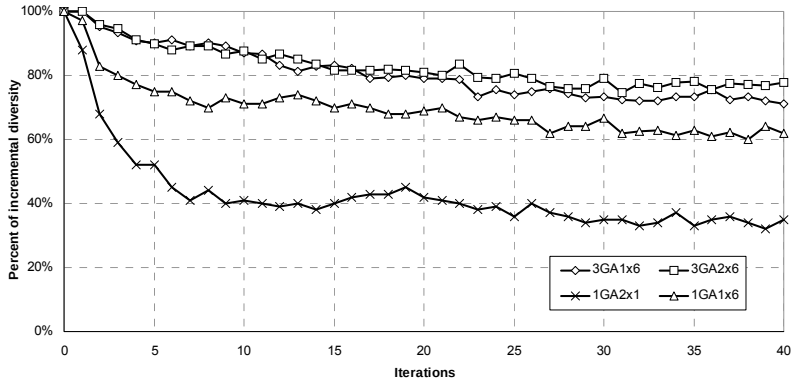


Fig. 3. 13 Incremental diversity per iterations.

3.6.2.5 Convergence, design points concentrations and local optima

All design points explored by each of the selected group of experiments (corresponding to the four mentioned NASA configurations) are separately shown in Fig. 3.14, where each axis represents one design space dimension in our 3D design space, i.e., mapping, architectural components and platforms. Moreover, for a fair comparison, each axis in Fig. 3.14 contains all *ordered* design-decision instance numbers (i.e., the canonical representations of the strings for the platform, architectural components and mappings dimensions) explored together by these four groups of experiments. It can also be seen in Fig. 3.14 that the design points explored in the 3 GA-based experiments are scattered over almost the whole design space (high coverage) and are characterized by a high accumulated diversity. The design solutions reached by the 1 GA-based experiments, on the other hand, have a lower accumulated diversity and are often concentrated in a single region of the explored design space (lower coverage). This indicates that the searching process is converging toward an optimum, and in this last case, toward a local optimum as already shown in Fig. 3.12.

At this point, we also would like to mention that although the size of the design space (approximately 10^{12} alternatives, as mentioned in Section 3) is much bigger than the number of individuals actually evaluated (maximum 410) in these experiments, the results in Fig. 3.12 and Fig. 3.14 clearly indicate that GA-based DSE in our experiments converge to (local or global) optima after only a relatively small number of iterations.

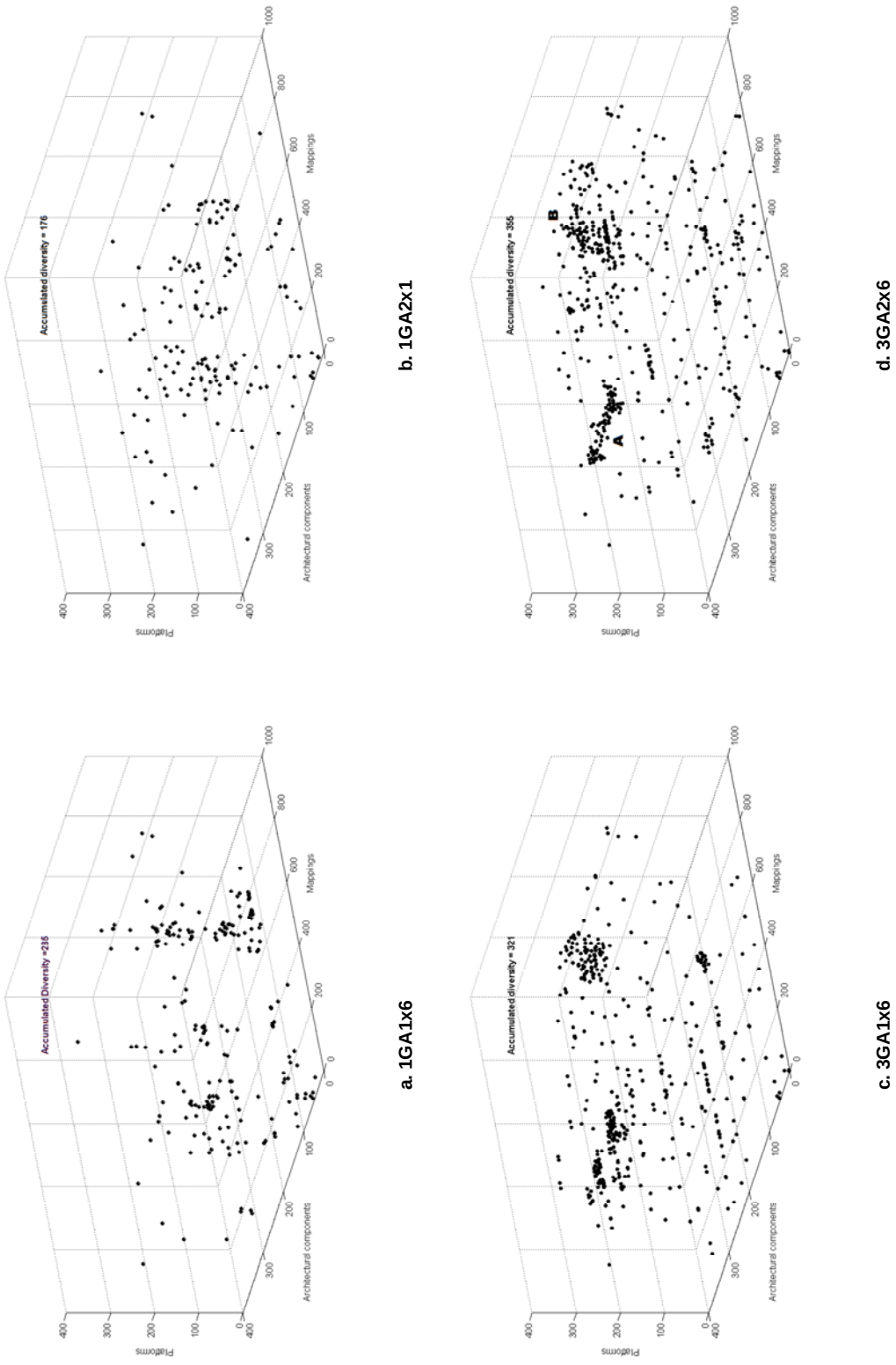
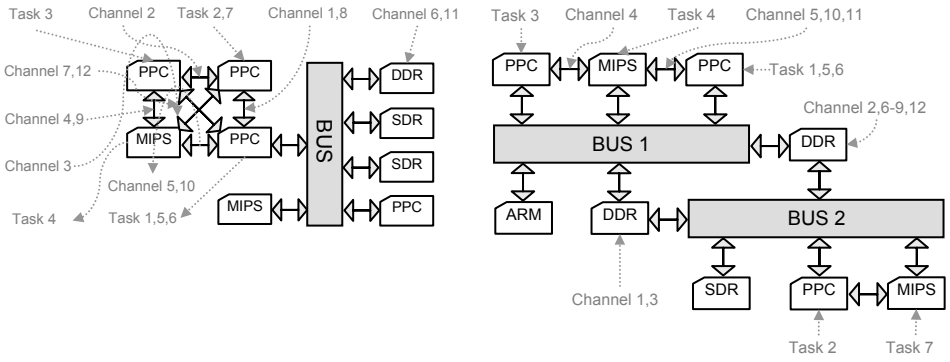


Fig. 3. 14 Explored design points by each selected NASA configuration.

3.6.2.6 The need for refinement

It should be noted that *design points concentration* – a visual indicator of the convergence process – can also be observed in the 3 GA-based experiments. But, unlike the 1 GA-based experiments, the convergence is toward a global optimum or toward a few optimal points. The existence of *several optimal points* can be illustrated for two NASA configurations based on multiple GAs shown in Fig. 3.14, where more than one design points concentration (or convergence) area can be identified. This is correct since different alternatives can often satisfy a given set of user restrictions. To illustrate the above, two design points (A and B) have been marked in Fig. 3.14d, and their respective architectures and mappings are shown in Fig. 3.15.



a. Design point A: 1371 packets/s. **b. Design point B: 1355 packets/s.**

Fig. 3. 15 Examples of design points found by DSE with pc=0.8, pm=0.3 after 40 iterations.

In this case, although both solutions (corresponding to each of the design points convergence regions) have similar performance (A achieving 1371 packets/sec and B 1355 packets/sec), their underlying platform architectures are however quite different. Moreover, they are also over-dimensioned in the sense that not all resources are actually used by the application. Therefore, in this case, designers can perform an additional optimization process in terms of architectural components and/or in terms of mapping. Such *refined* optimization can be performed in a next and more detailed phase of exploration experiments where, e.g., the platform is fixed and only the architectural components and mapping dimensions are explored more rigorously. Alternatively, additional objectives or fitness functions (such as the cost of designs) can also be taken into account in the optimization process. In the next section, we will further investigate refinement by fixing one dimension (platforms) and then conducting DSE in just a two dimensional space (architecture components and mapping).

3.6.3 Hierarchical refinement and analysis of 2D-DSE with NASA

Our second set of experiments presented in this chapter aims at demonstrating NASA's flexibility and capacity to perform the aforementioned refinement process. To this end, this set of experiments is focused on 2D design space explorations, where design decisions about mapping and architectural components are explored for a particular platform template, i.e., the platform dimension is fixed and no search algorithm is used in this dimension. Obviously, in order to model a specific platform template, designers should properly configure the platform string values for the number and types of element containers in each instantiated BTU as well as their connections with each other.

GA	Selection (S)		Proportional with etilism
	Crossover (C)		1-point, *pc* = 0.5
	Mutation (M)		Independent (M=6), *pm* = 0.5
ga	Selection (S)		Tournament without etilism
	Crossover (C)		2-point, *pc* = 0.8
	Mutation (M)		Simultaneous (M=1), *pm*= 0.3
Collecting iterations (δ_{arc})		2	architectural components dimension
Search iterations (I)		21	-
Population size (N)		10	Nr. of individuals per iteration
PE		≤ 6	ARM and hardware dedicated block
SE		≤ 3	DDR and SDR

Table 3. 2 Search module and target architecture parameter settings.

3.6.3.1 Fixed platform, variable architectural components and mapping

The selected target platform template and available type values for PE and SE are depicted in Fig. 3.16 and Table 3.2. This platform template can provide architecture models based on two AMBA buses connecting up to six PEs and three SEs. The execution time of the visual tracking application's tasks has been estimated using ADS, while we assume that the hardware dedicated block (which executes block matching operations of the target application) has a x10 speedup factor with respect to the SW implementation. Note that in this case study, plenty of platforms could have been analyzed in our refinement experiment. However, we believe that we selected a realistic MP-SoC platform template, consisting of several homogeneous processors completed

with a few coprocessors or hardware dedicated blocks in a bus-based architecture, rather than an MP-SoC based on various PE and NE types having different computational and communication characteristics.

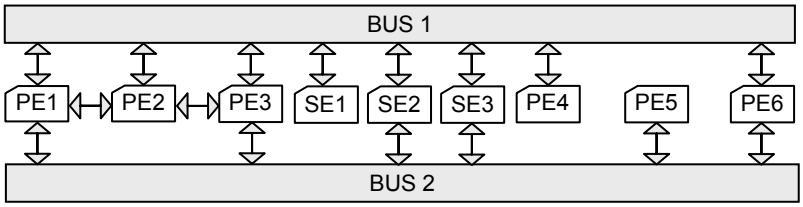


Fig. 3. 16 Target platform template for the second set of experiments.

3.6.3.2 2D NASA configuration for DSE

Three NASA configurations have been selected in this second set of experiments: 1GA+1Random, 1GA+1ga and 1GA+1GA. The used parameter settings of the genetic algorithms are illustrated in Table 3.2. For example, 1GA+1GA (or 1GA+1ga) refers to two identical (or different) genetic algorithms are used in the architectural components and mapping dimensions, respectively. On the other hand, in the cases of 1GA+1Random, a GA explores different architecture instances by varying the type of SEs as well as the location of the hardware dedicated block in different PE containers of the platform template (since the rest of PE share the same processor type), while a random search algorithm explores different functionality distributions onto system resources in a random fashion. It should be noted that although not included in this set of experiments, an extensive number of combinations of different search algorithms as well as GA parameters could have been used (as already shown in Fig. 3.10 for our first set of experiments). Therefore, the three selected configurations only represent a few samples of NASA's capacity and flexibility.

3.6.3.3 Results and discussion

The results of the above three configurations are shown in Fig. 3.17. The curves show for each of the configurations (i.e., 1GA+1GA, 1GA+1ga and 1GA+1Random) the average fitness values (obtained with twenty different sets of initial populations) reached by all individuals in each of the twenty iterations. From these results, it can be seen that 1GA+1Random can only sporadically reach a few design points that satisfy the real-time constraints. Moreover, it clearly cannot ensure convergence toward any global optimum. On the other hand, although both experiments based on two genetic algorithms can provide solutions that satisfy the input constraints, 1GA+1ga only needs to simulate an average number of 40 individuals before reaching the first individual that satisfies the

real-time constraint, while 1GA+1GA need to simulate an average number of 60 individuals. This may suggest that more efficient GA-based DSE experiments can be achieved by selecting and configuring appropriately GA parameters.

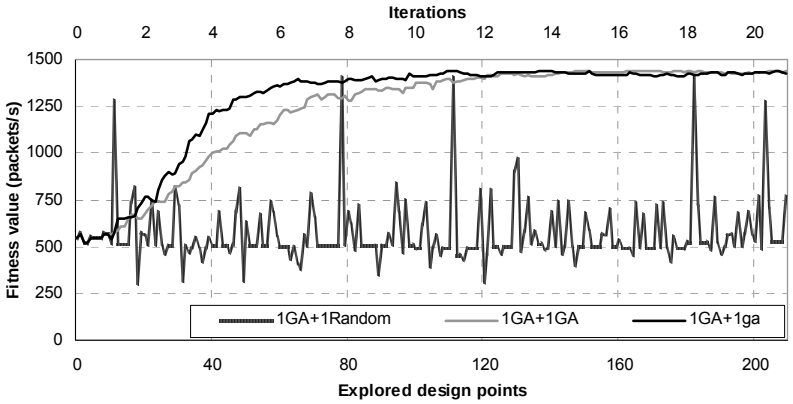


Fig. 3. 17 Comparative results obtained in the second set of DSE experiments.

Finally, we would like to highlight another important benefit of our framework. NASA provides not only information about the best solutions but also about all other explored design points in each experiment. This is a key element for better understanding the studied design space, i.e., the more design points are provided to the designer, the more information can be extracted from the explored design space, and therefore, it will allow designers to more easily compare the architectural characteristics of the evaluated design points. That is, it can be very useful for a designer to distinguish the architectural similarities of the design alternatives featuring good fitness values.

This last aspect can be illustrated in Fig. 3.18, which shows an example of typical NASA output after each DSE experiment. The set of simulated design points, corresponding to a 1GA+1ga experiment in this case, can form a surface that approximates the landscape of the explored design space. A 2D view of the resulting surface is shown for this experiment since it is based on 2D exploration, i.e., the x axes and y axes of the 2D view contain the explored instance numbers of the mapping and architectural components dimensions, having fixed the platform as mentioned. So, for example, 120 different mappings have been explored in this example. Note that the fitness value associated to each design point is color coded, ranging from red (high fitness value) to blue (low fitness value). Therefore, even when exploring a relatively small number of design points, the distribution of design points in the surface can clearly indicate the location of convergence region(s), while dark red areas can provide a good insight of where the

sweet spots (design points with higher fitness values) in the design space are located. Moreover, designers can select any design point of the surface, and examine information about that design point such as the parameter values for the mapping and/or architectural components dimensions. Finally, it should be stressed that all these experiments presented in this chapter have been performed in a fully automatic fashion, only providing parameter settings and constraints such as those shown in Table 3.1 and Table 3.2.

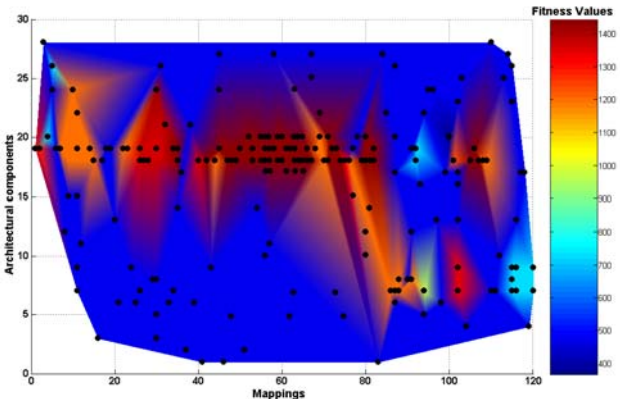


Fig. 3. 18 Example of NASA output.

3.7 Conclusions

In this chapter, we addressed the lack of a generic, flexible, and re-usable infrastructure to facilitate and support system-level MPSoC design space exploration (DSE) experiments. To this end, we have presented a system-level MPSoC DSE support infrastructure, called NASA. This highly modular framework uses well-defined interfaces to easily integrate different system-level simulation tools as well as different combinations of search strategies in a simple plug-and-play fashion. Moreover, we also described a new methodology aiming at multidimensional DSE, dimension-oriented DSE approach, which allows designers to configure the appropriate number of, possibly different and tailored, search algorithms to simultaneously co-explore the various design space dimensions. The result of both contributions is a flexible and re-usable framework/methodology for the systematic exploration of the multidimensional MP-SoC design space, starting from just a set of relatively simple user specifications.

In order to demonstrate distinct aspects of our approaches, we have presented a number of DSE experiments in this chapter. First, we compared NASA configurations using a

single search algorithm for all design space dimensions to configurations using a separate search algorithm per dimension. Here, we focused on search algorithms that are based on genetic algorithms (GAs). The experiments have shown that, compared to the more traditional approach of using a single search algorithm for all dimensions, the multidimensional co-exploration seems to be able to find better design points and ensure the convergence toward global optima. Furthermore, the multidimensional co-exploration has a higher diversity and coverage of design alternatives, producing higher quality DSE results. Finally, we have also illustrated NASA's capability and flexibility to integrate different kinds of search algorithms in DSE experiments.

However, in spite of the results presented in this chapter, there are still three aspects that must be proven for our proposed framework/methodology. First, we should have to demonstrate the capability of NASA to integrate other kinds of search algorithms. Second, additional deployment case studies should be carried out to illustrate that our approaches can also conduct multi-objective optimization problems. Finally, since the high total run-time is still being the main drawback of the DSE based on simulations, other exploration strategy alternative should be provided in order to improve the DSE efficiency (in term of total run-time) without losing accuracy in the evaluation. All these challenges will be addressed in the next chapter.

Chapter 4

Strategy and algorithms for mapping real-time application onto MPSoC

The main goal of this chapter is twofold. On one hand, we present a hierarchical DSE methodology for addressing the problem of mapping real-time applications onto MPSoC platforms, while multiple design objectives are taken into account. On the other hand, we configure the NASA framework to perform several DSE experiments following a novel hierarchical methodology. These experiments are aimed to validate the flexibility and capability of NASA framework to plug in different kinds of search techniques as well as to address multiobjective optimization problems. Finally, we compare the efficiency and quality of the results reached by our hierarchical DSE approach with the results obtained by other traditional DSE strategies.

4.1.1 Complexity of the application mapping on MPSoC

As pointed out in Chapter 1, the increasing levels of integration are leading to more complex embedded systems on chip, which can contain multiple SEs, NEs and a variety of PEs. In this context, the MPSoC platforms are emerging as an interesting solution in the development of modern SoCs, since they can provide a flexible and re-usable architecture to support different product versions (or family of products), and an architecture that can be easily modified in response to market needs, users requirements and product updates during the design and product life cycle.

However, even working with a fixed platform template, there still exist significant complexity and effort for the system designers to map an application onto such MPSoC. For example, in order to efficiently use these MPSoCs, the designers not only have to face the traditional mapping decisions such as HW/SW partitioning and resource usage policies, but also they need to explore design decisions[4] about the number and type of each architectural component, as well as the allocation of resource instances on the target architecture. That is, designers are now not only concerned with deciding which functionality of the application should be processed on which PE, but also with making decisions about where to produce and store the data. As a consequence, it is not too surprising that, this problem of the application mapping onto MPSoC turns out to be *NP-hard* [113], as in order to find the optimal mapping under a set of user constraints (e.g., power consumption, cost/area and real-time constraint), a vast number of design decisions should be explored.

4.1.2 Multi-objective optimization problem

Besides the large amount of design decisions that should be considered during the early design stages, the system designers also have to cope with contradictory design constraints. On one hand, many of the modern SoCs often target mass production and battery-based devices, and therefore should be cheap (in term of cost/area) and power efficient. On the other hand, they still need to show high (e.g., real-time) performance, and often support multiple applications and standards which requires high programmability. Basically, these design objectives can be roughly separated in two categories/leveles: primary and secondary objectives.

[4] All these design decisions can be clustered into design space dimensions, as explained in Chapter 3.

4.1.2.1 Primary objectives

Primary objectives concern the properties of the overall system and are typically used directly as optimization goal, i.e., they do not represent an intermediate, supportive cost metric, but drive the optimization process directly. The objectives listed in this category can universally be applied to DSE and are not domain-specific.

- **Cost**: the cost of a design could be measured as the sum of all component costs integrated in the system, based on the wholesale prices from different component manufacturers or based on the own manufacturing costs, which could be determined by the area consumption in the target technology and the packaging costs. Fixed costs, such as the fabrication costs of mask sets or the engineering costs for designing the system, cannot drive the optimization process and are often not included.

- **Power dissipation**: optimization for power more and more becomes the focus for DSE. On the one hand, high-end systems optimized for speed have to cope with the generation and flow of heat within the system that degrades the life-time of the components. On other hand, embedded systems not only focus on the minimization of the worst-case power dissipation, but also on the power leakage during idle periods of the system in order to decrease the costs for maintenance by extending the life-time of batteries.

- **Performance**: the speed of a design can be expressed by different metrics, such as the throughput achieved for computations and communications, the overall amount of data processed or transferred, the latency/response time for certain events (whether deadlines are met), the period length of a schedule of computations and communications, or the bare clock speed supported by the design.

4.1.2.2 Secondary objectives

Metrics in this category are either focused on the properties of only a part of the overall design or provide supportive information on the design, i.e., they reveal characteristics of the design that influence primary goals. The utilization of resources, for instance, could be seen as one component of the primary cost or power dissipation objectives. Secondary objectives are often problem-specific and facilitate the analysis of the overall design, pointing the designer to bottlenecks of the design. Examples of the common secondary objectives are listed below.

- **Utilization of computation resources**: the utilization of a resource determines the fraction of the overall execution time of a PE during which the resource is busy. Depending on the primary goal and the application domain, the goal of the optimization could be to maximize the utilization in order to exploit the silicon area as much as possible. That is, the system designers should optimize the efficiency of the resource usage avoiding over-dimensioned (or over-designed) architecture, i.e., a design solution is better than other ones if it achieves higher performance at the same cost/area or the same performance at a lower cost/area.

- **Load balancing in processing resources**: load balancing avoids overloading certain resources which would lead to excessive packet delays in terms of the communication paths and excessive processing delays at computation resources. Moreover, as demonstrated in Section 2.4.2.2, a load-unbalanced design could also generate additional and unproductive traffics (or access to shared resources) in the communication architecture due to the over-synchronization effects. Therefore, this latter influences heavily the power consumption.

- **Communication-specific metric**: optimizing power consumption of SEs and NEs consists of minimizing the number of memory accesses and/or minimizing the system load in NEs [114], for instance. Clustering tightly coupled application functions into the same processing resource reduces the SEs accesses (and power consumption), and often increases the SoC performance. Moreover, reducing the amount of data exchanged via shared SEs also optimizes the SE sizes, and therefore, cost/area of the design.

4.1.3 Existing strategies for exploring the design space of mapping

In order to cope with the aforementioned complexity of the mapping problem while considering multiple design goals, a significant number of approaches has been proposed during the last decades [35, 46, 92, 98, 99, 103, 107, 115, 116], which are often based on analytical methods and/or simulations.

The analytical model is suitable to explore large design space, because its small run-time characteristic enables it clearly to find the optimal mapping in a reasonable amount of time. However, it also suffers from two drawbacks. First, most of these approaches only focus on the importance of a subset of design variables in their analytical model, while

they assume fixed values for other design decisions and/or do not model some design parameters that form the axes of the design space. As a consequence, the resulting analytical model is often inflexible and hard to re-use for different studied cases. Second, analytical model often fail to consider *non-deterministic* system behaviour, such as unpredictable arbitration delay of communication architectures due to simultaneous access request by multiple competing PEs, which may lead to over-designed or sub-optimal solutions in the practice. As a result, such *non-linear* system behaviours make an accurate performance evaluation difficult without the use of simulation.

On the other hand, the simulators can take into account the simultaneous interactions of a huge number of design choices in each simulation (as illustrated in previous chapters), therefore evaluating accurately the dynamic system behaviours and providing detailed system metrics such as communication latency, performance, etc. Unfortunately, it is impractical to use these simulators to explore a large design space due to (i) prohibitively high simulation time, and (ii) high set-up effort for each new architecture and mapping

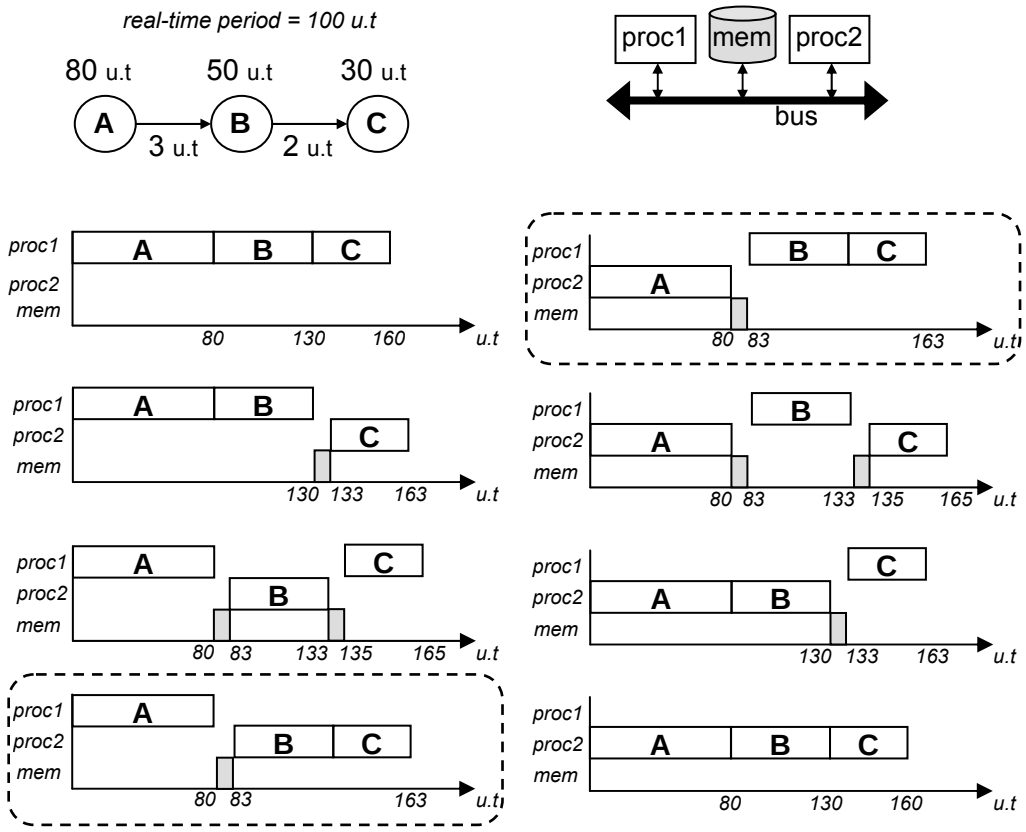


Fig. 4. 1 Example of pruning the design space of mapping.

model creation. For example, suppose that an application consisting of 7 tasks is mapped onto an architecture with 6 PEs. Only considering the design decision about the distribution of application's functionality on different PEs, it results in $6^7 = 279936$ alternatives. Thus, even for the most efficient high level simulation tool (in order of 2 seconds to simulate a single design point), it will require 6.48 days to evaluate all possible designs. As a consequence, the simulation tools are often coupled with some kind of search methods like evolutionary algorithms, simulated annealing, or ant colony algorithms, as these search algorithms only need to visit a limited number of design points to provide a convergence path toward an optimal solution, as is already demonstrated for GAs in Chapter 3. However, in spite of the efficiency achieved in these results, it should be noticed the total simulation time (or total run time) of each our DSE experiments is still in order of several hours for a moderate population size and number of iterations. That is, as explained in Section 3.6.1.4, a maximum of 410 simulations are performed for each DSE experiment, where CASSE requires on average 40 seconds to simulate a single design point in those cases. As a result, each DSE experiment requires 4.5 hours (41 iterations x 10 individuals per interation x 40 seconds per simulation).

Example 4.1. Fig. 4.1 depicts a real-time application consisting of 3 tasks and 2 channels (with their associated task execution times and inter-task communication times), and a set of Gant-charts for visualizing all its possible mappings on a target architecture. In this (fairly simple) example, we can realize that the real-time constraint only can be achieved by two design points (enclosed in the dotted line), which represent only the 25% of the design space. That is, a design space is often composed with an important number of the design points that cannot satisfy clearly the design goals. However, simulations are not needed to evaluate such design points, since an analytical estimation can determine their performance, and thus reducing the number of the design points that are worth of a more accurate simulation. This process of reducing the design space is known as pruning the design space, and it can improve significantly the efficiency of the DSE process, as will be illustrated in the next sections of this chapter.

4.1.4 Hierarchical DSE methodology

Based on the former analysis, it seems that none of both approaches is suitable to explore large design space of the mappings, since the analytical methods are not sufficiently accurate and the simulations are too time-consuming. Therefore, in order to improve the efficiency of system-level DSE, this chapter presents a hybrid DSE

methodology. Our approach is composed of two independent phases (analytical estimation and system simulation) communicating via a file-base interface, such that the resulting methodology is capable of combining the benefits of both analytical models and simulation tools (i.e., *speed* and *accuracy*).

In the first phase, a large design space is pruned. To this end, we have developed a set of analytical methods, which considers both deterministic and dynamic system behaviours, to rapidly explore and evaluate design points (in terms of multiple design objectives), as well as to eliminate those design points that cannot satisfy the user requirements. The output of this pruning process is a reduced number of mapping alternatives, which are evaluated accurately in the second phase by a system-level simulation tool in order to find the optimal mapping solution. The overview and implementation details of our hierarchical DSE methodology will be discussed in Section 4.4 and Section 4.5, respectively.

Finally, in order to demonstrate the benefits of our hierarchical DSE methodology, we have plugged the proposed analytical methods into NASA framework, performing thus several experiments following our hierarchical DSE methodology. The obtained experimental results will be illustrated and discussed in Section 4.6. This latter will allow us to validate and prove the flexibility and capacity of NASA framework to integrate different kind of search techniques, as well as to address and solve the multi-objective optimization problem, as claimed in Chapter 3.

4.2 Related work

In the literature, there exist numerous efforts on DSE aiming to address the problem of the application mapping on MPSoC [94, 99, 103, 107, 109, 116, 117]. In [103], a two-phase approach was proposed for the mapping of a real-time application on a homogenous multiprocessor architecture. Basically, they use a set of heuristic-based algorithms to explore distinct alternatives about both task binding and assignment on the architectural components, while an analytical estimation model was applied for the performance evaluation. Kim et al. [98] also proposed a framework based entirely on analytical methods for the DSE. In this case, they focused on the exploration of a design space associated to the HW/SW partitioning decision, as well as the choice of an appropriate scheduling policy for each PE in order to ensure that the application timing requirements are met. Unlike our approach, data dependencies and communication

overhead are ignored in [98], since they assume an ideal buffering, i.e., the number of messages that can simultaneously circulate on the system is not bounded. And in the case of [103], a simplified communication architecture model at the high level of abstraction is used to estimate the system traffic cost, assuming that a PE can send and receive any number of tokens concurrently, while the communication delay is simply proportional to the amount of communication data and user specified latency parameters for each component. From our point of view, their assumptions are not realistic enough for modelling modern embedded system, and more advanced models are required.

In order to overcome this lack of realism on the modelling of communication operations, analytical approaches considering the dynamic behaviour or resources sharing problem have been proposed [107] [115]. Basically, they propose an analytical model for each kind of architectural components that can be later composed to capture and analyze the complete system. However these authors also claim that, in spite that their models can achieve a good level of accuracy (with a reasonable estimation error), it is not easy to combine such models for large systems evaluation yet.

On the other hand, system-level simulation tool represents a natural alternative for a more accurate performance evaluation, and/or the analysis of more complex design problems in larger systems. In [116], Sesame [58] was coupled with genetic algorithms (provided by PISA framework [109] [111]) to address the mapping of multiple applications (with different timing requirements) executing concurrently on the same MPSoC. While in [99], CASSE was used in the DSE experiments aiming to find out the optimal mapping of a target application on a MPSoC that works with multiple clock domains. Lahiri et al. [35] also combined PTOLEMY [62, 118] with an iterative algorithm to determine the well-optimized configuration of arbitration schemes for the different NEs on the architecture, such that the system performance is maximized. However, in spite of achieving the desired solutions with a relatively low number of simulations, a high total run time (i.e., in order of hours) of each DSE experiment is still being the common denominator in their works.

Finally, and to the best of our knowledge, only a few approaches combining both analytical estimation model and simulation in a single framework have been proposed. One of them is the hierarchical DSE methodology in [46], which has a similar objective as the approach presented in this chapter. Their approach initially uses symbolic constraint satisfaction method to rapidly explore and prune a large design space. Subsequently, the remaining set of alternatives is individually evaluated using trace-driven simulations and

incorporating more design details (e.g., processor idle time, reconfiguration overheads, energy dissipation, available parallelism, etc). And then, detailed simulations are performed using low-level simulators to select the appropriate design solution. Unlike our methodology, the impact of the resources placement in the architecture, as well as the contention effects on the shared communication architecture are not explicitly explored.

Lee et al. [92] present a framework that determines the MPSoCs for the optimal mapping of a given real-time application, i.e., satisfying the real-time constraint while the system cost is minimized. This two-phase approach selects first an optimal set of PEs for the mapping of the target application. Then, by means of a static estimation method based on the queuing model, they explore and prune the design space of communication architectures. And finally, their framework provides a set of interfaces that facilitate the integration of different simulation tools, which can be used to accurately evaluate each solution in the reduced design space, and determine thus the Pareto-optimal set of design points achieving the user constraints. Unlike our approach, they address the DSE as a high-level synthesis problem. Moreover, their queuing model is limited to bus-based topology, while our analytical estimation model is not restricted to a particular architecture, but it is flexible enough to analyze different kind of communication architectures.

4.3 Preliminaries and problem statement

In this section, we define the concepts and key assumptions underlying our approach, and follow-up with a formal problem statement.

Definition 1. *Application model*. A real-time application is modelled as a task graph $TG=\{T, L, D\}$, where $T=\{t_1, t_2, \ldots, t_k\}$ is a set of k periodic tasks that must be executed in a certain order to produce the desired results under a real-time constraint *RTC* (expressed in seconds), $L=\{l_{12}, l_{13}, \ldots l_{jk}\}$ represents a set of h unidirectional channels or links that communicate the tasks with each other as well as indicate their data dependencies, and $D=\{d_{12}, d_{13}, \ldots d_{jk}\}$ specifies the amount of shared data associated to each link. Finally, a task without any input link is called *root*.

Assumption 1. We assume that the application partitioning is done at the functional block granularity, i.e., the basic unit for partitioning (or each task) is a function. Although a

finer partitioning could be done to exploit parallelism at algorithmic level, this latter is not addressed in this thesis, as pointed out in Chapter 2.

Definition 2. *Architecture template*. In this chapter, we discuss our hierarchical DSE methodology in a context of the architectures composed respectively with *p, m* and *n* number of PEs, SEs and NEs, where the connections between different components as well as the architectural topology (defined by the number and type of NEs) is already fixed. Since such description can be represented by several possible instantiations, we focus specifically on the templates consisting of several homogeneous RISC processors and shared SEs, completed with a few hardware dedicated blocks, in a bus-based architecture. As a consequence, different architecture instances can be derived from the same template by varying the number and type of PEs and SEs, as well as their allocation on the architecture.

Assumption 2. We assume that there is a library of configurable resource models for PE, SE and NE, which can be instantiated and configured properly to build the architecture templates mentioned in Definition 2. These components and their parameters of configuration (such as read/write latency, operating frequency, storage capacity, network arbitration policy, and so on) can also be used by the system designer in both estimation (or pruning) phase and simulation phase. For example, using such parameters, a PE can be configured as a generic RISC processor for flexible application support as well as dedicated hardware block for achieving specific performance, while setting appropriately the access latency and storage capacity parameters a SE can behave as a fast cache memory of small storage capacity or memory bank of large storage capacity but high access latency.

Definition 3. *Mapping*. It consists of the process of distributing the application functionality on the available resources of the target architecture. This process implies to address three sub-problems: (i) partitioning: refers to the selection of a suitable PE type for each application's task, (ii) assignment: decides the type and number of instances for PEs and SEs, their locations in the architecture template, as well as which task and channel should be assigned to which component instance (if more than one instance of a component type is present in the architecture), and (iii) scheduling: defines the order of execution for the different tasks assigned in a PE.

Assumption 3. We assume that all application's tasks are assigned to a set of PEs working in a pipeline fashion, where each PE can execute one task at a time, and task

duplication and migration are not considered. On the other hand, the inter-tasks synchronizations and communications can be assigned to (i) the SE: when two communicating tasks have been assigned to different PEs, the corresponding inter-tasks channels should be assigned to a shared SE accessible by such two PEs, and therefore a communication delay occurs, or (ii) inside the PE: if two communicating tasks (and their associated inter-tasks channels) reside on the same processor, it then is reasonable to neglect this communication time, without loss of generality.

Definition 4. *Communication delay or time*. The time (in seconds) taken for a communication transaction between a PE and SE is given by the sum of three parts: (i) *protocol delay* (cpd): latency associated with communication protocol, (ii) *contention delay* (ccd): waiting time due to simultaneous access attempts to the shared resources, and (iii) *SE access delay* (rwd): time taken to read/write the data from/to a SE.

Definition 5. *Computation delay or time*. It is the execution time (in seconds) of an application's task on a specific PE. Thus, all task's timing information in the *pt* available types of PE (provided by component library) should de first determined, i.e., each task has a set of computation time $\{wcet(ti)_1, wcet(ti)_2, \ldots, wcet(ti)_{pt}\}$, where $wcet(ti)_j$ is the worst-case execution time of task t_i on the PE type j. To measure the execution time of a task on a general purpose processor (e.g., ARM), we use an instruction set simulator (e.g., ARM suite developer). Note that the execution time of a specific task on a dedicated hardware implementation is assumed given, while this hardware block takes an infinite amount of time to execute any of the remaining application tasks. As a result, a profile table with *pt* rows and *k* columns can be obtained.

Assumption 4. We assume that the time required in the communication transactions is much smaller than the execution time. Otherwise, we consider that there is not possible benefit for a parallel implementation of the application in a multiprocessors architecture.

Problem statement. Given a real-time application (TG), an architecture template and a component library, our problem is to find the optimal architecture instance for the mapping of the target application, satisfying the *RTC* and achieving additionally a good triple trade-off among the efficiency in PE usage, PE load balancing and inter-PEs communication traffic minimization. Formally, the optimal mapping solution should satisfy the following design objectives:

Primary objective

The performance of the real-time application mapped on the target MPSoC platform should achieve the RTC.

Secondary objectives

- **Objective 1.** Minimize the number of resources used in the mapping solution, i.e.:

$$Min \{NPE \text{ and } NSE\} \tag{4.1}$$

where NPE and NSE represent respectively the number of PEs and SEs actually used in the MPSoC platform for mapping the real-time application.

- **Objective 2.** Maximize the efficiency of PEs usage (EPE), i.e.:

$$Max \{EPE\} \tag{4.2}$$

$$EPE = \frac{\sum_{i \in NPE} CMP_i}{NPE} * 100 \tag{4.3}$$

where CMP_i is the total computation time due to the tasks assigned on PEi.

- **Objective 3.** Minimize the load unbalancing in PEs (LuB), i.e.:

$$Min \{LuB\} \tag{4.4}$$

$$LuB = \frac{\sum_{i \in NPE} abs(CMP_i - EPE)}{NPE} * 100 \tag{4.5}$$

- **Objective 4.** Minimize the inter-PEs traffic (IPT), i.e.:

$$Min \{IPT\} \tag{4.6}$$

$$IPT = \frac{\sum_{d_{qh} \in VPLs} d_{qh}}{\sum_{d_{ij} \in D} d_{ij}} * 100 \tag{4.7}$$

where the denominator represents the total amount of data exchanged by all tasks of the application, while the numerator indicates the amount of data exchanged between tasks

mapped on different PEs. This latter is denotated as inter-clusters links (*VPLs*).

4.4 Overview of our hierarchical DSE methodology

The overall scheme of our hierarchical DSE methodology is depicted in Fig. 4.2. Two separated phases communicating via a file-based interface can be clearly distinguished: estimation phase and simulation phase. The goal of the first phase is to gradually reduce a large design space to arrive at a reduced number of potential mapping solutions, while the second phase targets a more accurate performance evaluation for each potential solution of this reduced design space. This way, our two-phase approach enables the system designers to exploit (i) the benefits of analytical techniques for a rapid exploration of large design space, (ii) the accuracy of the simulators for individual design point evaluation, and (iii) the ability to integrate different (or additional) analytical models and simulation tools for a hierarchical design space exploration, aiming to solve the problem of the real-time application mapping onto MP-SoC formulated in Section 4.3.

As will be explained with more details in the next section, the first phase is based on analytical methods, i.e., this phase does not use simulations, it rather uses a set of heuristic-based algorithms to explore rapidly the design space associated to the sub-problems of partitioning, scheduling and assigning, respectively. Moreover, an additional analytical model is used during this process to estimate roughly the performance of each design point (in terms of the computation and communication delay), such that if the resulting delays sum of a design point cannot satisfy the RTC, this design point will not be considered by the simulation phase. As a consequence, our estimation phase can prune dramatically the initial design space, and only a reduced number of potential mapping solutions is given as output.

Subsequently, these detected potential solutions are specified in a text file, being the description of each potential solution encoded in a numeric vector (or string) format. This way, each mapping description together with the application model and the architecture template are combined to generate an ESM, which is required and used by the simulator to evaluate more accurately the system performance of each potential solution. Finally, this simulation process is repeated iteratively either until a solution (that meets the user constraints) is found or for all potential solutions, and in this latter case, the optimal simulated solution is then selected as output.

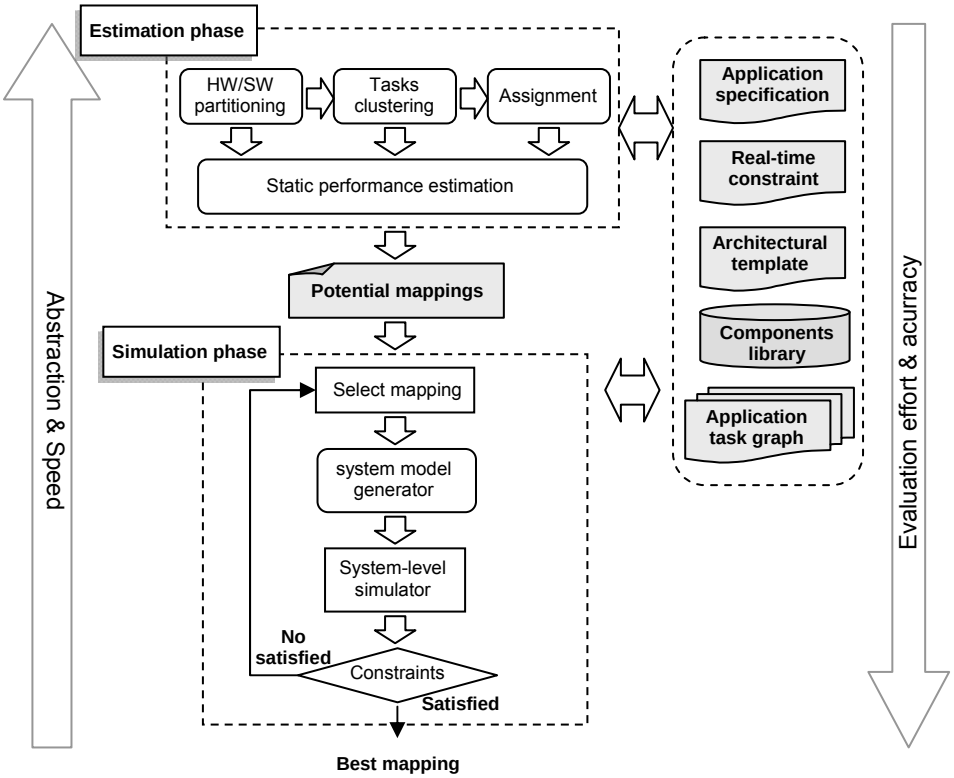


Fig. 4. 2 Overview of the hierarchical DSE methodology.

At this point, we would like to stress that DSE experiments using our hierarchical methodology can be perfectly carried out with NASA framework. In fact, we have performed a set of hierarchical DSE experiments configuring appropriately NASA input files, and plugging CASSE as well as the aforementioned analytical (or pruning) methods into NASA. Evidently, the file-based interfaces used in our hierarchical DSE methodology (see Fig. 4.2) are defined according to the requirement of NASA framework. All the results obtained in these hierarchical DSE experiments are shown in Section 4.6, where we also present a comparative analysis (in terms of run-time and quality of the mapping solutions) between our hierarchical DSE and other DSE based only on analytical models and/or simulations.

4.5 Heuristics-based algorithms and analytical model for hierarchical DSE

This section aims at explaining the distinct intermediate steps in each phase of our approach. Moreover, using examples, we illustrate some issues that arise in the process of mapping a real-time application onto a MPSoC.

4.5.1 Static performance estimation

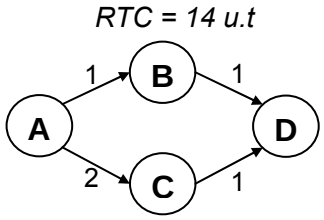
The goal of this phase is to explore and prune the design space, i.e., identify all potential solutions that meet the design objectives (listed in Section 4.3), while eliminating those alternatives that do not meet the requirements. To this end, an analytical estimation model and three heuristic-based algorithms have been applied currently: (i) HW/SW partitioning, (ii) tasks clustering, and (iii) clusters assignment.

4.5.1.1 HW/SW partitioning

For a given TG and available PE types, this algorithm selects a suitable PE type for each task, i.e., it decides whether a task is executed as software embedded on a processor or by using a hardware implementation to achieve specific performance requirements. To this end, all tasks are checked formally by the following condition:

$$wcet(t_i)_{proc} \leq RTC, \ \forall t_i \in T \tag{4.8}$$

where $wcet(t_i)_{proc}$ represents the worst case computation time on processor *proc* for the task t_i. Thus, if $wcet(t_i)_{proc}$ is longer than *RTC*, the task t_i is selected to run on a processor; otherwise, t_i is implemented in a specific hardware block[5]. As a result, this partitioning algorithm outputs two numeric vectors or strings: VNT and VCT.



		Tasks			
		A	B	C	D
wcet for each PE type	1(proc)	20	5	5	2
	2(HW)	10	-	-	-
Partitionning outputs	VCT	10	5	5	2
	VNT	2	1	1	1

Fig. 4. 3 Example of the HW/SW partitioning algorithm.

1) VNT is a vector containing the nature of each task. In this context, the nature of the task refers to the PE type selected for each task, i.e., hardware dedicated block or generic processor. Note that the VNT is actually a numeric vector, such that different types of PEs (e.g., processors and/or hardware dedicated blocks) are coded with different numeric values.

[5] We assume that the computation time of the task t_i on a dedicated hardware block can perfectly satisfy the aforementioned condition.

2) VCT is a vector defined as $\{c(t_1), c(t_2), \ldots, c(t_k)\}$, where $c(t_i)$ is the computation time of task t_i on the PE type selected for t_i, i.e., according to the nature of the task t_i specified in VNT.

Example 4.2. Fig. 4.3 depicts an application modelled as a task graph composed of 4 tasks and 4 channels. Suppose that 2 PE types are available in the component library: a generic processor (*proc*) and a HW dedicated block for task A, and RTC is 14 u.t. Then, according to the *wcet* values listed in Fig. 4.3 and (4.8), the HW/SW partitioning algorithm will select the task A for a hardware implementation, while the rest of the tasks will be run on the generic processor. As a consequence, the VNT and VCT will be delivered as outputs, which are also shown in Fig. 4.3. Finally, it should be noticed that no information about architecture topology/platform is taken into account in this step.

4.5.1.2 Tasks clustering

This step aims to solve the sub-problem of scheduling. Basically, we propose an algorithm that, considering the *RTC*, the outputs of the partitioning algorithm, and the data dependencies between tasks specified in *TG*, enables to schedule the tasks on *v logical clusters* or *Virtual Processors* (*VPs*) working in a pipeline fashion, such that each task (inside a *VP*) is executed according to the order in which it was scheduled in the *VP*.

The pseudo-code of our tasks clustering algorithm is described in Algorithm 1. First, the clustering algorithm uses the information contained in VNT to bind each task selected for hardware implementation to a new *VP*. Subsequently, it creates a new task graph (*TG'={T', L', D'}*) without the tasks bound to *VPs*. Note that all links associated to such bound tasks are not considered in *TG'*. Then, the clustering algorithm copies all links of the roots to the list of ready-links (R), such that each couple of tasks associated to each link of R that satisfies the two following conditions are bound to the current virtual processor (namely, VP_h):

$$c(t_i) + c(t_j) \leq RTC - c(\nu(VP_h)) \tag{4.9}$$

$$min\left\{ \max_{\forall d_{ij} | l_{ij} \in R} \{d_{ij}\} + \sum_{\forall t_k \in succ(t_j)} d_{jk} \right\} \tag{4.10}$$

where $\nu(VP_h)$ gives the set of tasks that are bound on VP_h. The first condition ensures

that $c(v(VP_h)) = \sum_{\forall t_i \in v(VP_h)} c(t_i) \leq RTC$ in any instant, and then, the objective of PE load balancing is guaranteed in each task clustering decision. In the second condition, the first term clusters into the same *VP* the couple of tasks (t_i, t_j) that share the most amount of data. If more than one option is possible, the second term selects an l_{ij} such that t_j is the task that shares the least amount of data with all its immediate successors (i.e., $succ(t_j)$). Consequently, the inter-*VPs* traffic is minimized. Note that as soon as a task t_j is scheduled into a VP_h, a new ready-link list R' is obtained from t_j, and these two conditions are newly applied to R'. This iterative process $F(t_j)$ is repeated until the R' is emptied. In this point, it should be noticed that a link of R' is removed either when it has been scheduled in VP_h, or it is impossible to schedule it into VP_h according to both conditions above, and then it has to move into *R*.

Once all tasks have been scheduled, the clustering algorithm outputs a set of virtual processors, $VPs=\{VP_1, VP_2, \ldots, VP_v\}$, where each *VP* contains a set of tasks that determine directly the nature of each *VP (tVP)*, i.e., whether a *VP* contains tasks that will be run on generic processor or implemented in hardware dedicated block. Finally, using the set of *VPs*, the clustering algorithm also generates a set of *inter-VPs* links, $VPLs=\{vpl_{12}, vpl_{13}, \ldots, vpl_{jv}\}$, where vpl_{ij} represents a link or channel between a task

Algorithm 1. Pseudo-code of task clustering algorithm

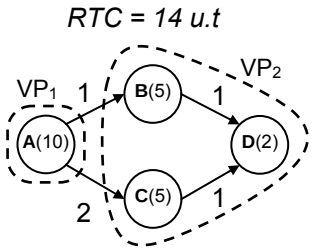
Input: TG={T,L,D}, VNT, VCT, RTC

```
 1:   R={Ø};
 2:   VPs={Ø};
 3:   for all t_i in T
 4:       if nature t_i in VNT is HW
 5:         select new VP; VP ← t_i; VPs ← VP;
 6:       end if;
 7:   Copy TG' ← TG;
 8:   Remove in TG' all tasks and links bound to existing VPs;
 9:   Select all roots and obtain R;
10:   Select new VP;
11:   while (R≠ Ø)
12:       select l_ij in R that satisfies conditions (4.9) and (4.10);
13:       if ( ∃ l_ij)
14:          VP_h ← t_i;
15:          Update VP_h computation time;
16:          Remove l_ij from R; F(t_j);
17:       else
18:          VPs ← VP_h;
19:          select new VP;
20:       end if;
21:   for all l_ij in L
22:       if ((ti, tj) belong to different VP)
23:          e_ij ← d_ij; vpl_ij ← (l_ij, e_ij);
24:          VPLs ← vpl_ij;
25:       end if;
```

Output: VPs, VPLs

belonging to VP_i and other task in VP_j, and has associated a communication cost, e_{ij}, which specifies the amount of sharing data (in bytes) in this link. All this information is used as input in the next step, as will be explained later.

Example 4.3. Following with the example shown in Fig. 4.3, the results of applying the tasks clustering algorithm are depicted in Fig. 4.4. First, since task A has been selected for a hardware implementation, it is assigned to a single cluster (VP_1). Moreover, the type of VP_1 (*tVP*) has also been set to HW. Subsequently, the rest of the tasks are checked according to (4.9) and (4.10) and clustered in VP_2. As mentioned before, each task is executed (inside a *VP*) according to the order in which it was scheduled, and in this example, the execution order of the tasks in VP_2 is B, C and D.



VPs	tVP	Eligible tasks	Conditions		c(VP)
			(4.9)	(4.10)	
VP_1	HW	A	A	-	10
VP_2	proc	B,C	B,C	B	5
		C	C	-	10
		D	D	-	12

Fig. 4. 4 Example of the tasks clustering algorithms.

4.5.1.3 Cluster assignment

The goal of this step is to assign the *VPs* to the PEs of the target MPSoC platform[6], while the assignment of *VPLs* on the SEs is carried out in the next step. Thus, this clusters assignment process implies to make three design decisions:

1. specify the type and number of PEs,
2. which task should be assigned to which particular PE instance, and finally,
3. decide the location of each PE instance in the architectural template.

In order to handle such design decisions, our proposed algorithm assigns firstly the *VPs* to PEs using a one-to-one assignment function, $\Lambda(VP)$, i.e., $\Lambda(VP_i) \neq \Lambda(VP_j), \forall\ VP_i \neq VP_j$. As a consequence, the number of PE instances actually used in the architecture template

[6] Note that the number of VPs provided by the tasks clustering algorithm is *v*, which can be greater than *p* (i.e., the number of PEs in the MPSoC platform). In our current implementation, however, we suppose that $p \geq v$.

(NPE) corresponds to the number of *VPs*, while the type of each PE instance is specified by the *VP* type (*tVP*) assigned to each PE instance. Once the *v VPs* have been assigned to *v* PEs, we generate subsequently all possible combinations of location for these *v* PE instances in the target architecture template. In this case, for an architecture template with up to *p* PE holders, the total number of allocation alternatives for the NPE is $p!/(p-v)!$.

On the other hand, although the inter-clusters links assignment is not performed in this step, the different combinations of type and allocation for SEs are indeed explored in this step. Note that this latter is needed by the analytical estimation model to determine the optimal combination of number, type and location of SE instances for the *VPLs* assignment. In this case, if up to *m* SEs can be instantiated in the architecture and there are *st* different types of SEs (*tSE*), the resulting number of possibilities is st^m.

Therefore, considering the above analysis for both PEs and SEs, the total number of possible scenarios (*SCs*) can be approximate as following:

$$\text{Total number of SCs} = \left(\frac{p!}{(p-v)!}\right) * st^m \tag{4.11}$$

In our implementation, all these *SCs* are exhaustively generated and listed as output in a text file. Each scenario is represented in a vector format as the following:

$SCi=\{idVP_1, idVP_2, \ldots, idVP_p; tSE_1, tSE_2, \ldots, tSE_m\}$

where a value of $idVP_j$ distinct than null indicates the identifier of the *VP* assigned to a PE instance located in the PE holder *j*, while tSE_q refers to the SE holder *q* that contains a SE instance of type *tSE*.

4.5.1.4 Static performance estimation

Given a set of scenarios, this step estimates analytically the performance of each scenario, and outputs those cases that meet the real-time constraint. To this end, we have proposed and used in our hierarchical DSE methodology an analytical estimation model, which takes into account both static and dynamic behaviour during the evaluation of each design point. As a result, our analytical model can prune rapidly a large design space with a minimal effect on the run-time of the framework. Nevertheless, it should be noticed that our goal is not to develop a sophisticated estimation model based on

complex mathematical formulations, but rather to propose a model accurate enough to roughly estimate the performance of different design alternatives and reduce rapidly the number of potential mapping solutions, since these latter will be evaluated more accurately by our system-level simulator in the further step.

Criterion for pruning the design space

In our approach, since we assume that the set of assigned PEs works in a pipeline fashion, each scenario performance is thus measured in terms of total PE time in order to decide whether the real-time constraint is satisfied, i.e., we check formally the following condition for each assigned PE of a scenario:

$$max\{CMP_i + CMM_i\} \leq RTC; \quad \forall i = 1..p \tag{4.12}$$

where CMP_i and CMM_i are the total computation time and total communication time due to the tasks assigned on PE_i, respectively. This way, the total PE time is calculated for each assigned PE of the scenario, and thus, a scenario is considered as a potential

Algorithm 2. Pseudo-code of clustering assignment algorithm and analytical performance estimation

Input: VPs, VPLs, MPSoC template

```
1:   Λ(VP_i): VP_i → PE_j;  ∀VP_i ∈ VPs;
2:
3:   Generate all possible scenarios (SCs);
     Potential_solutions = {Ø};
4:   for all scenarios
5:       for all PEs
6:           CMP_i ←calculate total computation time for PE_i;
7:       Generate latency table;
8:       links ← VLPs; infeasible_mapping = false;
9:       SEs={Ø};
10:      while ((links ≠ Ø) or (infeasible_mapping))
11:          Determine α(SE);
12:          if (∃ SE_q)
13:              SE_q ← link_h; remove link_h from links;
14:          else
15:              infeasible_mapping = true;
16:          end if;
17:      if (infeasible_mapping ≠ true)
18:          for all PEs
19:              calculate cpd, ccd, rwd;
20:              CCM_i ←calculate total communication time for PE_i;
21:          if (condition (4.12) is satisfied)
22:              Potential_solutions ← SC_h;
23:          end if;
24:      else
25:          infeasible_mapping = false;
26:      end if;
27:  Rank potential solutions in ascendant order of total PE time;
```

Output: Potential_solutions strings;

solution if the maximum total PE time (of all total PE time of a *SC*) can achieve the above condition.

Total computation time

In order to determine such a total PE time, CMP_i and CMM_i should be first calculated. The total computation time of a PE is estimated in our algorithm as follows:

$$CMP_i = \sum c(t_j); \quad \forall t_j \in PE_i \tag{4.13}$$

The latency table

Nevertheless, in order to determine the corresponding communication delays for each scenario, the *VPLs* should be firstly assigned to the SE instances. In this point, it is important to notice that the location of assigned PE instances and the communication delays are closely related with each other, since the characteristics of inter-PE communications as well as SE accesses depend on the *VP* assigned to each PE instance. That is, when the location of a PE instance and/or *VP* assignment change, the traffic characteristic on the system and the availability of accessible resources change as well. Consequently, these latter should be taken into account during the communication delay estimation. In our case, such availability or accessibility of resources is made explicit by means of the latency or connectivity table. More specifically, our assignment algorithm generates automatically for each scenario a latency table, which reflects the relationship between the PE location and the accessibility of shared resources. Thus, the latency table shows all accessible SEs for each PE instance located in the target architecture, as well as the remaining storage capacity and access latency per byte of each SE instance. An example of such a latency table is shown in Fig. 4.5.

Constraints for assigning *VPLs* to SEs

On the other hand, our analytical model estimates the communication delays in several steps. First, we assign iteratively the *VPLs* to a set of available SEs, *α(SE)*, where priority is given to vlp_{ij} with the highest e_{ij} value in each iteration. In this context, vlp_{ij} represents a link communicating PE_i with PE_j, and *α(SE)* refers to a set of SEs that satisfy these constraints:

(R1) each of them can be accessed or shared by PE_i and PE_j,

(R2) each of them still has sufficient storage capacity to hold e_{ij}, and

(R3) if a *VPL* can be assigned to several available SEs, the SE with the minimum access latency is selected.

Communication protocol delay and SE access delay

Therefore, the communication protocol delay (cpd) corresponding to each *inter-VPs* link is estimated by means of a set of latency parameters associated to each type of architectural components, which are specified in the user input file and also used in the architectural description file (as explained in Chapter 2). And the SE access delay (rwd) is calculated as a product of the amount of shared data (in the *VPL*) and the access latency of the SE where the *VPL* is assigned to.

Communication contention delay

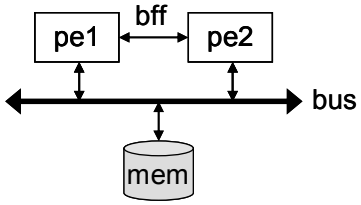
In order to estimate the contention delay (ccd), we firstly identify the path or NE assigned to each inter-cluster link, i.e., the NE used by the PE to access a SE. Thus, when several *VPLs* compete to use simultaneously the same shared NE, a communication overhead and contention delay occur, which is approximated to:

$$ccd = (\alpha\text{-}1)*com(NE_i) \tag{4.14}$$

where $com(NE_i)$ is the average SE access delay of all *VPLs* that share the network NE_i, and α is the total number of *VPLs* that:

(i) share the same NE_i, and
(ii) are associated to different PE sources and/or PE destinations.

Example 4.4. Suppose that the *VPs* and *VPLs* obtained in Example 4.3 are mapped on the target architecture shown in Fig. 4.5, where the latency table with the corresponding access latency and storage capacity (denoted in parentheses) are also illustrated. Applying our cluster assignment algorithm, two possible scenarios are generated by varying the assignment of *VPs* to PEs. In order to simply the explanation, we suppose that the location of SEs (i.e., bff and mem) cannot be changed in this example. As a result, the design space that should be explored is composed of eight possible mapping solutions. Applying our analytical estimation model in this (fairly simple) example, 75% of the design space can be rapidly pruned due to different infeasibilities, while feasible mapping solutions satisfying the condition (4.12) are also found. Therefore, this example demonstrates how our analytical estimation model (and pruning phase) can reduce the number of potential solutions to be simulated in the next phase, improving thus the efficiency of DSE in terms of run time.



	Latency table	
	bff(2)	mem(3)
pe1	0	1
pe2	0	1

Possible mapping	VPs mapping		VPLs mapping		Feasibility	CMM		max{CMM_i+CMP_i}
	VP_1	VP_2	VPL1(1)	VPL2(2)		pe1	pe2	
1	pe1(HW) CMP(pe1)=10	pe2(proc) CMP(pe2)=12	bff	bff	Infeasible(1)	-	-	-
2			bff	mem	Infeasible(2)	2	2	14
3			mem	bff	Feasible	1	1	13*
4			mem	mem	Infeasible(2)	4,5	4,5	16,5
5	pe2(HW) CMP(pe2)=10	pe1(proc) CMP(pe1)=12	bff	bff	Infeasible(1)	-	-	-
6			bff	mem	Infeasible(2)	2	2	14
7			mem	bff	Feasible	1	1	13
8			mem	mem	Infeasible(2)	4,5	4,5	16,5

Infeasible(1): Storage capacity limitation; Infeasible(2): Violation of VPLs assignment constraints;

* For sake of simplicity, we suppose that cpd=0;

$$pe1 \Rightarrow \left\{ \begin{array}{l} rwd = 0 \times 2 + 1 \times 1 = 1; \\ com(bus) = 1; \\ \alpha = 1; \end{array} \right\} \; ccd = 0; \Bigg\} \quad CMM_1 = 1; \quad CMP_1 + CMM_2 = 11;$$

$$pe2 \Rightarrow \left\{ \begin{array}{l} rwd = 0 \times 2 + 1 \times 1 = 1; \\ com(bus) = 1; \\ \alpha = 1; \end{array} \right\} \; ccd = 0; \Bigg\} \quad CMM_2 = 1; \quad CMP_2 + CMM_2 = 13;$$

$$max\{CMM_i + CMP_i\} \le RTC \qquad \forall i = 1..2; \qquad \Rightarrow \quad max\{CMM_2 + CMP_2\} = 13 \le 14;$$

Fig. 4. 5 Example of the cluster assignment algorithm and analytical estimation.

Our idea is that, the contention delay suffered by an inter-clusters link is directly proportional to the average SE access delay of the *VPLs* that compete for the same NE, such that the more *VPLs* (i.e., a high *α* value) share the same path, the more waiting-time (on average) is needed to access the data. However, we also would like to notice that, this way of estimating the contention delays can introduce certain error or difference with respect to the simulation results when, (i) the SE access delay of inter-clusters links (sharing the same NE) present high deviations with respect to the average SE access delay, and/or (ii) the number of inter-clusters links sharing the same NE increases, as will be depicted in the Section 4.6.

Once that all CMM_i have been estimated according to the aforementioned process, our algorithm eliminates all those scenarios that do not satisfy the condition (4.12). As a result, the remaining scenarios are ranked and listed in the ascendant order of the maximum total PE time, such that a scenario with a small maximum total PE time is evaluated first in the next simulation-based phase.

4.5.2 Potential solutions

In our framework, a text-file interface is used to link both estimation and simulation phases. Such interface enables to represent symbolically the design space composed by the potential solutions, where each alternative is encoded as two numeric strings. The format of these strings is described as the following:

$\{idPE_1, idPE_2, \ldots, idPE_k; idSE_{12}, idSE_{13}, \ldots, idSE_{jk}\}$
$\{tPE_1, tPE_2, \ldots, tPE_p; tSE_1, tSE_2, \ldots, tSE_m\}$

where $idPE_k$ indicates that the task t_k is mapped on the PE holder $idPE$, $idSE_{jk}$ refers that the link l_{jk} is mapped on the SE instance holder $idSE$, while tPE_p and tSE_m represent the PE instance type and the SE instance type allocated in PE holder p and SE holder m, respectively. Fig. 4.6 illustrates an example of such strings, which correspond to the case studied in the above examples.

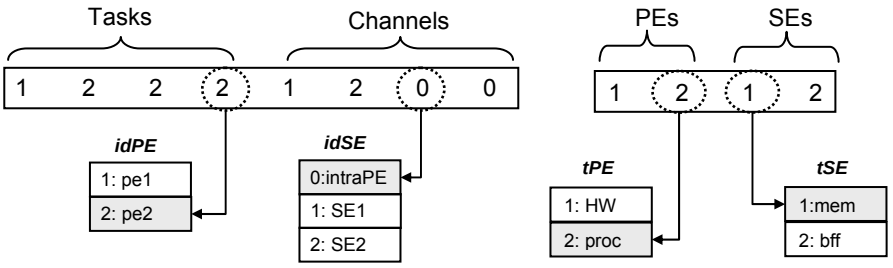


Fig. 4. 6 Example of mapping solution description strings.

It should be noticed that the choice of the representation (or encoding) scheme has a strong impact on the scalability and flexibility of the framework. And in our case, this choice is due to mainly three reasons:

(i) it allows us to modularly define a very large space, where our representation scheme warrants that each potential solution receives a unique encoding value,

(ii) both the vector length and the variable values are not fixed, but they are dynamically updated for each experiment according to the user specifications, and

(ii) last but not least, this format fits perfectly to the interface requirements of NASA. This way, our hierarchical DSE methodology as well as our analytical algorithms can be plugged into NASA infrastructure, as will be demonstrated in Section 4.6.

4.5.3 Global system simulation

In order to evaluate more accurately each potential solution of the pruned design space, we have used CASSE in our framework. As explained in Chapter 2, CASSE enables the system designers to include and configure (for each DSE experiment) numerous design variables, which often are not taken into account by the analytical models, such as bus arbiter policy, processor scheduler, memory map, different clock domains, size and number of data packets that can simultaneously circulate on the network, and so on. As a result, different quantitative information about the studied system can be gathered during each simulation, which provides to the system designer a more complete and realistic vision of the system load fluctuations, as well as their impact on the system performance for later evaluation and final decisions.

In this point, it should be noticed that in order to simulate each potential solution, the simulatable system model of each potential solution (i.e., an ESM consisting of an application model[7], architecture model and mapping model) should be generated first. To this end, we have used the Translator (presented in Section 3.5.5), which can carry out automatically this process, as already explained in Chapter 3. Evidently, the needed information about the task computation delay, as well as the number and type of each kind of components in the architecture are provided by the potential solution strings explained in Section 4.5.2.

Finally, each potential solution is evaluated in CASSE, where we use a pre-determined terminating condition to conclude the system-level simulations. Thus, the iterations eventually terminate either when a solution meeting the real-time requirement is found, or when no potential solution can achieve such user constraint. On the latter case, no solution is output by our approach.

[7] Note that the estimation phase only needs the task-graph (TG) of the application to carry out the design space exploration and pruning process, while the source code of the tasks is required explicitly by CASSE to perform the system-level simulations.

4.6 Experimental results

In this section, we present several sets of experimental results. Our aim is to compare our hierarchical DSE approach with other approaches based only on analytical model or on simulation tool, and therefore demonstrating the capacities and benefits of using our mixed and hierarchical approach for the kind of problem addressed in this chapter. Finally, we also would like to stress that all our DSE experiments are carried out with the NASA framework, illustrating thus its flexibility and capacity to integrate different kinds of techniques for performing system-level DSE experiments, as well as to solve the mapping problem taking into account multiple design objectives, as formulated in the problem statement of Section 4.3.

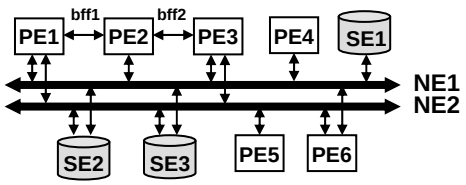
4.6.1 Analysis of the DSE efficiency

4.6.1.1 Target MPSoC platform and real-time application

The studied architecture template, the latency table associated to such template, as well as the possible configuration parameters of the different architectural components are depicted in Fig. 4.7. Basically, this MP-SoC template may consist of up to 6 PEs of types ARM-9 or hardware blocks, up to 3 SEs of either single (SDR) or double (DDR) data rate types, and up to 2 AMBA busses. The application that is mapped onto the MP-SoC is the visual tracking algorithm presented in Chapter 2.

4.6.1.2 NASA configurations for DSE experiments

Using such user specifications, we have configured NASA to perform several sets of experiments. In the first set of experiments, only the estimation phase (i.e., analytical estimation model) of our hierarchical approach has been used to find the optimal mapping. Since no simulations are needed in this set of experiments, it can be carried out without using CASSE. Note that a single solution (or the best estimated solution) is outputted in this case. Subsequently, we repeat the set of experiments using our two-phase approach. To this end, we have plugged into NASA both our analytical models and CASSE. In addition, we performed the last set of DSE experiments using two genetic algorithms (GAs), i.e., we apply our dimension-oriented DSE methodology (presented in Chapter 3) in order to co-explore the design space (consisting of the architectural components and mapping dimensions). It should be noticed that the platform dimension is not explored in these experiments since the MPSoC platform is already fixed, so that we had to set up appropriately the platform string values according to the template depicted in Fig. 4.7. Finally, the values of the GA parameters used in this last set of experiments are listed in Table 4.1.



	bff1 (128)	bff2 (128)	SE1 (Cap)	SE2 (Cap)	SE3 (Cap)
PE1	0	∞	Lat	Lat	Lat
PE2	0	0	Lat	∞	∞
PE3	∞	0	Lat	Lat	Lat
PE4	∞	∞	Lat	∞	∞
PE5	∞	∞	∞	Lat	Lat
PE6	∞	∞	Lat	Lat	Lat

Architecture element	Number	Types	Value
PE	≤ 6	RISC Accelerator	ARM-9 Dedicated hardware
SE	≤ 3	SDR DDR	Cap=256 KB; Lat=5 Cap=28 KB; Lat=1
NE	≤ 2	AMBA bus	32 bit width, Round robin arbiter, I/O buffered

Fig. 4. 7 Target architecture template and component configuration parameters.

Parameters	Number	Types	Values
Selector (*S*)	1	1	Proportional with elitism
Crossover (*C*)	1	2	2-points
C probability (*pc*)	5	-	[0.1, 0.3, 0.5, 0.8, 1.0]
Mutation (*M*)	1	1	Single mutation per individual
M probability (*pm*)	5	-	[0.1, 0.3, 0.5, 0.8, 1.0]
Population size	10	-	Nr. of individual per iteration
Iterations	21	-	-

Table 4. 1 GA parameters used in the third set of experiments.

4.6.1.3 Comparisons between different DSE approaches

In order to compare the experimental results obtained with different DSE approaches, we have selected four metrics:

(M 1) Number of explored design points.

(M 2) Total run time of the DSE experiment.

(M 3) Performance achieved by the optimal solution found in the exploration.

(M 4) Number of simulations until finding the optimal solution.

The most significant differences between the results obtained in the aforementioned experiments are summarized in Table 4.2. At first sight, it can be seen that for different

timing constraints (RTC = 25 frames/s and RTC = 28 frames/s, i.e., processing 1250 packets/s and 1400 packets/s, respectively), our hierarchical DSE approach not only can cover statically a large number of design points, but also can achieve a valid solution after only a reasonable small number of simulations. On the other hand, if only the analytical phase is applied in the DSE experiments, a large design space can be also explored analytically. However, a more accurate evaluation reveals that the best estimated solution (which achieves the real-time requirement according to the analytical model) cannot always satisfy actually such constraints (for example, as in the case of RTC = 28 frames/s). As already mentioned in Section 4.1.3, such difference or margin between the estimated and simulated result is inherent in any analytical estimation model, what emphases the need of a simulation-based phase in our approach.

	RTC = 25 frames/s				RTC = 28 frames/s			
	M 1	**M 2**	**M 3***	**M 4**	**M 1**	**M 2**	**M 3***	**M 4**
Estimation	279936	30 s	1252 (1266)	-	279936	30 s	1308 (1425)	-
Estimation + Simulation	279936	30 s	1252 (1266)	1	279936	90 s	1401 (1411)	3
Simulation	210	1800 s	1263 (-)	60	210	6000 s	1408 (-)	200

* Performance obtained in the simulation (estimation).

Table 4. 2 Comparison of the DSE efficiency between the three sets of experiments.

Finally, the DSE in the last set of experiments is guided by two GAs, which ensures the convergence to the optima points exploring only a reduced number of alternatives, as has already been demonstrated in Chapter 3. Note that a GA could have substituted perfectly our heuristics-based algorithms at the pruning phase as well. This way, an exhaustive exploration of all possible design combinations (in the assignment step) is not needed any more, therefore improving greatly the efficiency of our DSE experiments. However, in spite of the efficiency demonstrated in our GA-based DSE experiments, the long simulation time is still being the dominant factor in these experiments based only on simulations. In fact, CASSE requires on average 30 seconds to simulate a single design point in these experiments, while the GAs need an average time of 1800 seconds and 6000 seconds (i.e., for 60 and 200 simulations) to reach the first solution that meets the real-time constraints of 25 frames/s and 28 frames/s, respectively. That is, comparing with GAs-based DSE experiments, our hierarchical DSE approach can improve the efficiency of DSE (in terms of total run-time to reach the first valid mapping) by two orders of magnitude. Moreover, we also would like to notice that the results obtained in GA-

based experiments suffer from an important variance, since the GA is extremely sensitive to its parameters such as the initial population, probability associated to crossover (*pc*) and mutation (*pm*) operator, etc.

This last aspect can be illustrated in Fig. 4.8, where the gray dotted lines indicate the top and down boundaries closing the results obtained with 40 different initial populations and combinations of *pc* and *pm*, the black line represents the average performance in each iteration reached by the individuals of all GA-based experiments, while the gray lines are two particular examples extracted from this set of experiments. If the input arrival frame rate is 1450 packets/s and a minimum of 1250 packets/s has to be processed to satisfy the minimum RTC of the studied application (which is equivalent to processing 25 frames/s), it can be seen clearly that, the experiment labelled *pc=0.8pm=0.8* not only needs fewer simulations to reach a valid solution, but can also converge progressively to higher performance solutions, while the other experiment labelled *pc=0.5pm0.1* can hardly reach the minimum RTC after 20 iterations.

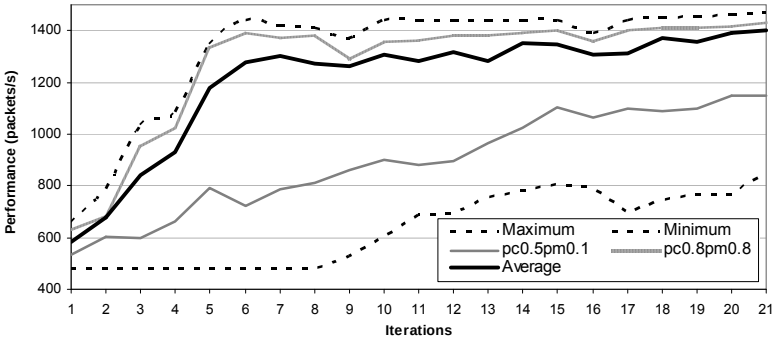


Fig. 4. 8 Results obtained in GA-based DSE experiments.

Moreover, the variation of the performance achieved in DSE experiments with different GA parameters configurations can be significant. And for the case shown in Fig. 4.8, such a margin between the maximum and minimum can be up to 65% (measured in sixth iteration). In our point of view, these data could suggest that, in order to find out the best configuration of the GA parameters for each studied case, the system designers should repeat the experiments several times with different combinations of GA parameters settings, therefore decreasing the attraction (in terms of time and efforts) of using GA in the DSE experiments.

4.6.2 Analysis of the quality of the optimal solutions

In this point, it also should be noticed that although our hierarchical DSE approach and GA-based DSE experiments can provide solutions satisfying different RTCs (i.e., primary design objective), the quality of such solutions can be also radically different. In this context, we measure the quality of the solution in terms of the secondary design objectives defined in Section 4.3, i.e.:

(O 1) Minimize the resources used in the architecture design.
(O 2) Maximize the efficiency of resource usage.
(O 3) Minimize the load unbalancing between PEs actually used in the solution.
(O 4) Minimize the total communication traffic in the system.

In Table 4.3, we list the mapping solutions obtained by our hierarchical DSE approach, as well as the first optimal mapping solution reached by the GAs-based experiment *pc0.8pm0.8* for different RTCs. More specifically, we indicate the type of PE and SE allocated in the component holders actually used in each solution, as well as the tasks and channels mapped on each of them. On the other hand, Table 4.4 analyzes the quality of the aforementioned solutions in terms of the aforementioned secondary design objectives.

RTC (frames/s)		Tasks mapping	Channels mapping
25	**Estimation + Simualtion**	Proc1={A}, HW3={D}, Proc2={B,C,E,F,G}	Buf1={1}, Buf2={3,4,5,9,10}
	Simulation	Proc1={B,C}, Proc3={E}, HW5={D}, Proc4={A,F,G}	SDR1={1,6,7,9,11,12}, DDR2={3,5,10}, DDR3={4}
28	**Estimation + Simulation**	Proc1={A}, HW3={D}, Proc2={B}, Proc6={C,E,F,G}	Buf1={1}, Buf2={3}, DDR1={2,9,10}, DDR2={4,5}
	Simulation	Proc1={B}, Proc5={C}, HW4={D}, Proc3={F,G}, Proc6={A,E}	SDR1={3,6,10,11}, DDR2={2,7,9}, DDR3={1,4,5}

Table 4. 3 Mapping solutions obtained in our DSE experiments.

Examining these data, it can be seen that for both RTCs, our approach of hierarchical and mixed estimation and simulation DSE can provide solutions using a lower number of PEs and SEs. On the other hand, the number of assigned PEs has also a direct impact on the system traffic (i.e., O 4) , since the more PEs are used, the less amount of inter-

tasks channels are assigned inside PEs, therefore increasing the volume of exchanged data via shared resources and lengthening the actual execution time of the tasks. In fact, an immediate effect of this latter result is also visible on the difference of performances obtained in the estimation and simulation, shown in Table 4.2. In this case, when the number of resources competing for shared resources increases, the occurrence of contention becomes probably more frequent and complex. As a result, the performance estimated with the analytical model is less accurate, increasing the margin between estimated and simulated results.

RTC (frames/s)		O 1*	O 2	O 3	O 4
25	Estimation + Simualtion	3 (0)	88.875 %	11.533 %	49.045 %
	Simulation	4 (3)	66.656 %	29.066 %	87.308 %
28	Estimation + Simulation	4 (2)	71.655 %	18.427 %	58.903 %
	Simulation	5 (3)	57.324 %	30.026 %	97.124 %

* Number of PE (SE) used in the solution.

Table 4. 4 Comparison of the quality of the mapping solutions.

To summarize, RTCs can be easily achieved by over-dimensioning the hardware resources, as occurs with the results obtained in GAs-based DSE experiments. However, the challenge is to deliver timing guarantees without such forms of over-provisioning, because this latter tends to introduce larger amounts of traffic load and/or repetitive overhead, saturating thus the system communication. Therefore, the results illustrated in this section reveal that our proposed methodology is profitable for searching/exploring large design spaces. More specifically, compared with the traditional GA-based search techniques, our hierarchical DSE methodology can improve the efficiency of DSE by two orders of magnitude as well as provide higher quality mapping solutions.

However, we would like to stress that these *proof-of-concept* DSE experiments are aimed to demonstrate and illustrate the key properties of our methodology for a target application and a particular MPSoC platform. Then, we plan to carry out more experiments using more applications and a wide range of architectures in the future, in order to validate and prove progressively the capabilities of the methodology presented in this chapter.

4.7 Conclusions

With the increasing complexity of embedded systems on chips, an enormous amount of new possibilities for the application mapping onto a target architecture is emerging, leading to an exponential growth of the design space. Such challenge demands new methodology capable to explore large design space in a fast and accurate way. In this chapter, we have focused on the problem of the mapping of a real-time application on a flexible architecture template, and more specifically, we have presented a hierarchical DSE methodology and a set of analytical methods to address this issue. We consider that the set of experiments conducted are quite complete and enough to prove the concept and to validate the framework, setting up the road for more experiments in the future in order to progressively show the full capabilities of our methodology.

Our proposed methodology consists of two independent phases. The first phase uses a set of heuristic-based algorithms for the design space exploration, as well as an analytical model for a rough performance estimation of the explored design points. The purpose of this phase is to prune rapidly a large design space, such that only a reduce number of potential solutions are retained for next phase. Then, a system-level simulation tool is used in the second phase, in order to evaluate accurately each of these selected solutions until a solution meeting RTC is found.

Finally, the experimental results presented in this paper reveal that our approach, compared to other approaches based only on either analytical estimation models or on simulations guided by GAs, not only can explore a large design space and reach a valid solution in a time-efficient and accurate way, but can also provide an optimal solution optimizing features such as resource usage efficiency, system traffic load and processor load balancing.

Chapter 5
Conclusions and future work

This chapter presents the main conclusions obtained from this PhD work and the future research areas that could be fostered from it.

5.1 Conclusions

The increasing level of integration on chips is leading to more complex embedded systems. As a result, system designers have currently to explore an exponential growing design space in order to reach an optimal design solution considering multiple design objectives. In this context, it is widely believed that traditional design methodologies and tools are not appropriate enough to explore efficiently such a large design space as the one brought about by modern SoCs. In order to cope with the design complexity of such embedded systems, System Level Design (SLD) methods have been proposed by the system design community in order to complement the traditional methods and to improve the productivity of the system designers.

Harnessing the benefits of working at system level, Design Space Exploration (DSE) is becoming a key task of SLD. Early DSE is of paramount importance, as early design choices heavily influence the success or failure of the final product, and can avoid wasting time and effort in further design steps without the possibility of meeting design requirements because of the inappropriate choices of design decisions.

Aiming at facilitating designers to explore a wide range of design options during the early design stages, an important number of system-level modelling and simulation tools has emerged. However, these tools only provide a partial solution, since the process of system-level DSE logically requires three interdependent components: (i) the search method to systematically travel through the design space, (ii) the evaluation technique to assess the quality (in terms of system-wide metrics) of each design point selected by the search method, and (iii) a mechanism to generate the system description that is used by the evaluation technique to provide the needed system-wide metrics. Although many DSE approaches have been proposed in the literature, to the best of our knowledge, there does not exist a generic infrastructure to facilitate and support system-level MPSoC DSE experiments. This calls for a unified framework integrating and coupling all the aforementioned components to systematically explore large design spaces, as well as new methodologies for performing DSE experiments in a time and effort efficient way.

This thesis provides some contributions that help addressing the above challenges, being our ultimate goals to reduce the effort of system designers in the design process as well as to improve the efficiency of the DSE in the early design stages. To this end we have contributed the following advances:

1) We have developed and proposed a novel *dimension-oriented DSE methodology* for co-exploring multidimensional design spaces. The idea underlying this approach is that a large design space can be decomposed/separated in design space dimensions, such that system designers can *select different combinations of (tailored) search strategies for exploring each dimension*. Two main advantages can be highlighted in our methodology. On one hand, it is flexible to be applied in different DSE experiments, where designers can choose to *simultaneously co-explore all dimensions*, *or to fix one* or more of them and to focus the exploration on just one or two dimensions. On the other hand, our methodology is *extensible*, since additional dimensions can be defined and incorporated in the design space if larger and/or more complex design spaces need to be explored.

2) We have implemented *a tool to automatically generate a large variety of architecture models*. In fact, as illustrated in Chapter 2, in order to evaluate each design point in a system-level simulator, a simulatable system model should be created first. Setting up and/or creating manually such a system model can be a very error-prone, time-consuming and labour-intensive and expensive task for the system designer, being therefore one of the bottlenecks in improving the designer's productivity, and severely limiting the amount of the design space that can be explored in a reasonable time. Thus, automating this process could definitely free designers from the effort to manually create such models, and could improve significantly the design productivity and efficiency. The experimental results illustrated in Chapter 3 have demonstrated that our *architectural platform generator* can provide a wide range of architectural alternatives to be explored in each DSE experiment, which enables designers to *identify/detect an optimal architecture (or a set of optima architectures) for implementing a target application*.

3) We have also developed the *NASA infrastructure, which is a generic and modular framework to facilitate and support system-level DSE experiments*. Unlike existing DSE approaches, NASA is not targeted to a particular DSE methodology, neither to a specific system-level simulation tool. In fact, NASA is a *unified framework* that can support our dimension-oriented DSE methodology, that allows for integrating our generator of architectural models and *enables designers to incorporate different system-level simulation tools as well as different combinations of search methods by means of a simple plug-in mechanism*. As a result, *NASA provides a flexible and re-usable environment to explore the multidimensional design spaces in a systematic and automatic way*.

4) In order to *illustrate, validate and assess* the capabilities and key properties of NASA framework, we deployed several sets of DSE experiments. The *experimental results* shown in this thesis demonstrate three major features of NASA: *flexibility* in the search process, *speed* of the tool, and *quality* of the design solutions delivered. NASA's flexibility to carry out fast DSE is shown with different kinds of search algorithms such as random search, heuristic-based search and genetic algorithms (GA). Moreover, DSE experiments conducted with several search algorithms show that NASA configurations *using multiple genetic algorithms (i.e., following our multidimensional co-exploration methodology) can find better design solutions and explore larger design spaces* as compared to more traditional approaches of using a single genetic algorithm for all dimensions.

5) We have also proposed a *hierarchical DSE methodology to address the problem of mapping a real-time application onto a target MPSoC platform*. This approach consists of two independent phases communicating via text files interface. The first phase is based on *analytical methods*, i.e., this phase does not use simulations, rather uses a set of heuristic-based algorithms to explore rapidly the design space associated to the sub-problems of partitioning, scheduling and assigning, respectively. Moreover, an additional analytical model is used in this phase to *estimate roughly the performance of each design point* (taking into account both deterministic and non-deterministic system behaviours), and thus *pruning dramatically the initial design space*, such that only a reduced number of potential mapping solutions is given as output. Subsequently, by taking into account several additional design decisions that are not considered in the analytical models, these *potential solutions are evaluated accurately in a system-level simulation tool* until a solution that meets the user's constraints is found.

6) Finally, in order to prove the benefits of our hierarchical DSE methodology, several sets of DSE experiments have been performed by *plugging the proposed analytical methods into NASA framework*. The results obtained in these experiments reveal that, compared to DSE experiments based only on analytical estimation and/or simulations, a hierarchical DSE methodology using our analytical methods not only *can improve the efficiency of DSE by two orders of magnitude, but also can reach higher quality mapping solutions*. Moreover, these experiments also validate the flexibility and capability of NASA framework for supporting different DSE

methodologies, as well as to address multi-objective optimization problems in a time-efficient and accurate way.

Nevertheless, we would like to stress that the sets of experiments illustrated in this thesis are aimed to prove and illustrate the key concepts/properties of our methodology. We consider that these proof-of-concept DSE experiments are quite complete to validate our framework. Although throughout the thesis we have used as reference real-time application a challenging visual object tracking algorithm, all concepts and methods development have been carried out without loss of generality. Following the overall direction established in this thesis, our further goal in this research line is to perform more experiments to demonstrate and validate progressively the capabilities of our methodology for different real-time applications, and even different application domains, and across an extensive range of MPSoC platforms.

5.2 Future work

- *More parallelization at algorithmic levels.* The automatic parallelization of sequential program code is still an open issue, and it has not been addressed in the current version of CASSE tool neither in the NASA framework. Therefore, the designer is required to manually parallelize the algorithm based on the knowledge about the algorithm and/or on their experience/expertise. A possible future work would be to implement a generator which should be capable to automatically parallelize –at different granularity levels– a sequential source code into parallel loops, task/function graphs and/or Kahn Process Networks. As a consequence, the efficiency of each DSE experiment as well as the productivity of system designers will improve in a significant way.

- *More primary design objectives.* DSE experiments shown in this thesis have demonstrated that our framework can explore large design spaces and reach optimal design solutions considering several design objectives. However, we explicitly have considered *only one primary design objective (i.e., performance)* in our experiments mainly due to the current limitation of CASSE tool. Therefore, more DSE experiments should be performed in order to prove the capability of our framework to address multi-objective optimization problems, i.e., considering additionally *other primary design objectives such as power consumption and area/cost*, and provide to designers a set of Pareto-optimal design points within the explored design space.

- *More simulators.* In order to cope with more design objectives, other system-level simulation tools being capable of providing multiple system-wide metrics (e.g., power, cost/area) can be plugged into NASA. Since one goal of NASA framework is to facilitate the integration of different system-level simulators, this proposed future work will also enable to see and quantitatively assess how much work is required to couple a new simulation tool in NASA.

- *More design space dimensions.* We have claimed that one of the strengths of NASA infrastructure lies in its flexibility, but this latter has been validated and assessed only partially. That is, we have demonstrated its ability to integrate different search techniques with empirical DSE experiments in a three dimensions design space. However, additional deployment case studies should be performed in order to, for instance, prove the benefits of our dimension-oriented DSE approach. In this case, an interesting future work could be to analyse the quality of the DSE process when new (and/or additional) design space dimensions are introduced into NASA.

- *Better modelling*. Finally, it is worth to underline that this work is located in a context of higher abstraction level, where the results obtained in these DSE experiments are based on abstract models of systems, and the accuracy of the evaluations depends directly on the simulation tool and/or analytical estimation model used for each case study. These models need to have been properly calibrated. As a consequence, system designers still need to further validate these models by means of, for example, comparing the estimated and evaluated results to results obtained in actual implementations of the system models in physical hardware MPSoC platforms (e.g., MPSoCs in FPGA, ASICs, or custom silicon targets). This refinement process towards lower abstraction system models is not addressed in the current version of NASA framework since it is behind the scope of this thesis. Therefore, other interesting future work would be to extend the NASA framework by adding additional capabilities such as gradual model refinement, automated synthesis, and RTL generation towards a defined target technology.

Chapter 6
Resumen en español

En este capítulo presentamos un resumen en español del contenido de esta tesis.

Resumen

A medida que la capacidad de integración en chip aumenta, los sistemas en chip (SoC) se vuelven cada vez más complejos, siendo bastante habitual en la actualidad encontrarnos con SoCs que integran una gran variedad de elementos de procesamiento, memorias, dispositivos I/O y elementos de comunicación. Para hacer frente a la complejidad de diseño de los modernos SoCs, los diseñadores de sistemas propusieron elevar el nivel de abstracción del proceso de diseño al nivel de sistema, donde la exploración del espacio de diseño (DSE) se ha convertido en una pieza clave en el proceso de diseño a nivel de sistema (SLD). Sin embargo, cabría preguntarse en este contexto si las metodologías de diseño existentes permiten al diseñador explotar todo el beneficio potencial de la DSE a nivel de sistema, o si se deberían plantear nuevas metodologías y/o técnicas alternativas para sacar el máximo provecho del SLD.

Esta tesis pretende precisamente responder a dicha cuestión. Concretamente, hemos desarrollo nuevas metodologías y novedosos algoritmos con el objetivo de aminorar el esfuerzo del diseñador de sistemas y lograr eficientes DSE en la etapa temprana del proceso de diseño. Asimismo, con el fin de validar nuestras técnicas y esquemas de trabajo, también hemos presentado una importante cantidad de experimentos de DSE en esta tesis. Estos resultados experimentales demuestran que, en comparación con las metodologías tradicionales, nuestras propuestas no sólo pueden mejorar la productividad del diseñador y la eficiencia de DSE a nivel de sistema, sino que también son capaces de obtener diseños de mayor calidad.

6.1 Introducción

El mundo está cambiando. Este hecho es aplicable a una gran variedad de aspectos de nuestro entorno, pero desde mi punto de vista, es en el ámbito tecnológico donde toma su máxima expresión. Sin embargo, a pesar de que los avances tecnológicos están introduciendo cambios en nuestras vidas, aportando nuevas fórmulas y soluciones optimizadas, también abren puertas a nuevos desafíos que esperan resolución.

Desde el punto de vista de los *clientes tecnológicos* (ya sean éstos los consumidores finales o consumidores intermedios), exigen cada vez más innovación y más funcionalidades. Este *apetito feroz* por lo más innovador no se debe a otra cosa que el propio avance de la tecnología, provocando una situación paradójica conocida como *ciclo vicioso*. Es decir, los consumidores siempre están demandando productos innovadores, porque lo que hace poco era innovador pasa a considerarse obsoleto tras un cierto periodo de tiempo relativamente corto, con lo cual el fabricante está obligado a recurrir al ingenio y a buscar soluciones para satisfacer estas necesidades, que terminan casi siempre colocando en el mercado un producto nuevo o una versión mejorada; y así se repite este mismo ciclo una y otra vez.

Desde el punto de vista de los *productores tecnológicos*, que abarcan desde las instituciones públicas y privadas hasta los fabricantes (es decir, todos aquellos que contribuyen a estos avances), cabrían dos tipos de reflexiones. Por un lado, para aquellos situados en la orilla empresarial de este sector, el anterior ciclo vicioso se convierte en un *ciclo virtuoso*, puesto que la constante demanda del mercado se traduce en unos clientes fieles y mercados cautivos, y en definitiva, les garantizan tener siempre un *trozo del pastel*. No obstante, desde el punto de vista de los *verdaderos productores tecnológicos*, es decir, las cabezas pensantes, el ciclo vicioso representa un verdadero desafío.

Por otro lado, este desafío no sólo proviene desde el lado de la demanda, sino también existen presiones desde el lado de la oferta. Esto último se traduce en que los diseñadores no sólo deben dar con la solución del problema planteado por los consumidores, sino que también dicha solución debe satisfacer al mismo tiempo unas restricciones empresariales (costes, *time-to-market*, tecnología disponible, economía de escala/alcance, etc.) Desafortunadamente, los modelos tradicionales de diseño

presentan cada vez más dificultades para conseguir la convergencia de ambas fuerzas, y consecuentemente, deben considerarse nuevas metodologías y herramientas de trabajo.

6.1.1 Diseño a nivel de sistema

6.1.1.1 Ventajas del diseño a nivel de sistema

Como se mencionó en el apartado anterior, las constantes demandas de nuevas funcionalidades exigen que los diseños sean cada vez más complejos, y esto último se traduce en lo que respecta a los diseñadores en *escribir más líneas de código*. Esto es, lo que antes era un componente muy sencillo, ahora se convierte en un procesador, y lo que inicialmente era un aglomerado de componentes pasa a ser un sistema multi-procesador completo. No obstante, los esfuerzos de los diseñadores no sólo se centran en el diseño, sino también en el tiempo dedicado a la simulación y la verificación. Si a esto último se le suma que la innovación es constante, entonces esa dimensión del esfuerzo se magnifica en términos descomunales.

En respuesta a estos desafíos, numerosos investigadores propusieron elevar el nivel de abstracción para el modelado, simulación y verificación del diseño. Trabajar a un mayor nivel de abstracción (pasando del RTL a nivel de sistema), aparte de manejar las informaciones con un mayor nivel de granularidad, ciertamente presenta otras importantes ventajas para el diseñador:

Reducción del esfuerzo. A nivel de sistema, el diseñador puede trabajar directamente con macro-bloques o componentes predefinidos, los cuales muchas veces son proporcionados por las propias librerías de las herramientas de trabajo, como ya se explicará más adelante. Estos componentes ya prediseñados (y verificados) proporcionan las prestaciones y funcionalidades necesarias al diseñador. Por consiguiente, con ellos los diseñadores pueden centrar sus esfuerzos en configurar y organizar el conjunto de macro-bloques para que realicen correctamente sus cometidos, ahorrando así tiempo y evitando duplicar esfuerzos innecesarios.

Mayor flexibilidad. Los componentes de las librerías son básicamente plantillas de diferentes tipos de recursos que podemos encontrar en cualquier sistema. Entre otros se tratan de elementos de procesamiento (PEs) que modelan el comportamiento de los procesadores, DSP, coprocesadores y aceleradores. Asimismo, también se pueden encontrar los elementos de almacenamiento (SEs), que pueden configurarse como diversos tipos de memorias. Finalmente, los elementos de comunicación (NEs) prestan

servicios de comunicación para los elementos que están unidos a ellos. Como todos estos elementos son altamente configurables a través de los parámetros asociados a cada uno ellos, podemos derivar fácilmente desde un diseño a otro sin más que reconfigurar dichos parámetros.

Abstracción de los detalles. Cuando el sistema crece, crece también el conjunto de detalles que el diseñador debe tener presente. Por ejemplo, cuando se modela un sistema en chip (SoC) compuesto de varios PEs, numerosas SEs y una estructura de comunicación multi-bus, el trabajo del diseñador de sistema está orientado a que el conjunto de estos elementos tengan el comportamiento esperado, es decir, que los PEs transfieran sus datos correctamente a las SEs que tienen acceso y asegurar el uso correcto del NE compartido evitando los problemas de contención. En definitiva, se centran en las informaciones a nivel de bloques y no tanto en lo que suceden dentro de cada uno de ellos.

Menor tiempo de simulación. Para trabajar a un mayor nivel de abstracción, también se precisa una nueva generación de herramientas de diseño y simulación para explotar al máximo todas las ventajas comentadas. Uno de los aspectos a destacar de estas herramientas es su esquema de simulación. A diferencia de los tradicionales simuladores para RTL, estos nuevos simuladores trabajan a nivel de transacciones (TLM). Trabajando a este nivel, el diseñador puede abstraerse de numerosos detalles de implementación, dando como resultado que las simulaciones sean mucho más rápidas.

Todas estas características justifican en gran medida el por qué de la emigración del nivel de trabajo o modelado de los diseñadores de sistema. No obstante, si bien es verdad que estas contribuciones pueden facilitar y acelerar el proceso de diseño, también implican nuevos desafíos no estudiados hasta la fecha, obligando a los investigadores a ofrecer soluciones acordes a estas exigencias.

6.1.1.2 Metodologías de diseño a nivel de sistema

Con el incremento de la complejidad de los sistemas se pone en evidencia cada vez más la ineficiencia de los esquemas basados en el flujo de co-diseño hardware/software, que si bien su aplicación sigue siendo posible en los complejos SoCs actuales, el esfuerzo y el coste requerido hace que esta opción sea cada vez menos atractiva.

Como consecuencia, en los últimos años se han seguido tres estrategias en la construcción de flujos de diseño de estos sistemas siguiendo el mismo tipo de pautas que, anteriormente se empleaban a nivel RTL: flujos de diseño *top-down*, *meet-in-the-middle*, y *bottom-up*.

La metodología basada en el flujo de diseño *top-down* sigue una trayectoria descendente. En ella, partiendo de unos requerimientos y especificaciones del sistema, el diseñador modela el sistema y, a continuación, va refinando progresivamente desde ese modelo *lógico* hasta llegar a una implementación física.

En cambio, el flujo de diseño *bottom-up* es justo lo contrario que el caso anterior. En este caso, el diseñador parte de una cierta implementación física ya definida, y va abstrayendo hacia niveles de abstracción superiores hasta encontrar con la aplicación que mejor se ajusta a estas características físicas.

Finalmente, el flujo *meet-in-the-middle* es un caso intermedio entre los dos anteriores, es decir, no se trata de un *top-down* ni de un *bottom-up* en sentido estricto. En este caso, el diseñador también parte de unos requerimientos y especificaciones de alto nivel, pero a su vez dispone de una determinada arquitectura configurable, por lo que el diseñador refina por un lado la aplicación y por otro va subiendo de nivel de abstracción, hasta que ambos procesos se encuentren en un punto intermedio.

Formalmente, existen unos conceptos que subyacen tras cada una de estas metodologías. En este ámbito se puede hacer referencia a la síntesis de alto nivel, el principio de la ortogonalización y el diseño basado en componentes. En el proceso de síntesis, el diseñador añade progresivamente detalles de diseño hasta lograr una solución física concreta. A pesar de utilizarse ampliamente en la actualidad, el principal inconveniente es su carácter *ad-hoc*, careciendo de la flexibilidad y reconfigurabilidad que comentábamos en los párrafos anteriores. Al mismo tiempo, la reciente tendencia de diseño basado en componentes aprovecha la disponibilidad de un conjunto de elementos ya predefinidos, y así, mediante la composición con los mismos, se obtiene el diseño del sistema que mejor se ajusta a las especificaciones de la aplicación. Finalmente, el concepto de ortogonalización propone la separación de la funcionalidad y la implementación y la computación de la comunicación, permitiendo de esta forma mayor libertad e independencia entre las diferentes etapas del proceso de diseño, ganando un tiempo valiosísimo con este planteamiento de trabajo en paralelo.

6.1.2 Exploración del espacio de diseño a nivel de sistema

6.1.2.1 Los objetivos de la exploración del espacio de diseño

Otra consecuencia del incremento de la capacidad de integración en chip es el crecimiento exponencial del espacio de diseño que el diseñador de sistema debe gestionar. En este contexto, un espacio de diseño se compone de un conjunto de puntos de diseño, donde cada uno de ellos representa una solución de diseño definido por diferentes *decisiones u opciones* de diseño, tales como la partición HW/SW, el mapeo de la aplicación, la topología de la plataforma arquitectural, el número y tipo de cada uno de los componentes de la arquitectura, etc. Evidentemente, cuanto mayor es el número de decisiones de diseño que debe tener en cuenta el diseñador, mayor será el espacio de diseño a explorar, y por consiguiente, mayor será el esfuerzo que conlleva al diseñador para encontrar una solución óptima de diseño.

Precisamente debido a todo lo comentado anteriormente, podemos afirmar que la exploración del espacio de diseño (DSE) a nivel de sistema se ha convertido en una tarea clave para el modelado y diseño de los sistemas empotrados modernos. A grandes rasgos, los principales objetivos del DSE a nivel de sistema es explorar grandes espacios de diseño de una forma eficiente y eficaz, y analizar distintos puntos de diseño en términos de un conjunto de métricas (tales como consumo de potencia, costes/área y prestaciones) con el fin de encontrar la solución óptima de diseño que satisfaga los objetivos especificados por el diseñador. Por otra parte, además de explorar una amplia gama de opciones de diseño, la DSE también permite que los diseñadores puedan analizar el impacto o la influencia de diferentes decisiones de diseño en el comportamiento global del sistema y, de esta forma, tomar las decisiones de optimización mucho antes de que el diseño final esté disponible.

Por lo tanto, podemos afirmar que una correcta DSE en una etapa temprana de diseño tiene suma relevancia en el éxito o el fracaso del producto final. Además, contribuye también al ahorro de tiempo y esfuerzo en el proceso de diseño, puesto que evita que los diseñadores tengan que trabajar sobre unas soluciones sin posibilidad de ajustarlas a los requerimientos de diseño debido a decisiones de diseño inadecuadas de partida.

6.1.2.2 Los componentes de la exploración del espacio de diseño

El proceso de DSE a nivel de sistema consiste en tres componentes interdependientes: (i) el método de búsqueda para viajar por el espacio de diseño de forma sistemática, (ii) la técnica de evaluación para analizar la calidad de cada punto de diseño seleccionado

por el método de búsqueda, y (iii) un mecanismo para generar la descripción del sistema que utiliza la técnica de evaluación para proporcionar las métricas necesarias. La interdependencia resultante entre estos tres componentes se muestra en la Fig. 1.1.

6.1.2.2.1 Los métodos de búsqueda

Se pueden mencionar dos tipos de métodos: estrategias para cubrir el espacio de diseño y los métodos que podan el espacio de diseño. Cabe señalar que ambas categorías pueden aplicarse de forma independiente o conjuntamente para llevar a cabo una DSE. De hecho, el Capítulo 4 de esta tesis presenta una metodología para DSE a nivel de sistema que combina técnicas pertenecientes a ambas categorías con el fin de mejorar la eficiencia de la DSE, como se explicará más adelante.

Estrategias para cubrir el espacio de diseño

Se pueden mencionar tres tipos de estrategias en este bloque: búsqueda exhaustiva, al azar y guiada. Mientras que las dos primeras técnicas se suelen implementar mediante un *script* de control específico para cada experimento de DSE (lo que les hacen inflexibles y difícilmente reutilizables), la búsqueda guiada se basa normalmente en la heurística.

- *Evaluar de forma exhaustiva todos los puntos de diseño.* Este enfoque evalúa todas las combinaciones posibles de los parámetros de diseño, por lo que es prohibitivo para aplicarse en grandes espacios del diseño. Aunque el espacio de diseño se puede reducir limitando la gama de parámetros de diseño, este tipo de búsqueda se lleva acabo sin ningún tipo de guía ni tampoco teniendo en cuenta las preferencias del diseñador.

- *Muestreo aleatorio del espacio de diseño*. Seleccionar y evaluar muestras aleatorias es una opción apropiada para explorar grandes espacios del diseño e intentar identificar tendencias. Además, este tipo de técnica también tiene la ventaja de que presenta una perspectiva imparcial sobre las características del espacio de diseño.

- *Incorporar conocimientos acerca del espacio de diseño.* Las estrategias de búsqueda en esta categoría incorporan conocimientos acerca del espacio de diseño en el proceso de búsqueda, con el fin de asegurar la convergencia hacia soluciones óptimas. El conocimiento puede actualizarse en cada iteración del proceso de búsqueda o puede ser una característica inherente del algoritmo de búsqueda.

Métodos que podan el diseño del espacio

Todos los métodos de exploración mencionados anteriormente pueden emplearse conjuntamente con las técnicas presentadas en este apartado, con el fin de reducir la complejidad de la búsqueda.

- *Exploración jerárquica.* Normalmente, este proceso se centra en eliminar rápidamente aquellos puntos de diseño que no pueden satisfacer las especificaciones de diseño, en lugar de asegurar una evaluación detallada de cada alternativa de diseño. En consecuencia, el conjunto de soluciones candidatas seleccionadas por el proceso anterior se evalúan con un mayor nivel de detalle por las herramientas de simulación en una fase posterior.

- *Submuestreo del espacio de diseño.* El submuestreo del espacio de diseño es una opción razonable si el diseñador está interesado en una DSE objetiva o cuando la búsqueda exhaustiva es prohibitiva. El patrón de submuestreo podría establecerse completamente al azar, basándose en algún tipo de patrón o función objetivo.

6.1.2.2.2 Las técnicas de evaluación

La esencia de las técnicas de evaluación es proporcionar un conjunto de métricas sobre la calidad de los sistemas o soluciones de diseño a los métodos de búsqueda y diseñadores, con el fin de que éstos últimos influyan en la toma de decisiones en el proceso de búsqueda de DSE. Para la DSE a nivel de sistema, se pueden emplear dos tipos de técnicas para evaluar cada punto de diseño: evaluación basada en la simulación y estimación basada en enfoques analíticos.

Evaluación basada en simulación

Las simulaciones son muy apropiadas para analizar los comportamientos dinámicos y esporádicos que ocurren en el sistema. Un claro ejemplo es el efecto de la contención, el cual suele producirse si varios PEs del sistema compiten simultáneamente por hacerse con el uso de un mismo recurso compartido. Esta cualidad es lo que precisamente permite que las evaluaciones basadas en simulaciones puedan revelar resultados más realistas con respecto a los obtenidos en las estimaciones analíticas.

Estimación basada en enfoques analíticos

Los métodos de estimación analítica son idóneos para aquellos casos en los que el sistema presente un comportamiento determinista, o que WCET es una suposición razonable para el sistema objeto de evaluación. A diferencia de la evaluación basada en

simulación, los modelos analíticos se caracterizan por un tiempo de ejecución corta. Sin embargo, el principal inconveniente de este enfoque sigue siendo la falta de precisión en la evaluación. Una revisión de los métodos de estimación analítica se puede encontrar en el capítulo 4 de esta tesis.

6.1.2.2.3 El generador de la descripción del sistema

Independientemente de la naturaleza de las técnicas de evaluación (es decir, utilizar la simulación o estimación analítica), en primer lugar se debe crear un modelo de sistema para evaluarlo adecuadamente. En particular, este último se convierte en un requisito cuasi-indispensable si las evaluaciones se basan en la simulación. En estos casos, para realizar la simulación y obtener información del sistema (entre las que se incluyen las medidas o métricas), las herramientas de simulación suelen requerir un archivo ejecutable del modelo de sistema (ESM), que puede considerarse como un modelo virtual o meta-modelo del sistema. Dicho modelo virtual no sólo debe incluir información acerca de la funcionalidad de la aplicación y de los detalles de la arquitectura, sino especificar también cómo ambos dominios se *fusionan*.

En este punto, cabe hacer hincapié en que el principio del Y-Chart facilita en gran medida la concepción y creación de estos ESM. De acuerdo con este principio, un sistema (y en nuestro caso, un ESM) se puede especificar mediante la combinación de tres modelos: un modelo de la aplicación, un modelo de la arquitectura y un modelo de mapeo. Esto último significa que el principio de Y-Chart separa explícitamente el modelado de la aplicación y el de la arquitectura. El modelo de la aplicación describe el comportamiento funcional de la aplicación utilizando, por ejemplo, redes de procesos de Kahn (KPN) o grafos de tareas. Al mismo tiempo, un modelo de la arquitectura especifica el diseño arquitectural (por medio de los componentes predefinidos y altamente configurables proporcionados por la biblioteca de las herramientas) y captura sus rendimientos o prestaciones. Por último, otro fichero describe cómo se mapea el modelo de la aplicación en el modelo de la arquitectura para su co-simulación con el fin de obtener métricas del sistema en análisis posteriores.

Comparado con otras propuestas tradicionales (como FPGA o simulaciones RTL), un ESM que se crea siguiendo el principio de Y-Chart presenta varias ventajas:

- **El diseñador puede disponer del ESM en una fase muy temprana del proceso de diseño.** Esta es una consecuencia implícita de trabajar a un nivel de abstracción

más alto que RTL, lo que implica estar pendiente de menos detalles de implementación y por tanto, se puede modelar los diseños del sistema con menor esfuerzo y tiempo.

- **El ESM representa una solución más flexible.** Dado que el meta-modelo del sistema contiene menos detalles y se construye de un modo modular, las modificaciones y optimizaciones pueden hacerse con suma rapidez, y los nuevos modelos pueden obtenerse en menos tiempo que con las otras soluciones tradicionales.

Por último, es oportuno advertir que tales ESM se pueden crear manualmente o automáticamente. Sin embargo, el proceso de la creación manual (y la configuración de cada modelo) puede ser una tarea muy propensa a errores, aparte de que requiere mucho esfuerzo y tiempo para el diseñador del sistema. Por tanto, creemos que la automatización de este proceso podría mejorar notablemente la productividad y la eficiencia de diseño DSE. Una propuesta para generar ESMs de forma automática se presenta en el capítulo 3.

6.1.3 Contribuciones de la tesis

6.1.3.1 Objetivos

Esta tesis se centra en las metodologías y las técnicas para la realización de la exploración del espacio de diseño de MPSoC a nivel de sistema de una forma eficiente y automática. Más específicamente, se proponen nuevas herramientas, métodos y algoritmos que facilitan la labor y reducen el esfuerzo y el tiempo dedicado por los diseñadores en la tarea de la DSE durante las etapas tempranas del diseño. En consecuencia, el objetivo último de esta tesis persigue ofrecer los marcos teóricos y prácticos para hacer frente a la complejidad de diseño de los SoCs modernos, mejorando así la eficiencia y la productividad de los diseñadores en el proceso de diseño a nivel de sistema.

6.1.3.2 Aportaciones originales

Las principales aportaciones de esta tesis son las siguientes:

- Se propone una infraestructura o herramienta software que proporciona el soporte necesario para la exploración del espacio de diseño de MPSoC a nivel de sistema. El

nombre de esta herramienta es NASA. La composición modular es la principal característica de NASA, la cual utiliza un conjunto de interfaces bien definidos para integrar, mediante el simple mecanismo de *plug-and-play*, distintas estrategias de búsqueda y diferentes herramientas de simulación a nivel de sistema en un mismo entorno de trabajo. Como consecuencia, se puede realizar cada experimento de DSE sin la necesidad de preparar scripts a la medida de cada experimento, sino que el diseñador simplemente tiene que configurar y especificar los valores apropiados en ficheros de entrada.

- Se propone un método para generar de forma gradual y automática el ESM requerido por los simuladores para la obtención de las métricas del sistema para evaluar las decisiones de diseño. De este modo, la totalidad del proceso de DSE (compuesto por la búsqueda, la generación de ESM y la evaluación de cada punto de diseño) se puede realizar de una manera automática y sistemática, mejorando así la productividad del proceso de diseño y la disminuyendo los esfuerzos del diseñador.

- Se presenta una metodología para la DSE a nivel de sistema llamada *DSE basada en las dimensiones del espacio de diseño*. De acuerdo con este enfoque, se puede desglosar explícitamente un gran espacio de diseño en diferentes dimensiones, las cuales podrían representar diferentes grupos de decisiones u opciones de diseño (ortogonales entre sí), como por ejemplo, el mapeo, los componentes arquitecturales y la topología de la plataforma. De este modo, el diseñador puede optar por explorar simultáneamente todas las dimensiones o *fijar* una o varias dimensiones y dedicarse a la exploración de las otras dimensiones. O dicho con otras palabras, el diseñador puede configurar el número y el tipo del algoritmo de búsqueda para *co-explorar* las diferentes dimensiones del espacio de diseño.

- Se han desarrollado unos algoritmos heurísticos para podar el espacio de diseño del mapeo de aplicaciones con estrictas restricciones de tiempo real en plataformas MPSoC. Concretamente, se propone un modelo de estimación analítico que no sólo es capaz de considerar múltiples objetivos de diseño durante el proceso de mapeo, sino que también puede evaluar cada punto de diseño teniendo en cuenta los comportamientos estáticos y dinámicos del sistema. Por otra parte, estos métodos analíticos forman parte de *una metodología de DSE jerárquica*. Esta metodología utiliza algoritmos heurísticos y el modelo de estimación (mencionados anteriormente) para eliminar rápidamente un gran número de puntos de diseño en la fase de poda,

mientras que el reducido número de soluciones remanentes las evalúa un simulador a nivel de sistema en una segunda fase. Finalmente, nuestros resultados experimentales revelan que esta metodología de DSE jerárquica puede mejorar significativamente la eficacia de DSE, así como encontrar soluciones de mapeo de mayor calidad.

6.1.4 Estructura de la tesis

En el capítulo 2 presentamos CASSE, un entorno de modelado y simulación a nivel de sistema basado en SystemC que se ha utilizado a lo largo de este trabajo. CASSE, desarrollado por los investigadores del IUMA en ULPGC, desempeña un papel clave en las metodologías propuestas en esta tesis, puesto que es la herramienta que nos permite modelar y simular las diferentes soluciones de diseño, así como proporcionar las métricas necesarias para nuestros experimentos de DSE. Por ello, presentamos en primer lugar los conceptos clave de CASSE, tales como los métodos aplicados para modelar de forma separada la aplicación de la arquitectura, la técnica de simulación, etc. Luego, explicamos cuáles son los aspectos de las interfaces de CASSE que un diseñador de sistemas debe manejar con el fin de crear y configurar un ESM. Por último, con el fin de demostrar las capacidades de CASSE para llevar a cabo diversos tipos de experimentos de DSE, concluimos este capítulo presentando un caso de estudio.

El capítulo 3 está dedicado a NASA y a la metodología de DSE basada en las dimensiones del espacio de diseño. En primer lugar, repasamos las principales limitaciones de las metodologías actuales de la DSE. A continuación, presentamos el panorama general de NASA y sus claves para superar las debilidades detectadas en los otros enfoques existentes. Posteriormente, entramos en detalle en la implementación de los diferentes componentes (o módulos) de NASA, y particularmente, nos centramos en explicar el concepto de DSE basado en dimensiones del espacio de diseño, y cómo se ha implementado dicha metodología en nuestro trabajo. Por último, con el fin de demostrar las capacidades de NASA e ilustrar los beneficios de nuestra metodología de DSE basada en las dimensiones del espacio de diseño, presentamos también un amplio repertorio de experimentos, donde analizamos y comparamos los resultados obtenidos en los experimentos basados en nuestro enfoque con los obtenidos siguiendo otras estrategias tradicionales de DSE.

El capítulo 4 se centra en la estrategia de DSE jerárquica. En primer lugar, analizamos las fortalezas y debilidades de DSE que utiliza técnicas de evaluación basadas en

simulación o modelo de estimación analítica. A continuación, presentamos una visión general de nuestra metodología de DSE jerárquica, y definimos los conceptos y el enunciado del problema que vamos a tratar en este capítulo. Posteriormente, explicamos cómo se ha implementado los distintos algoritmos de la metodología (tanto en la fase de estimación como en la fase de simulación) en nuestro trabajo. Asimismo, explicamos con una serie de ejemplos los distintos aspectos de nuestros algoritmos con el fin de proporcionar una mejor comprensión de los mismos. Por último, concluimos este capítulo con un conjunto de experimentos, donde comparamos la eficiencia y la calidad de los resultados obtenidos en los experimentos basados en nuestra metodología de DSE jerárquica y los resultados obtenidos siguiendo otras estrategias tradicionales de DSE.

Finalmente, en el capítulo 5 presentamos las principales conclusiones obtenidas en esta tesis, y también señalamos las líneas de investigación que pueden extender este trabajo en el futuro.

6.2 CASSE: entorno de modelado y simulación a nivel de sistema

6.2.1 Introducción

En una era tecnológica como ésta en la que estamos viviendo hoy en día, se hace impensable todo lo comentado hasta ahora sin el uso de las herramientas CAD. Es lógico pensar que conjuntamente con la aparición de los nuevos marcos teóricos y metodologías de diseño, aparezcan también las herramientas necesarias para dar soportes a dichos conceptos. De lo contrario, cabe preguntarse qué sentido tiene formular principios que luego no se pueden llevar a la práctica.

Actualmente, existe un número considerable de herramientas que permiten al diseñador trabajar desde un mayor nivel de abstracción. No obstante, cada herramienta tiene sus características propias y unas singularidades que les hacen diferentes una de las otras. A modo de ejemplo, presentan notables diferencias el lenguaje utilizado para el modelado de la aplicación, el flujo de diseño permitido, la librería de componentes al alcance del diseñador, los parámetros del sistema factibles de ser evaluados durante las simulaciones, el motor de simulación utilizado, el grado de automatización del proceso en su conjunto, etc. Obviamente, estas mismas características son las que a su vez les hacen diferentes una de las otras.

Volviendo a la presentación de las herramientas de simulación, el otro punto importante de la controversia es el grado de integración de las diferentes etapas del proceso de diseño en cada una de estas herramientas. Por ejemplo, las herramientas que trabajan a nivel de RTL se caracterizan por un alto nivel de automatización, llegando (en algunos casos) a concentrar todas las etapas del proceso en un único entorno de trabajo, es decir, desde la especificación hasta el *layout* de la implementación final, pasando por el diseño, síntesis, simulación y verificación. La ventaja de trabajar de este modo radica en que, por un lado, el diseñador no tiene que estar manejando simultáneamente varias herramientas CAD, las cuales muchas veces tienen unas interfaces radicalmente diferentes. Por otro lado, esta cadena de trabajo automatizada puede concatenar la salida de fase anterior con la entrada de la siguiente, proporcionando los formatos necesarios y evitando así posibles errores humanos en los pasos de unas herramientas a otras.

6.2.2 Herramientas de simulación a nivel de sistema basadas en SystemC

Como se mencionó en los bloques anteriores, un punto de diseño puede ser evaluado por las herramientas de simulación o modelos de estimación analítica, y la principal diferencia entre ambas técnicas es el grado de precisión lograda en la evaluación. Por un lado, es bien conocido que las herramientas de simulación permiten a los diseñadores controlar completamente el proceso de DSE en cada experimento. Por ejemplo, estas herramientas garantizan que dos esquemas de arbitraje se ejecuten exactamente bajo las mismas condiciones. Por otra parte, los simuladores son cada vez más realistas, permitiendo incorporar cada vez más parámetros de diseño en el modelo del sistema. Por ejemplo, actualmente existen entornos de evaluación donde el diseñador puede configurar diferentes topologías de arquitecturas de comunicación, modelar un SoC con varios dominios de reloj y ancho de banda de red, especificar el número y el tipo de componentes arquitecturales a utilizar en cada modelo de sistema, etc. Además, los simuladores también permiten evaluar de un modo más preciso los efectos de la contención en los recursos compartidos del sistema. En resumen, todas estas razones justifican en gran medida porqué las herramientas de simulación han sido ampliamente utilizadas en los experimentos de DSE a nivel de sistema.

Por otra parte, a pesar de que en la actualidad existen un gran número de herramientas basadas en diferentes lenguajes de modelado a alto nivel (por ejemplo, SystemC, SpecC, o SystemVerilog), se ha visto que poco a poco está proliferando una mayoría de herramientas basadas en SystemC y TLM. Este hecho se debe en gran medida a iniciativas como OSCI, compuesta por trece grandes empresas de la industria de la automatización de diseño electrónico, la cual ha promovido el desarrollo y la estandarización de las herramientas CAD basadas en SystemC y TLM.

6.2.3 Flujo de diseño e implementación interna de CASSE

CASSE es entorno de modelado y simulación basado en SystemC desarrollado por investigadores del IUMA en ULPGC, cuyo objetivo principal es ayudar a los diseñadores a explorar y analizar el funcionamiento y comportamiento global de un sistema en una fase temprana del proceso de diseño. Las propiedades claves de CASSE son:

- **Modelado de la aplicación y del mapeo.** Durante la fase de especificación es muy poco probable que el diseñador conozca qué partes de la aplicación se integrarán como software en procesadores, y qué partes se implementarán en bloques

hardware. Por lo tanto, CASSE debe permitir al diseñador modelar estos aspectos de la aplicación de un modo abstracto. Por otra parte, para realizar DSE de una forma eficiente, la aplicación debe modelarse de forma independiente de la arquitectura. Esto significa que dichos modelos deben construirse de forma ortogonal, y *fusionarse* explícitamente en un paso de mapeo aparte.

- **Una librería de componentes genéricos y altamente configurable.** Con el fin crear rápidamente diferentes modelos arquitecturales, y explorar así un gran espacio de diseño, CASSE debe proporcionar un conjunto de modelos de los recursos arquitecturales en su librería de trabajo de forma que el esfuerzo de crear nuevos modelos de arquitectura y derivar modelos de los existentes resulte mínimo. Los componentes de esta librería deben ser lo suficientemente genéricos y configurables, para permitir emular las funciones de un amplio repertorio de componentes arquitecturales. Finalmente, estos modelos de componentes también tienen que ser capaz de dar soporte al mapeo de la funcionalidad de la aplicación en ellos.

- **Análisis avanzado del rendimiento.** En las simulaciones a nivel de sistema, el diseñador debe analizar una gran cantidad de información y resultados acerca del funcionamiento del sistema antes de decidir si un punto de diseño cumple o no con los requisitos establecidos. Interpretar tal cantidad de información con el fin de encontrar los cuellos de botella y posibles vías de optimización no es una tarea trivial en el diseño de los SoCs modernos. Por lo tanto, CASSE debe proporcionar los mecanismos necesarios que faciliten la identificación de los puntos conflictivos dentro de cualquier diseño.

- **Facilidad para crear/configurar los modelos, rápida simulación del sistema y una buena precisión en las evaluaciones.** Como se comentó con anterioridad, para simular cada punto de diseño, el diseñador debe crear y configurar previamente el ESM correspondiente a dicho punto de diseño. Obviamente, cuanto más rápido se lleve acabo este proceso, mayor cantidad de alternativas de diseño pueden explorarse. Para conseguir este objetivo, CASSE debe cumplir con los siguientes requisitos:

  1) CASSE debe proporcionar una interfaz fácil de usar para usuario. Además, dicha interfaz debe poseer cierta capacidad de *scripting* con el objetivo de automatizar y acelerar las modificaciones en el ESM.

2) Para explorar grandes espacios de diseño, es preciso que cada simulación sea llevada acabo lo más rápidamente posible. En estos términos, CASSE debe proporcionar al menos un factor de mejora de x1000 con respecto a las simulaciones RTL si se desea obtener una mejora significativa en la DSE a nivel de sistema. Esto nos lleva a la velocidad de la simulación en el orden de los Mhz.

3) Obviamente, siempre es preferible para un diseñador trabajar con la mayor precisión posible. No obstante, a menudo trabajar con un cierto nivel de precisión puede ser suficiente para mejorar la eficiencia de la DSE y reducir los esfuerzos de crear el ESM. En este contexto, el fin último de las herramientas de simulación a nivel de sistema es evaluar rápidamente el mayor número posible de alternativas y proporcionar al diseñador la información necesaria que le ayude a decidir, por ejemplo, si un sistema es mejor o peor que otro en determinados aspectos.

6.2.3.1 Metodología de diseño de CASSE

Ya hemos comentado a lo largo de este documento la importancia de la herramienta de simulación en el proceso de diseño, porque en cierto modo, ésta limita las capacidades de actuación del diseñador y también es la que condiciona el flujo de diseño que debe utilizar. En esta tesis, nos hemos apoyado en CASSE, una herramienta CAD que permite cubrir las tareas del modelado de la aplicación, modelado arquitectural, mapeo, simulación, análisis y optimización progresiva bajo un único entorno de trabajo.

Una característica esencial de CASSE es que se maneja a través de interfaces basadas en ficheros de texto, es decir, no tiene una interfaz gráfica propiamente dicha que interactúa con el diseñador. Estos ficheros son utilizados por la herramienta durante la simulación para crear y configurar el modelo de la aplicación, el modelo de la arquitectura y el modelo del mapeo. Una vez compilado el conjunto de todos estos ficheros, la herramienta devuelve como resultado un ESM. No obstante, en las modificaciones sucesivas sobre los parámetros arquitecturales del sistema (no sobre el código de la aplicación), el diseñador sólo necesita retocar estos valores dentro del fichero correspondiente, evitándose así recompilar todo nuevamente. Este detalle permite realizar de forma totalmente automatizada exploraciones de numerosas combinaciones de parámetros del sistema, lo que supone un importante ahorro de tiempo.

El esquema del flujo de diseño de la herramienta se muestra en la Fig. 2.1. A continuación explicaremos muy brevemente las funciones de cada etapa que constituye este flujo de diseño.

Modelado de la aplicación. En primer lugar, el código de una aplicación se descompone en forma de grafo de tareas. Para CASSE, las tareas son entidades que ejecutan paralelamente sus procesos entre sí y se comunican a través de canales unidireccionales. Dos tareas conexas se comunican y se sincronizan a nivel TTL mediante las llamadas a las primitivas de sus puertos, consiguiendo de esta forma una separación explícita entre la computación y la comunicación. También hay que hacer hincapié en que la funcionalidad de las tareas está fijada en el momento de compilación, y que a diferencia de los parámetros arquitecturales del sistema, éstos últimos pueden modificarse sin necesidad de recompilar todos los modelos, como se verá más adelante.

Finalmente, cabe mencionarse que la paralelización de la aplicación en el grafo de tarea es un proceso manual, el cual depende en gran medida de las experiencias y conocimientos del diseñador sobre la aplicación en cuestión. Básicamente, el diseñador debe decidir dónde insertar los puntos de ruptura en el código fuente e introducir los puertos y los canales en dichos puntos. La automatización de este proceso de paralelización no es objeto de esta tesis, aunque puede considerarse como una posible línea de trabajo de cara al futuro.

Modelado de la arquitectura. Una plantilla de arquitectura consta típicamente de unos elementos de procesamiento, elementos de almacenamiento, elementos de comunicación, y dispositivos de I/O. Las plantillas o modelos de estos recursos arquitecturales están disponibles en la librería de la herramienta de CASSE, por lo que el diseñador sólo tiene que seleccionarlos y configurarlos para crear el modelo de sistema deseado. Esta manera modular de construir sistemas facilita la expansión, reconfiguración, y la reutilización de la plantilla, aportando así importantes ventajas en cuanto al tiempo y esfuerzo en el proceso de diseño.

Mapeo. Una de las principales ventajas de la herramienta es su flexibilidad a la hora mapear la funcionalidad de la aplicación en el modelo de la arquitectura. Es decir, CASSE soporta el mapeo directo de las aplicaciones TTL (es decir, tareas y canales) en los componentes arquitecturales y también la ejecución del sistema resultante sin necesidad de cambios en el código fuente (es decir, el código fuente original de las tareas se ejecuta directamente en el modelo de la arquitectura). Esta técnica se llama

emulación del código fuente (HCE), el cual evita el uso de modelos de hardware y Simuladores de Juegos de Instrucciones (ISS). Por tanto, reduce el esfuerzo del modelado y permite a su vez simulaciones más rápidas. Por otro lado, las informaciones de tiempo que refleja el coste de cómputo (de la funcionalidad en un PE) aún deben anotarse en las tareas. Esto último puede realizarse de modo automático o puede anotarse de forma manual en cada tarea.

Sin embargo, esta flexibilidad para el mapeo también representa un hándicap para los diseñadores. Es decir, si bien la versión actual de CASSE permite mapear manualmente (y selectivamente) cualquier aplicación en una plantilla arquitectural, CASSE no proporciona ninguna estrategia o técnica que permita que dicho proceso se realice de forma automática. Este proceso puede suponer unos esfuerzos considerables para el diseñador, sobre todo cuando la aplicación en cuestión tiene una complejidad considerable.

Simulación. Como ya se comentó con anterioridad, cada una de las posibles combinaciones de las opciones de diseño equivale a un punto de diseño en el espacio de diseño. Obviamente, cuando más puntos haya, mayor será el espacio a explorar y consecuentemente, más compleja será la labor de análisis del diseñador. Por tanto, la etapa de simulación tiene como misión principal la de proporcionar la información necesaria relativa al punto de diseño analizado, de modo que el diseñador pueda analizarla con posterioridad.

Sin embargo, a pesar de las numerosas métricas capaces de ser capturadas por CASSE (tales como el número de bytes transmitidos, los ciclos de ocupación/ocio de un PE, los ciclos de retardo, etc.), otros indicadores tan importantes como la potencia y coste/área no están siendo considerados en la versión actual de CASSE. Este punto de flaqueza de CASSE limita en gran medida los tipos de problemas de optimización que el diseñador puede tratar. Por esto entendemos que ésta puede ser una posible línea de investigación futura con el fin de mejorar las prestaciones de CASSE.

Análisis y optimización. El análisis de los resultados de las simulaciones ayuda a entender las propiedades cualitativas y cuantitativas del sistema en una fase temprana del proceso del desarrollo del producto. De este análisis se pueden extraer conclusiones relativas a diferentes opciones de diseño y su impacto en el funcionamiento del sistema. Finalmente, la eficiencia y la viabilidad de la implementación final dependerán también de estos trabajos de evaluación y optimización.

6.2.3.2 La estructura interna de CASSE

Como se puede ver en Fig. 2.7, CASSE está estructurada en tres niveles o capas.

- **Capa de interfaz con el usuario.** Esta capa funciona como la interfaz del usuario para controlar la herramienta. Como se mencionó con anterioridad, existen varios grupos de ficheros/modelos de descripción que permiten a los diseñadores controlar plenamente la creación y la configuración de cada ESM: el fichero del grafo de tareas (o aplicación), el fichero de la arquitectura y el fichero del mapeo.

- **Capa del núcleo de la herramienta.** Esta capa implementa la funcionalidad básica de la herramienta. Aparte de contener un programa de análisis que lee e interpreta los modelos de descripción (especificado por los diseñadores), esta capa también lleva incorporada dos librerías específicas: la biblioteca de aplicaciones (APP) y la librería de componentes arquitecturales (ARCH). Por otra parte, CASSE es capaz de llevar a cabo dos tipos de simulaciones: simulaciones funcionales y simulaciones de rendimiento. Mientras que las simulaciones funcionales sólo requieren el modelo de grafo de tareas, CASSE necesita leer y analizar todos los modelos para realizar las simulaciones de rendimiento. El resultado de este proceso es un ESM, es decir, un archivo ejecutable del modelo que describe el sistema en su conjunto.

- **Capa del núcleo del simulador.** Estos ESMs se ejecutan con el núcleo de SystemC, que constituye la tercera capa de la herramienta. Durante las simulaciones, los resultados y las estadísticas son monitorizados y guardados en los archivos de salida para su análisis posterior.

6.2.4 Caso de estudio: sistema de seguimiento visual

En esta sección, presentamos un conjunto de experimentos de DSE con el fin de demostrar las capacidades y el funcionamiento de CASSE. Asimismo, presentamos también la aplicación de referencia que hemos utilizado en todos los experimentos de DSE de esta tesis.

6.2.4.1 Algoritmo de seguimiento visual

La aplicación de referencia que hemos utilizado en este caso es un algoritmo de seguimiento visual basado en la técnica de correlación, el cual ha sido desarrollado por los investigadores del Instituto Universitario de Sistemas Inteligentes Autónomos y

Aplicaciones Numéricas en Ingeniería del ULPGC. A modo de resumen, resaltar que el algoritmo realiza un proceso de correlación de la imagen entrante con un patrón de referencia que el criterio de ajuste utilizado en dicho proceso es el conocido como la suma de la diferencia absoluto (SAD), como se recoge en la expresión (2.1).

El resultado S_{ij} que se recoge en la expresión (2.1) representa un valor de la distorsión, de modo que cuanto menor sea dicho valor mayor será la correlación entre la imagen y el patrón utilizado. Por otro lado, el valor mínimo de la distorsión es el utilizado para determinar el umbral de discriminación o actualización del patrón de referencia. Dicho en otras palabras, sólo se reemplazará o actualizará el patrón si existe una gran probabilidad de perder el objetivo, lo cual viene marcado por el valor de la mínima distorsión de la imagen actual con respecto al umbral de actualización. Utilizando este sistema de actualización dinámica del patrón de referencia, se reduce de forma considerable el número de operaciones de correlación innecesario y también, el número de accesos a las memorias.

En consonancia con las explicaciones del apartado de la metodología de diseño de CASSE, el algoritmo de la aplicación en cuestión es estructurado en primer lugar como un grafo de tareas. El requisito de tiempo real de la aplicación exige procesar 25 imágenes/s, donde el tamaño de las imágenes y el del patrón de referencia, son 320x240 y 24x24 píxeles, respectivamente. Finalmente, queremos hacer hincapié en que la importancia de este grafo está en que, por un lado, permite al diseñador identificar claramente el orden y la dependencia de las tareas y por otro lado, también revela la estructura de datos implicada en cada transacción. Es decir, una vez separados los puntos de comunicación, el diseñador puede tener una visión más completa de las fuentes y destinos de cada dato, lo que facilita en gran medida el mapeo de estos canales de comunicación a la memoria más adecuada.

Paralelamente al desarrollo del algoritmo de seguimiento, se ha modelado una plantilla de arquitectura, la cual no tiene carácter *ad-hoc* sino que es genérica, ya que se ha construido sin tener en cuenta las peculiaridades de las tareas y su interrelación, esto es se ha elegido de forma totalmente independiente de la aplicación. En la práctica, existen muchas implementaciones arquitecturales capaces de satisfacer los requisitos demandados por una aplicación dada. Obviamente, cada alternativa tendrá su pros y su contras. No obstante, nuestro objetivo en este capítulo no consiste en realizar un análisis comparativo de distintas alternativas, sino más bien centrarnos en la exploración del espacio de diseño de una plataforma arquitectural en concreto.

En el caso de este estudio, hemos propuesto una arquitectura muy genérica, que se basa en un procesador RISC programable, unos buses, diversas memorias compartidas y diversos módulos hardware que bien podrían actuar de aceleradores, DSP, ASIC o coprocesadores. También se han incorporado al sistema unos puentes que acomodan la comunicación entre los buses y, a su vez, adaptar los diferentes dominios de reloj. Todos los elementos se pueden encontrar en la librería de trabajo de CASSE, de modo que el trabajo del diseñador se limitará a organizarlos y configurarlos de la forma apropiada. Por ejemplo, un procesador RISC ARM-9 se puede obtener fácilmente configurando los parámetros de un PE: la política de planificación, el tiempo de cambio de contexto, el ciclo de lectura/escritura, y otros parámetros. Mientras que, por ejemplo, configurando adecuadamente la política de arbitraje y el tamaño de los búferes de I/O de un NE, se puede modelar un AMBA bus.

6.2.4.2 Resultados basado en una solución monoprocesador

Antes de comenzar con el proceso del mapeo, se ha compilado el código del algoritmo en ADS (ARM Developer Suite) versión 1.2, para un procesador de referencia, ARM922 a 200 Mhz. El código ensamblador devuelto por esta herramienta nos permite identificar un perfil de tiempo de ejecución para cada tarea del grafo. Esta información se muestra en Fig. 2.9 y en ella se puede observar que esta versión del estudio del sistema con un único procesador está muy lejos de conseguir el comportamiento de tiempo real.

6.2.4.3 Resultados basado en un SoC multiprocesador

Para cumplir con la restricción de tiempo real, se ha considerado una solución basada en múltiples PEs, es decir, en vez de empotrar todo el SW en un único procesador, se distribuirán las tareas entre varios PEs, aprovechando de esta forma el procesamiento en paralelo. Inicialmente, se utilizaron solamente tres PEs (PE 1, PE 2 y PE 4). Si tenemos en cuenta la información obtenida en ADS, vemos que la tarea SAD es muy costosa desde el punto de vista de cómputo. Además, dado que está muy acoplada a la tarea de búsqueda de mínimos, se ha decidido asignar ambas tareas en el mismo PE. Por otro lado, en cualquier sistema, un procesador de vídeo puede realizar más eficientemente la tarea de recepción y de conversión de formato que un procesador RISC, y por ello, estas tareas del grafo se mapearán en el PE 4, por ejemplo. Finalmente, las tareas restantes se agrupan en el PE 1 y las canales de comunicación se mapean en la memoria compartida.

En este caso, hemos realizado una exploración del espacio de diseño manteniendo la arquitectura del bus, el tipo y número de memorias, y la configuración del sistema, pero variando la frecuencia (o reloj) y el factor de aceleración de los coprocesadores. Y los resultados obtenidos se muestran en Fig. 2.11. En ella se puede observar, por ejemplo, que para un rango de frecuencias comprendido entre 200 y 800 Mhz, y unos factores de aceleración desde *x*1 hasta *x*10, el sistema es capaz de procesar 25 imágenes/s si se trabaja a 400 Mhz y el coprocesador tiene un factor de *x*10. Con otras combinaciones, sin embargo, difícilmente se puede procesar 25 imágenes en cada segundo.

Evidentemente, muchos puntos de diseño son capaces de lograr la restricción de tiempo real requerida, aunque muchas veces no todos ellos son deseables desde el punto de vista de implementación física. Por ello, se ha utilizado adicionalmente el criterio de mínima carga de bus en el sistema para elegir la solución óptima de entre todas las factibles. Este criterio tiene su razón de ser en el siguiente argumento: cuanto menos tiempo utiliza un PE el bus compartido, más PEs podrán tener acceso al mismo y por consiguiente, más tareas y/o aplicaciones pueden ejecutarse simultáneamente en el sistema. En este caso, al conocer de antemano perfectamente el volumen de datos implicado en el sistema, la solución de mínima carga en el bus se traduce automáticamente en la solución de mínima carga de sincronización. Consecuentemente, la solución inicial es aquella dada por la combinación de 400 Mhz y un factor de aceleración *x*10.

En este punto conviene explicar que la carga total del bus está compuesta de, por un lado, los datos procedentes de las operaciones de lectura y escritura y, por otro lado, de la información de sincronización que envían los PEs a las memorias compartidas vía bus, convirtiéndose de esta forma en un cuello de botella para el sistema. Este detalle es particularmente interesante cuando tenemos PEs con reparto no equilibrado de cargas de computación. Dicho con otras palabras, cuando un procesador necesita un dato de la memoria, primero testea la memoria para asegurar la presencia del dato: si el dato está disponible, lo consume; si no, el procesador puede optar por realizar otras operaciones o por continuar testeando la memoria.

Este último caso es comúnmente conocido con el nombre de la sobre-sincronización, que es causa de un elevado volumen de tráfico innecesario en el bus, penalizando el rendimiento del sistema. La Fig. 2.12 muestra las diferentes cargas de sincronización producidas por cada puerto de las tareas asociadas a cada PE. Si existe una distribución equilibrada de la carga de trabajo, circulará una cantidad mínima de tráfico de

sincronización por el sistema, siendo este caso representado como el *ideal*. En cambio, en el caso de nuestra solución, vemos perfectamente cómo determinados puertos acusan precisamente de este problema de sobre-sincronización.

Una solución para paliar el efecto de la sobre-sincronización es reubicar los canales accedidos por los puertos afectados por dicho fenómeno. En este caso, hablamos del puerto 2 del PE1 y del puerto 1 del PE 4. Por tanto, los canales asociados a dichos puertos se mapean nuevamente en unas memorias locales y, de este modo, dichos puertos evitan acceder al bus principal para disponer del dato de la memoria principal del sistema, disminuyendo considerablemente la carga del bus con estas dos medidas. Finalmente, queremos hacer hincapié en que todos estos mapeos se han realizado de forma manual, es decir, en función de los resultados obtenidos en cada simulación, modificamos el fichero del mapeo o creamos manualmente el nuevo modelo de mapeo correspondiente.

6.2.4.4 Análisis en un entorno multi-aplicación

Siguiendo con nuestro propósito de construir un CVS más complejo, otro objetivo prioritario en nuestro análisis es estudiar la posibilidad de mapear más de una aplicación en el sistema, manteniendo los comportamientos de tiempo real del sistema de seguimiento. Para ello, hemos modelado otra aplicación productor-consumidor con las características típicas de las aplicaciones de procesado de imágenes, en donde los flujos de datos son regularmente o periódicamente recibidos y procesados. Aquí lo que nos interesa es analizar el efecto del tráfico generado por esta aplicación en el sistema inicial y no tanto el procesado que realiza esta nueva aplicación.

Con este planteamiento en mente, hemos desarrollado una nueva aplicación para que genere cuatro tipos diferentes de tráfico:

- tráfico constante, donde los datos de ambas aplicaciones compiten por la misma memoria;
- tráfico periódico, es decir, donde la nueva aplicación accede a los recursos del sistema a una tasa específica, siendo la misma 10 ms y 20 ms; y
- finalmente, un tráfico aleatorio.

Las tareas de la nueva aplicación (productor-consumidor) y sus canales se mapean en la memoria compartida principal. Tras numerosos experimentos y simulaciones, obtuvimos

los resultados que se muestran en Fig. 2.14. En ella se puede observar que el rendimiento de nuestra solución anterior (es decir, para una configuración de 400 Mhz y un factor de aceleración de *x*10) se mantiene intacta si la nueva aplicación produce un tráfico constante con un tamaño de los paquetes inferior a 100 Kbytes, u otra que produce un trafico con periodicidad de 10 ms, siendo el tamaño de sus datos menor a 2 Mbytes.

6.2.5 Conclusiones

En este capítulo hemos presentado a CASSE, una herramienta de modelado y simulación a nivel de sistema basada en SystemC. Hemos explicado los aspectos claves para entender el funcionamiento de CASSE y también hemos presentado un conjunto de experimentos de DSE para demostrar las capacidades de esta herramienta CAD.

Los resultados obtenidos indican que CASSE es una herramienta potente que posee muchas cualidades para llevar a cabo de forma eficiente diferentes tipos de experimentos de DSE. Sin embargo, quedan todavía mucho trabajo por hacer para que todo el proceso de modelado y creación de ESM puede realizarse de una forma automatizada. Con esta idea en mente, los capítulos 3 y 4 se centrarán en proporcionar los mecanismos para reducir el esfuerzo del diseñador en el modelado/creación de cada ESM y también hacer más eficiente el proceso de la DSE a nivel de sistema.

6.3 Metodología e infraestructura para la exploración del espacio de diseño multidimensional

6.3.1 Introducción

El objetivo principal de la exploración del espacio de diseño a nivel de sistema consiste en evaluar una amplia gama de opciones de diseño, de modo que permita al diseñador evaluar el impacto de estas opciones en el rendimiento global de sistema y, por consiguiente, tomar las decisiones de diseño apropiadas en una etapa temprana del proceso de diseño.

Sin embargo, como ya se comentó anteriormente, el diseñador debe evaluar un gran número de opciones en el diseño de los SoCs modernos. Estas opciones de diseño consisten normalmente en el número y tipo de los componentes utilizados, la topología de la plataforma arquitectural, el mapeo de la aplicación en los recursos de la arquitectura, etc. De esta forma, podemos agrupar dichas opciones de diseño en dimensiones del espacio de diseño, como se muestra en el ejemplo de Fig. 3.1. Evidentemente, cuanto mayor es el número de opciones de diseño y el número de dimensiones, mayor será el tamaño del espacio de diseño que debe explorar el diseñador, y por tanto, mayor esfuerzo le supondrá en el proceso de la DSE.

Sin embargo, como ya se ha visto en el capítulo 2, crear manualmente cada ESM y explorar exhaustivamente grandes espacios de diseño puede ser totalmente inviable, incluso trabajando con herramientas de simulación a nivel de sistema. De hecho, como se apuntó en el capítulo 1, la herramienta de simulación sólo constituye el medio para evaluar de forma individualizada cada modelo de sistema, y por ello, aún se precisa integrar estas herramientas junto con los mecanismos de búsqueda en una infraestructura común para llevarse acabo el proceso de DSE. Aunque muchos trabajos han abordado esta cuestión proponiendo una gran variedad de técnicas, podemos resaltar tres elementos comunes en todos ellos:

- Muchas de las infraestructuras propuestas en la literatura están diseñadas para dar soporte a un simulador (o modelo de estimación analítico) en concreto, lo que dificulta la reutilización.

- La configuración de los experimentos de DSE puede ser muy laboriosa, hasta el punto de que en muchas ocasiones, el diseñador debe escribir un script a la medida de cada experimento.

- A pesar de la gran variedad de arquitecturas susceptibles de utilizarse en cada experimento, muchos trabajos se centran exclusivamente en un tipo de ellos. Por otra parte, los diseñadores deben crear manualmente cada nuevo modelo de arquitectura, siendo esta tarea muy propensa a errores y también uno de los principales cuellos de botella en la productividad del diseñador del sistema, limitando así el tamaño del espacio de diseño que se puede explorar en un tiempo razonable.

En resumen, y basándonos en nuestros estudios, podemos afirmar que existe una carencia en cuanto a la propuesta de una infraestructura software genérica y reutilizable, que proporcione el soporte necesario para los experimentos de DSE a nivel de sistema. Esta laguna demanda la necesidad de un entorno de trabajo, capaz de integrar diferentes tipos de simuladores y técnicas de búsqueda, para la realización de diversos tipos de experimentos de DSE de una forma eficiente y sistemática. Por otra parte, dicha infraestructura también debe incorporar algún tipo de mecanismo que sea capaz de generar una amplia variedad de arquitecturas, de modo que permita explorar automáticamente grandes espacios de diseño.

En este capítulo presentamos una infraestructura genérica para DSE a nivel de sistema llamada NASA. El objetivo principal de NASA es proporcionar un marco de trabajo común, único y modular para la realización de experimentos de DSE a nivel de sistema, que a su vez sea capaz de incorporar diferentes simuladores y estrategias de búsqueda por el simple mecanismo de *plug-in*. Además, también hemos integrado en NASA un generador de ESM, con el objetivo de liberar al diseñador de crear manualmente tales modelos para cada experimento, y mejorando de esta forma, la productividad de los diseñadores de sistema. Por otro lado, NASA también provee el soporte necesario para una nueva metodología de DSE basada en las dimensiones del espacio de diseño, permitiendo al diseñador explorar sistemáticamente grandes espacios de diseño multi-dimensionales mediante la configuración de un sencillo fichero de entrada. Finalmente, las informaciones de salida proporcionadas por NASA no sólo incluyen las soluciones de diseño que cumplen las restricciones exigidas (como rendimiento, coste/área, potencia, etc.), sino que el diseñador también puede disponer de información sobre todos los puntos explorados durante la DSE para cualquier análisis y evaluación a posterior.

6.3.2 Implementación de NASA

Se pueden resaltar cuatro propiedades claves de NASA:

- **Modularidad:** NASA está compuesto por una serie de módulos, que interactúan entre sí por medio de unas interfaces bien definidas. Esta propiedad trae consigo dos consecuencias. Por un lado, permite que cada módulo pueda comportarse como si fuese una caja negra dentro de NASA. Por otro lado, se pueden incorporarse fácilmente nuevos módulos en NASA por el simple mecanismo de *plug&play*, mientras que los nuevos módulos tengan las interfaces requeridas por NASA.

- **Flexibilidad:** NASA introduce una nueva metodología para la exploración del espacio de diseño, *DSE basada en las dimensiones del espacio de diseño*, la cual permite al diseñador *co-explorar* espacio de diseño multidimensionales, es decir, utilizar explícitamente un algoritmo de búsqueda para co-explorar (de forma separada) cada una de las dimensiones que conforman el espacio de diseño. En la implementación actual se distinguen tres dimensiones o niveles: plataforma, componentes arquitecturales y mapeo. De este modo, el diseñador de sistema puede optar por co-explorar simultáneamente todas dimensiones, o mantener los valores de una o varias dimensiones (por ejemplo, fijar una plataforma en concreta) mientras explora las opciones de diseño en las dimensiones restantes.

- **Reutilización:** Dada una serie de especificaciones de diseño, NASA puede llevar a cabo la DSE de una forma totalmente autónoma. Esto significa que los diseñadores no tienen que preparar los scripts para cada experimento ni tampoco crear cada modelo de sistema de forma manual, sino que sólo necesita centrarse en la configuración de los parámetros de diseño (especificados en el fichero de entrada) para comenzar un nuevo experimento de DSE.

- **Extensibilidad:** La propiedad de la modularidad, así como el uso de las interfaces en NASA, posibilita la inclusión de nuevos módulos y funcionalidades en nuestro entorno de trabajo. Estos módulos pueden, por ejemplo, explorar nuevas dimensiones sin la necedad de modificar los módulos existentes.

El esquema general de NASA se muestra en la Fig. 3.2. En esencia, se pueden distinguir seis módulos principales: módulo de búsqueda (*Search*), módulo de comprobación de validez (*Feasibility Checker*), generador de la plataforma arquitectural (*Architectural platform generator*), módulo adaptador del simulador (*Translator*), simulador (*Simulator*),

y módulo de evaluación (*Evaluator*). A continuación, explicaremos las interfaces utilizadas en NASA así como los módulos mencionados anteriormente.

6.3.2.1 Las interfaces de NASA

Las interfaces de NASA se basan en ficheros de textos, los cuales son creados automática y dinámicamente a partir de las especificaciones del usuario en el fichero de entrada. Básicamente, NASA utiliza tres tipos de interfaces:

- **Fichero intermedio de la arquitectura:** se utiliza para enlazar el módulo adaptador del simulador con el generador de la plataforma arquitectural.

- **Fichero de métrica de evaluación:** es utilizado por el módulo de evaluación para proporcionar las métricas suministradas por el simulador al módulo de búsqueda.

- **Ficheros de opciones de diseño:** se utilizan en el interior de los módulos de búsqueda y de comprobación de validez, así como para comunicar ambos módulos entre sí.

En la implementación actual de NASA, utilizamos un fichero de opciones de diseño y uno de métrica de evaluación por cada dimensión del espacio de diseño explorada en el experimento, estando representadas las informaciones contenidas en dichos ficheros en formato de *string* (o cadena de valores numéricos). Por ejemplo, en el caso de la DSE de tres niveles mostrada en la Fig. 3.3a, la dimensión de la plataforma recoge en un único fichero las decisiones de diseño sobre la topología de la arquitectura, tipos de NEs y las propiedades de conectividad entre los elementos que conforma la arquitectura. Mientras por otro lado, la dimensión de componentes arquitecturales y de mapeo poseen (cada una de ellas) su propio fichero de opciones de diseño correspondiente. Sin embargo, en el caso de que el diseñador decida no utilizar un algoritmo de búsqueda por dimensión (como se muestra en la Fig. 3.3b), unos adaptadores son automáticamente puestos a la entrada y salida del módulo de búsqueda, con el fin de preservar la interfaz de un fichero por dimensión.

También queremos hacer hincapié en que la longitud del *string* en estos ficheros también es variable. Usando nuevamente el ejemplo mostrado en la Fig. 3.3, se puede ver claramente que la longitud de la cadena de valores correspondiente a la dimensión del mapeo depende del número de las tareas y de los canales de comunicación de la

aplicación. De forma análoga, la longitud del *string* de la dimensión de componentes arquitecturales está directamente asociada al máximo números de PEs y SEs permitidos en una plataforma, siendo estos valores especificados por el usuario en el fichero de entrada.

La última interfaz de NASA es el fichero intermedio de la arquitectura, y se utiliza para concentrar en un único fichero las decisiones de diseño relativas al modelado de la arquitectura (de cada punto de diseño). Estas decisiones de diseño proceden de los *strings* de la dimensión de la plataforma y de la dimensión de componentes arquitecturales. La finalidad de este interfaz es doble:

- Lo utiliza el módulo adaptador del simulador para generar el modelo de la arquitectura de cada punto de diseño en cuestión.

- También es fundamental para el módulo de comprobación de validez, el cual necesita esta información para evaluar la validez de cada mapeo.

6.3.2.2 Módulo de búsqueda

La misión de este módulo es buscar y seleccionar, de forma iterativa, los puntos de diseño (dentro del espacio de diseño) para que el simulador pueda evaluarlos. Como ya se mencionó con anterioridad, NASA aplica un nuevo concepto para la DSE, que es la DSE basada en las dimensiones del espacio de diseño. Básicamente, este concepto posibilita la co-exploración del espacio de diseño por medio de múltiples y diferentes tipos de algoritmos de búsqueda (por ejemplo, uno por dimensión del espacio de diseño), o emplear un número inferior de algoritmos de búsqueda para la DSE (por ejemplo, utilizar un único método de búsqueda para todas las dimensiones).

Llegados a este punto, conviene resaltar la definición de *co-exploración* en nuestra metodología de DSE basada en dimensiones del espacio de diseño. En nuestro caso, a pesar de que se emplea un algoritmo de búsqueda por dimensión, esto no implica que cada uno de los algoritmos de búsqueda lleve a cabo la DSE de forma independiente, sino que al contrario, los resultados proporcionados por los algoritmos de búsqueda son utilizados conjuntamente para guiar la DSE. Formalmente, esto significa que se puede especificar una solución (o punto de diseño del espacio de diseño) *dp* como la concatenación de un conjunto de opciones de diseño $\{d_{pla}, d_{arc}, d_{map}\}$, los cuales corresponden a cada una de las *k* dimensiones del espacio de diseño, donde d_{map}

representa una opción de diseño (o *string*) de la dimensión del mapeo, mientras que d_{arc} y d_{pla} describen las decisiones de diseño de las dimensiones de componentes arquitecturales y de la plataforma, respectivamente.

Cuando se utilizan varios algoritmos de búsqueda para co-explorar el espacio de diseño, hay muchas formas de conjuntar las decisiones de diseño de las distintas dimensiones para obtener la especificación de los puntos de diseño. Por un lado, se puede optar por la técnica piramidal (Fig. 3.4a), que consiste en enlazar cada *string* de la dimensión de la plataforma con todos los *strings* de la dimensión de componentes arquitecturales, y a su vez, cada *string* de ésta última dimensión se asocia con todas los posibles strings de la dimensión del mapeo. Por esto, podemos afirmar que el número total de puntos de diseños generados por esta técnica crece de forma exponencial con respecto al número de dimensiones exploradas. Por otro lado, también se puede emplear la técnica de asociación *uno-a-uno* (Fig. 3.4b). En este caso, cada opción de diseño de una dimensión se concatena exclusivamente con un *string* de otra dimensión y, por tanto, el número total de puntos de diseños generado con esta técnica es igual al número de *strings* especificados en cada fichero de opción de diseño, suponiendo que todos los ficheros contengan el mismo número de *strings*.

No obstante, aunque la técnica de asociación *uno-a-uno* supone una drástica reducción del número de puntos de diseño a explorar, ésta no está exenta de inconvenientes. Desde nuestro punto de vista, el problema de la *infra-exploración* es el más relevante. En esencia, este tipo de problema surge cuando se descarta prematuramente una opción de diseño (una determinada plataforma, por ejemplo) basándose en una cantidad de información insuficiente. Por ejemplo, considérese el punto de diseño $A = \{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ formado por los strings de la plataforma d^A_{pla}, componentes arquitecturales d^A_{arc}, y mapeo d^A_{map}, mientras que $B = \{d^B_{pla}, d^B_{arc}, d^B_{map}\}$ es otro punto de diseño obtenido por la concatenación de otro conjunto distinto de opciones de diseño. Si tras una única simulación de cada una de las dos alternativas se obtiene que el rendimiento de A es mejor que el de B, entonces no podemos afirmar tajantemente que d^A_{pla} sea mejor que d^B_{pla}, aunque sí se puede inferir que la combinación de las opciones de diseño para $A = \{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ es mejor que la de $B = \{d^B_{pla}, d^B_{arc}, d^B_{map}\}$. Dicho con otras palabras, no podemos garantizar que $\{d^A_{pla}, d^A_{arc}, d^A_{map}\}$ sea mejor que $\{d^B_{pla}, d^B_{arc}, d^C_{map}\}$, donde d^C_{map} representa otro mapeo distinto para B.

Para solucionar el problema de la infra-exploración, hemos introducido una variante en la técnica de asociación uno-a-uno. En este caso, se propone que solamente se cambie y actualice en cada iteración de búsqueda los *strings* correspondientes a la dimensión del nivel de abstracción más bajo (la dimensión de mapeo en nuestro caso). Mientras por otro lado, los algoritmos de búsqueda de las restantes dimensiones recolectan y conservan las métricas obtenidas con cada uno de los puntos de diseño sin introducir cambios en sus correspondientes *strings* durante un número determinado iteraciones, a las que llamamos iteraciones de recolecta (δ). Entonces, sólo cuando se ha alcanzado el número de iteraciones de recolecta especificado para cada dimensión, los *strings* de cada dimensión son actualizados, y volviendo el proceso a comenzar de nuevo. En este punto, hay que hacer hincapié en que cuanto mayor es el nivel de abstracción de una dimensión (como por ejemplo, la dimensión de plataforma), mayor es la cantidad de alternativas de diseño que se puede generar de una opción de diseño (por ejemplo, se pueden derivar múltiples variantes arquitecturales de una plataforma en particular) y, como consecuencia, mayor deberá ser el valor de las iteraciones de recolecta asignado a dicha dimensión.

6.3.2.3 Módulo de comprobación de validez

Durante el proceso de búsqueda llevado a cabo por el módulo de búsqueda se pueden seleccionar puntos de diseño que no se ajustan al conjunto de especificaciones de diseño impuesto por el diseñador. Por ello, el objetivo principal del módulo de comprobación de validez será la detección y reparación de las soluciones inválidas encontradas durante el proceso de búsqueda.

El proceso de comprobación de validez también se realiza de forma jerarquizada, es decir, se deben verificar en primer lugar los *strings* de la dimensión de plataforma, seguido por la comprobación de los *strings* de la dimensión de componentes arquitecturales, y finalmente, se analizan las opciones de diseños de la dimensión de mapeo. Dicho de otra manera, no se puede verificar la validez de un mapeo si previamente no se ha asegurado de que la arquitectura (que da soporte al mapeo) ha sido correctamente construida.

Obviamente, dada que la naturaleza de las informaciones codificadas en cada dimensión es diferente, el tipo de invalidez a detectar dependerá también de la dimensión en cuestión. Por ejemplo, en la dimensión de plataforma se debe validar la topología y la conectividad de los componentes entre sí, mientras que en la dimensión de

componentes arquitecturales se debe verificar si el tipo y el número de PEs y SEs satisfacen las especificaciones del usuario. Por ejemplo, si un diseñador está interesado en la exploración de alternativas arquitecturales que contengan un máximo de 2 procesadores ARM y el algoritmo de búsqueda ha seleccionado un punto de diseño con 4 procesadores ARM, entonces el módulo de comprobación de validez deberá ser capaz de detectar y corregir este tipo de puntos de diseño inválidos. Finalmente, es imprescindible asegurar, en la dimensión de mapeo, que cualquier canal que comunica una pareja de tareas entre sí esté mapeada en SEs que pueden ser compartidos o accesibles por los PEs en los que están mapeadas dichas tareas.

Una vez que se hayan detectadas las opciones de diseños inválidas, se puede aplicar una gran variedad de técnicas de reparación. No obstante, cabe mencionar que en función de la técnica utilizada, una misma opción de diseño inválida puede convertirse en diferentes soluciones válidas. En nuestra implementación actual, hemos optado por un algoritmo de reparación basado en la distancia mínima, el cual es capaz de reparar un *string* inválido introduciendo el mínimo número de modificaciones necesarias. Volviendo al ejemplo del mapeo mencionado en el párrafo anterior, si se ha mapeado un canal de comunicación en un SE que no es accesible por uno (o ninguno) de los PEs en que están mapeadas las dos tareas, nuestro algoritmo tratará de reubicar el canal a un SE compartido por los PEs en que están mapeados las dos tareas; si no existe ningún SE que cumpla con dicha condición, entonces se declara dicho mapeo como una opción de diseño inválida. Obviamente, en vez de reubicar el canal de comunicación, podríamos haber reasignado una de las dos tareas a otro PE. Sin embargo, una tarea normalmente puede comunicarse con otras varias tareas a través de múltiples canales de comunicación y, por consiguiente, cualquier cambio en el mapeo de una tarea implica necesariamente que se vuelva a comprobar los canales asociados a dicha tarea, pudiendo este proceso convertirse en el peor caso en un bucle infinito.

6.3.2.4 Generador de la plataforma arquitectural

La tarea principal de este módulo es crear una descripción de la arquitectura para cada punto de diseño válido seleccionado en las fases anteriores. Las descripciones arquitecturales resultantes son listadas en un único fichero, el fichero intermedio de la arquitectura, el cual es utilizado más tarde (i) para la comprobación de validez de los *strings* de la dimensión del mapeo, y (ii) como fichero de entrada del módulo de adaptador del simulador para generar los correspondientes modelos de arquitecturas requeridos para realizar las simulaciones.

Básicamente, una descripción de arquitectura combina las informaciones contenidas en los *strings* de la dimensión de la plataforma y de la dimensión de los componentes arquitecturales. Asimismo, el proceso de creación de esta descripción de arquitectura consta de dos pasos: fase de la generación de la plantilla topológica y fase de la generación de la instancia arquitectural.

Unidad Básica de Topología

La Unidad Básica de Topología (BTU) es el cimiento sobre el cual construimos la descripción de la arquitectura. Como puede verse en la Fig. 3.5, la BTU es un patrón lógico compuesto por un contenedor de inter-conector que contiene siempre un único NE, y un número variable de contenedores de elementos que pueden albergar tanto PEs como SEs. El número total de contenedores de elemento en cada BTU depende directamente de la especificaciones del diseñador, de modo que, si éste indica que sólo deben utilizarse 3 PEs and 2 SEs en su DSE, entonces el máximo número de contenedores de elemento será 5. Por otro lado, el diseñador también puede configurar el número de conectores que puede tener cada contenedor de elemento, es decir, los contenedores de inter-conector no pueden conectarse directamente entre sí, mientras que los contenedores de elemento no sólo pueden conectarse entre sí, sino que también puede actuar de puente entre dos NEs. Finalmente, cabe subrayar que todos los contenedores en un BTU (tanto contenedores de inter-conector como de elementos) deben estar etiquetados para facilitar su identificación, como también se muestra en la Fig. 3.5.

Generación de la platilla topológica

Una vez creadas las BTUs, estas son numeradas y replicadas un número de veces para generar una meta-plataforma, la cual es la base para generar o derivar las plantillas topológicas. Es importante resaltar que el número de veces que debe replicar la BTU depende del número máximo de NEs especificados por el diseñador para la DSE y de la conectividad de cada contenedor de elemento. En este contexto, la conectividad hace referencia al número y direcciones de los conectores que puede tener cada contenedor de elemento. De esta forma, si un diseñador decide habilitar los conectores (del contenedor de elemento) en las tres direcciones, entonces las BTUs pueden replicarse hacia los tres ejes *(x, y, z)* de coordenadas, formando así una estructura de meta-plataforma tridimensional (3D). En el caso contrario, las BTUs se replican en el mismo plano, formando una meta-plataforma bidimensional (2D), como se muestra en la Fig. 3.6.

El siguiente paso tras la creación de la meta-platforma sería generar una plantilla topológica para cada punto de diseño seleccionado (por el módulo de búsqueda) y verificado (por el comprobador de validez). Toda la información necesaria para generar una plantilla está presente en el *string* de la dimensión de plataforma. Esto es, cada cadena contiene los valores para las distintas decisiones de diseño vinculadas a este nivel del espacio de diseño, tales como el número y tipo de NE en la plataforma, la cantidad de contenedores de elemento utilizados, qué conectores están activados y qué tipo de componentes (es decir, PE o SE) está asignado en cada contenedor de elemento. No obstante, quisiéramos insistir en que estas informaciones han sido cuidadosamente comprobadas (por el comprobador de validez), con el objeto de reparar cualquier tipo de inconsistencias en la plataforma, como por ejemplo, BTUs aisladas, conectores que no enlazan con ningún otro elemento, etc.

Generación de la instancia arquitectural

En esta fase continuamos refinando las plantillas topológicas generadas en la fase anterior para obtener las instancias arquitecturales. De modo análogo, utilizamos los valores contenidos en el *string* de la dimensión de los componentes arquitecturales para describir el tipo de cada PE y SE contenido en la plantilla topológica de cada punto de diseño. Es decir, especificamos si un PE debe configurarse como un procesador ARM o MIPS, por ejemplo y, por otra parte, si debemos configurar un SE para que se comporte como una memoria de un único puerto o doble puerto, etc.

6.3.2.5 Módulo adaptador del simulador

Para poder integrar un simulador en NASA, es requisito indispensable que el simulador permita especificar un modelo de sistema (o lo que es lo mismo, el ESM de un punto de diseño) por medio de ficheros de texto. Esto último significa que, si el simulador integrado en NASA sigue el esquema Y-Chart, primero debemos generar cada uno de los modelos que conforma el ESM, es decir, el modelo de la aplicación, el modelo de la arquitectura y el modelo del mapeo. Ésta es precisamente la función del módulo adaptador del simulador, es decir, a partir de la especificación de la aplicación, el fichero intermedio de la arquitectura y el *string* del mapeo, este módulo es capaz de generar automáticamente los respectivos modelos (de aplicación, de la arquitectura y del mapeo) para cada punto de diseño seleccionado en el proceso de exploración. Consecuentemente, el simulador puede combinar estos modelos y crear el ESM correspondiente para realizar la simulación y evaluación pertinente. Separar

explícitamente este módulo del resto de los módulos aporta numerosas ventajas a NASA, y las dos más importantes son:

- Por un lado, podemos particularizar el formato de los modelos generados. Es decir, típicamente existen unas diferencias sustanciales en el lenguaje utilizado por cada simulador para describir los distintos modelos requeridos, y el módulo adaptador debe ser capaz de generar los modelos requeridos acorde al formato o al lenguaje utilizado por cada simulador en particular.

- Por otro lado, y como consecuencia de la anterior, sólo debemos preocuparnos en crear un nuevo adaptador si decidimos utilizar otro simulador en NASA, mientras que los restantes módulos permanecen inalterados. Por ello, se puede considerar el módulo adaptador de simulador como una capa o interfaz entre los módulos independientes del simulador (módulos de color gris en la Fig. 3.2) y módulos dependientes del simulador (módulos de color negro en la Fig. 3.2).

6.3.2.6 Simulador

El simulador que hemos utilizado e integrado en NASA es CASSE, el cual ya se ha presentado en el capítulo 2 de esta tesis. Recordar simplemente que CASSE cumple con los requerimientos exigidos por NASA en cuanto a la descripción de los distintos modelos por medio de ficheros de textos.

6.3.2.7 Módulo de evaluación

A través de las simulaciones, el diseñador puede disponer de una gran cantidad de información sobre la ejecución del sistema, la cual es, sin duda, de gran utilidad para cualquier análisis posterior. Por ejemplo, a menudo se utilizan los resultados obtenidos sobre el rendimiento, costes/área, y/o consumo de potencia para calcular el conjunto de puntos que determinan la frontera óptima de Pareto en los problemas de optimización multiobjetivo. Pues bien, la esencia de este módulo consiste precisamente en proporcionar esta retroalimentación (es decir, la calidad del resultado de cada punto de diseño) al módulo de búsqueda, incidiendo de esta forma en el proceso de la DSE.

Nuevamente, separar el módulo de evaluación del módulo de búsqueda proporciona enormes ventajas a NASA en cuanto a la flexibilidad y reutilización. Por ejemplo, un diseñador puede interesarse por diferentes objetivos de optimización en cada DSE. Este aspecto se soluciona en NASA mediante la simple sustitución y/o modificación de la

función objetivo que pondera o cuantifica la calidad de cada punto de diseño, manteniéndose inalterado el resto de los módulos de NASA. Asimismo, el diseñador también puede elegir el número de funciones objetivo utilizado en cada DSE, es decir, una única función de evaluación para todos los algoritmos de búsqueda, o una función objetivo diferente para cada algoritmo.

Sin embargo, hay que tener muy presente que cuando se utilizan varias funciones objetivo conjuntamente, éstas deben definirse de un modo coherente entre sí, ya que de lo contrario, se puede inducir a conflictos y divergencia en el proceso de DSE. Asimismo, esta coherencia también debe hacerse explícita entre cada pareja de algoritmo de búsqueda y su función objetivo. En nuestra implementación actual, hemos definido un conjunto de funciones objetivo jerarquizadas, que pueden utilizarse con la técnica de asociación uno-a-uno presentada en el módulo de búsqueda.

6.3.3 Conclusiones

Basándose en nuestros estudios hasta la fecha, hemos detectado una cierta carencia de propuestas en la literatura sobre una infraestructura genérica, flexible y reutilizable que proporcione el soporte necesario para la realización de experimentos de DSE a nivel de sistema sobre MPSoC. Con el objetivo de cubrir este vacío en mente, hemos desarrollado un entorno de trabajo llamado NASA, que se caracteriza por su composición modular a través de unas interfaces bien definidas, las cuales posibilitan y facilitan la integración de diferentes tipos de algoritmos de búsqueda y herramientas de simulación mediante el simple mecanismo de *plug&play*.

Por otra parte, NASA también incorpora una nueva metodología de DSE basada en la co-exploración del espacio multidimensional de diseño, de modo que cada diseñador puede configurar el número y tipo de técnica de búsqueda más adecuada para cada DSE en particular. De este modo, el diseñador sólo debe configurar correctamente un conjunto de especificaciones de diseño en el fichero de entrada, para llevar a cabo de forma totalmente automatizada y sistemática la exploración del espacio multidimensional de diseño sobre MPSoC.

Para demostrar la capacidad de NASA, hemos realizado una serie de experimentos (i) con un único algoritmo de búsqueda en NASA, (ii) también con múltiples algoritmos de búsqueda idénticos, y finalmente, (iii) con diferentes estrategias de búsqueda para cada dimensión del espacio de diseño. Los resultados experimentales revelan que,

comparado con los métodos tradicionales de DSE basado en un único algoritmo de búsqueda, nuestra técnica de co-exploración parece ser capaz de encontrar mejores soluciones, a la vez que asegura la convergencia hacia los óptimos globales del espacio de diseño. Asimismo, estos resultados también indican que nuestra propuesta es capaz de cubrir un mayor rango y diversidad de alternativas de diseño, logrando de esta forma unos resultados de DSE de mayor calidad.

6.4 Estrategias y algoritmos para el mapeo de aplicaciones en MPSoC

6.4.1 Introducción

El incremento de la capacidad de la integración en chip ha posibilitado la aparición de complejos sistemas empotrados compuestos de múltiples SEs, NEs y PEs. Las ventajas de estos MPSoCs son múltiples. Por un lado, facilita una plataforma flexible y reutilizable para diferentes versiones de un producto (o familia de productos). Por otro lado, se pueden actualizar y modificar con suma facilidad para adaptarse a las exigencias del mercado y de los clientes potenciales.

Sin embargo, a pesar de que estas plantillas presentan una arquitectura (más o menos) fijada, los diseñadores de sistemas aún requieren explorar un vasto dominio de opciones de diseño sobre estas plataformas, con el fin de encontrar el modo idóneo de mapear una aplicación específica en una de estas plantillas de arquitectura. Por ejemplo, las alternativas a explorar pueden abarcar co-diseño HW/SW, esquema de arbitraje en el bus, número y tipo de cada componente utilizado en la arquitectura, y la ubicación de determinados recursos en lugares estratégicos dentro de la plataforma, etc.

6.4.2 Problema de optimización multiobjetivo

Además de la gran cantidad de opciones de diseño que el diseñador debe considerar en las fases tempranas de diseño, éste también tiene que sopesar un conjunto de restricciones u objetivos de diseño. Por un lado, la mayoría de los SoCs de la actualidad se diseñan para fabricarlos en masa e integrarlos en dispositivos que utilizan baterías de pequeñas dimensiones. Esto es, los SoCs modernos deben ser baratos (en términos de costo/área) y eficientes (en términos de consumo de energía). Por otra parte, también tienen que proporcionar altos rendimientos (por ejemplo, prestaciones de tiempo real) y a la vez suficiente flexibilidad para dar soporte a múltiples aplicaciones. Básicamente, estos objetivos de diseño pueden clasificarse a grosso modo en dos categorías o niveles: objetivos primarios y secundarios.

6.4.2.1 Los objetivos primarios

Los objetivos primarios tienden a estar vinculados con algunas propiedades del sistema en general, y a menudo son utilizados directamente como los objetivos a optimizar. Es decir, no representan a ninguna métrica o parámetro de coste intermedio, sino que son

aplicados directamente para resolver los problemas de optimización. Los objetivos primarios comúnmente utilizados en la DSE se enumeran a continuación.

- **Coste**: el coste de un diseño puede medirse como la suma del coste de todos los componentes que están integrados en el sistema. El coste del sistema a su vez puede desglosarse en varios componentes: el coste de los componentes utilizados en el sistema y el coste del fabricante de chip, el cual podría estimarse a partir del área de silicio consumido y el coste del encapsulado.

- **Consumo de energía**: la optimización en el consumo de energía y disipación de potencia se está convirtiendo actualmente en el foco de atención en el diseño de SoCs. Por un lado, los sistemas optimizados para alcanzar altos rendimientos tienden a consumir una elevada cantidad de energía, lo que aumenta la generación y disipación del calor dentro del sistema, degradando así la vida útil de los componentes. Por otra parte, los diseñadores no sólo deben centrarse en minimizar la disipación del calor cuando el sistema está en periodo de procesamiento, sino también tienen que asegurar que el consumo de la energía sea mínimo durante los períodos de inactividad del sistema, con fin de alargar la vida útil de las baterías.

- **Rendimiento**: la prestación o rendimiento de un diseño puede expresarse mediante diferentes indicadores, por ejemplo, el throughput, la cantidad total de datos procesados o transferidos y la latencia o el tiempo de respuesta ante determinados eventos.

6.4.2.2 Los objetivos secundarios

Los objetivos englobados en esta categoría se centran a menudo en determinadas partes o aspectos concretos del diseño. Generalmente suelen utilizarse para proporcionar información de apoyo al diseñador, es decir, revelan características del diseño que influyen en los objetivos primarios. Por ejemplo, el grado de utilización de los recursos podría revelar datos sobre el uso efectivo del área de silicio (es decir, el objetivo primario de coste) y sobre el consumo de potencia. A continuación, se enumeran algunos ejemplos de objetivos secundarios.

- **La eficiencia en el uso de los elementos de procesamiento**: la utilización de un elemento de procesamiento puede calcularse como el porcentaje del tiempo en el que el recurso está ocupado. Básicamente, optimizar la eficiencia en el uso de un

elemento de procesamiento implica generalmente explotar al máximo el uso efectivo del área de silicio. Es decir, los diseñadores del sistema deberían evitar en la medida de lo posible sobre-dimensionar la arquitectura. Dicho de otra forma, una solución de diseño es mejor que otra si se logra un mayor rendimiento con el mismo coste/área o el mismo rendimiento a un menor costo/área.

- **Equilibrio en el reparto de la carga de trabajo**: un reparto equitativo de la carga de trabajo evita sobrecargar determinados recursos. Por otro lado, un desequilibrio en el reparto de la carga de trabajo podría generar retrasos en el envío/recepción de datos en la arquitectura de comunicación, creando de esta forma un retraso en el procesamiento de la información en otros elementos de procesamiento. Asimismo, como se demuestra en el apartado 2.4.2.2, un diseño con un reparto no equitativo de la carga de trabajo también podría generar tráficos adicionales (y contraproducentes) en la arquitectura de comunicación debido a los efectos de la sobre-sincronización. Por lo tanto, podemos concluir que el reparto de la carga de trabajo influye directamente en el consumo de energía del diseño.

- **Objetivos relacionados con la comunicación**: una manera de optimizar el consumo de energía de los SEs y NEs consiste en reducir al mínimo el número de accesos a SEs y reducir al mínimo la carga del sistema. Una solución podría consistir en agrupar aquellas funciones de la aplicación fuertemente acopladas entre sí en el mismo elemento de procesamiento, ya que de esta forma, se puede reducir el acceso del PE a los SEs del SoC. Por otra parte, la reducción de la cantidad de datos intercambiados a través de los SEs compartidos también optimiza el tamaño de los SEs y, por ello, el coste/área del diseño.

6.4.3 Estrategias actuales para la DSE

Como recalcábamos en los capítulos anteriores, en la DSE el diseñador puede utilizar dos tipos de mecanismos para evaluar cada punto de diseño: la evaluación basada en las herramientas de simulación y los modelos de estimación analíticos.

Independientemente de la naturaleza implícita del mecanismo de evaluación, ambos han sido ampliamente estudiados y aplicados en la práctica. Por ejemplo, los modelos de estimación analítica se caracterizan por un tiempo de ejecución reducido, lo que les hacen idóneos para la exploración de grandes espacios de diseño de una forma eficiente desde el punto de vista del *run-time*. Sin embargo, estos modelos no están exentos de

inconvenientes. Por un lado, los métodos analíticos suelen enfatizar la importancia y/o influencia de un subconjunto de variables en sus modelos de estimación, mientras que omiten o asumen valores fijados de antemano para las restantes decisiones de diseño. Como consecuencia, muchos de estos modelos se implementan para casos de estudio muy concretos, lo que resalta su inflexibilidad e incapacidad para reutilizarse en otros casos. Por otro lado, los métodos analíticos poseen unas capacidades limitadas (o nulas, en algunos casos) para evaluar el impacto de los comportamientos *no deterministas* del sistema, como por ejemplo, el retardo debido al tiempo de espera para acceder a un recurso compartido por múltiples PEs. Como resultado de esto, los diseñadores de sistemas se ven obligados a recurrir en muchas ocasiones a las herramientas de simulación con el fin de obtener unas estimaciones más precisas.

Como alternativa de los métodos analíticos están las herramientas de simulación a nivel de sistema. Como ya se explicó en el capítulo 2, estos simuladores generalmente permiten que cada diseñador confeccione su modelo de la arquitectura, que suelen realizarse con una composición de unos modelos de componentes configurables (proporcionados por la librería de la propia herramienta). De este modo, el diseñador puede controlar un amplio rango de parámetros de diseño (como la frecuencia o reloj del sistema, esquema de arbitraje en la arquitectura de comunicación o retardo en la escritura/lectura de un dato en la memoria), y realizar unas estimaciones más precisas teniendo en cuenta el comportamiento dinámico dentro del sistema en todo momento. No obstante, estos simuladores a nivel de sistema no son adecuados para explorar grandes espacios de diseño debido fundamentalmente a que: (i) presentan un tiempo de simulación prohibitivamente alto (en comparación con los métodos analíticos), y además, (ii) suelen requerir bastante esfuerzo para crear y configurar el modelo del sistema utilizado en cada DSE.

Como ejemplo, supongamos que se desea mapear una aplicación de 7 tareas en una arquitectura de 6 PEs. Solamente considerando las posibles combinaciones de asignación de las tareas en los PEs, llegamos a que el número de alternativas asciende a 6^7=279936. Entonces, incluso para el más eficiente simulador a nivel de sistema que simulara, por ejemplo, cada punto de diseño en 2 segundos, aún requeriría 6.48 días para evaluar todo este espacio de diseño. Por ello, en la práctica los diseñadores suelen a menudo utilizar estos simuladores conjuntamente con algunos métodos de búsqueda como los algoritmos evolutivos, métodos de gradientes, etc, puesto que dichos algoritmos de búsqueda sólo necesitan explorar un número reducido de puntos de

diseño para asegurar cierto nivel de convergencia hacia el óptimo (y/o los óptimos) global del espacio de diseño.

6.4.4 Metodología de DSE jerárquica

Basándonos en los análisis anteriores, se puede asumir que los dos métodos de evaluación citados con anterioridad tienen sus inconvenientes para explorar grandes espacios de diseño. Es decir, mientras que los métodos analíticos no reúnen el nivel de precisión que se exige en la mayoría de los casos, los simuladores consumen demasiado tiempo para la evaluación de cada modelo de sistema. Por ello, en este capítulo presentamos una estrategia híbrida que combina estos dos métodos de evaluación, siendo ésta capaz de llevar a cabo la DSE a nivel de sistema, y proporcionar al diseñador de sistema la solución óptima para el mapeo de una aplicación en una plataforma MPSoC, es decir, no sólo logrando satisfacer las restricciones de tiempo real, sino que también optimiza la eficiencia en el uso de los recursos del sistema, minimiza el tráfico en el sistema y asegura un reparto equilibrado de la carga computacional entre los distintos PEs disponibles.

A grandes rasgos, nuestra metodología consta de dos etapas independientes, las cuales se comunican entre sí a través de interfaces basado en ficheros de textos. En la primera fase, utilizamos una serie de métodos analíticos para podar el espacio de diseño. De esta forma, el objetivo perseguido en esta fase no es una evaluación de los puntos de diseño con alto nivel de precisión, sino que más bien está orientada hacia una rápida DSE, a la vez que elimina los puntos de diseño que no cumplen explícitamente con los restricciones impuestas por el diseñador. Asi, el resultado obtenido en esta fase es un conjunto reducido de soluciones potenciales o *candidatos*, los cuales se evalúan de un modo más preciso por la herramienta de simulación en la segunda fase de nuestra metodología.

Finalmente, quisiéramos insistir en que la separación de la fase del podado y la fase de simulación aporta dos importantes ventajas a nuestra metodología. Por un lado, garantizamos que el proceso de DSE se pueda realizar de una forma mucho más eficiente (en término de *run-time* dedicado en cada experimento de DSE), a la vez que se asegura un cierto nivel de precisión en la evaluación de las soluciones candidatas. Por otro lado, dicha diferenciación hace que nuestra propuesta sea mucho más flexible, ya que no sólo permite al diseñador realizar una DSE de forma jerarquizada eligiendo y utilizando los métodos de búsqueda más apropiados para cada caso, sino también

facilita la integración de diferentes y nuevas técnicas o simuladores, siempre en cuanto que éstos satisfagan los requisitos de interfaz propuestos en nuestra metodología.

6.4.5 Algoritmos y modelo de estimación para la metodología de DSE jerárquica

En esta sección, explicaremos las distintas etapas intermedias dentro de cada fase de nuestra metodología, así como las interfaces utilizadas para acoplar las distintas partes entres sí.

6.4.5.1 Fase de estimación analítica

Como ya se mencionó con anterioridad, el objetivo principal de esta fase es la exploración y poda del espacio de diseño, esto es, reducir el espacio de diseño a un número de soluciones candidatas (en términos de restricciones de tiempo real) para su posterior evaluación en la fase de simulación. Con este objetivo en mente, se han implementado e integrado tres algoritmos basados en técnicas heurísticas y un modelo de estimación analítico en esta fase.

6.4.5.1.1 Co-diseño hardware/software

Dado un conjunto de tareas y varios tipos de PEs que puede utilizar el diseñador, este algoritmo selecciona el tipo de PE más apropiado para la ejecución de cada una de las tareas. Concretamente en nuestro caso, se decide si una tarea debe procesarse como software embebido en un procesador genérico ARM o es conveniente elegir una implementación hardware a la medida para la misma con el objeto de alcanzar unos específicos requerimientos de rendimiento.

Básicamente, comprobamos el tiempo de cómputo de cada tarea en el procesador genérico de acuerdo a la condición (4.8), de modo que si una tarea satisface dicha condición, ésta será susceptible de ser embebida en el procesador; de lo contrario, se requerirá una implementación hardware para su procesamiento. De esta forma, la salida del algoritmo será un par de vectores que contienen la naturaleza de cada una de las tareas y el tiempo de ejecución asociado al tipo de PE elegido. En este contexto, la naturaleza se refiere al tipo del PE seleccionado (procesador genérico o implementación hardware) para cada una de las tareas.

6.4.5.1.2 Agrupación de tareas

Dado un grafo de tareas, una restricción de tiempo real (RTC) y los vectores obtenidos en la etapa anterior, este algoritmo permite agrupar las tareas en un mínimo número de procesadores virtuales (VPs) o grupos lógicos, al mismo tiempo que garantiza que tiempo total de cómputo de cada VP no supere la RTC. Por otro lado, una característica importante de este algoritmo es que tiende a agrupar las tareas que comparten mayores volúmenes de información entre sí en un mismo VP. Puesto que cada uno de los VPs se asignan a un PE en el paso siguiente, esta cualidad del algoritmo nos aporta tres importantes ventajas:

- al mantener grandes flujos e intercambios de información en un mismo PE, se evita sobrecargar con excesivos volúmenes de datos en la arquitectura de comunicación del sistema.

- asimismo, y como consecuencia de la anterior, se evita producir grandes retardos debido a las contenciones entre los distintos PEs que compiten por los recursos compartidos del sistema.

- finalmente, el algoritmo asegura explícitamente un cierto nivel de equilibrio en el reparto de la carga computacional entre los distintos PEs disponibles.

Una vez concluido este proceso de agrupación de tareas, este algoritmo proporciona al diseñador una lista de VPs (con las correspondientes tareas asignadas en cada uno de los VPs), el tipo asociado a cada VP (el cual viene determinado por la naturaleza de las tareas asignadas a cada VP), y también una lista de enlaces inter-VPs (VPLs, con la indicación de la cantidad de información compartida en cada caso).

6.4.5.1.3 Asignación de los grupos lógicos

En esta etapa se lleva acabo la asignación de los VPs a los PEs, mientras que en el paso siguiente se realiza la asignación de los VPLs. El algoritmo que implementa esta funcionalidad debe tener presente tres tipos de problemas: (i) especificar el tipo y número de los PEs a utilizar, (ii) en qué instancia de cada tipo de PE se debe asignar cada VP, y finalmente, (iii) tomar decisiones sobre la ubicación de los PEs en la plantilla arquitectural.

Con este objetivo en mente, se ha implementado un algoritmo que asigna en primer lugar cada VP a un único PE, es decir, el número y tipo de los PEs vienen determinado por el número y tipo de VPs obtenidos en las etapas anteriores. Una vez realizado esta asignación, generamos exhaustivamente todas las posibles combinaciones de ubicación de los *v* PEs en una plantilla arquitectural que puede albergar un máximo *p* PEs. De esta forma el número total de posibles alternativas es $\frac{p!}{(p-v)!}$.

En este punto, es importante señalar que aunque este algoritmo no trata la problemática de asignación de VPLs en los SEs, sí que tiene en cuenta las posibles combinaciones de tipo y ubicación de SEs dentro de la plataforma, puesto que el modelo de estimación analítico necesita disponer de esta información para la estimación del coste o tiempo de comunicación asociado a la asignación de un VPL en un SE particular. En este caso, si el diseñador dispone de *st* tipos diferentes de SE y se puede utilizar un máximo de *m* SEs en la plataforma, tenemos que el número de posibles combinaciones para los SEs se eleva a st^m. Y por tanto, el número total de posibles escenarios se puede calcular con la expresión (4.11), siendo todos estos escenarios generados y listados en un fichero de salida, donde se representa y codifica cada escenario en un formato de vector.

6.4.5.1.4 Modelo de estimación analítico

Este modelo estima analíticamente el rendimiento de cada uno de los escenarios detectados en el etapa anterior, con el fin de obtener un conjunto reducido de soluciones candidatas que satisfagan la restricción del tiempo real y los objetivos secundarios citados en los apartados anteriores. En nuestra implementación actual, utilizamos el criterio (4.12) para tomar dicha decisión, el cual tiene en cuenta conjuntamente el tiempo de computación y de comunicación asociado a cada PE.

Sin embargo, para poder estimar el tiempo de comunicación incurrido en cada escenario, primeramente debemos asignar el conjunto de VPLs en los SEs. Con este propósito, nuestro modelo confecciona para cada escenario una tabla (o matriz) de conectividad que expresa las condiciones de accesibilidad para cada par de PE y SE presente en la arquitectura. Por tanto, esta tabla indica, por un lado, cuáles son los SEs que puede acceder un PE; y por otro lado, contiene información relativa a la capacidad de almacenamiento remanente de cada SE en cualquier momento y, asimismo, cada valor de la matriz contiene un parámetro que indica el tiempo de lectura/escritura de un byte en los SEs utilizados.

Con esta información, se ha implementado un algoritmo que va asignando de forma iterativa los VPLs a los SEs, dando mayor prioridad en este proceso a los VPLs con mayor volumen de datos implicados en las transacciones. Una vez concluido este proceso, el tiempo de acceso a la memoria de un PE a un SE puede calcularse simplemente como el producto del volumen de datos implicado y el parámetro de lectura/escritura del SE correspondiente. No obstante, aparte del tiempo de acceso a la memoria, el tiempo de protocolo de comunicación asociado a cada transacción también forma parte del retardo de comunicación. Asimismo, también es necesario incluir los posibles retardos dinámicos que pueden surgir en el sistema. Para ello, se ha utilizado la expresión (4.14) para estimar de una forma sencilla y aproximada los posibles impactos de eventual contención en el sistema. La idea subyacente es que cuanto más VPLs utilizan un determinado elemento de comunicación, mayor será la probabilidad de que los PEs asociados a cada VPL sufran un retardo de contención, y por consiguiente, mayor será el tiempo que debe esperar cada PE para acceder a los datos que necesita.

6.4.5.2 Fichero de soluciones candidatas

En nuestra implementación, se ha utilizado una interfaz basada en fichero de texto para enlazar la fase de estimación analítica con la fase de simulación. Concretamente, cada solución candidata es representada y codificada como un vector (o *string*) numérico, de modo que cada cadena de valores puede representar de modo simbólico y unívoco cada punto de diseño en el espacio de diseño.

Es importante subrayar que la elección del esquema (o formato) de representación de los puntos de diseño tiene un fuerte impacto en la escalabilidad y flexibilidad de cualquier esquema de trabajo. De hecho, el formato utilizado en este trabajo permite por un lado (i) definir grandes espacios de diseño de forma modular, pero asegurando a su vez que cada punto de diseño tenga una representación única, como ya se comentó con anterioridad; y por otro lado, (ii) tanto la longitud como los valores contenidos en el *string* no están fijados de antemano, sino que son creados o actualizados automáticamente a partir de las especificaciones de diseño en los ficheros de entrada. Finalmente, también nos hemos decantado por este formato por razones de compatibilidad con NASA, de modo que estos resultados puedan utilizarse conjuntamente en NASA para realizar experimentos de DSE jerárquicas.

6.4.5.3 Fase de simulación

El objetivo de esta fase es evaluar de un modo más preciso el rendimiento de las soluciones candidatas proporcionadas por la fase de exploración y poda analítica. Para ello, utilizamos nuevamente CASSE para llevar a cabo dichas simulaciones y evaluaciones. En este contexto, CASSE evalúa un único punto de diseño en cada simulación, de modo que nuestro algoritmo termina automáticamente las simulaciones desde el momento en que encuentra un solución válida (que cumple los objetivos de diseño impuestos). En el caso contrario, si tras simular todas las soluciones candidatas, ninguna puede alcanzar el nivel de rendimiento exigido, nuestro algoritmo no entregará ninguna solución en su salida.

Es importante recordar que durante la simulación el diseñador puede controlar (y explorar también) numerosas variables de diseño que no se suelen tener en cuenta en los modelos de estimación analítica, tales como el esquema de arbitraje en los elementos de comunicación, mapa de memoria, diferentes dominios de reloj, tamaño y número de tokens que pueden circular concurrentemente por la arquitectura de comunicación, etc. Asimismo, las simulaciones también facilitan una visión más detallada y completa de las fluctuaciones de cargas de trabajo y flujos de transferencias en el sistema, así como de sus impactos sobre el rendimiento del mismo, los cuales pueden analizarse más tarde por el diseñador para tomar las decisiones finales.

6.4.6 Conclusiones

El incremento de la capacidad de integración de los sistemas empotrados actuales posibilita que cada vez más funcionalidades puedan mapearse en un único chip, lo que implica a su vez una expansión exponencial del espacio de diseño. Este desafío demanda nuevas metodologías que permitan explorar este tipo de espacio de diseño de forma rápida y sin sacrificar en exceso el nivel de precisión.

Con este objetivo en mente, presentamos nueva metodología de DSE jerárquica en este capítulo. Concretamente, nos centramos en el problema del mapeo de las aplicaciones de tiempo real en plantillas de arquitectura (o plataforma) basado en MPSoC. Nuestra propuesta consta de dos fases. La primera de ellas consiste en un conjunto de algoritmos que utilizan técnicas heurísticas para la DSE. Asimismo, también hemos implementado un modelo analítico para estimar de forma aproximada el rendimiento de los puntos de diseño encontrados. El propósito de esta fase es podar grandes espacios de diseño de una forma rápida, de manera que sólo un conjunto reducido de soluciones

potenciales sean seleccionadas para la siguiente fase. Seguidamente, en la segunda fase, utilizamos un simulador a nivel de sistema para evaluar de un modo más preciso cada una de las alternativas elegidas en la primera fase, con el fin de encontrar la solución óptima del mapeo teniendo en cuenta los objetivos de diseño impuestos. Finalmente, los resultados experimentales realizados en NASA revelan que, comparado con otras propuestas de DSE basadas sólo en el modelo de estimación analítica o sólo en la simulación, nuestra metodología no sólo puede explorar un mayor espacio de diseño (a la vez de alcanzar soluciones óptimas en el menor tiempo posible), sino también es capaz de encontrar soluciones de mayor calidad en términos de eficiencia en el uso de los recursos, minimización en los tráficos de comunicación del sistema, y reparto equilibrado de las cargas de trabajos en los PEs.

Por último, queremos hacer hincapié en que los experimentos realizados hasta la fecha (y presentados en este capítulo) tienen el fin último de demostrar los conceptos de nuestra metodología de trabajo. No obstante, somos conscientes de que la validación de nuestro esquema de trabajo está basada en una única aplicación y en un escenario MPSoC muy específico y acotado, por lo que creemos firmemente que en nuestra línea de investigación futura se deben utilizar más aplicaciones y mayor variedad de arquitecturas para ir progresivamente validando y cuantificando las fortalezas y debilidades de los métodos y *frameworks* propuestos en esta tesis.

6.5 Conclusiones y trabajo futuro

6.5.1 Conclusiones obtenidas

El creciente nivel de integración en chip permite la entrada en escena de sistemas integrados mucho más complejos. Como resultado, los diseñadores de sistemas tienen que explorar un espacio de diseño en continuo crecimiento con el fin de obtener la solución de diseño óptima teniendo en cuenta varios objetivos. En este contexto, creemos que las metodologías y las herramientas tradicionales de diseño no son apropiadas para explorar eficientemente estos grandes espacios de diseño ni para el diseño de los SoCs modernos. Con el fin de hacer frente a la complejidad de diseño de estos sistemas empotrados, el nivel de abstracción de trabajo se eleva de RTL a un nivel de sistema, donde la DSE se está convirtiendo en una tarea clave en el diseño a nivel de sistema.

Al mismo tiempo que el nivel de abstracción del diseño se ha elevado al nivel de sistema, ha emergido un número importante de herramientas CAD que asisten al diseñador en las tareas del modelado y simulación a nivel de sistema. Sin embargo, a pesar de las numerosas virtudes de estas herramientas, que facilitan a los diseñadores explorar una amplia gama de opciones de diseño durante una fase temprana de diseño, aún se debe generar un modelo del sistema (ESM) con el fin de simular cada alternativa de diseño. La creación del ESM es un proceso que requiere mucho tiempo y esfuerzo, aparte de ser un proceso muy propenso a errores si el diseñador tiene que crear manualmente cada ESM. En consecuencia, podemos afirmar que esto último representa un enorme obstáculo para mejorar la productividad de los diseñadores de sistema, así como ralentiza y entorpece la exploración de grandes espacios de diseño a nivel de sistema. Por otra parte, no debemos olvidar que estos simuladores a nivel de sistema sólo representan una de las piezas para llevar a cabo la DSE, por lo que estas herramientas aún deben integrarse en algún tipo de infraestructura para realizar la exploración del espacio de diseño de una manera sistemática y eficiente.

Esta tesis ofrece diversas técnicas y metodologías que ayudan al diseñador a abordar los problemas mencionados anteriormente, siendo nuestro objetivo último reducir el esfuerzo de los diseñadores en el proceso de diseño a nivel de sistema, así como mejorar la eficiencia de la DSE en la fase inicial de diseño. Para ello, proponemos en primer lugar una metodología de DSE para co-explorar los espacios de diseño multidimensionales. La idea que subyace de esta metodología es que un espacio de

diseño es susceptible de descomponerse en varias dimensiones del espacio de diseño, de manera que los diseñadores puedan seleccionar diferentes estrategias de búsqueda para explorar cada una de las dimensiones. En segundo lugar, hemos implementado una herramienta para generar automáticamente una gran variedad de modelos de arquitecturas. Por lo tanto, ésta libera a los diseñadores de tener que crear manualmente estos modelos y, como consecuencia, los diseñadores pueden explorar con suma facilidad una amplia gama de alternativas arquitecturales en cada experimento de DSE. En tercer lugar, también hemos desarrollado NASA, la cual es una infraestructura genérica y modular que facilita la realización de los experimentos de DSE a nivel de sistema. Por otra parte, NASA es un marco de trabajo unificado, es decir, soporta nuestra metodología de DSE basada en las dimensiones del espacio de diseño, permite la integración de nuestro generador de modelos arquitecturales y también permite a los diseñadores incorporar diferentes herramientas de simulación a nivel de sistema así como diferentes combinaciones de métodos de búsqueda por el simple mecanismo de *plug&play*. Como resultado, NASA proporciona un entorno flexible y reutilizable para explorar los espacios multidimensionales de diseño de una forma completamente sistemática y automática. Por último, también hemos propuesto una metodología de DSE jerárquica para abordar el problema del mapeo de las aplicaciones con restricción de tiempo real en plataformas MPSoC. Combinando las ventajas de los métodos de estimación analítica y evaluaciones basadas en las simulaciones a nivel de sistema, nuestra metodología de DSE jerárquica no sólo puede explorar y podar rápidamente un gran espacio de diseño, sino que también puede obtener soluciones de diseño de mayor calidad teniendo en cuenta varios objetivos de diseño.

Por último, para demostrar y validar diferentes aspectos de nuestro trabajo, también hemos incluido varios conjuntos de experimentos de DSE en esta tesis. Los resultados obtenidos en estos experimentos revelan que las metodologías y las técnicas presentadas en esta tesis pueden efectivamente mejorar la productividad del diseñador y la eficiencia de la DSE a nivel de sistema. No obstante, queremos hacer hincapié en que los experimentos realizados hasta la fecha (y presentados en esta tesis) tienen el fin último de demostrar los conceptos y técnicas de nuestra metodología de trabajo. Asimismo, somos conscientes de que la validación de nuestro esquema de trabajo está basada en una única aplicación y en unos escenarios de MPSoCs muy específicos y acotados, por lo que creemos firmemente que en nuestra línea de investigación futura, se deberá utilizar más aplicaciones y mayor variedad de arquitecturas para ir progresivamente validando y cuantificando las fortalezas y debilidades de los métodos y *frameworks* propuestos en esta tesis.

6.5.2 El trabajo futuro

Automatizar la transformación del código de una aplicación en un grafo de tareas sigue siendo la asignatura pendiente de CASSE y del esquema de trabajo de NASA. Por lo tanto, el diseñador se ve obligado a segmentar manualmente el código de la aplicación basado en su conocimiento y experiencias anteriores. Un posible trabajo futuro podría ser desarrollar un generador que sea capaz de transformar automáticamente el código fuente de una aplicación en un grafo de tareas y/o KPN. Como consecuencia, la eficiencia de cada experimento de DSE así como la productividad de los diseñadores del sistema podrían mejorarse significativamente.

Por otro lado, se ha visto que los experimentos de DSE mostrados en esta tesis han demostrado que nuestra metodología permite explorar grandes espacios de diseño y obtener soluciones óptimas de diseño teniendo en cuenta varios objetivos de diseño. Sin embargo, a pesar de esto, dichos experimentos sólo han considerado de manera explícita un único objetivo primario de diseño (es decir, el rendimiento). Por lo tanto, se tendría que realizar más experimentos de DSE con el fin de probar la capacidad de nuestras metodologías para abordar los problemas de optimización multiobjetivo, en donde se tiene en cuanta, por ejemplo, otros objetivos primarios de diseño tales como el consumo de energía y costes/área, y de este modo, proporcionar al diseñador un conjunto de puntos de diseño que conforman la frontera de Pareto dentro del espacio de diseño explorado.

Por último, hemos resaltado que uno de los puntos fuertes de NASA se encuentra en su flexibilidad, pero este aspecto sólo se ha validado parcialmente. Es decir, hemos demostrado su capacidad para integrar diferentes técnicas de búsqueda, aunque se podría realizar más estudios para demostrar los beneficios de nuestra metodología de DSE basada en las dimensiones del espacio de diseño. Más específicamente, un interesante trabajo de investigación en el futuro podría ser analizar la calidad de la DSE cuando se añada una nueva dimensión del espacio de diseño en NASA.

References

[1] ITRS. International Technology Roadmap for Semiconductors: 2007 Edition. http://www.itrs.net/Links/2007ITRS/Home2007.html

[2] J. Krasner, "Embedded Software Development Issues and Challenges", *Embedded market forecasters*, 2003.

[3] G. Spirakis, "Designing for 65 nm and beyond", *Keynote Address at Design Automation and Test in Europe (DATE'04)*, 2004.

[4] A. D. Pimentel, P. Lieverse, P. van der Wolf, L. O. Hertzberger, and E. Deprettere, "Exploring embedded-systems architectures with Artemis", *Computer, vol. 34, no. 11, pp. 57-63,* Nov. 2001.

[5] D. Densmore, R. Passerone, and A. Sangiovanni-Vincentelli, "A Platform-Based Taxonomy for ESL Design", *IEEE Design & Test of Computers, vol. 23, no. 5, pp. 359-374,* May. 2006.

[6] G. Martin, B. Bailey, and A. Piziali, "ESL Design and Verification: A prescription for Electronic System Level Methodology", *Morgan Kaufmann Series, Elsevier,* 2007.

[7] T. Grötker, S. Liao, G. Martin, and S. Swan, "System Design with SystemC", *Kluwer* 2002.

[8] S. Pasricha, "Transaction Level Modeling of SoC with SystemC 2.0", *Synopsys User Group Conference*, 2002.

[9] L. Cai, S. Verma, and D. Gajski, "Comparison of SpecC and SystemC Languages for System Design", *Technical Report CECS-TR-03-11, UCI,* May. 2003.

[10] Accellera. SystmVerilog 3.0 Accellera's Extensions to Verilog, http://www.accellera.org.

[11] C. Araujo, M. Goes, E. Barros, S. Rigo, R. Azevedo, and G. Araujo, "Platform designer: An Approach for modelling multiprocessor platforms based on SystemC", *Design Automation for Embedded Systems, vol. 10, pp. 253-283, Springer,* 2006.

[12] V. Reyes, "Métodos y herramientas para el diseño a nivel de sistema de plataformas multiprocesador heterogéneas para multimedia", *PhD thesis,*

Department of Electronic Engineering, University of Las Palmas de Gran Canaria, 2008.

[13] K. Keutzer, S. Malik, R. Newton, J Rabaey, and A. Sangiovanni-Vincentelli, "System Level Design: Orthogonalization of Concerns and Platform-Based Design", *IEEE Transactions on Computer-Aided Design of Circuits and Systems, vol. 19, no. 12, pp.1523-1543,* Dec. 2000.

[14] J. A. Rowson and A. Sangiovanni-Vincentelli, "Interface-Based Design", *in Proc. DAC*, pp. 9-13, Jun. 1997.

[15] B. Kienhuis, E. Deprettere, K. Vissers, and P. van der Wolf, "An Approach for Quantitative Analysis of Application-Specific Dataflow Architectures", *In Proc. IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors, pp. 338-249,* Jul. 1997.

[16] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, "A methodology for architecture exploration of heterogeneous signal processing systems", *In Proc. IEEE Workshop on Signal Processing Systems, pp. 181-190,* Oct. 1999.

[17] Toshiba R-CUBE project, 2005, www.semicon.toshiba.co.jp/eng/r_cube

[18] W. Cesário, D. Lyonnard, G. Nicolescu, Y. Paviot, A. Jerraya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor SoC Platforms: A Component-Based Design Approach", *IEEE Design & Test of Computers, vol. 19, no. 6, pp. 52-63,* Dec. 2002.

[19] L. Cai, P. Kritzinger, M. Olivares, and D. Gajski, "Top-Down System Level Design Methodology Using SpecC, VCC and SystemC", *In Proc. DATE*, Ago. 2002.

[20] W. Tibboel, V. Reyes, M. Klompstra, and D. Alders, "System-Level Design Flow Base on a Functional Reference for HW and SW", *In Proc. DAC, pp. 23-28,* 2007.

[21] M. Gries, "Methods for evaluating and covering the design space during early design development", *Integration, the VLSI Journal, vol. 38, no. 2, pp. 131-183,* Dec. 2004.

[22] M. Gries, "Algorithm-architecture trade-offs in network processor design", *PhD thesis, Swiss Federal Institute of Technology (ETH) Zurich, Switzerland,* 2001.

[23] A. Baghdadi, N. Zergainoh, W. Cesario, T. Roudier, and A. Jerraya, "Design space exploration for hardware/software codesign of multiprocessor systems", *In Proc. Int. Workshop on Rapid System Prototyping (RSP), pp. 8-13,* 2000.

[24] C. Ykman-Couvreur, J. Lambrecht, D. Verkest, F. Catthoor, A. Nikologiannis, and G. Konstantoulakis, "System level performance optimization of the data

queuing memory management in high-speed network processors", *In Proc. DAC,* pp. 518-523, 2002.

[25] S. Blythe and R. Walker, "Toward a practical methodology for completely characterizing the optimal design space", *In Proc. Int. Symposium on System Synthesis, pp. 8-13,* 1996.

[26] S. Chaudhuri, S. Blythe, and R. Walker, "A solution methodology for exact design space exploration in a three dimensional design space", *IEEE Transactions on Very Large Scale Integration Systems (VLSI), vol. 5, no. 1, pp. 69-81,* 1997.

[27] M. Auguin, L. Capella, F. Cuesta, and E. Gresset, "CODEF: a system level design space exploration tool", *In Proc. Int. Conference on Acoustics, Speech, and Signal Processing, vol. 2, pp. 1145-1148,* 2001.

[28] R. Szymanek and K. Kuchcinski, "Design space exploration in system level synthesis under memory constraints", *In Proc. EUROMICRO, vol. 1, pp. 29-36,* 1999.

[29] M. Schwiegershausen and P. Pirsch, "A system level design methodology for the optimization of heterogeneous multiprocessors", *In Proc. Int. Symposium on System Synthesis, pp. 162-167,* 1995.

[30] S. Blythe and R. Walker, "Efficient optimal design space characterization methodologies", *ACM Transactions on Design Automation of Electronic Systems, vol. 5, no. 3, pp. 322-336*, 2000.

[31] R. Dutta, J. Roy, and R. Vemuri, "Distributed design space exploration for high-level synthesis systems", *In Proc. DAC, pp. 644-650*, 1992.

[32] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing on-chip communication architectures", *IEEE Transactions on Computer aided Design of Integrated Circuits and Systems, vol. 20, no. 6, pp. 768-783,* 2001.

[33] D. Bruni and L. Benini, "Statistical design space exploration for application-specific unit synthesis", *In Proc. DAC, pp. 641-646,* 2001.

[34] D. Gajski, F. Vahid, S. Narayan, and J. Gong, "System-level exploration with SpecSyn", *In Proc. DAC, pp. 812-817,* 1998.

[35] K. Lahiri, A. Raghunathan, and S. Dey, "Efficient exploration of the SoC communication architecture design space", *In Proc. Int. Conference on Computer Aided Design (ICCAD), pp. 424-430,* 2000.

[36] V. Srinivasan, S. Radhakrishnan, and R. Vemuri, "Hardware software partitioning with integrated hardware design space exploration", *In Proc. DATE, pp. 28-35,* 1998.

[37] I. Ahmad, M. Dhodhi, and C. Chen, “Integrated scheduling allocation and module selection for design-space exploration in high-level synthesis”, *In IEE Computers and Digital Techniques, vol. 142, no. 1, pp. 65-71,* 1995.

[38] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, “Design space exploration of network processor architectures”, *In P. Crowley, M. Franklin, H. Hadimioglu, P. Onufryk (Eds.), Network Processor Design: Issues and Practices, Morgan Kaufmann Publishers, vol. 1, pp. 55-89,* 2002.

[39] T. Blickle, J. Teich, and L. Thiele, “System-level synthesis using evolutionary algorithms”, *In IEEE Design Automation for Embedded Systems, vol. 3, no. 1, pp. 23-58,* 1998.

[40] M. Palesi and T. Givargis, “Multi-objective design space exploration using genetic algorithms”, *In Proc. Int. Symposium on Hardware/Software Co-design (CODES), pp. 67-72,* 2002.

[41] G. Ascia, V. Catania, and M. Palesi, “Design space exploration methodologies for IP-based system-on-a-chip”, *In Proc. Int. Symposium on Circuits and Systems, vol. 2, pp. 364-367,* 2002.

[42] G. Ascia, V. Catania, and M. Palesi, “A framework for design space exploration of parameterized VLSI systems”, *In Proc. ASP-DAC/VLSI Design, pp. 245-250,* 2002.

[43] C. Erbas, S. C. Erbas, and A. D. Pimentel, “A multiobjective optimization model for exploring multiprocessor mappings of process networks”, *In Proc. CODES/ISSS,* pp. 182-187, 2003.

[44] B. De Smedt and G. Gielen, “WATSON: a multi-objective design space exploration tool for analog and RF IC design”, *In Proc. IEEE Custom Integrated Circuits, pp. 31-34,* 2002.

[45] R. P. Dick and N. K. Jha, “MOCSYN: Multiobjective core-based single-chip system synthesis”, *In Proc. DATE, pp. 263-270,* 1999.

[46] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, “Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation”, *In Proc. Languages, compilers and tools for embedded systems: software and compilers for embedded systems,* Jun. 2002.

[47] G. Hekstra, G. L. Hei, P. Bingley, and F. Sijstermans, “TriMedia CPU64 design space exploration”, *In Proc. Int. Conference on Computer Design: VLSI in Computers and Processors, pp. 599-606*, 1999.

[48] D. S. Rao and F. Kurdahi, “Hierarchical design space exploration for a class of digital systems”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 1, no. 3, pp. 282-295*, 1993.

[49] Z. J. Jia, T. Bautista and A. Núñez, “Real-time visual tracking system modelling in MPSoC using platform-based design”, *In Proc. IS&T/SPIE Symposium on Electronic Imaging*, Jan. 2009.

[50] Z. J. Jia, A. D. Pimentel, M. Thompson, T. Bautista, and A. Núñez, “NASA: A Generic Infrastructure for System-level MP-SoC Design Space Exploration”, *In Proc. ESTIMedia*, Oct. 2010.

[51] M. Volckmann, “Electronic System Level Tools: Global Market Demand Analysis”, *White paper of Venture Development Corporation*, www.vdc-corp.com

[52] IEEE 1666TM SystemC standard, 2006, http://www.systemc.org

[53] Transaction Level Modelling Standard 2.0, 2008, http://www.systemc.org

[54] L. Cai and D. Gajski, “Transaction Level Modeling: An Overview”, *In Proc. Int. Conference on Hardware/Software Codesign and System Synthesis, pp. 19-24,* 2003.

[55] TLM 2.0 user manual and kit. Available from the Open SystemC Initiative (OSCI) at: www.systemc.org

[56] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, “A methodology for architecture exploration of heterogeneous signal processing systems”, *Journal of VLSI Signal Processing for Signal, Image and Video Technology, vol. 29, no. 3, pp. 197-207*, 2001.

[57] J. E. Coffland and A. D. Pimentel, “A software framework for efficient system-level performance evaluation of embedded systems”, *In Proc. Symposium on Applied Computing, pp. 666-671,* Mar. 2003.

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

[59] C. Erbas, “System-Level Modelling and Design Space Exploration for Multiprocessor Embedded System-on-Chip Architectures”, *PhD thesis, Informatics Institute, University of Amsterdam*, 2006.

[60] A. D. Pimentel, “The Artemis workbench for system-level performance evaluation of embedded systems”, *Int. Journal of Embedded Systems, vol. 3, no. 3, pp. 181-196,* 2008.

[61] T. Kogel, A. Wieferink, D. Bussaglia, R. Leupers, G. Ascheid, H. Meyr, D. Bussaglia, and M. Ariyamparambath, “Virtual Architecture Mapping: A SystemC based Methodology for Architectural Exploration of System-on-Chip Designs”, *In Proc. Int. workshop on Systems, Architectures, Modeling and Simulation (SAMOS), pp. 138-148,* 2003.

[62] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, “Taming heterogeneity – the Ptolemy approach”, *In Proc. vol. 91, no. 1, pp.127-144,* Jan. 2003.

[63] E. A. Lee and T. M. Parks, “Data flow process networks”, *In Proc. vol. 83, no. 5, pp.773-799,* 1995.

[64] E. A. Lee and D. G. Messerschmitt, “Synchronous Data Flow”, *In Proc. vol. 75, no. 9, pp.1235-1245,* 1987.

[65] J. T. Buck, “Scheduling Dynamic Data flow Graphs with Bounded Memory using the Token Flow Model”, *PhD thesis, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley*, 1993.

[66] G. Kahn, “The semantics of a simple language for parallel programming”, *In Proc. of the IFIP, pp. 471-475,* 1974.

[67] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, “Metropolis: An integrated electronic system design environment”, *Computer, vol. 36, no. 4, pp.45-52,* 2003.

[68] http://daedalus.liacs.nl/Site/Daedalus%20home.html.

[69] T. Austin, E. Larson, and D. Ernst, “SimpleScalar: An infrastructure for computer system modeling”, *IEEE Computer, vol. 35, no. 2, pp. 59-67,* Feb. 2002.

[70] T. Taghavi, M. Thompson, and A. D. Pimentel, “Visualization of Computer Architecture Simulation Data for System-level Design Space Exploration”, *In Proc. Int. Symposium on Systems, Architectures, Modelling and Simulation (SAMOS '09),* Jul. 2009.

[71] T. Taghavi and A.D. Pimentel, “Visualization of Multi-Objective Design Space Exploration for Embedded Systems”, *In Proc. Euromicro Conference on Digital System Design (DSD'10)*, Sep. 2010.

[72] P. van Stralen and A. D. Pimentel, “A High-level Microprocessor Power Modelling Technique based on Event Signatures”, *Journal of Signal Processing Systems, vol. 60, no. 2, pp. 239-250, Springer*, 2010.

[73] P. van Stralen and A. D. Pimentel, “Signature-based Microprocessor Power Modelling for Rapid System-level Design Space Exploration”, *In Proc. IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia '07), pp. 33-40*, 2007.

[74] P. van der Wolf, E. de Kock, T. Hendriksson, W. Kruijtzer, and G. Essink, “Design and Programming of Embedded Multiprocessors: An Interface-Centric Approach”, *In Proc. CODES+ISSS, pp. 206-217,* Sep. 2004.

[75] M. Thompson, H. Nikolov, T. Stefanov, A. Pimentel, C. Erbas S. Polstra, and E. Deprettere, “A Framework for Rapid System-level Exploration, Synthesis, and Programming of Multimedia MP-SoC”, *In Proc. CODES+ISSS, pp. 9-14,* Oct. 2007.

[76] S. Verdoolaege, H. Nikolov and T. Stefanov, “PN: a tool for improved derivation of process networks”, *EURASIP Journal on Embedded Systems*, 2007.

[77] S. Gheorghita, S. Stuijk, T. Basten, and H. Corporaal, “Automatic Scenario Detection for Improved WCET Estimation”, *In Proc. DAC, pp. 101-104,* Jun. 2005.

[78] T. Meyerowitz, A. Sangiovanni-Vincentelli, M. Sauermann, and D. Langen, “Source-Level Timing Annotation and Simulation for a Heterogeneous Multiprocessor”, *In Proc. DATE, pp. 276-279,* Mar. 2008.

[79] V. Reyes, W. Kruijtzer, T. Bautista, and A. Núñez, “A Unified System-level Modelling and Simulation Environment for MPSoC design: MPEG-4 Decoder Case study”, *In Proc. DATE,* pp. 1-6, 2006.

[80] W. Wolf, B. Ozer, and T. Liu, “Smart Cameras as Embedded Systems”, *IEEE Computer, vol. 35, no. 9, pp. 48-53,* 2002.

[81] H. Ziegler, B. So, M. Hall, and P. C. Diniz, “Coarse-Grain Pipelining on Multiple FPGA Architectures”, *In Proc. Symposium on Field Programmable Custom Computing Machines (FCCM’02), pp. 77-86,* 2002.

[82] J. Schlessman, C. Y. Chen, B. Ozer, K. Fujino, D. Itoh, and W. Wolf, “Hardware/Software Co-Design of an FPGA-based Embedded Tracking”, *In Proc. IEEE System Conference on Computer Vision and Pattern Recognition Workshop,* 2006.

[83] S. C. Wong, M. Jasiunas, and D. Kearney, “Towards a reconfigurable tracking system”, *In Proc. Int. Conference on Field Programmable Logic and Applications, pp. 456-462,* 2005.

[84] M. W. Eklund, G. Ravichandran, M. M. Trivedi, and S. B. Marapane, “Real-Time Visual Tracking Using Correlation Techniques”, *In Proc. Workshop on Applications of Computer Vision, pp. 256-263,* 1994.

[85] R. Canals, A. Roussel, J. Famechon, and S. Treuillet, “A Biprocessor-Oriented Vision-Based Target Tracking System”, *IEEE Transactions on Industrial Electronics, vol. 49, no. 2, pp. 500-506,* 2002.

[86] N. Sawasaki, T. Morita, and T. Uchiyama, “Design and Implementation of High-Speed Visual Tracking System for Real-Time Motion Analysis”, *In Proc. Int. Conference on Pattern Recognition, vol. 3, pp. 478-483,* 1996.

[87] Z. J. Jia, M. Ferrer, C. Travieso, and J. Alonso, “Biometric base on the ridges of the palm skin over the head of the second metacarpal bone”, *IEE Electronics Letters, vol. 42, no. 7, pp 391-393,* 2006.

[88] Z. J. Jia, M. Ferrer, J. Alonso, C. Travieso, and F. Arbelo, “Autenticación de personas a partir de la biometría de la región dígito palmar”. *Vector Plus, Fundación Universitaria de Las Palmas, no. 27, pp. 27-34,* 2006.

[89] C. Guerra, “Contribuciones al seguimiento de objetos precategóricos”, *PhD thesis, Departament of Intelligent Systems, University of Las Palmas de Gran Canaria*, 2005.

[90] ARM Developer Suite, Version 1.2, www.arm.com

[91] V. Reyes, T. Bautista, G. Marrero, P. P. Carballo, and W. Kruijtzer, “CASSE: A System-level Modelling and Design Space Exploration Tool for Multiprocessor Systems-on-Chip”, *In Proc. Euromicro Symposium on Digital System Design (DSD’04), pp. 476-483,* 2004.

[92] C. Lee, S. Kim, and S. Ha, “A Systematic Design Space Exploration of MPSoC based on Synchronous Data Flow Specification”, *Journal of Signal Processing System, vol. 58, no. 2, pp. 193-213,* Feb. 2010.

[93] D. Shin, A. Gerstlauer, J. Peng, R. Dömer, and D. Gajski, “Automatic Generation of Transaction-Level Models for Rapid Design Space Exploration”, *In Proc. CODES+ISSS'06, pp. 64-69,* Oct. 2006.

[94] Z. J. Jia, A. Pimentel, M. Thompson, T. Bautista, and A. Núñez, “A Generic Infrastructure for System-level MP-SoC Design Space Exploration”, *submitted to IEEE Transactions on Computer-Aided Design of Circuits and Systems* (on Jul. 2010).

[95] G. Beltrame, C. Bolchini, L. Fossati, A. Miele, and D. Sciuto, “ReSP: A Non-Intrusive Transaction-Level Reflective MPSoC Simulation Platform for Design Space Exploration”, *In Proc. Asia and South Pacific Design Automation conference (ASPDAC'08), pp. 673-678,* 2008.

[96] A. Cassidy, J. Paul, and D. Thomas, “Layered, multi-threaded, high-level performance design”, *In Proc. Design, Automation and Test in Europe (DATE'03), vol. 1, pp. 10954-10959,* Mar. 2003.

[97] J. Teich, T. Blickle, and L. Thiele, “An Evolutionary Approach to System-Leel Synthesis”, *In Proc. Workshop on Hardware/Software Co-Design (Codes/CASHE'97), pp. 167-171,* 1997.

[98] M. Kim, S. Banerjee, N. Dutt, and N. Venkatasubramanian, “Design Space Exploration of Real-time Multi-media MPSoCs with Heterogeneous Scheduling Policies”, *In Proc. CODES+ISSS'06, pp. 16-21,* Oct. 2006.

[99] Z. J. Jia, T. Bautista, A. Núñez, C. Guerra, and M. Hernández, “Design space exploration and performance analysis for the modular design of CVS in a heterogeneous MPSoC”, *In Proc. Conference on Reconfigurable Computing and FPGA (ReConFig 2008), pp. 193-198,* Dec. 2008.

[100] G. Ascia, V. Catania, and M. Palesi, “A GA-Based Design Space Exploration Framework for Parameterized System-On-A-Chip Platforms”, *IEEE Trans. on Evolutionary Computation, vol. 8, no. 4, pp. 329-346,* Aug. 2004.

[101] M. Palesi and T. Givargis, “Multi-objective Design Space Exploration Using Genetic Algorithms”, *In Proc. Symposium on Hardware/Software codesign (CODES'02), pp. 67-72,* May. 2002.

[102] www.multicube.eu

[103] Z. J. Jia, T. Bautista, and A. Núñez, “Real-Time Application to Multiprocessor-System-on-Chip Mapping Strategy for a System-Level Design Tool”, *IEE Electronics Letters, vol. 45, no. 12, pp. 613-615,* 2009.

[104] D. Lyonnard, S. Yoo, A. Baghdadi, and A. Jerraya, “Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System-on-Chip”, *In Proc. Design Automation Conference (DAC'01),* Jun. 2001.

[105] S. Abdi and D. Gajski, “Automatic Generation of Equivalent Architecture Model from Functional Specification”, *In Proc. Design Automation Conference (DAC'04),* Jun. 2004.

[106] F. Angiolini, J. Ceng, R. Leupers, F. Ferrari, C. Ferri, and L. Benini, “An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration”, *In Proc. Design, Automation and Test in Europe (DATE'06), pp. 1145-1150,* Mar. 2006.

[107] L. Thiele, I. Bacivarov, W. Haid, and K. Huang, “Mapping Applications to Tiled Multiprocessor Embedded Systems”, *In Proc. Conference on Application of Concurrency to System Design (ACSD 2007), pp. 29-40,* Jul. 2007.

[108] G. Palermo, C. Silvano, and V. Zaccaria, “A flexible framework for Fast Multi-Objective Design Space Exploration of Embedded Systems”, *In Proc. PATMOS 2003, vol. 2799, pp. 249-258,* Sep. 2003.

[109] S. Künzli, L. Thiele, and E. Zitzler, “A Modular Design Space Exploration Framework for Embedded Systems”, *In Proc. Computer & Digital Techniques, pp. 183-192*, 2005.

[110] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, “PISA – A Platform and Programming Language Independent Interface for Search Algorithms”, *In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb and L. Thiele, editors, Evolutionary*

Multi-Criterion Optimization (EMO 2003), vol. 2632/2003 of LNCS, pages 494-508. Springer-Verlag Heidelberg, 2003.

[111] http://www.tik.ee.ethz.ch/sop/pisa/

[112] J. Madsen, T. K. Stidsen, P. Kjarulf, and S. Mahadevan, "Multi-Objective Design Space Exploration of Embedded System Platforms", *In Proc. IFIP, vol. 225, pp. 185-194,* 2006.

[113] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", *W. H. Freeman & Co.*, 1990.

[114] J. L. Ayala, "Power estimation and power estimation policies for processor-based system", *PhD thesis, Department of Electronic Engineering, Universidad Politécnica de Madrid,* 2005.

[115] S. Künzli, F. Poetti, L. Benini, and L. Thiele, "Combining Simulation and Formal Methods for System-Level Performance Analysis", *In Proc. Design Automation and Test in Europe (DATE '06),* Mar. 2006.

[116] P. van Strelen and A. D. Pimentel, "A Trace-based Scenario Database for High-level Simulation of Multimedia MP-SoCs", *In Proc. Conference on Embedded Computer Systems: Architectures, Modelling and Simulation (SAMOS '10), pp. 11-19,* Jul. 2010.

[117] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors", *ACM Computing Surveys (CSUR), vol. 31, no. 4, pp. 406-471,* 1999.

[118] J. Buch, S. Ha, E. A. Lee, and D. D. Masserchmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems", *Journal on Computer Simulation, Special Issue on Simulation Software Management, vol. 4, pp. 155-182,* Apr. 1994.