Degree Project in Computer Science and Engineering, specializing in Embedded Systems
Second cycle, 30 credits

Design space exploration for co-mapping of periodic and streaming applications in a shared platform

ZHANG YUHAN
Design space exploration for co-mapping of periodic and streaming applications in a shared platform

ZHANG YUHAN

Master's Programme, Embedded Systems, 120 credits
Date: October 9, 2023
Supervisor: Rodolfo Jordão
Examiner: Ingo Sander
School of Electrical Engineering and Computer Science
Swedish title: Validering av designlösningar för utforskning av rymden för samkartläggning av periodiska och strömmande applikationer i en delad plattform
Abstract

As embedded systems advance, the complexity and multifaceted requirements of products have increased significantly. A trend in this domain is the selection of different types of application models and multiprocessors as the platform. However, limited design space exploration techniques often perform one particular model, and combining diverse application models may cause compatibility issues. Additionally, embedded system design inherently involves multiple objectives. Beyond the essential functionalities, other metrics always need to be considered, such as power consumption, resource utilization, cost, safety, etc. The consideration of these diverse metrics results in a vast design space, so effective design space exploration also plays a crucial role.

This thesis addresses these challenges by proposing a co-mapping approach for two distinct models: the periodically activated tasks model for real-time applications and the synchronous dataflow model for digital signal processing. Our primary goal is to co-map these two kinds of models onto a multi-core platform and explore trade-offs between the solutions. We choose the number of used resources and throughput of the synchronous dataflow model as our performance metrics for assessment.

We adopt a combination method in which periodic tasks are given precedence to ensure their deadlines are met. The remaining processor resources are then allocated to the synchronous dataflow model. Both the execution of periodic tasks and the synchronous dataflow model are managed by a scheduler, which prevents resource contention and optimizes the utilization of available processor resources. To achieve a balance between different metrics, we implement Pareto optimization as a guiding principle in our approach.

This thesis uses the IDeSyDe tool, an extension of the ForSyDe group’s current design space exploration tool, following the Design Space Identification methodology. Implementation is based on Scala and Python, running on the Java virtual machine.

The experiment results affirm the successful mapping and scheduling of the periodically activated tasks model and the synchronous dataflow model onto the shared multi-processor platform. We find the Pareto-optimal solutions by IDeSyDe, strategically aiming to maximize the throughput of synchronous dataflow while concurrently minimizing resource consumption.

This thesis serves as a valuable insight into the application of different models on a shared platform, particularly for developers interested in utilizing
IDeSyDe. However, due to time constraints, our test case may not fully encompass the potential scalability of our thesis method. Additional tests can demonstrate the better effectiveness of our approach. For further reference, the code can be checked in the GitHub repository at *.

**Keywords**

Design Space Exploration, Periodically activated tasks, Synchronous dataflow, IDeSyDe

---

*https://github.com/forsyde/IDeSyDe/tree/develop*
Sammanfattning

Allt eftersom inbyggda system utvecklas, blir komplexiteten och de mångfacetterade kraven av produkter har ökat avsevärt. En trend inom detta område är urval av olika typer av applikationsmodeller och multiprocessorer som plattformen. Dock begränsad design utrymmes utforskning tekniker ofta utföra en viss modell, och kombinera olika applikationsmodeller kan orsaka kompatibilitetsproblem. Dessutom inbyggt systemdesign i sig involverar flera mål. Utöver de väsentliga funktionerna, andra mått måste alltid beaktas såsom strömförbrukning, resurs användning, kostnad, säkerhet, etc. Övervägandet av dessa olika mått resulterar i ett stort designutrymme spelas så effektiv designutrymmes utforskning också en avgörande roll.


Vi använder en kombinationsmetod där periodiska uppgifter ges företräde för att säkerställa att deras tidsfrister hålls. Den återstående processorn resurser allokeras sedan till den synkrona dataflödesmodellen. Både utförandet av periodiska uppgifter och den synkrona dataflödesmodellen är hanteras av en schemaläggare, vilket förhindrar resursstrid och optimiserar utnyttjandet av tillgängliga processorresurser. För att uppnå en balans mellan olika mått, implementerar vi Pareto-optimering som en vägledande princip i vårt tillvägagångssätt.

Denna avhandling använder verktyget IDeSyDe, en förlängning av ForSyDe gruppens nuvarande verktyg för utforskning av designutrymme, efter Design Space Identifieringsmetodik. Implementeringen är baserad på Scala och Python, körs på den virtuella Java-maskinen.

Experimentresultaten bekräftar den framgångsrika kartläggningen och schemaläggningen av den periodiskt aktiverade uppgiftsmodellen och det synkrona dataflödet modell på den delade flerprocessorplattformen. Vi finner Pareto-optimal lösningar av IDeSyDe, strategiskt inriktade på att maximera genomströmningen av synkront dataflöde samtidigt som resursförbrukningen minimeras.

Denna uppsats fungerar som en värdefull inblick i tillämpningen av
olika modeller på en delad plattform, särskilt för utvecklare IDeSyDe. På grund av tidsbrist kanske vårt testfall inte är fullt ut omfattar den potentiella skalbarheten hos vår avhandlingsmetod. Ytterligare tester kan visa hur effektiv vår strategi är. För ytterligare referens, koden kan kontrolleras i GitHub.

Nyckelord

Designutrymmesutforskning, Periodiskt aktiverade uppgifter, Synkron dataflöde, IDeSyDe

*https://github.com/forsyde/IDeSyDe/tree/develop
Acknowledgments

This thesis was conducted at KTH. First, I am very grateful to Professor Ingo Sander for providing this master’s project. This thesis has been an invaluable learning experience for me. I have gained a deeper understanding of Design Space Exploration and have become proficient in using the software tool IDeSyDe. This paper allows me to apply theory to practice.

I would also like to express my sincere appreciation to my supervisor, Rodolfo Jordão. His guidance and support have been instrumental throughout this project. When I initially started, I was unfamiliar with IDeSyDe, but he patiently provided me with valuable guidance and transformed my initial anxiety into confidence.

Lastly, I want to thank my family for their unwavering support. During times of difficulty, they have been my strong foundation and source of encouragement.

Stockholm, October 2023
Zhang Yuhan
# Contents

1 Introduction
   1.1 Background ............................................. 1
   1.2 Problem .................................................. 2
   1.3 Purpose ................................................... 3
   1.4 Goals ..................................................... 3
   1.5 Research Methodology ................................. 4
   1.6 Structure of the thesis ............................... 4

2 Background
   2.1 Design Space Exploration ......................... 5
   2.2 Design Space Identification ....................... 8
      2.2.1 Design models and decision models .......... 10
      2.2.2 Identification stage .............................. 10
      2.2.3 Bidding and exploration stage ................. 10
      2.2.4 Reverse identification stage ................... 11
   2.3 Analyzable Application Models ................. 11
      2.3.1 Independent Periodic tasks ................... 11
      2.3.2 Synchronous Dataflow ......................... 14
   2.4 Design Constraints and Performance Metrics .... 15
   2.5 Platform Structure ................................. 16
   2.6 Related work ......................................... 17
   2.7 IDeSyDe ................................................. 18
      2.7.1 Structure of IDeSyDe ......................... 18
      2.7.2 Design model ..................................... 19
      2.7.3 Decision model and Standard Decision Model .... 20
      2.7.4 Critical parameters of IDeSyDe .............. 22
      2.7.5 Periodic tasks in IDeSyDe .................... 22
      2.7.6 SDF in IDeSyDe ................................. 23
      2.7.7 Platform in IDeSyDe ............................ 24
References 59

A Main equations 63
# List of Figures

1.1 Design flow ................................. 2  
2.1 Kienhuis’ Y-chart ............................... 6  
2.2 An example of a Pareto front .......................... 7  
2.3 Function of DSI and DSE in design flows. (Jordao et.al, 2023) .......................... 9  
2.4 Enhanced overview of the DSI-based DSE activity and the information flow of IDeSyDe. The dashed arrow from $M$ to $M'$ represents direct DSE methods and tools for comparison with DSI-based DSE and IDeSyDe. (Jordao et.al, 2023) .......................... 9  
2.5 Periodic task ................................. 13  
2.6 Rate-monotonic schedule for the task set: (5, 2), (2, 1). .......................... 14  
2.7 SDF graph example ............................... 15  
2.8 Tiled-based multicore platform with time-division multiplexing arbitrated communication .......................... 17  
2.9 Structure of IDeSyDe before version 0.5.0. .......................... 18  
3.1 IDeSyDe Tool Test Conditions .......................... 29  
4.1 States of Real-time operating system (RTOS) for a task .......................... 33  
4.2 Simulation effect ............................... 34  
4.3 RTOS for mock solution ............................ 35  
4.4 Idea of DSE by IDeSyDe ............................ 36  
4.5 Example SDF Graphs with WCET and WCCT table .......................... 42  
5.1 The first solution by hand ............................ 44  
5.2 The second solution by hand ............................ 46  
5.3 Test Method ................................. 47  
5.4 Solution by IDeSyDe ............................... 48
List of Figures
# List of acronyms and abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP</td>
<td>Constraint Programming</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>CSDF</td>
<td>Cyclo-Static Data Flow</td>
</tr>
<tr>
<td>DSE</td>
<td>design space exploration</td>
</tr>
<tr>
<td>DSI</td>
<td>design space identification</td>
</tr>
<tr>
<td>HSDF</td>
<td>Homogeneous Synchronous Dataflow</td>
</tr>
<tr>
<td>MDE</td>
<td>model-driven-engineering</td>
</tr>
<tr>
<td>MoC</td>
<td>Model of Computation</td>
</tr>
<tr>
<td>MPSoCs</td>
<td>Multi-Processor Systems-on-Chips</td>
</tr>
<tr>
<td>RM</td>
<td>Rate-Monotonic</td>
</tr>
<tr>
<td>RTOS</td>
<td>Real-time operating system</td>
</tr>
<tr>
<td>SDF</td>
<td>Synchronous Dataflow</td>
</tr>
<tr>
<td>TDM</td>
<td>Time-Division Multiplex</td>
</tr>
<tr>
<td>WCCT</td>
<td>Worst-Case Communication Time</td>
</tr>
<tr>
<td>WCET</td>
<td>Worst-Case Execution Time</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter introduces the research background of the thesis, including the description of current challenges, an introduction to the design flow for embedded systems, and an overview of the related notions, such as design space exploration (DSE). It outlines the thesis’s purpose and objective, presenting the research’s motivations. Additionally, this chapter briefly describes the methodology and outlines the structure of the thesis.

1.1 Background

With the development of technology, the requirements and complexity of embedded systems are increasing. One of the trends is to co-map the different types of application models to the shared multiprocessors which can extend its applicability.

However, there are three challenges: The first one is the inherent properties of models: Different types of models might not be trivially compatible, and combining them may generate undefined behavior.

The second issue is the current limited technology: During the design process, the techniques are used to analyze a particular model. For example, although in recent years, significant progress has been made in studying Synchronous Dataflow (SDF), they focus on correctness rather than effectiveness.

The third problem is the vast design space. Real-life embedded system design always involves multi-objectives. Different system metrics need to be considered, such as cost, power consumption, security, and so on. These factors contribute to complex and vast design space, with multiple potential solutions.
Figure 1.1: Design flow

Figure 1.1 shows the design flow considered in this thesis. It involves several significant stages, including defining application models, determining the platform structure, designing design constraints, and specifying the performance metrics. The application model describes the system’s functions and can take the form of a model of computation, while the platform defines the structure of the hardware. Design constraints encompass specific limitations or requirements, which can be real-time performance, power, and energy consumption, etc. Performance metrics mainly evaluate the performance of the system. They can involve different aspects, such as Worst-Case Communication Time (WCCT), speed, etc.

Once these metrics are considered, DSE becomes the cornerstone of the design process. It takes various parameters, including application models, platform structure, design constraints, and performance metrics, as inputs.

DSE maps and schedules these application models to the platform. Different parameters lead to diverse mapping methods, generating a range of solutions. These solutions act as the outputs of DSE, guiding subsequent phases like synthesis, compilation, and implementation phases.

1.2 Problem

The project mainly focuses on 1) Verify that it is possible to combine a periodically activated tasks model and a SDF Model of Computation (MoC) in a shared platform, and 2) If exists, find the trade-off between the solutions.
1.3 Purpose

The project intends to improve the current situation based on the current research. This project benefits companies focusing on embedded systems and the Internet of Things. It can not only help their tasks meet deadlines to ensure the normal progress of tasks but also achieve trade-offs between system metrics. Electronic equipment with rich functions and security guarantees can facilitate our lives and improve life quality.

In addition, this project uses software development and an extension of the design space exploration tool, IDeSyDe. The solution of mapping SDF models and periodic task models to the shared platform, trade-off the metrics, and related contents to IDeSyDe are introduced in this thesis, so it is beneficial for embedded developers and people who are interested in IDeSyDe.

1.4 Goals

The goal of this project is to try to combine a synchronous dataflow model and a periodically activated tasks model in the shared platform. If at least one combination exists, find the trade-off between the solutions. Generally, the goals can be divided into three parts: mastering the notions and technology related to the thesis, becoming familiar with the rationale of IDeSyDe [1] and ForSyDe IO [2], and finally implementing and testing it.

Specifically, the goals have been divided into the following six sub-goals:

1. Gain a solid understanding of concepts or knowledge related to the thesis, including SDF, periodically activated tasks, and DSE, etc.

2. Acquire proficiency in using the design space exploration tool, IDeSyDe, with ForSyDe IO serving as the test files.

3. Design a model for combining SDF and periodic tasks.

4. Find an illustrative example platform.

5. Map the model onto the platform architecture with the established scheduling mechanism.

6. Implement the code using IDeSyDe and test the results.
1.5 Research Methodology

This section presents the assumptions and research methods. The concrete contents will be expressed by Chapter 3.

The methods that are used in this project are mainly carried out in two parts: theory study and experiments.

The theory part will be based on the literature [3] [4] and the supervisor’s a priori solution [5]. It includes how to deal with the periodically activated tasks model and synchronous dataflow models in the platform and combine them in a shared platform. The hypothesis will be validated through software development and extension of the ForSyDe group’s current design space exploration tool, IDeSyDe [1]. This extension allows the student to validate any DSE related to the research question and hypothesis, and thus also serve as the medium to collect data regarding the existence of a solution regarding the research question.

1.6 Structure of the thesis

Chapter 2 provides essential information about DSE, including methods for finding trade-offs and related work. It introduces detailed modeling. Additionally, it introduces snippets of the code of IDeSyDe. Chapter 3 presents the methodology and method used to solve the problem. Chapter 4 details the implementation method employed to carry out the DSE process. Chapter 5 presents and analyzes the results generated manually and using IDeSyDe. Chapter 6 discusses the overall condition of IDeSyDe. Chapter 7 offers the thesis conclusion based on the findings of the trade-offs and provides insight for future work.
Chapter 2

Background

DSE is a vital stage for designing embedded systems. This chapter focuses on DSE and design space identification (DSI) methodology. It presents the concrete process and various existing solutions of DSE. Additionally, this chapter describes some vital code snippets of DSE implemented by IDeSyDe.

2.1 Design Space Exploration

Embedded systems are widely used in various fields such as consumer electronics, aerospace, office automation, communications, etc. With the development of technology, embedded systems face increasing challenges, which makes them more complex and demanding. More different applications are considered to add to mapping to the processors, not only do they need to meet their deadlines, but there are many system metrics such as power consumption, cost, execution time, and so on. Balancing these metrics leads to a vast design space.

Meanwhile, for the design of embedded systems in real life, different parameters in case of high performance and low cost are in conflict, so it leads to potential solutions with different focuses. DSE plays a crucial role in this context. By exploring the space, DSE can help find one or more solutions that satisfy the goals.

Kienhuis et al. provided a Y-chart approach that connects DSE and performance analysis. It gives valuable insight, which leads to a more optimized embedded design. The Y-chart is depicted in Figure 2.1 [6].
The fundamental concept of the Y-chart is to provide insight into dealing with different dimensions of the system, which involves three specific aspects: application, architecture, and DSE. The Y-chart can simply be comprehended as the process of mapping the applications onto the multiprocessor architecture and then analyzing the performance. The mapping step involves various choices, such as task allocation, schedule, parameters, etc. Once the mapping is finished, the system needs to be evaluated, namely performance analysis, which involves performance metrics such as throughput, used resources, and other relevant parameters. Performance numbers are used to evaluate the performance model.

The design of embedded systems always involves multiple objectives which leads to multi-dimensional optimization problems. To find one or more suitable models, the Y-chart approach is an iterative process. During each iteration, performance analyses for a single design point would be from three perspectives: accuracy, speed, and effort. Performance numbers play a significant role which can help to identify the requirements that are met and can be compared with other solutions. Given the vast design space, DSE has to do many iterations. During the process of iteration, the parameters of DSE, such as mapping, schedule, or architecture, can be adjusted to fine-tune.

In this context, the Pareto dominance relation is typically used to identify the optimal trade-off solutions for multi-objective problems [4]. The rule of Pareto dominance is to identify solutions that improve at least one objective without degrading the performance of any other objectives.
Solutions are compared based on their ability to dominate each other, where “dominate” signifies superior performance metrics compared to another solution. Figure 2.2 provides an example to elucidate the concept of Pareto dominance further.

This example considers two decision variables: execution time and cost. A solution is considered better if both values are lower. Six solutions are presented, and we select specific points to illustrate Pareto dominance.

In the graph, Solution A is denoted as a point (5, 1000), and Solution B is shown as a point (8, 800). In terms of execution time, solution A is shorter than B, but solution B has a lower cost, so they can not dominate each other. Solution D is represented as a point (35, 1000). When compared with Solution A, they share the same cost value, but solution A has a shorter execution time. Consequently, Solution A outperforms Solution D. According to this file, solutions A B C D dominate other solutions, but can not dominate each other, so they form the Pareto front. This front represents a set of solutions not surpassed by any other solution in the specified objectives.

![Figure 2.2: An example of a Pareto front](image-url)

Figure 1.1 has introduced the process of DSE. It uses analyzable application models, platform architecture, and design constraints as the input maps the models to the platform, and schedules the communication and execution as the output. The concrete components which are used in the thesis
will be introduced in the following section.

## 2.2 Design Space Identification

DSI is a composable and tuneable method to explore the design space [7].

Figure 2.3 shows how DSI enables generic DSE in design flows. Figure 2.4 gives more details of DSI in DSE activity.

For the Design domain and Decision domain domain, they are the universe where all the design or decision models exist. The transformation between models involves DSI.

The process of DSI mainly consists of three steps [8]:

1. Identification stage: This is the stage used for identifying the model from the MDE description files [8]. This stage does not involve any exploration.

2. Bidding and exploration stage: During this step, the system would strategically choose the optimal combination of suitable solvers to get the best trade-off solutions.

3. Reverse identification stage: This stage generates MDE models based on DSE models of the previous stage and MDE models from the input.

The concepts and equations, which we reference in [9], will be further explained in the following sections.
Design models $\mathcal{M}$: model-driven-engineering (MDE) models, e.g. Simulink, ForSyDe IO

Explored $\mathcal{M}'$: enriched MDE models, e.g. ForSyDe IO (with schedules and mappings)

Identification procedure using $R_{ident}$

Decision models $\mathcal{X}$: parameters and associated functions

Solved decision models $\mathcal{Y}$: enriched parameters and associated functions

bidding procedure and exploration

Explorers $\mathcal{E}$, based on:
Problem-specific solution algorithms and/or heuristics, Constraint Programming, GA, ...

Figure 2.4: Enhanced overview of the DSI-based DSE activity and the information flow of IDeSyDe. The dashed arrow from $\mathcal{M}$ to $\mathcal{M}'$ represents direct DSE methods and tools for comparison with DSI-based DSE and IDeSyDe. (Jordao et.al, 2023)
2.2.1 Design models and decision models

During the process of DSE, models like those used to define applications, platform architecture, and other features are design models. In this thesis, the design model can be seen as a directed graph that depicts the components, relationships, and some other necessary information. Jordao et al. advocates different properties’ definition have their concrete names for clear semantics. For example, using Models of Computation [10] for applications, models of Architectures [11] for the platforms, and Object Constraints Language [12] for other features [7]. These models are typically used by MDE techniques and tools.

Models that can be used for exploration can be seen as decision models. It always contains parameters, and functions used for design space [7]. We can use SDF as an example. For the design model of SDF, depicts a directed graph, we can know the numbers of actors and channels, the execution time for their specific processor, and so on. But we don’t know the execution order of SDF, whether it happens deadlock or concurrent. The decision model adds these parameters or functions.

2.2.2 Identification stage

The identification ruleset \( r \) is key to driving this process. The production of the new decision model set \( \mathcal{X}' = r(\mathcal{M}, \mathcal{X}) \) needs the design model \( \mathcal{M} \) or a set of decision models \( \mathcal{X} = \{X_1, \ldots\} \) with the rules \( R = \{r_1, \ldots, r_n\} \).

\[
\mathcal{X}_{r,i+1} = r(\mathcal{M}, \mathcal{X}_i) \tag{2.1}
\]

The identification stage is a step-wise process that continues until \( \mathcal{X}_{r,i+1} = \mathcal{X}_{r,i} \), indicating that the identification stage has reached a fix-point and is complete. To ensure the accuracy of the identification stage, it needs to be mentioned that the identified elements in the next step must contain or be consistent with the elements identified in the previous step. This ensures a correct and reliable identification process.

2.2.3 Bidding and exploration stage

Once the identification stage is finished, explorers will be used for exploring the design space of identified decision models. Explorers encompass various methods, such as Constraint Programming (CP), and heuristic algorithms, which offer different options to choose from based on the properties of
the solvers. Consequently, different combinations may arise for the same identified decision models.

Two attributes are essential for this stage. The first attribute determines whether the solver is capable of exploration, while the second attribute relates to the results after exploration, which would return a set of explored decision models.

It is mentioned that there are various explorers, so they need to place bids. The successful explorer is the one who can dominate others.

If different explorers do not dominate each other, then the tool will choose one randomly. The successful explorer is then chosen based on the bidding process to proceed with the exploration which returns a set of explored decision models.

### 2.2.4 Reverse identification stage

The reverse identification stage aims to enrich the input design models. The key to its implementation is the reverse identification rule, denoted as $\rho$. This rule operates as a function, generating a set of new explored design models, denoted as $M'$. These new models are derived from the input design models and the explored decision models $\mathcal{Y}$, as described in [8]. This process is governed by a set of reverse identification rules denoted as $\mathcal{R}_{rev}$, and can be expressed using Equation 2.2:

$$M' = \bigcup_{\rho \in \mathcal{R}_{rev}} \rho(\mathcal{Y}, M)$$ (2.2)

### 2.3 Analyzable Application Models

This section introduces independent periodic tasks models and synchronous dataflow models which are used in this thesis.

#### 2.3.1 Independent Periodic tasks

The implementation of embedded systems is a complex and multi-faceted project that requires the collaboration of multiple individuals or teams, who are in charge of different elements. Finally, all these parts are integrated. In one perspective, these individual parts are referred to as tasks. However, it’s important to note that the integration of functionality into tasks is not
always a one-to-one mapping. It represents one approach to understanding the integration process.

Periodic tasks [13] play a crucial role in embedded systems, especially in real-time systems. Periodic tasks are executed periodically and associated with deadlines.

The reaction to these deadlines can be divided into three types:

1. **Hard deadline**: The task must meet its deadline, otherwise it can lead to severe consequences, such as safety issues or system failure. Examples include tasks in safety-critical systems like brake control in the driving system.

2. **Firm deadline**: Missing a firm deadline is meaningless for the system, but it may not cause catastrophic problems.

3. **Soft deadline**: The deadline of the task is more flexible and negotiable, where the task still has value even if it finishes after its deadline.

This dissertation specifically considers the context of hard real-time systems. Additionally, the thesis considers the independent periodic task, which implies there are no precedence relations or resource constraints between them.

A set of all tasks is denoted by \( T = \{ \tau_1, ..., \tau_n \} \), where \( n \) means the total number of tasks. Each periodic task \( \tau_i \) is defined as a tuple \( (\phi_i, T_i, C_i, D_i) \), where \( \phi_i \) represents phase of the task, \( T_i \) is the period of a task, \( C_i \) denotes the maximum execution time of the task, and \( D_i \) means Relative Deadline.

\( \phi_i \) means the time of task can be executed for the first time, namely the initial offset. \( T_i \) decides the frequency of the task that is executed.

Figure 2.5 visually illustrates the relationship between these parameters for tasks. If the phase is zero \( (\phi_i = 0) \) and the Relative Deadline equals the period of the task \( \tau_i \) \( (D_i = T_i) \), we simply write \( (T_i, C_i) \) as the tuple.

For every interval of time, the execution order of the periodic task set would repeat. This time interval is known as the Hyperperiod \( H \), which is the least common multiplier of the periods of all tasks. It is the time span for the repeated task set, which is important for scheduling.

\[
H = \text{lcm}(\tau_1, ..., \tau_n)
\]
The resource priority of tasks can be classified as either static priority or dynamic priority. In the case of static priority, the priority of the task is fixed, which means a higher priority task will always get more resources than another, in any situation. The allocation of resources is predictable and remains constant during execution. On the other hand, dynamic priority allows the priority of the task can vary based on different situations, which is more flexible.

In this project, we choose the fixed-priority scheduling algorithm: Rate-Monotonic (RM) algorithm [14] that assigns the priority of the task based on their task periods, as shown in Equation 2.4. According to this relationship, the task with longer periods has lower priority. Under the RM algorithm, higher-priority tasks are executed before lower-priority tasks, so preemption is used to suspend lower-priority tasks when a higher-priority task executes. One of the advantages of RM algorithm is its predictability, which makes it easy to know the execution order.

\[ T_1 > T_2 \rightarrow pri_1 < pri_2 \] (2.4)

Figure 2.6 illustrates an example following the RM algorithm used to schedule the independent task set \((5, 2), (2, 1)\).

In this example, the release time of both tasks is zero, so they are eligible for execution when the program starts. Since the period of \(T_2 = 5\) is shorter than \(T_1 = 2\). So \(T_2\) has higher priority that executes first. In this case, the hyperperiod is calculated as \(H = \text{lcm}(T_1, T_2) = 10\). This process repeats the execution every 10-time units.


Figure 2.6: Rate-monotonic schedule for the task set: (5, 2), (2, 1).

### 2.3.2 Synchronous Dataflow

A data flow graph is a directed graph that consists of a set of nodes and arcs. Each node in the graph means an actor, and each arc represents the channel that connects the actors. In order for the node to execute (fire), the input arcs of the node must have enough data tokens. The dataflow graph follows the First-In-First-Out principle, ensuring a consistent and orderly data flow.

Dataflow exhibits a trade-off between expressiveness and analysability. High expressiveness means the model can represent a wide range of situations and system behaviors. High analysability means the properties of the model can be proved or derived, making it easy to analyze.

SDF is a kind of high-analysis data flow that was proposed by Lee and Messerschmitt [15] [16].

SDF finds extensive application in multimedia processing, digital signal processing tasks, such as signal filtering, video processing, telecommunications, etc. due to its deterministic behavior and efficient throughput analysis.

An SDF graphs $G(A, C, E)$ contains a set of actors $A$, channels $C$, and edges $E$, where each edge connects an actor to a channel or vice versa. The static scheduling property of SDF ensures that the number of tokens consumed and produced by actors is fixed. This analysability enables designers to foresee and determine the behavior of the application in advance, making it especially suitable for real-time streaming media applications.

Figure 2.7 demonstrates an example of an SDF graph. The graph is mainly made of a set of circles and a set of channels. A circle represents an actor, and an arc represents a channel. The numbers on the arcs are the numbers of
tokens. The direction of the arrow indicates from the source to the target. The number of the source of arcs represents how many tokens would be produced. The number of the target of arcs represents the number of consumed tokens.

In Figure 2.7, actor $a$ requires one token from the input channel to fire and subsequently produces one token for channel $ab$ and three tokens for channel $ac$. Actor $b$ consumes one token and produces three tokens, while actor $c$ fires, only when there is at least one token in channel $bc$ and one token in channel $ac$ simultaneously. This guarantees to fulfill the requirements and then actor $c$ can fire and produce one token.

Ignoring the unconnected channels and their productions and consumption rates, the firing order for this example should be $ABCCC$, resulting in the repetition vector $q^T = [q_a, q_b, q_c] = [1, 1, 3]$.

![Figure 2.7: SDF graph example](image)

Homogeneous Synchronous Dataflow (HSDF) is a special case of SDF graph. For HSDF, the consumed tokens and produced tokens of each actor are the same, resulting in a token rate of each actor being one. It guarantees the dataflow is balanced and predictable. Due to its property, HSDF can simplify the analysis and optimization process when mapping application models to the shared platform without considering the complexity of different token rates. So choosing HSDF for the SDF model in the thesis enables a streamlined method to find combination methods and the trade-offs between metrics.

### 2.4 Design Constraints and Performance Metrics

Design constraints refer to clarified requirements or limitations that designers need to fulfill during the design process. These constraints design the boundaries of DSE. There are two compliance levels that can be considered: satisfy or optimize [17]. Satisfy means the solution must satisfy the constraints, which optimization is more flexible, which tries to find best-effort
solutions. In this context, the optimization can be viewed as the performance metrics. For the purposes of this thesis, our objectives are to minimize the used resources and maximize the throughput of SDF. Therefore, the throughput of SDF and resources used are considered as the performance metrics. The number of available used resources can be seen as the design constraint.

### 2.5 Platform Structure

In the context of embedded systems, two primary processor configurations are prevalent: single-processor systems and multiprocessor systems. A single-processor is often referred to as a uniprocessor. It only has one central processing unit (CPU). In contrast, multiprocessor systems can have multiple CPUs for processing the concurrent tasks.

In this thesis, we choose the tile-base multi-core platform. For this configuration, every tile has its dedicated resources, including memory for data storage, CPU for computation, and network interface to guarantee each processor operates independently and efficient communication among processors.

The selection of Time-Division Multiplex (TDM) buses for Multi-Processor Systems-on-Chips (MPSoCs) is determined based on worst-case communication time analysis.

TDM bus divides the bus into many time slots. Each slot is allocated to a specific processor to guarantee reliable data transmission. It is a static schedule, which means it is fully predictable.

Fig 2.8 shows the tiled-based multicore platform with time-division multiplexing communication. It can be seen from the figure that the dedicated slots cycle, once all slots execute completely. It is worth noting that the divided slots can be executed repeatedly in each cycle. Every processing node that sends or receives has its own one or more slots, namely communication.
Figure 2.8: Tiled-based multicore platform with time-division multiplexing arbitrated communication

### 2.6 Related work

The thesis focuses on the exploration of different models. Based on this theme, we investigate multiple tools proposed for DSE, with the common idea of finding solutions that fulfill the design constraints. However, these DSE tools differ in two aspects: different parameters they handle (such as application models, platform models, and performance metrics), as well as the exploration techniques they employ.

Forget et al. worked on scheduling the communicating periodic tasks [3], while Lee et al. proved the algorithm for scheduling SDF systems onto single or multiple processors [16]. Jeong et al. proposed a parallel scheduling method based on an evolutionary algorithm for multiple SDF graphs onto Heterogeneous processors. These studies, however, consider only one kind of application model.

FORMULA [18] utilizes an SMT solver instead of CP for solving design constraints and finding available design instances

Octopus [19] leverages existing tools for exploration and focuses on minimizing memory size and maximizing throughput when mapping tasks.

Khalilzad et al. proposed a framework that deals with both SDF and independent tasks to the heterogeneous multiprocessor platform. However, the
application models are partitioned without proper sharing [5]. Bamakhrama and Stefanov developed techniques for applying real-time scheduling theory for periodic tasks to Cyclo-Static Data Flow (CSDF) [20], but they only focus on the throughput.

In conclusion, this thesis solved content not covered by other literature by using the IDeSyDe tool, which follows the CP approach for space exploration. This thesis maps and schedules SDF applications and independent periodic tasks onto the same tile TDM-based platform. Simultaneously, it considers the throughput of SDF and usage resources as the performance metrics.

2.7 IDeSyDe

This section provides an overview of the IDeSyDe framework and presents essential code snippets relevant to this thesis, particularly focusing on the definition of models.

2.7.1 Structure of IDeSyDe

IDeSyDe is a multi-module structure, which can be categorized into two kinds of modules: Identification modules to identify the elements of design models or decision models and Exploration modules to explore the space. Figure 2.9 shows the dependency relationships between modules. The root module, named “core”, serves as the core part of the modules. It includes the fundamental and crucial traits, such as “DesignModel”, “DecisionModel”, “IdentificationModule”, “Explorer” and so on. They are the key components of this project, and the important traits will be introduced in the following.

![Figure 2.9: Structure of IDeSyDe before version 0.5.0.](image-url)
Module “common” builds upon the module “core” and defines the data structures for decision models. It enables the exchange of correct decision models between different modules and handles identification processes.

Module “blueprints” provides an additional set of routines, data structures, and interfaces built on top of module “core” and “common”, which can be used externally in a multi-language exploration process. Module “choco”, “minizinc” and “forsyde io” are at the same level and all of them depend on “core”, “common”, and “blueprint”. The “choco” module uses the Choco-solver for exploration through constraint programming using the Open-Source Java library. Module “forsyde io” imports the Java library forsyde-io-java.

Module “minizinc” uses the MiniZinc solver to explore the design space. Module “cli” depends on all the modules mentioned before. It uses CLI Solver, which is used for exploration. For module “test”, aims to explore whether there are available and suitable solutions, so it depends on all the other modules. The parameters for testing will be set here.

2.7.2 Design model

The trait for a design model in the design space identification methodology, as defined in [7]. “DesignModel” is a system model, which contains original data types. We can see “DesignModel” as the “database” which contains all the information we need. Different “DesignModel”’s define different rules for concrete functions. Listing 2.1 is “DesignModel”, which is founded in the “core” module. For the concrete “DesignModel”, "identifiers" are necessary which guarantee two elements of type ElementT are always compared for equality and uniqueness.

In the context of the thesis, four different types of elements are crucial for the design model: SDF graph, periodic task, order elements, and platform.

Listing 2.1: DesignModel

```
trait DesignModel {
    type ElementT
    type ElementRelationT

    def merge(other: DesignModel): Option[DesignModel]

    def elements: Set[ElementT]
}
```
def elementID (elem : ElementT): String

def elementIDs: Set[String] = elements.map(  
elementID  )

def +( other : DesignModel) = merge(other)

def category: String

def header: DesignModelHeader = DesignModelHeader(  
uniqueIdentifier,  
Set(),  
Set(),  
elementIDs  )  
}

2.7.3 Decision model and Standard Decision Model

[7] also defines the trait for a decision model in the design space identification methodology. Listing 2.2 shows the code which is also defined in the “core” module. The information of “DecisionModel” is used for design space exploration. “DecisionModel” is an instance of the design model, which adds parameters, decision variables, or analysis techniques to implement it.

Listing 2.2: DecisionModel

```scala
trait DecisionModel {

    type ElementT

    def uniqueId: String

    def coveredElements: Set[ElementT]

    def dominates(other: DecisionModel): Boolean = header.domines(other.header)

```
def elementID(elem: ElementT): String

def coveredElementIDs: Set[String] = coveredElements.map(elementID)

def header: DecisionModelHeader = DecisionModelHeader(uniqeIdentifier, None, coveredElementIDs)

Listing 2.3 “StandardDecisionModel” is the decision model extension, defined in the “common” module. Listing 2.3 is “StandardDecisionModel”. It is a simple model and the type only involves Strings. Compared with other “DecisionModel” descendants, the pros of “StandardDecisionModel” is that it gives more possibilities from an implementation perspective. By that, we mean that sharing if the identification procedure is implemented in a multi-tool manner, using String as both [ElementT] and the ID makes consistency easy. Consequently, it also advances higher decoupling between “DecisionModel”s and “DesignModel”s. So “StandardDecisionModel” is preferred to use widely, it maximizes reusable design space.

Listing 2.3: StandardDecisionModel.scala

trait StandardDecisionModel extends DecisionModel
{
    type ElementT = String

    def elementID(elem: String): String = elem
}
2.7.4 Critical parameters of IDeSyDe

The identification stage is significant for DSE. The identification is an iterative stage to identify more elements for decision models. These two critical parameters need to be defined to satisfy this purpose: \textit{coveredElements}, \textit{coveredElementRelations}.

\textit{coveredElements}: It describes the elements that need to be used.

\textit{coveredElementRelations}: It depicts the relation between the elements.

A set of \textit{coverElements} and \textit{coveredElementRelations} can check whether the elements are increasing to guarantee the identified process properly: It terminates and converges to a single, definitive value, which is the correct value \cite{21}. Take Listing 2.4 a “SDFApplicationWithFunctions” as an example. The SDF-directed graph is mainly composed of actors and channels, so \textit{coveredElements} is made of a set of actors and a set of channels. The relationship in the graph from source to target is described by \textit{coveredElementRelations}.

\begin{verbatim}
final case class SDFApplicationWithFunctions (  
  val actorsIdentifiers : Vector[String],  
  val channelsIdentifiers : Vector[String],  
  val topologySrcs : Vector[String],  
  val topologyDsts : Vector[String],  
  ...  
) extends StandardDecisionModel  
  with ParametricRateDataflowWorkloadMixin  
  with InstrumentedWorkloadMixin {  
  val coveredElements = (actorsIdentifiers ++  
    channelsIdentifiers).toSet  
  val coveredElementRelations = topologySrcs.zip(  
    topologyDsts).toSet
\end{verbatim}

2.7.5 Periodic tasks in IDeSyDe

The file “CommunicatingAndTriggeredReactiveWorkload” is a decision model for communicating periodically activated processes. In this thesis, the types of tasks are independent periodic tasks, with a fixed priority. Four
parameters are defined: *periods*, *offsets*, *relativeDeadlines*, *affineControlGraph*. These variables are about the concept of periodic tasks, which were introduced before. The code of periodic tasks is available on GitHub. This code calculates the hyperperiod, and priority and considers the task’s offset.

### 2.7.6 SDF in IDeSyDe

The file “SDFApplicationWithFunctions” is the decision model for synchronous dataflow graphs. Subsection 2.3.2 has introduced the significant concepts of SDF.

The parameters, such as actors, channels, consumed and produced numbers of tokens, etc. need to be defined to calculate the execution of the SDF graph. These parameters are defined in the file “SDFApplicationWithFunctions” as follows:

**actorsIdentifiers**: It is a set of actors. Distinct identifiers of actors to avoid confusion among them.

**channelsIdentifiers**: Each channel has its own identifier to guarantee the correct computation.

**topologySrcs**: The sources for every edge triple in the SDF graph.

**topologyDsts**: The destination for every edge triple in the SDF graph.

**topologyEdgeValue**: The tokens produced or consumed for each edge triple in the SDF graph.

**actorSizes**: The size for each actor’s instruction(s) in bits.

**channelNumInitialTokens**: The number of initial tokens of the channel.

**channelTokenSizes**: The size of the channel that contains tokens.

**minimumActorThroughputs**: The fixed throughput expected to be done for each actor, given in executions per second.

The SDF graph is encoded \((A \cup C, E)\). In this file, \(A\) is “actorsIdentifiers”, \(C\) is channelsIdentifiers, and \(E\) represents the edges, \(e \in E\). The edges are encoded into these three parameters: “topologySrcs”, “topologyDsts” and

“topologyEdgeValue”. \( e = (b, a, 3) \) indicates edge \( e \) transfer three tokens from channel \( b \) (producer) to channel \( a \) (consumer).

The SDF decision model mainly involves various elements, including the original SDF graph, the repetition vector, actor computation, checking for consistency to avoid the deadlock, concurrent analysis, and determining the firing order to ensure correct execution.

The process is given by tuples, the form as (src actors index, dst actors index, lumped SDF channels, size of the message, produced, consumed, initial tokens).

The specific code can be found on GitHub: 

### 2.7.7 Platform in IDeSyDe

It mainly contains two files, “PartitionedCoresWithRuntimes” and “Shared-MemoryMultiCore”. “PartitionedCoresWithRuntimes” is a decision model for configuring the partitioned cores with runtimes, which involves defining the available processors and schedulers, specifying scheduling policies, and determining whether processors execute in a bare-metal environment.

“SharedMemoryMultiCore” is designed for a multi-core hardware system, which includes components such as “processingElems” for cores, “storageElems” for memory, and “communicationElems” for network interfaces. The platform file is “PartitionedSharedMemoryMultiCore” is to synthesize these two parts.

The source code for this is available on 

### 2.7.8 IdentificationModule in IDeSyDe

Listing 2.5 shows the trait “IdentificationModule”. It defines two functions \( \text{identificationRules} \) for the identification stage and \( \text{integrationRules} \) for the reverse identification stage.

\( \text{identificationRules} \) abstract the design model to the decision model with a set of parameters that can be used for exploration. As introduced before, \( \text{identificationRules} \) produces a new decision model based on a set of design models and a set of decision models. \( \text{integrationRules} \) can be seen as the inverse of \( \text{identificationRules} \), it is the reverse identification rule, which produces a design model.

\[ \text{identificationRules} \]

\[ \text{integrationRules} \]

---


†https://github.com/forsyde/IDeSyDe/blob/master/scala-common/src/main/scala/idesyde/identification/common/models/platform/PartitionedSharedMemoryMultiCore.scala
2.8 Summary

This section provides an overview of the central issue addressed in the thesis and introduces the essential components required for space exploration. It also offers a brief of the existing tools in the field of Design Space Exploration. Additionally, it delves into the crucial code snippets within IDeSyDe that will play a pivotal role in the execution of this thesis, offering fresh insights and perspectives for the project’s advancement.
Chapter 3

Methods

In Section 3.1, the thesis elaborates on the research methodology and process employed throughout the thesis. Section 3.2 introduces the application of IDeSyDe and outlines the criteria for conducting DSE using this tool.

3.1 Methodology and research process

The methodology is mainly carried out in two parts: theory study and experiments.

In the theoretical part, we delve into various modeling concepts and current DSE tools. For the experiments, we use IDeSyDe [1] to implement the codes and ForSyDe IO [2] used for the test.

The research process can be divided into five phases.

- Literature review: Prior to embarking on code implementation, a comprehensive understanding of relevant concepts and related work is key.

- Familiarization with IDeSyDe and ForSyDe IO: Proficiency in using IDeSyDe and ForSyDe IO is fundamental for code implementation and testing. Allocate time for acquiring the necessary skills to effectively utilize these tools.

- Solution Concept Determination: It involves conceptualizing a solution and integrating solution ideas into IDeSyDe.

- Code implementation: The implementation phase comprises several key tasks, such as designing the model for combining SDF and independent periodic tasks, choosing the model for the platform and constraints,
mapping the model onto the platform architecture, and establishing a scheduling strategy.

• Testing and Analysis: After the functional codes are completed, rigorous testing and manual analysis of the results are performed to ensure the solution’s effectiveness.

3.2 Test environment and Software

To run the program successfully, the machine requires installing a Java Runtime Environment for Java 11. It can work on Linux, Windows, the Oracle distribution, corretto, and graalvm [22]. The software of the project is Visual Studio which is a basic complete set of development tools.

3.2.1 Software Tools

The software analysis tools use the IDeSyDe, which is essentially DSI. This thesis focuses on extending the tool. There are six key points about developing the tool. For getting more details, you can visit *.

• If there does not exist the appropriate tool can connect with IDeSyDe, a new “DesignModel” needs to be created as the basic trait.

• If the equations, parameters, constraints, etc. do not exist in any module, it needs to create at least one “DecisionModel” as outlined here.

• New explorers may need to be created to explore some decision models, i.e. the design space exploration.

• To identify the new decision model and connect it to the existing available model, new identification rules need to be created.

• For every new “DesignModel”, new identification rules and integration rules are required to be created.

• Once a new identification or integration rule is created, the register is necessary. There are two options to register: One is to register it into an existing “IdentificationModule” file in the IDeSyDe modules, and the other is to create a new file used for registration.

*https://forsyde.github.io/IDeSyDe/extensions
3.2.2 Evaluation framework

For evaluating the framework, we would load different combination models from ForSyDe IO to test in the “test” module. ForSyDe IO, which is mainly used for the library. The main user group is tool developers. ForSyDe IO contains xmi and fiodl files which mainly describe the system graph. We would load the different models with various conditions.

Figure 3.1 shows the condition tested by the IDeSyDe tool. The input type of IDeSyDe should be the ForSyDe IO description file. If the input file does not belong to the type of ForSyDe IO, it requires ConverSyDe to convert it to the fiodl file and verifies it again. ConverSyDe’s usage method is to download the jar on the website * and run it.

Once the input file fits the requirement, IDeSyDe starts to work as we defined. About the test of IDeSyDe, it requires downloading the jar file from the website † to install IDeSyDe. How to perform DSE by IDeSyDe is introduced on ‡.

After performing DSE, it produces many JSON files. There is a subfolder of IDeSyDe named “run”. It contains four folders in total. “inputs”, “identified”, “explored” and “reversed”. For the evaluation, the focus is on checking the files in the "explored" folder, which provides valuable insights into the allocation methods offered by IDeSyDe.

![Figure 3.1: IDeSyDe Tool Test Conditions](https://github.com/forsyde/forsyde-io/releases)

---

* [https://github.com/forsyde/forsyde-io/releases](https://github.com/forsyde/forsyde-io/releases)
† [https://github.com/forsyde/IDeSyDe/releases](https://github.com/forsyde/IDeSyDe/releases)
‡ [https://forsyde.github.io/IDeSyDe/usage/quickstart.html](https://forsyde.github.io/IDeSyDe/usage/quickstart.html)
Methods
Chapter 4

Implementation

This chapter introduces the solution idea and concrete method implemented by IDeSyDe.

4.1 Solution Idea

This thesis follows the DSI methodology, which is introduced in Section 2.2. Implementing the proof-of-concept for this thesis involves two key components:

Selecting the appropriate methodology that combines the SDF and periodic task model and mapping and scheduling the combined model.

4.1.1 Combination Method

Regarding the insight into the combination method:

We plan the primary and secondary priorities of models. Periodic task models have higher priority than SDF in any case. For the concrete execution of the periodic task, it uses RM algorithm, in which the shorter deadline has higher priority. So in this combination method, the periodic task can execute as normal by RM algorithm without being affected by SDF. SDF would utilize the remaining CPU.

Specifically, the idea of the combination code is based on the Periodic Task Model and SDF Model. We unify them by introducing variable actor durations, allowing tasks to dictate the timing parameters of the SDF model. Therefore, for the combining solution, we do not need to consider all the equations being solved for tasks or SDFs. What is significant for the project
is taking the remaining utilization of any CPU, and three parameters drive the combination approach: serverPeriod, serverBudget, and actorDurations.

The equations are listed in the following:

$$\text{newActorDuration} = \frac{\text{actorDuration}}{1 - \text{utilizationOfCpu}}$$ \hfill (4.1)

$$\text{serverPeriod} = \text{hyperPeriod}$$ \hfill (4.2)

$$\text{serverBudget} = \text{hyperPeriod} \times (1 - \text{utilizationOfCpu})$$ \hfill (4.3)

Since tasks always take precedence over SDF, serverPeriod is set equal to the hyperPeriod to maintain the real-time analysis for the tasks.

utilizationOfCpu refers to the utilization of the processor for periodic tasks. serverBudget is the allocated time that the SDF component can utilize.

newActorDuration represents the visual execution time for actors. We adjust the newActorDuration based on the remaining CPU time. Therefore, during execution, the actorDurations may appear to increase "visible" because processor utilization of SDF must remain less than or equal to 100%.

### 4.1.2 Real-Time Operating System

When it comes to ensuring the mapping and correctness of execution, Real-time operating system (RTOS) plays a pivotal role. The RTOS manages the tasks or actors in five distinct states during their execution:

**New**: This state indicates a task or actor is created but not yet started.

**Ready**: In this phase, the task or actor is prepared to execute, but awaiting their turn in the scheduling queue.

**Running**: When the task is in a running status, it signifies the task is executing on the processor.

**Waiting**: When the task or actor lacks of feasible resources to execute, it would be in a waiting state. They remain in this state until the required resources become available.

**Completed**: The task transitions to this state when it has finished its execution.
The relation of these five states is shown in Figure 4.1. The life-cycle of a task starts with its creation, at which point it enters the “Ready” state. This thesis adopts RM algorithm, where tasks with short deadlines receive higher priority and are scheduled for execution. However, if a task with higher priority becomes “Ready”, it can preempt the currently running task. If the running task encounters a resource shortage, it will be blocked, and “Waiting” until the necessary resources become available. Once a task completes its execution, it is removed from the task context.

### 4.1.3 Application Example

Figure 4.2 offers a visual case study that demonstrates the application of the combination method.

In this case, we assume the presence of one HSDF and three periodic tasks set \((5, 1), (10, 2), (10, 6)\) that are mapped to two processors, denoted as \(P_0\) and \(P_1\). The SDF graph contains four actors, each with a Worst-Case Execution Time (WCET) of two, and a WCCT of one. According to the property of SDF, when actor \(A\) fires, it produces tokens for \(B\) and \(C\) simultaneously. \(D\) fires both channels \(cd\) and \(bd\) have enough tokens concurrently.

**Task Allocation:** In this example, tasks \(\tau_1\) and \(\tau_2\) assigned to run on \(P_0\), while \(\tau_3\) run on \(P_1\) concurrently. \(\tau_1\) would take precedence, and then \(\tau_2\) is executed, based on RM scheduling.

**Actor Allocation:** According to execution order, we allocate actor \(A\), \(C\), \(D\) to \(P_0\), \(B\) to \(P_1\). Actor \(D\) on \(P_1\) requires tokens from both channel \(ab\) and channel \(ac\), necessitating communication from actor \(B\) on \(P_0\). Upon completion of actor \(D\)’s execution, SDF initiates a new round of execution.

Figure 4.3 shows the application of the RTOS for this case.

Because the overall principle remains unchanged, we would briefly explain this figure. At first, \(\tau_1\), \(\tau_2\), \(\tau_3\), and SDF actors \(A\), \(B\), \(C\), and \(D\) are initially in the “Ready” state.

Due to \(\tau_1\) and \(\tau_3\) having the highest priority for their processor, once
the execution begins, they are in a “Running” state. τ₂ runs on P₀ after τ₁ “Completed”. After τ₂ completes its task, and no pending tasks, actor A starts to run. When A completes its execution and in the “Ready” state. “Completed” to “Ready” can be a instant point. The red point represents the “Completed”.

It’s worth mentioning that when an actor or task is running, a higher-priority task can interrupt the running task. In this case, when task D is running, a higher-priority task like τ₁ preempts the processor upon its release. In such a scenario, D returns to the “Ready” state until the scheduler reschedules its execution.

![Diagram showing simulation effect](image-url)

**Figure 4.2: Simulation effect**
4.2 Implementation Overview with IDeSyDe

This section provides a high-level summary of the implementation process using the IDeSyDe tool, which is illustrated in Figure 4.4. We choose the periodic task models and SDF models as case studies and TDM multi-core platforms, and every core has a runtime with fixed priority scheduling [8]. The programming language is Scala. The nodes represent design models, decision models, and explorers. The marked $r_i$ arcs mean the identification rules. $p_i$ arcs represent the reverse identification rules. In accordance with the principles of DSI, the process can be segmented into three steps: identification stage, exploration stage, and reverse identification stage.
4.2.1 Identification Stage in the IDeSyDe Implementation Process

The identification stage, which is a stepwise procedure, encompasses the transition from the system graph to Periodic Task and SDF to Platform. This process involves iteratively identifying elements until no further elements can be identified. The identification is primarily driven by seven identification rules, denoted as $r_1$ to $r_7$, which are crucial for its implementation.

The System Graph, namely the design model used in this context is a directed graph supporting loops and multiple edges between vertices, which is always implemented by Simulink or ForSyDe IO.

Specifically, rules $r_1$, $r_2$, $r_3$, and $r_4$ are similar, all of them are designed to partially identify different parts within the system graph.

To provide more details about the individual rules:

- $r_1$ identifies SDF from SDF Graphs.
- $r_2$ identifies Periodic Task from periodic task Graphs.
- $r_3$ identifies Partitioned Runtimes from the runtime component in the graph. It ensures the runtime component of Partitioned Runtimes have a one-to-one mapping with all core components of the system graph [8].
- $r_4$ identifies Hardware Topology from the platform component. The purpose of $r_4$ is to guarantee that every core in Hardware Topology has correct data and instruction mappings by accessing at least one memory component [8].

Rule $r_5$ identifies Periodic Task and SDF based on the identified decision models, which integrates the Periodic Task and SDF if they are compatible.

Rule $r_6$ identifies Partitioned Platform based on the identified decision
models, which integrates the *Partitioned Runtimes* and *HW topology* if they are compatible [8].

Rule $r_7$ partially identifies *Periodic Task and SDF* to *Partitioned Platform*. To guarantee the space exploration, $r_7$ would inspect whether the *Periodic Task and SDF* models are consistent before getting the final decision model.

### 4.2.2 Exploration and reverse identification stage in the IDeSyDe Implementation Process

After the identification step, the identification stage finishes and they start exploring, namely *Periodic Task and SDF to Platform DSE*. Finally, the reverse identification rule $\rho_1$ would expand the input *System Graph* with explored *Periodic Task and SDF to Platform DSE*.

For the exploration stage, we follow the rule of Pareto-front to find the best trade-offs for the exploration stage. To implement this method, our approach involves adopting a two-step branching strategy [9]:

In the first step, all possible numbers of cores can be considered for exploration. It does not involve branching. Once an available solution is examined, the branching strategy comes to the second step. The algorithm selects the minimum value of $n_{\text{cores}}$ and creates branches to explore the solution space further. It aims to start constructing the Pareto-optimal solutions from the minimum number of cores. This two-step branching strategy guarantees the compacting mappings [23] to decrease exploration time.

We also refer to [24] to explore the solutions. In essence, the exploration is an iterative process. When the new solution is found, it will compete with the initial non-dominated solutions. If the new solution can dominate the original solution, (based on the objectives used in this thesis, which are used resources and throughput of SDF), it would substitute the original solution and become the new non-dominated solution. This process continues until no new dominant solutions are found, resulting in a set of solutions fitting the Pareto-optimal solutions.

### 4.3 Software design

This section introduces pivotal parameters that need to be used and then introduces code implementation. This thesis focuses on the identification stage concentrated in the “common” module.
Regarding the decision model, we utilize the extension of the decision model called “StandardDecisionModel” from the “common” module. All the decision models which include SDF, periodic task, and platform architecture extend the trait “StandardDecisionModel”.

The final test procedure can be implemented using the “test” module or through command-line interfaces in both Windows and Linux environments, which allows us to conduct comprehensive testing and evaluation of the developed models and algorithms.

### 4.3.1 Solution parameters

Some critical parameters used for this thesis are introduced in the following.

We can categorize them as the models:

For the platform model:

- $P$ is processing node. This thesis uses tile-based multiprocessor, $n_{\text{tiles}} \in \{1, \ldots, |P|\}$.
- $\text{wcet}$ WCET is related to the processors which indicates the longest execution time during execution.
- $\text{wcct}$ WCCT means the longest communication time for actors of SDF between the processing nodes. The local communication does spend time, namely the tokens exchange in the same processing node $\text{wcct} = 0$.

For the SDF graph $G(A, C)$, we use the solution and notations of [9], which we give a brief overview of next.

- $A = \{a_1, \ldots, a_{|A|}\}$, where $a_i$ is an actor, $A$ represents the set of actors.
- $C = \{c_1, \ldots, c_{|C|}\}$, where $c_i$ is a channel and $C$ are the set of channels.
- $\text{tok}_{i\text{in}}$ represents the initial tokens for the channel. In SDF graph, it is typically represented by a dot.
- $f_i \in \mathcal{J}$, where $f_i$ is the firing actor, $\mathcal{J}$ are the set of all firings actors.
- $\text{Th}_G$ For the SDF graph, the throughput of SDF means the rate at which data is produced over a given period of time.

For independent periodic task set $S = \{\tau_1, \ldots, \tau_i\}$:

- $i$ is the total number of tasks. $\tau_i$ means the $i^{th}$ task.
- $H_P$ represents the hyperperiod.
- $u \in [0, 1]$, $u$ represents the utilization of periodic tasks. During execution, the ratio of periodic tasks is used on the processor.

For mapping and scheduling model:

- $m^a_i$ Because we use TDM bus, there are multiple slots and mapping of the actor to the tiles is $m^a_i \in P$.
- $m^\xi_i$ Mapping of channel to the tiles $m^\xi_i \in P$. 
Mapping of periodic task to the tiles $m^p_i \in P$.

$n_{\text{cores}} \in P$ represents the number of used resources, where $P$ is the set of all possible resources. The value of $n_{\text{cores}}$ is determined by counting the distinct values in the following set:

$$n_{\text{cores}} = \text{numberofValue}(m^q_1, \ldots, m^{[A]}_a, m^p_1, \ldots, m^{|P|}_p)$$

(4.4)

### 4.3.2 Periodic Task Model and SDF

In the context of our combination method, we make an assumption that task precedence takes precedence over other considerations. In this approach, SDF dynamically utilizes available resources, and the execution start and end times are flexible. To illustrate this execution method for SDF in our thesis, we liken it to a sporadic server. A sporadic task means the task has unpredictable arrival times. The sporadic server is a scheduling mechanism that handles and allocates dynamically sporadic tasks and efficiently utilizes system resources. So the combination model is named “TaskdAndSDFServer”. It’s important to note that we employ this server analogy solely for the purpose of describing the execution method of SDF in our thesis; we do not involve an actual server in our implementation.

Listing 4.1 substantiates an example of the execution process of “TaskdAndSDFServer”.

The code is based on the “SDFApplicationWithFunctions” for the SDF task and “CommunicatingExtendedDependenciesPeriodicWorkload” for periodic tasks. For the implementation, SDF does not have a fixed execution time like a periodic task, so it is regarded as a server that two extra parameters “sdfServerperiod” and “sdfServerBudget” are added to know the execution status better.

Listing 4.1: TaskdAndSDFServer

```scala
final case class TaskdAndSDFServer(
  sdf: SDFApplicationWithFunctions,
  task: CommunicatingExtendedDependenciesPeriodicWorkload,
  sdfServerperiod: Vector[Rational],
  sdfServerBudget: Vector[Rational],
) extends StandardDecisionModel {
```

Let $th_i$ be the throughput of actor $a_i$ in a tile, without any periodic tasks in this tile. If the mapped tasks to this tile take $u_{P_i}$ utilization, then the new server-based throughput $th_i^s$ is upper bounded by Equation 4.4,

$$th_i^s = \frac{1}{th_i} = \frac{1}{\frac{th_i}{1-u_{P_i}}}$$

(4.5)

The utilization $u_{P_i}$ refers to the utilization rate of periodic tasks on the processor $P_i$. $C_{ij}$ represents the computation time of the j-th periodic task for processor $P_i$. $T_{ij}$ represents the period of the j-th periodic task for processor $P_i$. The utilization equation is shown by Equation 4.5.

$$u_{P_i} = \sum_{j=1}^{n} \frac{C_{ij}}{T_{ij}}$$

(4.6)

### 4.3.3 The platform for the SDF and Periodic Task Model

If the decision model of Periodic Task and SDF and Partitioned Platform are produced successfully, it would continue to combine Periodic Task and SDF and Partitioned Platform. This procedure involves mapping and scheduling the combination model to the platform, which is called “PeriodicWorkloadAndSDFServersToMultiCore”. The code in the ”common” module can be accessed by *

### 4.3.4 Identification rules

The identification stage plays a critical part in DSI. Defining identification rules is essential to get the decision model for exploration.

*https://github.com/forsyde/IDeSyDe/blob/develop/scala-common/src/main/scala/idesyde/common/PeriodicWorkloadAndSDFServersToMultiCore.scala
As introduced in 4.1, seven rules are required in this thesis. Specifically, different components of identification rules are defined and implemented in different files.

To partially identify periodic tasks from the system graph, \( r_1 \) is defined in “WorkloadRules”. For SDF identification, \( r_2 \) is defined in “ApplicationRules”.

The identification rules for platform \( r_3, r_4, r_6 \) are defined in “Platform-Rules”, the identification rule of task and SDF \( r_5 \), and the identification rule for the task and SDF to the platform \( r_7 \) are defined in the “MixedRules” trait. The implementation of these rules is carried out “CommonIdentificationModule” file. All these files are implemented in the “common” Module, which can be accessed at the following GitHub repository: *

\[ u = \frac{1}{3} + \frac{1}{5} + \frac{4}{10} + \frac{5}{30} = \frac{11}{10} \]

### 4.3.5 Test model

After implementation, we need to verify whether we combine the independent task model and SDF, and find the best trade-off solutions between metrics.

Based on the design goals of the independent periodic task, SDF, and tile-based multi-core platforms aim to maximize the throughput of SDF, and minimize the used resources. I selected two HSDFGs along with their corresponding WCET and WCCT, as depicted in Figure 4.5. Independent periodic tasks set are \( \{(3, 1), (5, 1), (10, 4), (30, 5)\} \), and two processing nodes. We need to use the ForSyDe IO types of files for the test. The link to access the test files is †. There are a total of 4 files used together for this case study: “periodicTask.fiodl” which corresponds to the periodic task set; “sdf1.fiodl” the description file for SDF1; “sdf2.fiodl” the description file for SDF2; and “bus_small_platform.fiodl” which describes the two-core platform for these components.

Here are the following considerations, when deciding the parameters: Two SDF graphs to demonstrate that the system can handle multiple SDF models. The different WCETs for processors are meant to test whether the system correctly identifies efficient mapping strategies. WCCT is to add the complexity for exploring the solution.

For this task set, the processor utilization is:

\[ u = \frac{1}{3} + \frac{1}{5} + \frac{4}{10} + \frac{5}{30} = \frac{11}{10} \]

*https://github.com/forsyde/IDeSyDe/tree/develop/scala-common/src/main/scala/idesyde/common
†https://github.com/forsyde/IDeSyDe/tree/develop/examples_and_benchmarks/small_and_explainable/yuhan_zhang_thesis
We ensured that the utilization of the periodic task set is greater than 1, which indicates a single processor cannot execute these task sets simultaneously. This validates whether IDeSyDe gives the correct mapping methods, i.e., all the solutions must choose more than one processor for this case. We chose to use two processing nodes to observe how the system explores different mapping types with consistent parameters, i.e., when the number of used resources and throughput is the same, to check if it can still find different mappings while obtaining the same optimal result.

To provide a comprehensive comparison and understand the thesis better, the solution generated by hand is also provided. Both solutions will be presented and analyzed in detail in Section 5.

Additionally, we have another test with fewer constraints based on the last case. However, since the last case is more representative and comprehensive, this thesis will only introduce and focus on the last test case.

By conducting this test with different models and platform settings, we can identify optional solutions that fit the requirements. This empirical evaluation is vital for validating the DSE solutions for co-mapping of periodic and streaming applications in a shared platform.

![Example SDF Graphs with WCET and WCCT table](image-url)
Chapter 5

Results and Analysis

In this chapter, we present and analyze two types of solutions provided for the given case - one created manually and the other generated by IDeSyDe.

5.1 Key Assumptions and Constraints

There are some specific assumptions that need to be highlighted:

This thesis does not support the concept of "migration," meaning that both periodic tasks and actors can only execute on their specific processors during the execution.

Regarding communication, all local communication assumes zero latency ($wcct = 0$). It means the communication between tasks or the actors residing on the same processor has no overhead. However, for communication between processing nodes, due to bus transfer, the WCCT is set to 1 ($wcct = 1$) in this case study.

5.2 Major results

Figure 5.1 and Figure 5.2 show two solutions by hand. Some SDF graphs have a very long time until they settle in a steady state. Since we are in this special case that we run only between task windows, it can make it even worse and have a very long time before arriving at a steady state. It makes no sense. So in this manual solution, there is a thirty-time unit interval, which represents the hyperperiod used for selecting tasks. Figure 5.4 shows the best solution from IDeSyDe.
## 5.3 Analysis

We will showcase and analyze handwriting solutions and software solutions.

### 5.3.1 Solution by hand

Figure 5.1 is the first manual solution that assumes $\tau_1$, $\tau_2$, actors $A \, B \, C \, E$ execute on the $P_0$; $\tau_3$, $\tau_4$ and actor $D$ fire on the $P_1$.

![Figure 5.1: The first solution by hand](image)

This is the first solution that comes to mind. The reason why I chose this execution method is that maximizes processor utilization at first. But when we continued to execute, the operation was different from what we expected.

WCCT and WCET are shown by the table of Figure 4.5. $wcct = 1$, it means different processing nodes need one-time units to communicate. For this solution, only the actor exchanges from $D$ to $E$ need to communicate. The execution sequence of the period task and SDF are fixed. Independent periodic tasks use RM algorithm. Based on our idea, tasks execute according to their own rules without being affected by SDF. In this case study, $H_{p_0} = 15$, $H_{p_1} = 30$. For $P_0$, SDF follows the execution order: $A \, B \, B \, C \, C \, E$. If we only consider the operation of SDF, it needs six time units for one iteration. In this case, for the first cycle, SDF1 is finished after 12 cycles. For this situation, the execution of $E$ needs the tokens from actor $D$. From the table, we can know it needs a one-time unit to communicate. So for the actors execute on $P_0$, 14 cycles can finish for the first iteration.

For SDF run on $P_0$, the utilization is

$$1 - u_{P_0} = 1 - (\frac{1}{3} + \frac{1}{5}) = \frac{7}{15}$$

For actor $A$ on $P_0$,

$$newADuration = \frac{1}{\frac{7}{15}} = \frac{15}{7}$$
For actor B on P₀,
\[ \text{newBDuration} = \frac{2}{15} = \frac{30}{7} \]

For actor C on P₀,
\[ \text{newCDuration} = \frac{2}{15} = \frac{30}{7} \]

For actor E on P₀,
\[ \text{newEDuration} = \frac{1}{15} = \frac{15}{7} \]

The remaining utilization for SDF on P₁ used:
\[ 1 - u_{p₁} = 1 - \left( \frac{4}{10} + \frac{5}{30} \right) = \frac{17}{30} = \frac{13}{30} \]

For actor D on P₁,
\[ \text{newDDuration} = \frac{1}{13} = \frac{30}{13} \]

For P₀, the longest duration path is
\[ \text{newADuration} + \text{newBDuration} + \text{newCDuration} + \text{newEDuration} \]
\[ = \frac{15}{7} + \frac{30}{7} + \frac{30}{7} + \frac{15}{7} = \frac{90}{7} \]

For P₁, the longest duration path is
\[ \text{newCDuration} = \frac{30}{13} \]

In this case, P₀ and P₁ bottleneck each other. Therefore, the inverse throughput of both is the bottleneck, which is \( \frac{90}{7} \).

So the throughputs of SDF₁ and SDF₂ are 0.0777.

From the figure, we can see there is some remaining space of P₁ wasted. The utilization is not good. Analyzing the figure, it is mainly because the WCCT is considered.

Based on the first solution, the second manual solution was proposed: \( \tau₁, \tau₂ \), actors A B C are allocated to P₀; \( \tau₃, \tau₄ \) and actors D E are allocated on
the \( P_1 \). Figure 5.2 illustrates the execution of a new solution.

The same allocation method for periodic tasks is maintained, but we alter how we assign actors in the SDF model. In this new allocation scheme, actors belonging to the same SDF are all assigned to the same processor, without involving considerations related to the WCCT.

So the utilization of SDF is the same as the solution 0:

\[
1 - u_{p0} = \frac{7}{15}
\]

For actors \( A, B, \) and \( C \), they also have the same newactorDuration values as Solution 0: \( \frac{15}{7}, \frac{30}{7}, \frac{30}{7} \).

Actors \( A, B, C \) consist SDF 1, all of them execute on \( P_0 \), according to the equation:

\[
th_{p0}^{-1} = th_{p0}^{-1} = th_{p0}^{-1} = newADuration + newBDuration + newCDuration = \frac{15}{7} + \frac{30}{7} + \frac{30}{7} = \frac{75}{7}
\]

\[
th_{sd1} = \frac{1}{th_{sd1}^{-1}} = \frac{1}{th_{p0}^{-1} + th_{p0}^{-1} + th_{p0}^{-1}} = \frac{225}{7} = \frac{7}{225} \approx 0.03111
\]

So the throughput of SDF 1 is 0.03111.

For actor \( D \) on \( P_1 \),

\[
newDDuration = \frac{1}{\frac{13}{30}} = \frac{30}{13}
\]

For actor \( E \) on \( P_1 \),

\[
newEDuration = \frac{2}{\frac{13}{30}} = \frac{60}{13}
\]
\[ th_{p_1}^{-1} = th_{p_2}^{-1} = newDDuration + newEDuration = \frac{90}{13} \]

\[ th_{sdf2} = \frac{1}{th_{p_1}^{-1} + th_{p_2}^{-1}} = \frac{13}{180} \approx 0.072 \]

So the throughput of SDF2 is 0.072.

Upon reviewing the figure, it becomes evident that the overall utilization of processors has improved significantly. However, the initial hand-written solution outperforms the second solution in terms of the throughput of SDF1 and SDF2. The reason may be that the different actors have different WCETs for different processors.

Moreover, there exist numerous potential strategies for allocating actors and periodic tasks. For instance, we can swap the processors on which tasks and actors run or reassign processors to them. There are multiple feasible solutions to explore, demanding a considerable amount of time to identify the optimal ones. It is low-efficiency and error-prone.

### 5.3.2 Solution by IDeSyDe

This thesis chose *fiodl* files for the input of the test. For this case study, it needs four files “SDF_1.fiodl”, “SDF_2.fiodl” for SDF Graphs, “periodic-Task.fiodl” for the independent periodic tasks set, and “small_bus_platform.fiodl” for the platform. Figure 5.3 shows the instructions for running this example for users. It provides the solution faster than the hand-written solution.

The rationale of IDeSyDe is that the tool finds a feasible solution on the first try; after that, it can only improve the solutions, until nothing more exists. Finally, it shows the solutions, which include the Pareto-optimal set.

![Figure 5.3: Test Method](image)

For this case, IDeSyDe provides the best solution. We can know the situation of scheduling by checking the parameter `processSchedulings` in the
Figure 5.4 shows the best trade-off solutions by IDeSyDe.

Solution by IDeSyDe schedules task $\tau_1$, $\tau_3$ and actors $A$ $D$ $E$ to $P_0$ and schedules tasks $\tau_2$ $\tau_4$ and actors $B$ $C$ to $P_1$.

For SDF run on $P_0$, the remaining utilization is

\[
1 - u_{P_0} = 1 - \left(\frac{1}{3} + \frac{4}{10}\right) = \frac{4}{15}
\]

For SDF run on $P_1$, the remaining utilization is

\[
1 - u_{P_1} = 1 - \left(\frac{1}{5} + \frac{5}{30}\right) = \frac{19}{30}
\]

For actor $A$ on $P_0$,

\[
newADuration = \frac{1}{\frac{4}{15}} = \frac{15}{4}
\]

For actor $D$ on $P_0$,

\[
newDDuration = \frac{1}{\frac{4}{15}} = \frac{15}{4}
\]

For actor $E$ on $P_0$,

\[
newEDuration = \frac{1}{\frac{4}{15}} = \frac{15}{4}
\]

For actor $B$ on $P_1$,

\[
newBDuration = \frac{3}{\frac{19}{30}} = \frac{90}{19}
\]
For actor C on P₁,

\[ newCDuration = \frac{3}{\frac{17}{30}} = \frac{90}{19} \]

\[ newBDuration = \frac{2}{\frac{1}{15}} = \frac{15}{2} \]

For P₀, the longest duration path is

\[ newADuration + newDDuration + newEDuration = \frac{15}{4} + \frac{15}{4} + \frac{15}{4} = \frac{45}{4} \]

For P₁, the longest duration path is

\[ newBDuration + newCDuration = \frac{90}{19} + \frac{90}{19} = \frac{180}{19} \]

Since SDF₁ and SDF₂ are both in P₀ and P₁, they bottleneck each other. Therefore, the inverse throughput of both is the bottleneck, which is \( \frac{180}{19} \).

\[ th_{sdf1} = th_{sdf2} = \frac{1}{\frac{180}{19}} \approx 0.10555 \]

Compared with the manual solution, we can find that all solutions choose the two processors, and as for the throughput, the solution by IDeSyDe outperforms the solution by hand.
Chapter 6

Discussion

In this section, we will discuss the results of the solution from IDeSyDe and the handwriting for the given case. We will then conclude what the pros and cons of IDeSyDe.

Based on our test design, all solutions are required to choose two processors, and in this regard, IDeSyDe gives the correct allocation.

Upon comparison between manual solutions and IDeSyDe’s solution, it is apparent that the solution generated by IDeSyDe outperforms the handwriting solution in terms of throughput of SDF. Additionally, it significantly expedites the solution-finding process, saving the time.

It can be summarized that the advantages of finding solutions by IDeSyDe:

• High efficiency. To find the best trade-off solution that fits Pareto-front, it needs to calculate the different solutions in different situations. Manual solutions are time-consuming, especially when dealing with more models with higher computational complexity. IDeSyDe, on the other hand, efficiently calculates different solutions in various scenarios, eliminating time-consuming.

• Cover more solutions. IDeSyDe can easily find more solutions than manual calculations. Human calculations are not easy to discover all possible solutions, leading to potential oversight of better trade-offs. It means, that although we perceive we find the best trade-off solution, it not be true for a vast space exploration. IDeSyDe, with its systematic approach, guarantees that a wider range of solutions is considered, which reduces the risk of missing optimal trade-offs.

But we have to acknowledge getting solutions still needs some time, because of the overhead of IDeSyDe. The overhead is mainly associated with
repeatedly initializing and executing *Identification* modules and *Exploration* modules processes [25]. Moreover, it still requires manual calculation, although the calculations are not complex.

In conclusion, IDeSyDe is a convenient tool for DSE, which simplifies these manual tedious procedures. The output files generated by IDeSyDe illustrate all the solutions, and we can get the allocation method from the JSON file. It has high efficiency and ensures accuracy. Meantime, IDeSyDe provides better solutions with guaranteed performance metrics, ensuring reliable and accurate results.
Chapter 7

Conclusions and Future work

7.1 Conclusions

In conclusion, the validation of design space exploration solutions for co-mapping periodic and streaming applications on a shared platform is a changeling, and a significant undertaking, and the results have been promising. We applied the theory that we proposed and techniques to achieve this combination. In this dissertation, the IDeSyDe tool is used, and following the DSI methodology, for tackling the problem.

Our evaluation employed HSDF and independent periodic tasks as application models, and TDMP multiprocessors as the platform. The tests show our success in mapping and scheduling the combination model of periodic models and SDF onto the shared platform. We effectively found the trade-off solutions between metrics, meeting the goals and improving the performance of finding the optimal solutions.

However, due to the time limitation of the thesis, only the initial goals have been achieved. There are still some aspects that can be improved, and some other methods can be tried that we could do in the future. Concrete work will be introduced in the following.

This presents an excellent avenue for future research, particularly for persons who are devoting embedded development, especially focusing on extending the IDeSyDe tool.

7.2 Limitations

While the thesis has achieved significant progress in exploring the co-mapping of periodic and streaming applications on a shared platform, there are some
limitations that should be acknowledged:

Limited Empirical Scalability Testing: Due to limited time, only a small special case, where the value of one decision variable is fixed, was conducted. While this test successfully demonstrated the combination’s viability and found the best solution, it is imperative to acknowledge that relying solely on one case is insufficient. More broader and more extensive set of cases should be considered to provide a more comprehensive and robust validation.

Model Choice: In this dissertation, we only consider two different types of models: an independent periodic tasks model and a synchronous dataflow model. To reduce the complexity of the combination, we choose HSDF as the SDF input model. However, in real embedded systems, these two kinds of models may not fully capture the complexities and variations present in real-world embedded systems with diverse application models.

Lack of Real-world Applicability and Implementation: The thesis mainly uses theory and simulation. Besides, this thesis only considers "Partitioned scheduling". Specifically, in this thesis, the mapping of tasks is fixed, instead of task migration. It eliminates the complexity of combination. However, it may not be suitable for various scenes.

7.3 Future work

In this section, we outline future work, focusing on addressing some of the remaining issues, limitations, and other insights of the thesis.

The future work can be categorized into five main aspects:

As highlighted in the limitations section, our scalability testing has been limited in scope. Specifically, we have primarily focused on a single, special case where one variable is held constant. To enhance the robustness and credibility of our project, a more diverse range of test cases can be conducted. Getting some results similar to the example of Pareto optimization given in this thesis, incorporating multiple scenarios would increase the persuasiveness of this thesis.

Dynamic task allocation: One of the future works involves dynamic task allocation techniques. Considering the attributes of flexibility, efficiency, and fault tolerance of dynamic task migration, dynamic allocation techniques can be adapted in future work. This technique may involve adaptive scheduling and optimizing task allocation to dynamically adjust.

Consider more details: For the implementation section, the prototype works well, but further considerations could enhance this design. The point
is to provide the range throughput for actors of SDF, which offers more constraints during DSE.

Enlarge the types of models: The thesis selects HSDF and independent periodic tasks, along with the TDMB multiprocessors as the platform. Future work could explore other types of models to enlarge the scope of the thesis. For example, We can choose other types of SDF, such as Cyclo-Static Data Flow (CSDF) [26], which is flexible because it can handle the different consumption and production rates of actors. For tasks, we can consider dependent tasks, which would be applicable to more scenarios.

Add other co-mapping approaches: This thesis adopts the co-mapping approach where periodic tasks execute first, and SDF only executes when there are the remaining resources. A possible feasible method could treat the execution of periodic tasks and SDF models as separate entities, following round-robin scheduling. We define the total time unit per execution and allocate it to the proportion for both models. In this case, SDF always has a higher priority than periodic task models.

Addressing these aspects in future work enhances applicability, and provides valuable contributions to the field of co-mapping and embedded system design.
Conclusions and Future work
References


Appendix A

Main equations

This appendix gives the significant equations that are used throughout this thesis.

The concrete details of utilization of periodic tasks Equation (A.1) set and new server-based throughput of SDF Equation (A.2) is shown in Section 4.

\[ u_{Pi} = \sum_{j=1}^{n} \frac{C_{ij}}{T_{ij}} \]  \hspace{1cm} (A.1)

\[ th_i^* = \left\lceil \frac{th_i}{1 - u_{Pi}} \right\rceil \]  \hspace{1cm} (A.2)