中国科学技术大学

专业硕士学位论文

（专业学位类型）

轻量级仿真平台硬件平台建模及应用的设计与实现

作者姓名：

专业领域：

校内导师：

企业导师：

完成时间：　二〇二二年三月二十九日

University of Science and Technology of China

A dissertation for master's degree

(Professional degree type)

Design and implementation of hardware platform modeling and application of lightweight simulation platform

Author:

Speciality:

Supervisor:

Advisor:

Finished time: March 29, 2022

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文，是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____ 签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

控阅的学位论文在解密后也遵守此规定。

☑ 公开 ☐ 控阅（____ 年）

作者签名：_____ 导师签名：_____

签字日期：_____ 签字日期：_____

摘　要

随着片上系统的设计复杂性不断提高，在设计前期通过对系统层次进行软、硬件的划分对片上系统各方面性能的影响日趋增加，迫切需要高效快捷的性能分析和验证方法学。而 ESL 设计方法不仅提供快速的仿真验证方法，还提供了详细的性能分析指标，现如今已经成为 SoC 设计领域最先进的设计方法。

由于片上系统设计复杂性的提高，在方案设计的过程中考虑到系统设计的每个方面变得不那么可能，因此我们需要自动化的工具或者流程去完成前期对巨大的设计空间的剪枝，设计空间探索（Design Space Exploration，DSE）成为了一种很好的方法去实现这一功能。但在如今系统级仿真过程中，仿真系统的复杂度越来越大，从而导致系统的复杂程度不适于进行快速的架构探索，所以我们需要建立一个轻量级的仿真平台，并在这个平台的基础上进行快速的架构探索。针对以上问题，本文做了以下几部分工作：

1. 设计并实现了轻量级仿真平台中硬件平台模块，包括硬件平台构建模块以及硬件模型建模。与仿真平台软件部分以及总线互联模块共同组成整个轻量级仿真平台，以用例文件和硬件配置文件作为输入，完成业务仿真整个流程，并最终输出仿真结果以及存储仿真过程中信息的数据库文件，提高了业务仿真的效率。
2. 设计并实现设计空间探索流程中的训练集生成模块以及仿真平台预测模型，参与实现了多目标的设计空间探索，得到了最终的帕累托最优解。并通过对结果的对比分析，得出方案设计的相应结论。

本文详细介绍了包含轻量级仿真平台硬件平台建模和设计空间探索流程中训练集生成模块以及仿真平台预测模型的需求分析、概要设计、系统设计与实现和系统测试与结果分析。本文为业内在前期芯片架构方案设计阶段提供新的一整套解决方案，并设计并实现了一个轻量级仿真平台，为一些快速的设计方案的实现提供了更加方便的平台。

关键词：电子系统级设计；轻量级仿真平台，Simpy；多目标优化；设计空间探索，硬件平台建模

ABSTRACT

With the increasing complexity of system-on-chip design, the influence of software and hardware division on the performance of system-on-chip is increasing day by day in the early stage of design, so efficient and efficient performance analysis and verification methods are urgently needed. The ESL design method not only provides fast simulation verification methods, but also provides detailed performance analysis indicators, and has become the most advanced design method in the field of SoC design.

Due to the increase in the complexity of system-on-chip design, it has become impossible to consider every aspect of system design in the process of scheme design. Therefore, we need automated tools or processes to complete the pruning of the huge design space in the early stage. Design Space Exploration (DSE) has become a good way to achieve this function. But in the current system-level simulation process, the complexity of the simulation system is getting bigger and bigger, which makes the complexity of the system unsuitable for rapid architecture exploration. Therefore, we need to build a lightweight simulation platform. Based on the rapid structure exploration. In response to the above problems, this article has done the following parts:

1. The hardware platform module in the lightweight simulation platform is designed and implemented, including hardware platform construction module and hardware model modeling. Together with the software part of the simulation platform and the bus interconnection module, it forms the whole lightweight simulation platform, takes the use case file and hardware configuration file as the input, completes the whole process of business simulation, and finally outputs the simulation results and the database file storing the information in the simulation process, which improves the efficiency of business simulation.
2. Design and implement the training set generation module and simulation platform prediction model in the design space exploration process, participate in the multi-objective design space exploration, and obtain the final Pareto optimal solution. Through the comparative analysis of the results, the corresponding conclusions of the scheme design are obtained.

This paper introduces in detail the demand analysis, outline design, system design and implementation, system test and result analysis of the training set generation module and simulation platform prediction model in the process of hardware platform modeling and design space exploration of lightweight simulation platform. This article

provides a new set of solutions for the industry in the pre-chip architecture design stage, and designs and implements a lightweight simulation platform, which provides a more convenient platform for the realization of some rapid design solutions.

Key Words: Electronic System Level; Lightweight simulation platform; Simpy; Multi-objective optimization; Design Space Exploration，Hardware platform modeling

目　录

第 1 章　绪论 · · · · · 1
1.1　选题意义及选题背景 · · · · · 1
1.2　国内外研究现状及发展趋势 · · · · · 2
1.2.1　仿真建模技术的发展现状 · · · · · 2
1.2.2　芯片系统性能分析发展现状 · · · · · 4
1.2.3　设计空间探索流程的发展现状 · · · · · 4
1.3　本论文的主要工作 · · · · · 4
1.4　本文的组织结构 · · · · · 5
第 2 章　相关技术简介 · · · · · 7
2.1　ESL 建模介绍 · · · · · 7
2.2　SimPy 仿真框架介绍 · · · · · 7
2.3　多目标优化介绍 · · · · · 8
2.4　SQLite 数据库介绍 · · · · · 9
2.5　仿真平台整体介绍 · · · · · 9
2.5.1　仿真平台整体架构 · · · · · 9
2.5.2　仿真平台业务仿真流程 · · · · · 9
2.6　本章小结 · · · · · 14
第 3 章　需求分析 · · · · · 15
3.1　系统需求分析简介 · · · · · 15
3.2　功能性需求 · · · · · 16
3.2.1　GeneralFifo 模块功能需求 · · · · · 16
3.2.2　Processor 模块功能需求 · · · · · 17
3.2.3　Memory 模块功能需求 · · · · · 18
3.2.4　普通用户 1 需求 · · · · · 19
3.2.5　高级用户需求 · · · · · 20
3.2.6　普通用户 2 需求 · · · · · 21
3.3　非功能需求 · · · · · 22
3.3.1　可扩展性需求 · · · · · 22
3.3.2　可维护性需求 · · · · · 22
3.3.3　性能需求 · · · · · 23
3.4　本章小结 · · · · · 24

第 4 章　概要设计 · · · · · 25
4.1　系统整体结构 · · · · · 25
4.2　仿真平台硬件模型生成模块概要设计 · · · · · 26
4.2.1　硬件平台构建模块概要设计 · · · · · 27
4.2.2　GeneralFifo 模块概要设计 · · · · · 28
4.2.3　Memory 模块概要设计 · · · · · 30
4.2.4　Processor 模块概要设计 · · · · · 31
4.3　设计空间构建模块及仿真预测模型概要设计 · · · · · 32
4.3.1　设计空间构建模块概要设计 · · · · · 33
4.3.2　仿真平台预测模型概要设计 · · · · · 33
4.4　仿真信息数据库概要设计 · · · · · 35
4.5　本章小结 · · · · · 38
第 5 章　详细设计与实现 · · · · · 39
5.1　仿真平台模块硬件平台构建模块的设计与实现 · · · · · 39
5.2　GeneralFifo 模块设计与实现 · · · · · 41
5.3　Processor 模块设计与实现 · · · · · 42
5.4　仿真平台 Memory 模块设计与实现 · · · · · 48
5.5　设计空间探索相关模块的设计与实现 · · · · · 50
5.5.1　训练集生成模块的设计与实现 · · · · · 50
5.5.2　预测模型生成模块设计与实现 · · · · · 51
5.5.3　设计空间探索结果分析 · · · · · 53
5.6　本章小结 · · · · · 56
第 6 章　系统测试与结果分析 · · · · · 57
6.1　仿真平台测试 · · · · · 57
6.1.1　测试环境 · · · · · 57
6.1.2　测试工具和方法 · · · · · 57
6.1.3　硬件模块功能测试 · · · · · 57
6.1.4　硬件平台构建模块测试 · · · · · 60
6.1.5　仿真平台运行模块测试 · · · · · 61
6.1.6　仿真平台性能测试 · · · · · 63
6.2　本章小结 · · · · · 63
第 7 章　结果与展望 · · · · · 64
7.1　论文总结 · · · · · 64
7.2　问题和展望 · · · · · 64

附录 A　缩略词对照表 66

插 图 清 单

图 1.1 软硬件协同设计的基本思路 · · · · · 1
图 1.2 基于 SystemC 的软硬件协同设计流程图 · · · · · 3
图 1.3 主要工作示意图 · · · · · 5

图 2.1 ESL 设计流程图 · · · · · 7
图 2.2 仿真平台功能模块示意图 · · · · · 9
图 2.3 仿真平台业务仿真流程图 · · · · · 10
图 2.4 任务数据流图示例图 · · · · · 10
图 2.5 调度器模块结构图 · · · · · 11
图 2.6 MasterPortMgnt 实现流程图 · · · · · 13

图 3.1 GeneralFifo 模块用例图 · · · · · 16
图 3.2 Processor 模块用例图 · · · · · 17
图 3.3 Memory 模块用例图 · · · · · 18
图 3.4 普通用户 1 用例图 · · · · · 19
图 3.5 普通用户 1 执行用例文件用例图 · · · · · 19
图 3.6 高级用户用例图 · · · · · 21
图 3.7 普通用户 2 用例图 · · · · · 21

图 4.1 系统整体框架图 · · · · · 25
图 4.2 硬件平台架构图 · · · · · 26
图 4.3 硬件平台生成图 · · · · · 27
图 4.4 硬件平台生成活动图 · · · · · 28
图 4.5 GeneralFifo 流程图 · · · · · 29
图 4.6 Memory 模块结构图 · · · · · 30
图 4.7 Processor 模块结构图 · · · · · 31
图 4.8 设计空间探索流程图 · · · · · 33
图 4.9 训练集生成模块流程图 · · · · · 34

图 5.1 硬件平台树状结构示意图 · · · · · 39
图 5.2 硬件模块实例化模块类图 · · · · · 40
图 5.3 GeneralFifo 类图 · · · · · 41
图 5.4 任务执行流程图 · · · · · 43
图 5.5 Processor 模块类图 · · · · · 44

图 5.6 三种数据类型类图 · · · · · 45
图 5.7 数据类型关系图 · · · · · 45
图 5.8 DMA 模块流水线时序图 · · · · · 46
图 5.9 HAC 模块三级流水时序图 · · · · · 47
图 5.10 Bank 模块交织方案 · · · · · 49
图 5.11 Memory 模块处理流程图 · · · · · 49
图 5.12 设计空间探索模块整体流程图 · · · · · 50
图 5.13 训练集文件格式示例 · · · · · 51
图 5.14 仿真结果与预测结果对比图 · · · · · 52
图 5.15 设计空间探索结果仿真目标值结果对比图 · · · · · 53
图 5.16 50 组设计方案时延结果对比 · · · · · 54
图 5.17 50 组设计方案核利用率结果对比 · · · · · 54
图 5.18 50 组设计方案核一致性结果对比 · · · · · 55
图 5.19 仿真平台结果分析图 · · · · · 55

图 6.1 硬件平台构建输出消息 · · · · · 61
图 6.2 仿真运行输出消息 · · · · · 62

表 格 清 单

表 3.1 硬件模块功能需求列表 · · · · · 15
表 3.2 用户需求列表 · · · · · 16
表 3.3 性能需求表 · · · · · 23

表 4.1 SimRecord 表 · · · · · 35
表 4.2 HacJobTable 表 · · · · · 35
表 4.3 HacJobTable 表 · · · · · 36
表 4.4 AlaKernelTaskTbl 表 · · · · · 36
表 4.5 WFEJobTable 表 · · · · · 36
表 4.6 Pl2UsageRecord 表 · · · · · 37
表 4.7 SimUsageTable 表 · · · · · 37
表 4.8 DBMUsageRecord 表 · · · · · 37

表 5.1 仿真平台设计空间 · · · · · 51
表 5.2 多目标预测模型误差值 · · · · · 52
表 5.3 最优设计方案设计参数 · · · · · 55

表 6.1 GeneralFifo 模块测试表 · · · · · 58
表 6.2 Processor 模块测试表 · · · · · 58
表 6.3 Memory 模块测试表 · · · · · 59
表 6.4 硬件平台构建模块测试表 · · · · · 60
表 6.5 仿真平台运行模块测试表 · · · · · 62
表 6.6 仿真平台性能测试表 · · · · · 63

第 1 章　绪　　论

1.1　选题意义及选题背景

随着片上系统（System on Chip,SoC）的迅速发展，一个 SoC 芯片上被嵌入了越来越多的微处理器和芯片，导致片上系统的设计复杂性不断提高。在上市时间迅速减少以及功耗降低的需求背景下，抽象层次较低的 RTL 设计已经不能满足 SoC 设计的需求。现在设计抽象层次已经被推向电子系统级别（Electronic System Level,ESL），可以在 SoC 设计过程中进行快速原型设计和早期验证[?]，或者在设计过程能够在设计初期高效迭代探索出比较准确的设计方向或大致几种比较优秀的方案就显得尤为重要。如果在芯片的 RTL 设计和验证过程中才发现芯片的体系结构或软硬件划分无法满足系统功能、性能和成本的要求，就会出现大量的重复工作。ESL 设计可以根据实际需求提供比 RTL 设计更快的仿真速度和不同抽象层次的时序精度，从而更好的评估 SoC 系统的整体性能。软硬件协同设计采用统一的语言描述系统功能，在系统实现之前可以对整个系统进行仿真，以便于尽早的发现功能问题。然后根据整个系统的约束对系统进行硬件划分。然后软件部分和硬件部分可以进行同步开发，最后再一起协同验证。采用软硬件协同设计可以大大减短芯片的开发周期，提高芯片性能，降低开发成本。现如今软硬件协同设计的基本思路如图 1.1 所示：

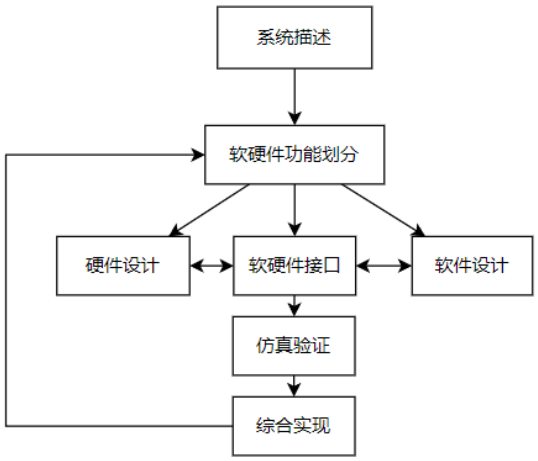

图 1.1　软硬件协同设计的基本思路

仿真模型模型的建立和验证是处理器设计周期中一个重要阶段。当前，大多数的处理器仿真模型为 C/C++ 所写，仿真模型的建立基于 SystemC[?]编写，用于统一硬件模型设计以及上层软件平台，这虽然节省了一定的沟通成本，以及保持体系结构设计及验证阶段模型的一致性。但灵活性较差，为了追求验证结果的精度，系统复杂度较大。因此，为了较快的完成架构空间探索，我们需要重新设

计一个轻量级，且灵活可配置的仿真平台。

在硬件/软件协同设计过程[?]中提高设计生产率的趋势之一是提高抽象水平。如今，大多数设计流程的实现过程都是基于寄存器传输级别（RTL）的描述。一方面，用于仿真的语言（例如 SystemC）提供了更高的抽象级别，另一方面，通过高层次合成（High Level Synthesis,HLS）[?]的能力也提供了更高的抽象级别。高级综合的目的是双重的。首先，通过在更高层次上描述系统，可以获得生产率的提高。其次，由于设计转换发生在更高的层次上，因此有更大的探索设计空间的潜力，这将导致更好的设计。这种发展的重要性直接影响到设计成本。

较大的设计空间、较高的性能目标要求以及紧迫的上市时间要求对系统的设计提出了极大的挑战。通常，设计空间探索 (DSE)[?] 在早期开发阶段执行，以在满足整体系统设计要求的多个高级设计配置之间进行选择。自动的设计空间中寻找和探索可行的解决方案。

本仿真设计平台基于 Python 的 Simpy[?]包构建的仿真环境。Python 语言具有简易快速的特点，我们可以快速的重构整个仿真平台。同时基于这种仿真环境构建的仿真平台具有仿真速度快的优点，我们在设计空间探索过程中可以快速进行，满足设计空间探索的要求。在本文中我们打通了在极大的设计空间中探索能使得系统性能最优的设计参数的设计空间探索流程。

1.2　国内外研究现状及发展趋势

1.2.1　仿真建模技术的发展现状

随着片上系统系统的复杂度不断提升，高层建模技术由于其更快的建模速度、更短的仿真时间以及更高的抽象程度逐渐被大多数研发水平高的机构、高校、电子设计自动化（Electronic Design Automation，EDA）公司所重视，并开发出多种高水平的仿真模拟工具以及仿真平台，具有代表性的有 Synopsys 的 CoWare，Mentor 的 Vista，Carbon 的 SoC Designer 等等。这些工具为芯片设计过程中遇到的许多问题提供了很多的解决方案，从各个模块模型的建立、系统架构的分析到功能验证以及时序分析，进一步分析功耗和时延等性能指标[?]，甚至给出了高层次综合（High Level Synthesis，HLS）方案。

ESL 设计作为系统级设计方法，其核心的思想是从算法级模型转变过来的。ESL 设计方法学和 ESL 设计工具最早是在 20 世纪 90 年代提出的。随着 SoC 设计技术的发展，ESL 设计越来越受到重视，真正能够执行设计流程的 ESL 设计工具是近几年才出现的。

目前硬件建模仿真平台一部分基于 C/C++ 编写，只能进行串行或多线程仿真和验证，后期的硬件原型建模需要通过硬件描述语言（如 Verilog）完成。另外一

部分用 SystemC 完成，SystemC[?]是一种系统级建模语言，在 C++ 语言的基础上扩展了一系列的硬件库。SystemC 更擅长描述复杂的算法和控制流程，能够更清楚的描述出整个电子系统的复杂行为。SystemC 由 Accellera 定义和推广,SystemC 适用于系统级建模、架构探索、性能建模、软件开发功能验证和高层次综合，通常与电子系统级设计和事务级建模（Transaction Level Modeling,TLM）相关联。SystemC 语言既可以按照硬件设计的要求进行时钟精确并行设计，而且还可以根据软件设计人员的要求进行更高抽象层级的串行设计，还能兼容 C 语言以及 C++ 语言，并且有专门的标准接口处理 SystemC 语言与硬件语言的连接问题, 因此很适合作为连接软件设计和硬件设计的桥梁[?]。SystemC 可以在系统级用高级语言同时描述软件行为和硬件行为，从而实现软硬件协同设计[?]。图 1.2 是基于 SystemC 的软硬件协同设计流程图。

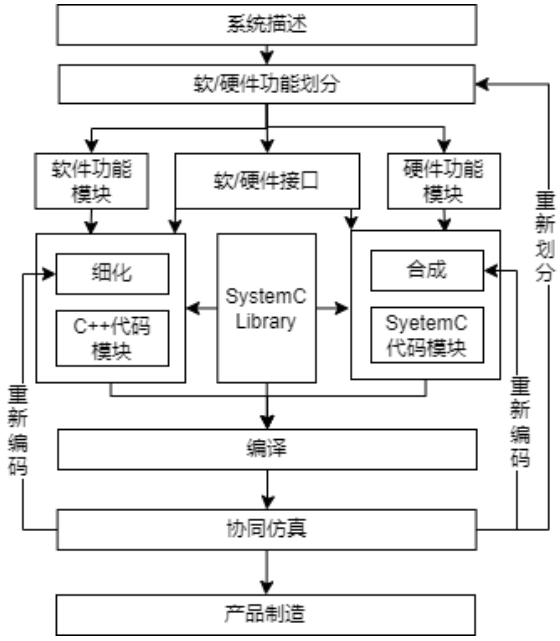

图 1.2　基于 SystemC 的软硬件协同设计流程图

基于 SystemC 的实现方式相对于前者的优点是将仿真平台的建立以及后期硬件模型验证统一起来，可以加快开发流程，节约设计流程和设计成本。在架构探索阶段，通过对前期简易建模的仿真分析，从而得出符合设计要求的架构设计，并快速对体系结构进行仿真，得到影响系统性能的因素。在功能验证方面，采用 SystemC 事务级模型对 SoC 高层模型进行设计和验证[?]。基于 SystemC 的标准事务级建模支持不同系统之间的模型交换。电子系统级设计抽象的描述系统级芯片的行为，能够满足芯片设计早期深入设计芯片系统仿真的需要[?]。这已经成为 SoC 设计领域前沿的设计方法。

现在又提出了新的仿真工具 Simpy，SimPy 最初基于 Simula 和 Simscript 的思想，但使用标准的 Python。它结合了两个以前的软件包，即 Simula 风格的 SiPy

和 Simscript 风格的 SimPy。SimPy 是一个基于标准 Python 的，以进程为基础的离散事件仿真框架。SimPy 中的进程是由 Python 中的生成器构成，生成器的特性是可以模拟仿真具有主动性的对象。但现在 Simpy 基本很少用于大型平台建模，主要用于某个具体场景仿真建模即单场景仿真[?]，如使用 SimPy 仿真框架进行基于森林的供应链建模[?]。

1.2.2　芯片系统性能分析发展现状

现如今对于芯片性能进行评估主要从两个方面进行：第一种方法是使用仿真器，如 Worm_sim、NoCSim 、NIRGAM 等等。还有一个方法是建立分析模型，如基于排队论或链路拆解原理的分析模型[?]。前者由于非定制的原因仿真精度不足但仿真速度较快，后者仿真精度高但速度较慢。现在已经有了能应用于 CPU 等复杂 SoC 的仿真工具，如应用广泛的 Gem5[?]。它由研究机构和企业合作开发，既支持应用程序仿真也支持全系统仿真，具有精确的时钟周期，可配置并与工业化标准体系结构和 CPU 模型集成。

1.2.3　设计空间探索流程的发展现状

设计空间探索是指根据系统设计所关心的参数而对不需要的参数进行系统分析和修剪[?]，虽然 DSE 适用于所有类型的系统，但主要涉及到的主要是电子系统和嵌入式系统。现代嵌入式系统通常是具有异构的多处理器系统架构。它们由多个处理器组成，从完全可编程的内核到用于时间紧迫任务需求而完全专用的硬件模块。这一类系统的组件越来越多的集成到单个芯片上，从而产生了异构的多处理器片上系统（MPSoC）架构[?]。为了应对此类系统的复杂性，近 15 至 20 年来，电子系统级设计方法这种新设计方法随之出现。并用此建立里一些高级模型，使建模工作最小化，并优化了模型的执行速度。在此基础上，我们能够在非常早期的设计阶段就应用设计空间探索。在此探索阶段，可以探索各种不同的设计替代方案。但设计空间探索的过程也极具挑战性，因为探索的设计空间很大。例如，用于探索应用程序到处理资源的不同映射[?]以及不同调度策略及系统参数对系统性能的影响。

1.3　本论文的主要工作

本文详尽的阐述了一个基于 Simpy 仿真环境搭建的仿真平台中硬件平台的构建、硬件模型的建模以及基于这个仿真平台实现的设计空间探索流程。该仿真平台能够快速的让用例在硬件平台上运行并给出仿真结果及仿真运行过程中的任务执行细节以及硬件资源的利用情况，打通了设计空间探索的流程，并最终给

出设计参数集合的帕累托最优解集。本文将从以下几个方面进行阐述：

1. 仿真平台硬件模块的整体架构设计，主要包括硬件平台配置文件的读取以及根据硬件配置信息实例化硬件 IP 模型，以及用例文件的任务信息的读取及整合，为整个仿真平台的运行打下基础。
2. 仿真平台 Processor 模块的整体架构设计，主要是各个硬件模块执行任务的逻辑以及硬件模块的实现，还包括了整个系统中交互的基础模型 General-Fifo 的实现。
3. 仿真平台 Memory 模块的整体架构设计，主要是数据在 Memory 模块上的读取和存储的逻辑实现，以及 Memory 模块与总线互联模块的交互设计。
4. 设计空间探索流程的实现，主要包括基于仿真平台实现仿真过程的预测模型，并基于预测模型实现设计空间探索流程，最终给出帕累托最优解集，并对设计空间探索结果进行分析。

本人的工作在整个项目中的结构示意图如图 1.3 所示。在整个项目过程中，本人的主要工作如下：

1. 设计并实现基于 Simpy 的仿真平台，完成仿真平台设计中的硬件平台构建模块、Processor 模块、Memory 模块、数据库模块以及整个仿真平台各个模块之间的交互部分。
2. 根据仿真平台实现预测模型，并基于硬件模块实现设计空间探索流程，并根据最终给出的帕累托最优解集给出相应的结果分析。

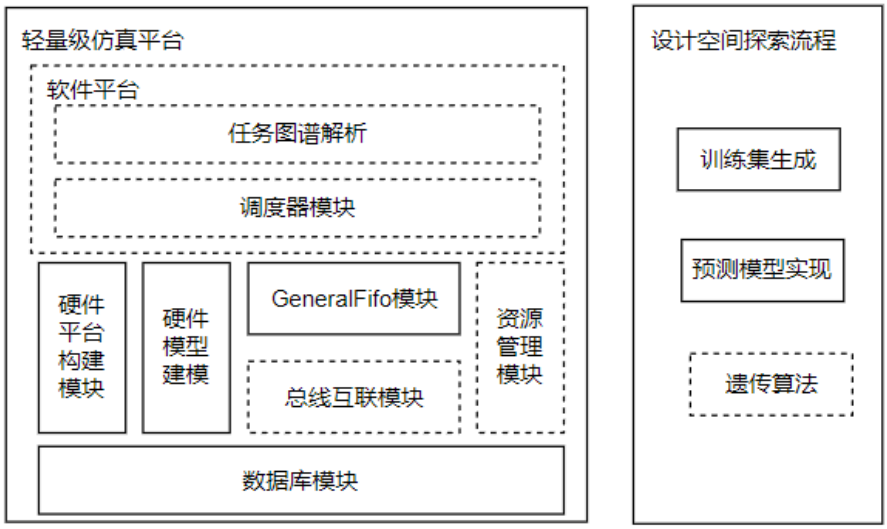

图 1.3　主要工作示意图

注：实线模块为本人主要工作

1.4　本文的组织结构

本文主要分为 7 个章节，每个章节的具体内容安排如下：

第一章：绪论：介绍不同模型仿真及系统仿真的发展，以及设计空间探索的国内外发展现状，并介绍了本文的研究内容。

第二章：相关技术介绍：对仿真平台及设计空间探索的设计实现所用到的关键技术进行详细的阐述，一方面介绍了系统建模的基础知识以及我们所采用的仿真环境 Simpy 框架，另一方面介绍了仿真平台的整体流程以及相关模块的整体设计。

第三章：需求分析：基于仿真平台的功能及性能需求及设计空间探索结果分析准确性的需求的基础上，对仿真平台及设计空间探索的需求分析进行详细的阐述。

第四章：概要设计：这部分主要是在需求分析的基础上对轻量级仿真平台中硬件平台建模的设计及设计空间探索模块的设计进行简要的阐述，对仿真平台的整体架构及各个模块的功能划分进行了详细的说明。通过概要设计，为整个应用的实现做好铺垫。

第五章：详细设计与实现：基于概要设计的方案，结合软件工程的的开发思想，详细的介绍了仿真平台硬件平台的搭建、各个硬件模型的建模、整个用例任务的执行、设计并实现了各个硬件模块之间数据及消息交互相应的接口、设计空间探索流程的具体实现。同时，我们对设计空间探索流程最终给出的设计参数的集合进行了分析并给出相应的结论。

第六章：系统测试及结果分析：在实现了仿真平台及设计空间探索流程之后，我们基于仿真平台的执行以及结果准确性等方面对整个仿真平台进行了测试，确保仿真平台能达到预期的结果。

第七章：总结和展望：在本章，笔者对这篇论文进行简单的总结，并对仿真平台及设计空间探索流程中的不足进行了描述，并对仿真平台及设计空间探索流程提出了未来的展望。

第 2 章　相关技术简介

本章简要介绍与本文密切相关的几种技术及框架，整个系统基于这几种技术进行开发。在本章，我们将对 ESL 建模、Simpy 仿真框架以及一些相关技术进行详尽的阐述。

2.1　ESL 建模介绍

ESL(Electronic System Level) 设计和验证方法是一种电子设计方法。基本方法是使用高级语言或使用图形化工具对整个系统的行为进行建模。在新的语言不断出现的情况下，现在常常使用 SystemC 来作为一种抽象的建模语言。

ESL 是许多世界领先的片上系统设计公司的既定方法，并且现在更多地用于系统设计。ESL 最初是被使用为一种“无实现链接”的算法建模方法，现在正在逐渐变成一套互补的方法，支持嵌入式系统的设计、验证和调试，一直到定制 SoC 的硬件和软件实现[?]。ESL 设计流程图[?]如图 2.1 所示。

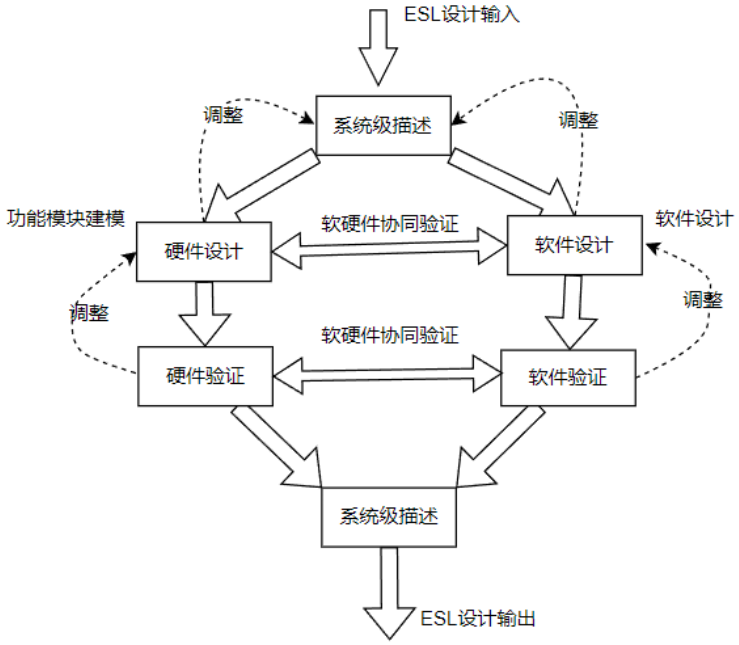

图 2.1　ESL 设计流程图

2.2　SimPy 仿真框架介绍

SimPy 是一个基于标准 Python 的基于过程的离散事件仿真框架。它的事件调度机制基于 Python 的生成器 Generator 来实现的，可用于异步网络或者用来实现多代理系统（具有模拟和真实通信）。

SimPy 中的进程是由 Python 中的生成器函数来定义的。所有的进程都存在

于一个仿真环境中。它们通过事件机制与仿真环境来彼此交互。在它们的生命周期中，它们会创建事件并创建 yield 语句以等待进程被触发。当一个进程产生一个事件时，该进程被挂起。SimPy 中的进程在事件发生时被触发（我们说事件被触发）。多个进程可以等待同一个事件。SimPy 通过记录它们创建并等待该事件的顺序，并以相同顺序恢复这些进程。同时，SimPy 中一个重要的事件类型为 Timeout，此类事件在一定量的（模拟）时间后触发。它们使仿真环境中的进程在给定的时间内休眠（或保持其状态）。所有事件都可以调用 Environment 进程所在的的方法来创建。

SimPy 提供了三种类型的容器，可用于同步进程或对拥塞点进行建模。第一种为 resource，支持优先级和抢占的共享资源；第二种为 container，用于在过程之间共享同种类物质的资源，无论该种类资源是连续的还是离散的；第三种为 store，用于存储支持特定对象请求的可能无限数量的对象的共享资源。此外，我们还可以自定义资源类型的实现。

2.3　多目标优化介绍

在我们对芯片架构进行设计空间探索的过程中，系统架构的目标参数通常都不止一个。在多目标优化问题中，多个目标参数之间可能会存在冲突，并且可能不能同时达到最优。以遗传算法为代表的许多进化算法[?]，都具有能够生成多个点并且能向多个方向进行搜索的特点，因此十分适合这种求解的最优解所在的搜索空间非常复杂的多目标优化问题[?]。

多目标优化问题是在给定约束条件的前提下，求多个目标函数的最大或最小的问题[?]。在多个目标函数中，有的可能是最小化目标函数，有的可能是最大化目标函数。多个目标之间可能会拥有不同的单位，也可能在优化某个目标时损害其他目标。但这并不意味着多目标优化问题可能没有最优解，事实上是可以有的，为了求出比较合理的解，这里引入多目标优化问题的合理解集——Pareto 最优解[?]。

在解决单目标优化问题的时候，这一类问题一定有唯一的最优解，但当解决多目标优化问题的时候，多个目标之间可能互相影响，所以最优解有可能不止一个。所以多目标问题的最优解有如下定义：

给定一个多目标优化问题 minf(x)，设 X* 属于 Ω，如果不存在 X 属于 Ω，使得满足以下条件：对于 f(x) 的任意子目标函数 fi(x) 都有 fi(X)≤fi(X*)，同时至少存在一个子目标函数 fi(x) 使得 fi(X)< fi(X*) 那么，我们称 X* 是一个强 Pareto 最优解[?]。

2.4　SQLite 数据库介绍

SQLite 因为其小巧、方便使用以及符合 ACID 事务特性的优点而受到开发人员的广泛喜爱，它是目前使用最多的小型数据库之一[?]。SQLite 由 D. Richard Hipp 设计开发。在编写导弹驱逐舰上运行的设备控制程序的时候，他产生了设计 SQLite 的想法[?]。SQLite 支持当今几乎所有的主流操作系统，如 Android、IOS 和 Windows 等，还能在 Python、Java、C# 等语言下使用，同时也支持 ODBC 接口。相比于传统数据库如 MySQL、PostgreSQL 等，SQLite 对于数据的处理速度要更加迅速[?]，而且它运行时只需要占用非常低的资源，通常只占用几百 K 的计算机内存，因此被许多嵌入式产品所采用。

2.5　仿真平台整体介绍

整个轻量级仿真平台由整个一起完成，为了更好的介绍设计实现的硬件平台相关的模块和了解仿真平台进行业务的整体流程，接下来主要介绍下仿真平台的整体架构、业务仿真进行的流程以及业务仿真在其他模块上的执行。

2.5.1　仿真平台整体架构

轻量级仿真平台建模整体分为硬件平台模块建模以及软件平台模块。硬件平台模块分为:Processor 模块、总线互联模块以及 Memory 模块；软件平台模块主要分为：输入处理模块以及调度器模块。仿真平台功能模块示意图如图 2.2 所示：

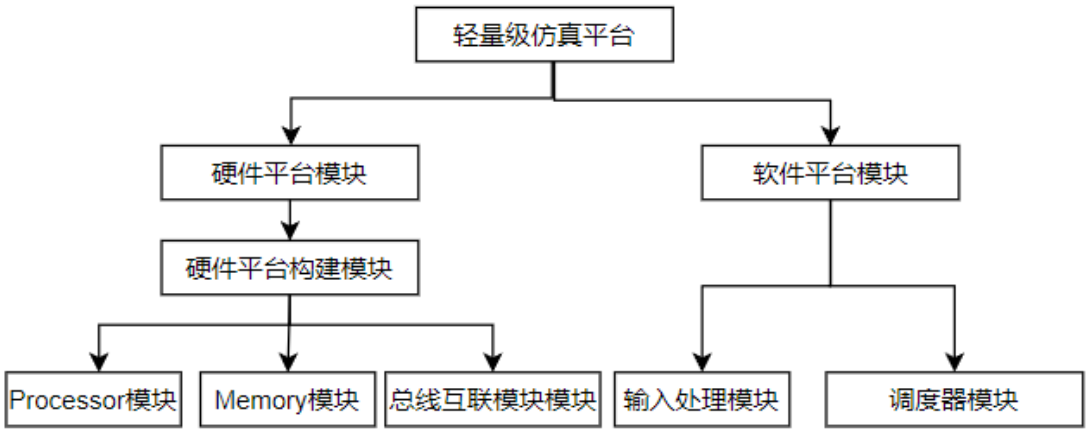

图 2.2　仿真平台功能模块示意图

2.5.2　仿真平台业务仿真流程

仿真平台的实现是为了在芯片正式生产之前对芯片系统架构以及功能进行验证，所以需要通过真实的用例在仿真平台上执行验证。仿真平台的业务仿真流程图如图 2.3 所示：

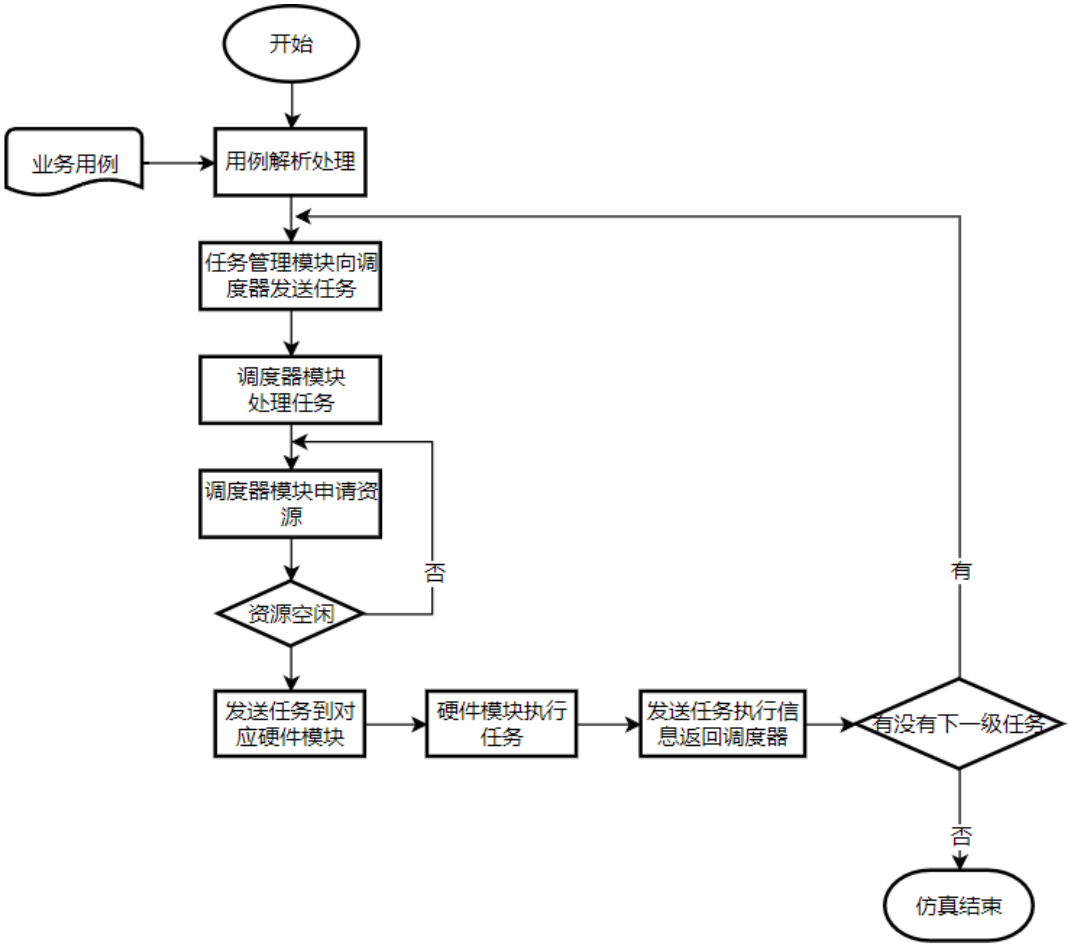

图 2.3　仿真平台业务仿真流程图

接下来简要的介绍整个业务仿真流程的执行，为了更好的理解本文后面设计的硬件平台模块，下面着重介绍仿真平台与其有交互模块的设计原理以及功能。

轻量级仿真平台的输入任务模型是实例级别的数据流图模型，任务之间以数据为触发关系，任务之间的触发为实时触发。当任务的输入数据全部就绪后，该任务实例才会开始执行，一旦任务执行完毕，其输出数据被触发搬运。整体的任务的数据流图示例图如图 2.4 所示。

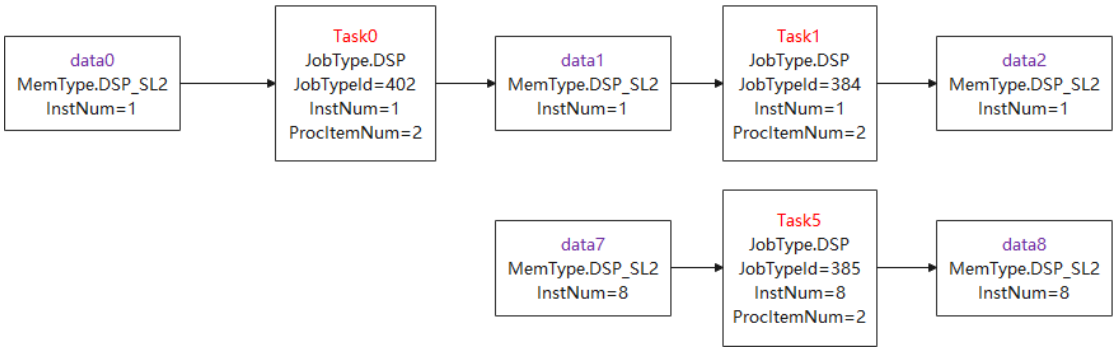

图 2.4　任务数据流图示例图

仿真平台的任务用例输入类型是描述了所有任务相关信息的 Excel 文件，包

括任务类型、任务执行的所属调度器、任务的输入输出数据，数据的大小类型等等。平台初始化时通过 SaeSimulaor 单实例中的 load_task_praph 方法解析任务信息文件，并通过 TaskGraphMgnt 类管理。

TaskGraphMgnt 类通过 ExcelReader 类里的方法读取输入任务用例的 Excel 表，将任务信息遍历读取，根据任务用例中任务具体信息将任务根据自身信息生成不同类型的任务对象，将数据信息生成 DataInst 类对象。在 Graph 类中由多种类型的节点，如图 2.4 任务流图示例图所示每个节点之间双向连接。为每个 TaskInst 节点配置 input_data 和 output_data 两个字典，为每个 DataInst 节点配置 producer 和 consumer 两个字典，在 Graph 类中读取任务实例和数据实例，并根据在任务用例文件中任务和数据之间关系，分别将任务实例和数据实例填入不同节点的各自的两个字典中，并且将数据和任务之间依赖关系作为边设为 edge_list。在读取 excel 的过程中构造图的结构，并不断地将 TaskInst 和 DataInst 以及结合他们之间的映射关系加入 Graph 对象中。

TaskGraphMgnt 模块将所有任务全部解析并读取到 Graph 对象中后，当仿真系统发出仿真执行的指令后，TaskGraphMgnt 模块按照任务顺序将任务发送到调度器模块。

调度器模块负责整体任务调度，观察并接收 TaskGraph Manager 中就绪任务，根据任务信息将任务分到不同子模块去分配资源，并分配到具体硬件上去执行任务，同时接收硬件返回的 response，将任务信息同步，释放资源并触发下一级任务。调度器模块还负责管理 device 资源和 memory 资源。调度器模块分为任务分配 TOP 模块、任务调度 SCH 模块、资源管理 RES 模块几个子模块。调度模块具体结构图如图 2.5 所示。

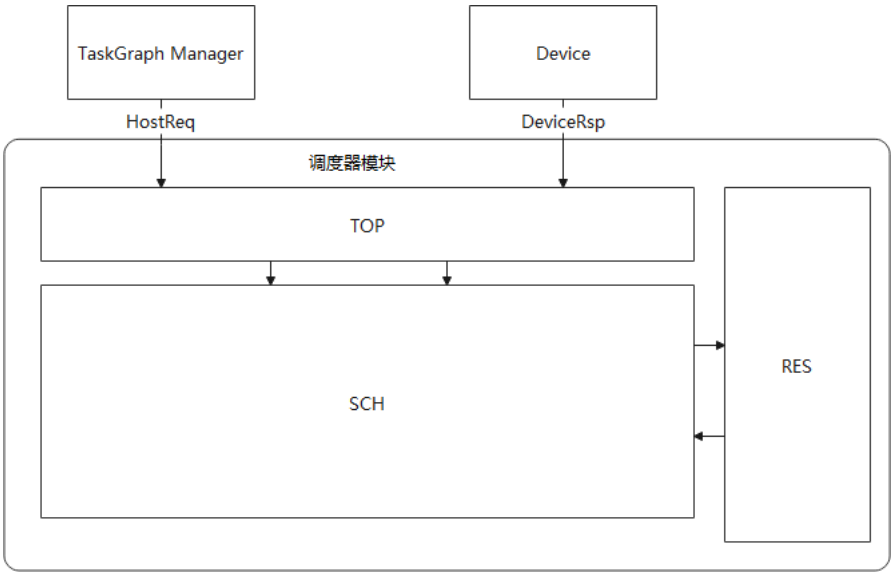

图 2.5　调度器模块结构图

TOP 模块负责接收任务以及硬件模型以及内部子模块发来的响应。通过任务信息，以及响应类别，通过不同的线程将不同类别的任务分别分到各个子模块处理。

SCH 模块负责给任务分配资源（包括硬件模型资源 CU 以及硬件 IP、内存资源等等），根据任务携带的信息将任务分配给之前分配的硬件 IP 进行具体的处理。

RES 模块负责管理各个核的私有内存（包括 L1D、PL2）；公共内存资源（如 SL2 等）；管理 Device 资源，每个 ALA 支持 32 个 Device，每个 D evice 分为多个 CU，CU 为每个硬件模型外的任务等待队列；管理硬件资源（如 DMA、HAC 等等）。其他模块通过调用这个模块的方法来申请和释放资源。

调度器模块通过 Schdule 模块读取 TaskGraphMgnt 对象里的 Graph 任务图对象读取出来。Graph 对象形式是有向无环图类型，Schdule 模块先遍历有向无环图，找到 DAG 图中没有入度的节点即为第一个任务或数据，将实例输入到调度器模块，由 TOP 模块接收判断类型，根据信息类型转发到 SCH 模块。

TOP 模块负责接收外部模块发送过来的任务图中实例信息以及硬件模型处理任务完成后发送回来的响应，按照实例或者响应分开处理。RES 模块负责管理各个核的私有内存（包括 L1D、PL2）；公共内存资源（如 SL2 等）；管理 Device 资源。SCH 模块负责向 RES 模块申请管理的硬件资源和内存资源。并将任务信息或数据信息发送到申请好的硬件 IP 中进行处理。

在调度器模块发送任务到具体硬件上执行后，当发生数据搬运时，模块与模块之间的数据交互是通过总线互联模块进行的。

在仿真平台中，在不同模块之间通过总线进行通讯，而总线在不同模块上的端口则是 Port 模块。针对端口设计的可扩展性，我们以 Moudule Port Base 为基础，采用 Master/Slave 方式。在仿真平台中，所有端口采用双向绑定机制，即在 Master 端可以调用 Slave 端的方法.

仿真平台中现在主要有两种模块间通讯接口，一种为硬件模型模块间的接口：Virtual Master Port 和 Virtual Slave Port，另一种是总线内部子模块之间的流水线通讯接口 VBusPipeLineMasterPort 与 VBusPipeLineSlavePort。

Master Device 在硬件实例化的时候创建对应的 VirtualMasterPort，Slave Device 在硬件实例化的时候创建对应的 VirtualSlavePort。Master Device 一般是 Processor 模型，硬件 IP 实例化时通过调用 ProcessorBase 的方法 add_port 创建 Port 端口。Slave Device 一般是 Memory 模块。在平台构建时，VirtualMasterPort 和 VirtualSlavePort 通过硬件平台配置表上的信息进行绑定。这两种端口都由 PortMgnt 进行管理，

在仿真平台硬件模块中，所有数据信息的传输都是通过总线模块进行，所有

的数据传输都是通过 Port 模块拆分封装成 Flit 格式的数据包经过总线模块从一个硬件模块发送到另一个硬件模块。对应不同模块之间总线模块是不可视的，能接触的只有 Port，当 Master 设备发送数据到对应 Port 上的操作如图 MasterPort-Mgnt 实现流程图所示。

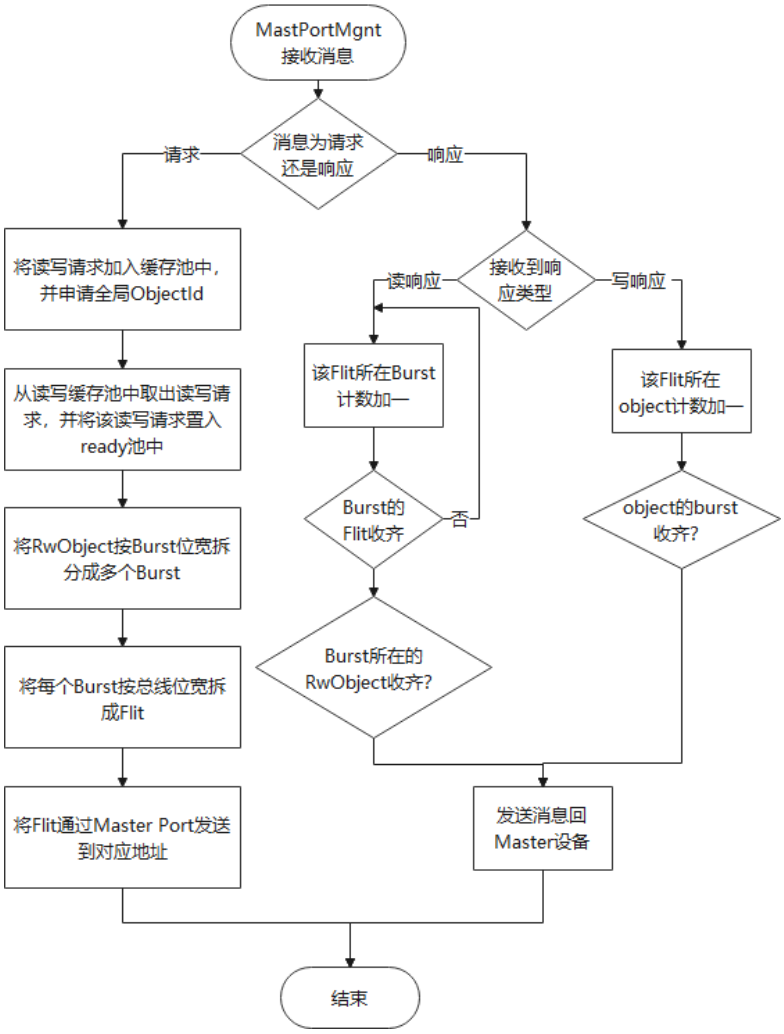

图 2.6　MasterPortMgnt 实现流程图

总线处理模块的主要功能是地址译码、优先级仲裁以及模拟每个 flit 的传输。BUS 模块中一共有四种通道，分为写通道、读地址通道、写响应通道和读数据通道。每个通道都有一套独立的译码模块和仲裁模块，各个模块之间通过端口连接，使用 Fifo 实现数据消息的传输。总线模块主要分为总线输入模块 InputStage、总线译码模块、总线仲裁模块以及总线输出模块。

InputStage 模块在整个 BUS 模块中的作用是作为外部 Master Device 与内部沟通的桥梁。主要的工作流程是接收 Master Device 发送来的 flit，并将 flit 发送到对应的译码模块；以及将来自仲裁模块的 flit 发送回 Master Device。

OutputStage 模块在整个 BUS 模块中是模块内部与 Slave 设备沟通的桥梁。主要的工作的流程是接收来自总线模块内部发送来的 flit，并将 flit 发送到对应的

Slave 设备的 Port 模块；或者将 Slave 设备 Port 返回的响应发送到对应的 Decoder 模块上。

BUS 模块的每个通道都有独立的译码模块，但读数据通道和写响应通道的译码模块都是对响应的译码，处理逻辑相同，我们将读数据通道和写响应通道的译码器模块合二为一。

Arbiter 模块的作用是对同一通道来自不同 Decoder 模块发送过来的 flit 进行传输顺序的仲裁。现在仲裁模块的算法采用的轮询算法，但在轮询算法的基础上回尽可能的保证同一个 Burst 的 flit 能够连续发送。BUS 模块的的类图如图 2.8 所示：

2.6　本章小结

本章从仿真平台建模开始介绍了 ESL 建模的概念，并且详细介绍了我们在实际建模过程中使用的仿真框架 SimPy。接下来介绍了在设计空间探索中使用的遗传算法工具库 Geatpy，介绍了其基本结构以及使用，同时说明了在我们实际设计空间探索过程中由于目标值有多个，介绍了多目标优化的基本知识以及使用遗传算法的原因。最后介绍了我们所使用的持久化数据的 SQLite 数据库。

在了解了相关技术之后，在第三章我们将阐述该仿真平台及设计空间探索的需求分析。

第 3 章　需 求 分 析

本章基于系统级仿真建模的基本需求，在此基础上，按照后续设计空间探索的需求以及保证仿真平台性能的非功能需求，对仿真平台以及设计空间探索流程的需求分析进行了详细的阐述。需求分析是非常关键以及重要的一步，为后续应用的具体实现有着指导作用。

3.1　系统需求分析简介

基于 Simpy 的仿真平台是一个轻量级的系统级仿真平台，需要完成对原有业务行为的仿真，可以适配现有的复杂的基于 SystemC 的 SAE 仿真平台的输入（例如任务用例、硬件配置文件等等）。这个仿真平台只是针对于部门内部员工进行使用，我们需要输出在用例仿真的时候一些硬件平台的具体信息（如核一段时间的利用率、内存碎片等等）和任务执行的具体细节。仿真平台也需要支撑后续的设计空间探索流程的实现。为了设计空间探索流程能够更简易使用，我们需要将整个流程做成可配置的。

在平台设计的时候，设计人员对各个硬件模块有相应的功能需求。在我设计的模块中，主要包含 GeneralFifo 模块、Processor 模块以及 Memory 模块。这几个模块的功能需求如表 3.1 所示：

表 3.1　硬件模块功能需求列表

模块	需求说明
GeneralFifo 模块	能够完成消息队列基础功能及事件触发
Processor 模块	Processor 模块各种模型的功能正确、与其他模块成功交互
Memory 模块	数据存取时序正确且数据不丢失、与其他模块成功交互

在硬件模块设计过程中，设计人员需要完成对各个模块的功能需求的实现，保证模型在实例化时能够成功搭建且保证功能正确，并且实现各个模块的执行流程细节在日志中可视。

在平台使用过程中，针对不同员工工作性质的不同，他们对仿真平台的需求也不同。结合上述功能和针对不同用户进行分析，得出用户需求列表 3.2。在这里我们将只使用仿真平台运行用例的员工对应为普通用户 1，需要配置硬件平台执行用例的员工对应为高级用户，需要执行设计空间探索流程的员工对应为普通用户 2。

表 3.2　用户需求列表

ID	需求说明	需求来源
UR1	执行用例文件	普通用户 1、高级用户
UR1.1	执行过程的异常报错	普通用户 1、高级用户
UR1.2	修改硬件平台配置	高级用户
UR1.3	硬件模型扩展开发	高级用户
UR2	读取用例执行结果	普通用户 1、高级用户
UR3	执行设计空间探索流程	普通用户 2
UR3.1	配置设计空间	普通用户 2
UR3.2	配置进化算法参数	普通用户 2
UR4	设计空间探索结果对比结果	普通用户 2

3.2　功能性需求

功能需求主要用来列举用户对产品所预期的功能和服务。本产品主要的功能是进行系统级业务行为仿真以及对整个系统架构的一些方面进行自动化的设计空间探索，因此需要通过需求分析对不同用户所需要的功能进行分析和阐述。

首先，我们根据表 3.1 依次从 GeneralFifo 模块、Processor 模块和 Memory 模块三个方面对这三个模块的功能需求进行详细的阐述：

3.2.1　GeneralFifo 模块功能需求

GeneralFifo 模块在仿真平台中的功能是一个可以即时触发 event 的先入先出消息队列。它需要满足接口可见且队列存取 event 可被其他模块触发的功能。该模块的用例图如下图 3.1 所示：

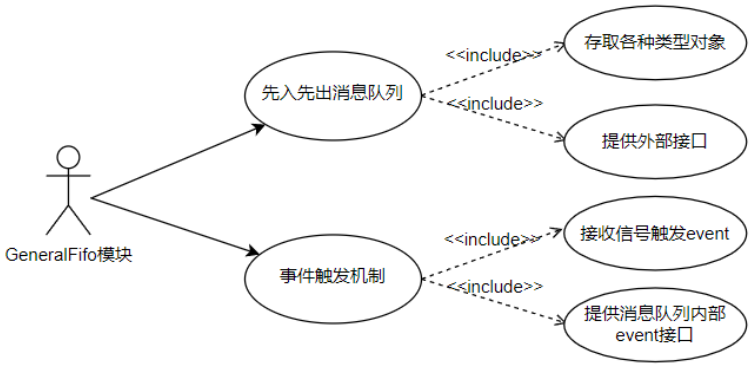

图 3.1　GeneralFifo 模块用例图

图 3.1 列举了 GeneralFifo 模块的功能需求。GeneralFifo 作为一个先入先出队列需要满足先入先出队列的所有功能，能够存储各种类型的对象，提供队列内部的一些基础接口如:队列深度、存对象方法以及取对象方法等等。其次，GeneralFifo

为了满足仿真平台能够实现各个模块之间即时的消息传递，GeneralFifo 需要实现当队列接收到消息时，便触发相应的事件的功能，并且 GeneralFifo 模块需要提供模块内部的事件相关接口，以便其他模块可以接受到队列内部的事件触发。

3.2.2　Processor 模块功能需求

Process 模块包括 DSP、HAC 以及 DMA 三种不同的硬件模型，Processor 模块需要实现各个硬件模型各自的功能需求以及硬件模块与其他模块之间的交互功能。Processor 模块的用例图如下图 3.2 所示：

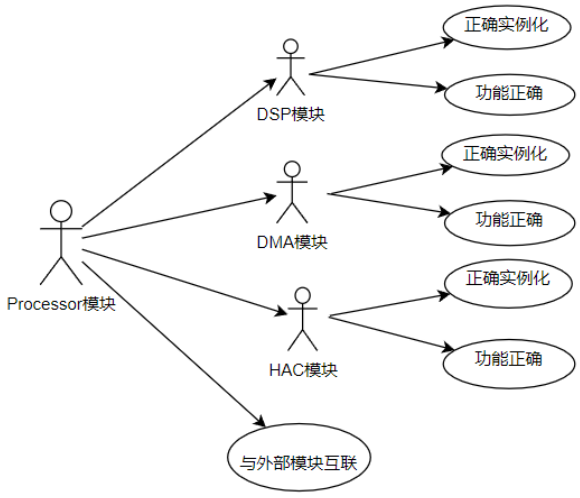

图 3.2　Processor 模块用例图

图 3.2 列举了 Processor 模块的整体需求，包括 DSP、HAC 和 DMA 三种硬件模型的功能需求以及所有硬件模型与其他模块的交互功能需求，包含与 Memory 模块和调度器模块的交互。下面就这几个方面分别来详细介绍 Processor 模块的功能需求。

DSP 模块需要实现在仿真平台构建时，根据硬件平台配置文件中的信息正确实例化，保证具体 DSP 实例能够通过总线连接到整个仿真平台中且 DSP 实例的配置信息正确。同时 DSP 模块在实例化后需要实现自己的硬件功能：接收并处理来自 DMA 实例或者调度器模块的任务实例，并根据任务实例中提取出的信息正确执行任务，并发送消息到下一级硬件实例中。

HAC 模块与其他 Peocessor 模块的一样需要在仿真平台构建时能够正确实例化。同时 HAC 模块在实例化后需要实现自己的硬件功能，HAC 模块作为一个用于处理某一种特定的任务的硬件模块，其需要接收并处理来自调度器的任务实例。接收需要根据任务信息从特定地址的内存块取出数据，然后以一定的时延处理任务，并在任务处理结束后将数据存储到特定地址的内存中。在这个过程中需要保证任务处理时延正确、搬入搬出的数据正确。

DMA 模块同样需要在仿真平台实例化的时候能够正确实例化。与此同时，

DMA 模块需要完成其模块的功能，即与 Memory 模块进行交互，进行数据的搬入搬出，保证在数据的搬入搬出的过程中数据的正确。并且 DMA 模块能够和其他模块进行消息交互，与 DSP 模块之间形成任务的执行流水线。

Processor 模块需要能与其他模块进行消息以及数据的交互，需要保证消息传递的及时性以及正确性，保证数据传输时源地址和目的地址的正确以及数据本身的正确。

3.2.3　Memory 模块功能需求

Memory 模块需要实现能够成功实例化且模块功能正确，以及需要保证能够和其他模块进行消息交互以及数据传输的功能。Memory 模块的用例图如下图 3.3 所示：

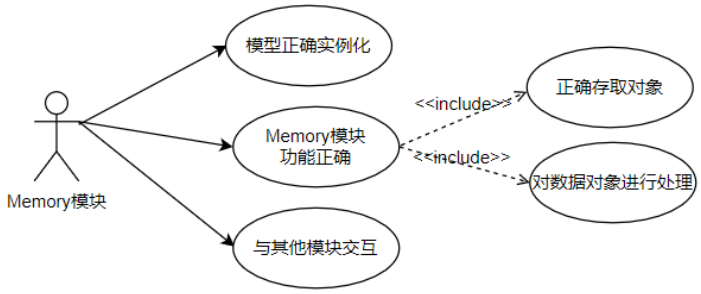

图 3.3　Memory 模块用例图

如图 3.3 所示，Memory 模块需要实现模型能够正确实例化、Memory 模块基础功能以及外部模块之间消息交互的功能。这几部分需求的具体内容如下：

1、Memory 模块能够在硬件平台搭建时根据硬件平台配置文件的信息去实例化，并通过模块上实例化的 Port 与总线连接，从而与整个硬件平台连接在一起。

2、Memory 模块需要保证其在实例化后，Memory 实例的功能正确：Memory 模块能够接收来自其他模块的存取对象的请求，并根据请求的类型以及具体任务信息执行任务，在执行数据的存取时保证数据以及存取地址的正确性。

3、Memory 模块在执行数据存取的时候需要通过总线与其他内存模块进行数据的交互、通过 GeneralFifo 消息队列与调度器模块中的内存管理模块、DMA 实例以及 HAC 实例进行消息的交互，通知数据存取任务的开始以及结束。

在介绍完仿真平台中硬件平台的功能需求之后，我们从仿真平台的使用用户的角度来介绍仿真平台的功能性需求。如表 3.2 所示，仿真平台使用所涉及到的需求用户分为普通用户 1、高级用户和普通用户 2 三类。下面将对这三种用户的需求进行详细的分析。

3.2.4　普通用户 1 需求

普通用户 1 即是简单使用仿真平台进行业务仿真的用户。普通用户 1 的用例图如下图 3.4 所示：

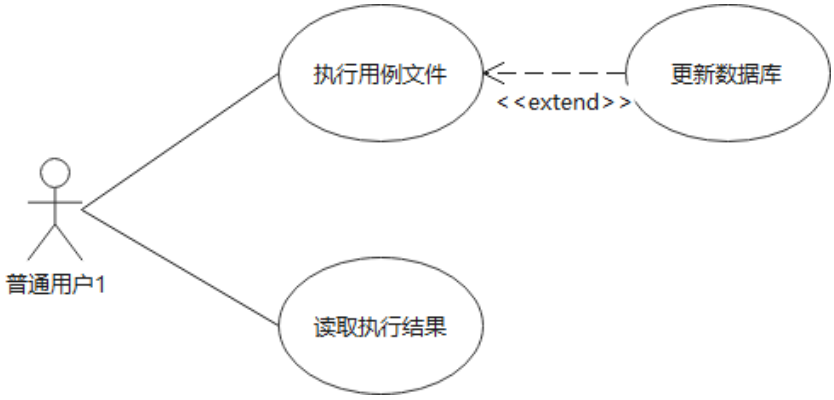

图 3.4　普通用户 1 用例图

图 3.4 列举了普通用户 1 对仿真平台的执行用例文件功能、读取执行结果功能以及仿真结果及仿真执行信息数据存储的需求，接下来对每个功能的具体需求进行阐述。普通用户 1 的执行用例文件用例图如图 3.5 所示：

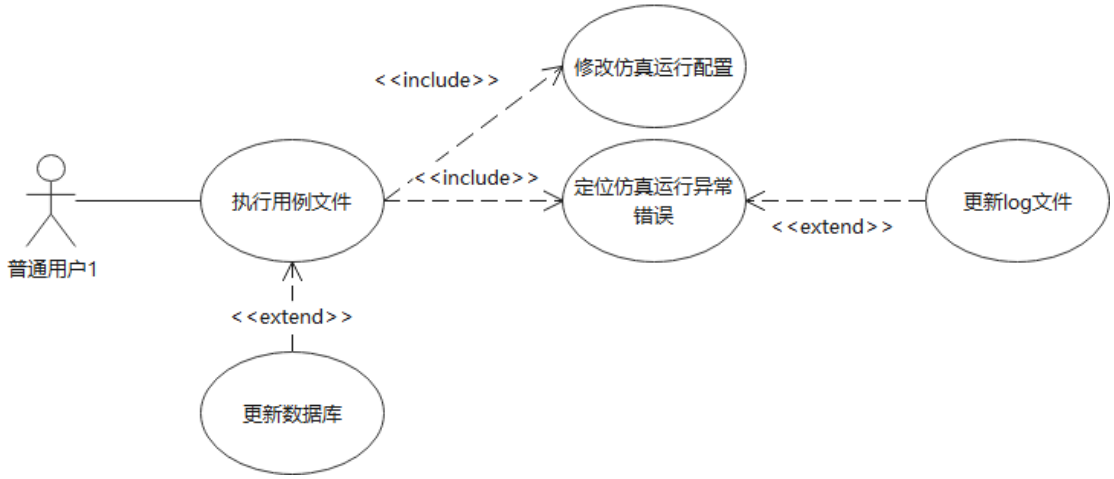

图 3.5　普通用户 1 执行用例文件用例图

如图 3.5 执行用例文件用例图所示。执行用例文件功能分为修改仿真运行配置、定位仿真运行异常错误和查看用例执行结果信息几个子功能。执行用例文件功能及其子功能的需求具体内容如下：

1、仿真结果时序正确。仿真时序正确要求每个任务的执行时间以及整体任务的执行调度顺序与输入的任务用例文件中任务信息以及任务间的依赖关系一致。仿真执行的输入是用例 Excel 文件，用例文件是将真实的任务在单板上的执行流程、执行信息以及执行结果体现在文件中，通过解析文件信息，将任务真实的具体信息及依赖关系作为仿真输入，通过任务调度平台的正确调度使得整个仿真最终得以正确执行。

2、仿真运行可配置。因为我们需要的执行用例主要用于基带芯片上，所以用例可能存在多小区多信道的用例同时执行或者不同的用例同时混合执行，所以我们需要在仿真运行前配置用例执行次数、仿真执行的用例文件名称、硬件配置文件名称以及 log 文件和数据库文件的位置，这样就能适配不同情况下的仿真。用户可以仅仅、修改执行脚本中的数据即可对整个仿真执行过程进行修改，从而得出用户所需的不同的仿真结果。

3、定位仿真执行异常错误。任务用例在仿真过程中可能会遇到各种情况的错误，有些是用例文件的任务信息错误，有些是任务执行时的错误。我们需要能够迅速定位到错误位置，我们通过任务执行过程中细致的 log 记录，可以迅速定位到任务执行哪一级出现错误以及可以查询到各个模块之间数据传输在总线上哪一级出现问题，可以迅速地定位到问题的产生地方以及解决问题。

以上详细的介绍了普通用户 1 对仿真平台执行仿真用例的具体需求，包括在仿真过程中通过数据库将详细的任务执行信息记录下来。

用户执行用例仿真的最终需求还是为了得到用例仿真结果信息。我们在仿真过程中通过数据库记录硬件平台的具体信息（如核一段时间的利用率、内存碎片等等）和任务执行的具体细节（任务的提交时间、开始时间、结束时间等等）。用户可以通过访问数据库文件来查看用例执行结果信息，从而得出自己所需要的的信息。数据库文件需要记录多次的仿真结果，不同的仿真结果以 SimId 为标志区分，并记录仿真执行的平台、仿真的时钟频率、执行的用例文件等等该次仿真的具体信息。这样我们可以将我们需要的执行的多次用例通过脚本依次执行，最后也可以查看到每次仿真执行后的具体信息。

3.2.5　高级用户需求

高级用户即是那些深度使用和开发该仿真平台的用户，除了普通用户 1 的需求，他们还需要对仿真平台的配置修改需求。高级用户的用例图如下图 3.6 所示：

图 3.6 列举了高级用户仿真运行、读取仿真运行结果信息、修改硬件平台配置和硬件模型扩展开发的需求。接下来我们对高级用户比普通用户 1 多出的两种功能需求进行详细的阐述。

因为在芯片设计过程中，高级用户需要尝试手动修改仿真平台中硬件平台中不同的硬件配置去运行仿真，因此我们的硬件配置需要高级用户能够很方便能够去修改。仿真平台中硬件平台的构建是通过解析输入的硬件配置文件来进行的，仿真平台输入的硬件配置文件类型包括 Excel 表及 XML 文件，用户可以通过修改硬件配置文件中的硬件配置信息来达到他们对硬件配置进行修改的需求。

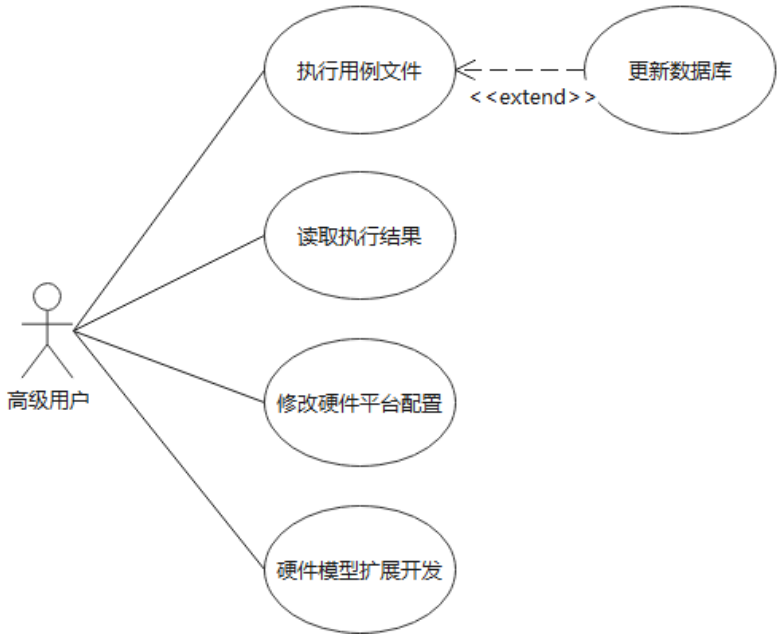

图 3.6　高级用户用例图

高级用户包括仿真平台的开发人员，芯片上的硬件不是统一不变，不同的硬件行为不同，因此当出现不同的硬件时，我们需要对新的硬件进行建模。但现有的芯片中的硬件类型大致就几种，我们基于硬件类型对各种硬件进行区分，为了这种建模操作更加方便，我们统一构建了各种类型硬件的基类，建模时用户只需基于这几种基类进行扩展开发即可。这样高级用户就可以很方便的仿真平台进行二次开发。

3.2.6　普通用户 2 需求

普通用户 2 即是那些使用设计空间探索流程的开发人员，他们只关注设计空间探索流程的输入输出以及一些参数设置。普通用户 2 的用例图如下图 3.7 所示：

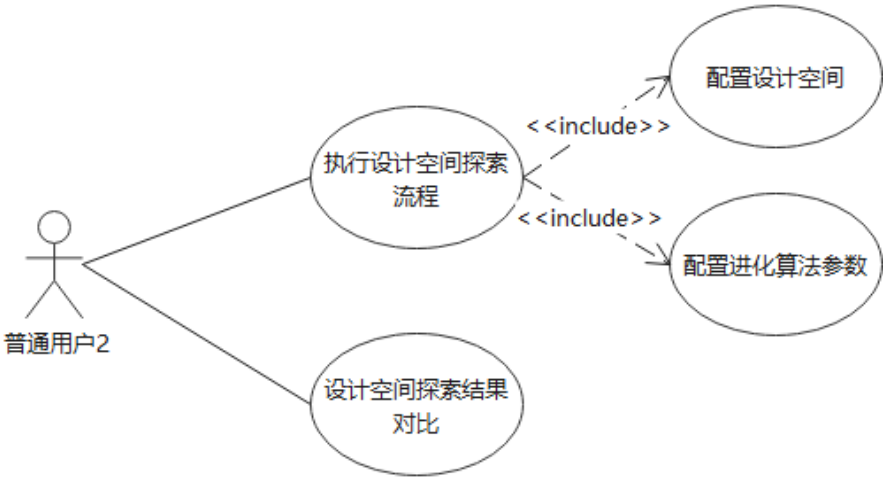

图 3.7　普通用户 2 用例图

由图 3.7 可见，普通用户 2 关注设计空间探索流程的输入输出以及流程中一些参数设置，对设计空间探索流程中具体执行过程并不关心。设计空间探索流程的输入是目标值及其组成的设计空间和仿真模型。在流程中可以调整一些进化算法的参数，如进化算法代数以及交叉变异的概率等等。我们通过在设计空间探索流程执行脚本中设置相应的参数，就可以方便的调整设计空间探索流程中遗传算法模块相应的参数，去得到对应的结果。最后能输出一个种群的帕累托最优解集。

普通用户 2 可以通过设置的结果分析脚本使最后输出的脚本与原设计参数下执行结果进行对比，更加方便可视化的得出用户所需要的结论。以便用户得出所需的设计参数或者进行下一轮的设计空间探索流程，接下来对系统的非功能需求进行阐述。

3.3　非功能需求

仿真平台及设计空间流程整个系统在满足功能需求的基础上，还要根据系统的实际情况，满足非功能需求。对于整个系统而言，非功能需求主要体现在可扩展性、可维护性和性能需求上。这些非功能需求的作用对项目来说也是至关重要的，只有同时满足功能性需求和非功能性需求，该应用才能在项目中顺利地发挥其作用。

3.3.1　可扩展性需求

系统主体的仿真平台整体逻辑比较复杂，涉及 5 大模块，在各个模块内部有很多部分功能类似，代码复杂，有很多模块需要复用。同时由于硬件设计的变化，我们需要对硬件模块在后续开发中要不断进行扩展，整个项目的可扩展性要求高，所以为了实现可扩展性需求，我们将各大模块代码模块化处理，在外部设计接口，统一通过接口进行交互。模块内部功能相似的模块统一设计基类，通过继承基类进行扩展开发。减少代码之间的循环依赖和降低代码之间的耦合，从而提高系统的开发效率以及后续的扩展功能，使得系统满足可扩展性需求。

3.3.2　可维护性需求

在仿真建模以及仿真平台的搭建过程中，系统功能的实现并不能一劳永逸，在新的硬件模块的增加后，仿真平台的功能以及性能是否满足要求，需要后续的不断对其进行维护，才能保证仿真平台的稳定性和可用性。为了实现仿真平台的可维护性，我们通过将仿真平台功能组件化，以保证后续模型的添加在功能实现方面可以实现复用，更好的兼容现在的平台。降低了模型的开发难度以及方便了

后续开发人员对功能和组件的维护，提高了开发效率。

同时，因为仿真平台的输入也一直在变，我们设置了一个迭代平台，可以每天在服务器上进行不同用例的自动化仿真，并给出仿真结果，降低了开发人员的人工成本。

3.3.3　性能需求

仿真平台主要是为了进行业务行为仿真，设计空间探索的目的是为了得到在各个方面结果都表现优秀的设计方案解集。所以评价仿真平台性能的指标包括仿真结果的时序准确以及仿真执行的速度，评价设计空间探索流程的性能是观察其输出结果是否在各个方面都体现了优秀的性能以及设计空间探索流程的效率。在这里我们也将这些作为性能需求的评价指标。

首先，在仿真平台方面，我们将输入的用例文件解析为以任务和数据为节点的有向无环图，这保证了任务执行的顺序，任务执行时间的准确则体现了硬件模型行为的准确。为了仿真能够更快的执行，我们简化了硬件模型的行为，以软件行为替代硬件行为，在内存模型上我们简化了数据的格式，将真实的数据存取过程只是体现在仿真时延的增加，而不是真实的进行存取。且采用了 Simpy 仿真框架，从而实现仿真速度的提升。

其次，在设计空间探索方面，我们采用进化算法，并通过进化算法的选择和具体的算法设置保证结果是全局最优而不会陷入局部最优。我们同时也需要保证设计空间探索流程的速度，保证整个设计空间探索流程能在很短时间闭环，以方便设计人员能在设计空间探索流程结束后很快的对设计探索的结果进行分析，得出所需要的设计架构。由于我们采用进化算法，我们在每一代中，对于相应的参数，我们都要通过仿真得到对应的仿真结果并且提取出我们需要的结果。因此我们通过在设计空间探索过程中将仿真模型通过机器学习简化为仿真预测模型，这样能大大减少每一次的对对应设计参数得出结果的时间，从而达到极大的提高设计空间探索流程的效率的需求。性能需求表如下表 3.3 所示：

表 3.3　性能需求表

模块	需求项	需求值
仿真平台构建模块	仿真硬件平台搭建成功的时间	<=5s
仿真执行模块	单次单小区仿真执行的时间	<=15s
仿真执行模块	仿真结果时序与用例一致	true
设计空间探索模块	单次一代遗传算法执行时间	<=500ms
设计空间探索模块	设计空间探索结果全局最优	true

3.4　本章小结

本章从实际场景出发，根据业务仿真的真实场景和设计空间探索流程的真实场景，详细分析了整个系统的功能需求和非功能需求，在对产品的需求进行了简介之后，给出了用户需求列表。在功能需求中，根据不同用户对系统的需求，对功能需求进行了详细的阐述，描述了用例文件执行、查看仿真结果信息、执行设计空间探索流程和得到设计空间探索结果对比等功能需求。接着根据系统的实际情况，从可扩展性、可维护性和性能需求几个方面来详细阐述系统的非功能需求。这一章对仿真平台及设计空间探索流程的需求进行了比较明确地阐述，为后续的系统概要设计打下了基础。

第 4 章　概 要 设 计

本章是在需求分析的基础上对在整个仿真平台我所负责设计实现的硬件平台相关模块及设计空间探索模块的设计进行简要的阐述。在此章节中，我们可以清晰的了解到硬件平台构建模块、硬件模型生成模块 (Processor 模块和 Memory 模块)、GeneralFifo 模块设计及设计空间探索的流程，为具体实现打好基础。

4.1　系统整体结构

整个系统整体结构主要包括两个部分：仿真平台部分，包括硬件平台构建、调度器模块、任务图解析及仿真部分；设计空间探索部分，包括通过脚本迭代生成训练集、训练预测模型及运用遗传算法实现设计空间探索。系统的整体架构图如图 4.1 所示：

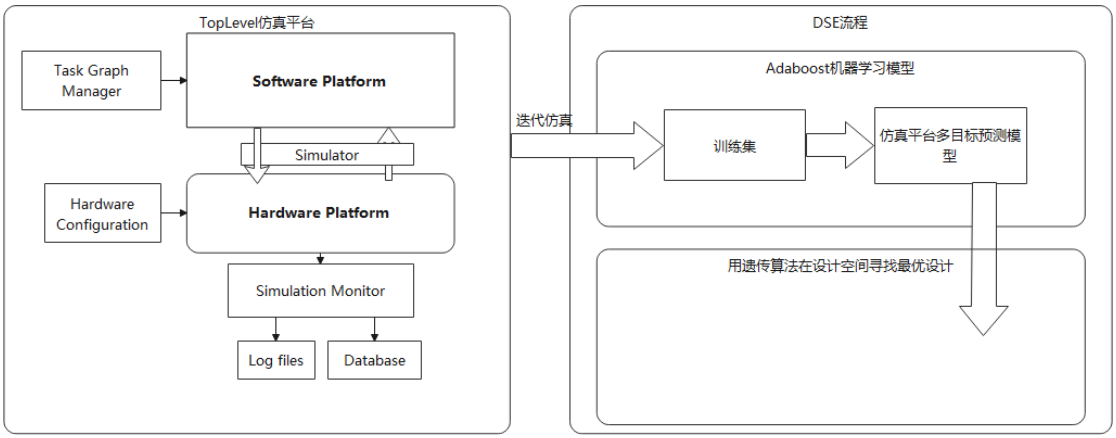

图 4.1　系统整体框架图

如系统整体框架图可以看出，该系统是一个以轻量级仿真平台为基础，通过遗传算法对比较大的设计空间进行探索，寻找在实际 SoC 设计时的最优设计。我们首先设计实现轻量级仿真平台，轻量级仿真平台通过读取硬件配置表实例化硬件平台。随后仿真平台读取输入的任务图，由 TaskGraphMgnt 模块管理任务实例并通过触发关系发送任务实例到调度器模块。调度器模块接收任务，然后申请硬件资源并将任务实例调度分发到各个硬件模块上执行任务，在整个任务调度及实现过程中输出日志文件和数据库文件。

通过在仿真平台上快速进行仿真，得出相对于设计空间较小的训练集文件。训练集文件中包括硬件平台配置，如总线位宽、一个组中 DSP 数量、内存大小等等。还包括仿真输出的数据库中提取的仿真信息，如仿真时延、核利用率、核一致性等等。

然后基于生成的训练集文件，通过现有的机器学习 Adaboost 模型[?]对训练

集文件训练出仿真平台多目标预测模型[?]，最后我们运用遗传算法对设计空间进行迭代剪枝，最后得出最优的设计种群。

系统主要包括仿真平台和设计空间探索流程两个部分，在仿真平台设计部分，我主要负责硬件平台部分的硬件平台构建模块、GeneralFifo 模块以及 Processor 模块的是实现，这部分主要实现各种硬件模块的建模以及整个硬件平台的搭建。在设计空间探索模块，我主要负责分析并设计设计空间的参数以及参数范围，通过设计脚本文件实现仿真平台进行迭代仿真从而得到训练集文件，并使用训练集文件训练得到简易的仿真平台训练模型。

所以下面就我所负责实现的仿真平台中的硬件平台模块、设计空间探索模块的训练集生成和预测模型模块以及仿真结果数据库部分分别来简要介绍各自模块的设计。

4.2　仿真平台硬件模型生成模块概要设计

TopLevel 仿真平台主要实现硬件平台各个模块的实例化，任务图的读取以及执行。整个仿真平台模块主要分为调度器模块、Memory 模块、总线模块、Processor 模块、用于各模块之间数据以及消息传输的 GeneralFifo 模块。仿真平台中硬件平台的整体架构图如图 4.2 所示：

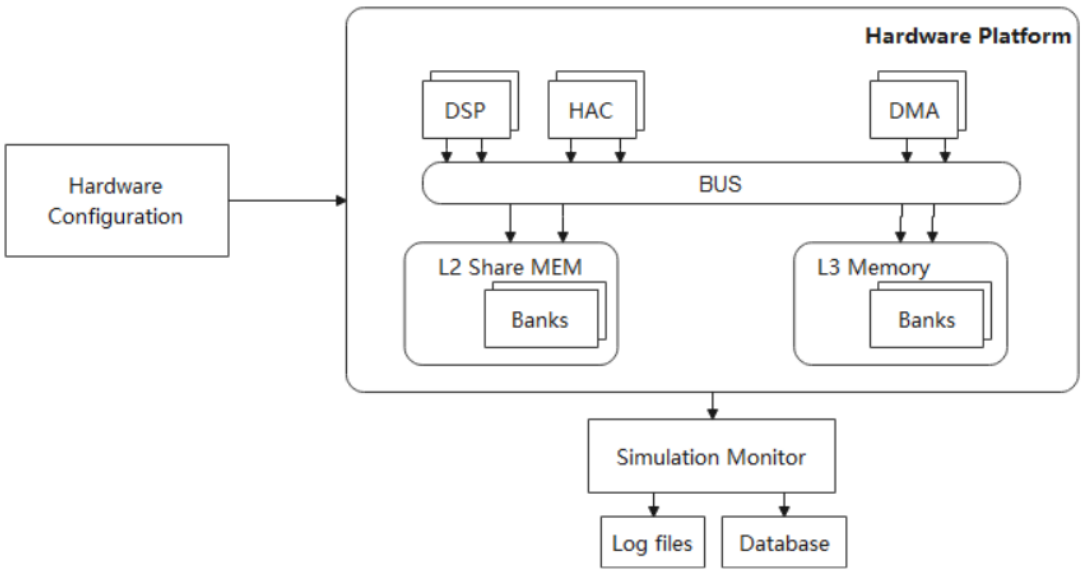

图 4.2　硬件平台架构图

仿真平台中主要包括三个部分 Processor 模块（包括 DSP、HAC 以及 DMA）、总线互联模块以及 Memory 模块。硬件平台在构建过程中读取硬件配置文件，根据硬件配置文件中的配置信息将各个硬件模块挂接在总线之上，从而得到整个硬件平台。硬件模型在执行任务的同时通过任务的开始和结束时间的触发去记

录硬件信息以及在任务执行时记录任务执行信息输出到数据库之中，同时在执行的节点输出 Log 信息，便于在仿真运行出错的时候能能够快速地寻找的问题所在。

下文主要对仿真平台中设计实现的的硬件平台构建模块、硬件模型生成模块 (Processor 模块和 Memory 模块) 以及 GeneralFifo 模块设计进行简要的介绍。

4.2.1　硬件平台构建模块概要设计

平台开发人员根据仿真平台用户的需求进行建模，提供在系统仿真过程中所需要的各种硬件模型，用户根据需求构建系统仿真所需的硬件平台，仿真平台根据用户配置的信息实例化各个硬件模块并构建成相应的硬件平台。在构建硬件平台过程时, 用户通过修改硬件平台配置文件中的硬件模块实例信息以及各个实例之间的连接方式，其中包括每个硬件模块的配置信息和总线的路由映射。硬件平台的构建过程由下图 4.3 所示。

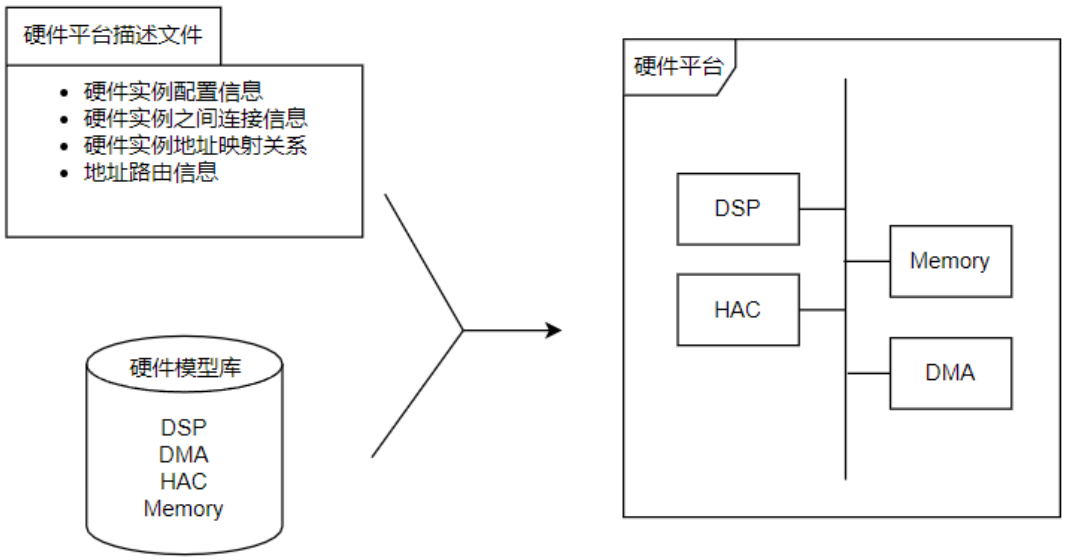

图 4.3　硬件平台生成图

在仿真平台中，整个硬件系统的构成是一个树状结构的层次化架构，在仿真平台中硬件平台的构建过程是对该树状结构进行深度优先遍历，所有硬件模型都基于 GeneralModule 基类实例化。在硬件平台构建过程中根据硬件平台配置文件中的配置信息，会最先实例化最底层叶子节点的各个具体的硬件模型，再去构建上层的分区组件直至整个硬件平台实例化完成。在此过程中若构建出现问题就会报错通知用户，说明硬件配置文件中信息配置错误，用户需要修改相应硬件配置参数重新搭建。硬件平台构建的活动图如图 4.4 所示：

如图 4.4 所示，硬件平台在构建过程中仿真平台需要从硬件模型库根据硬件配置信息中实例化具体的硬件模型。平台开发人员需要对各种硬件模型进行建

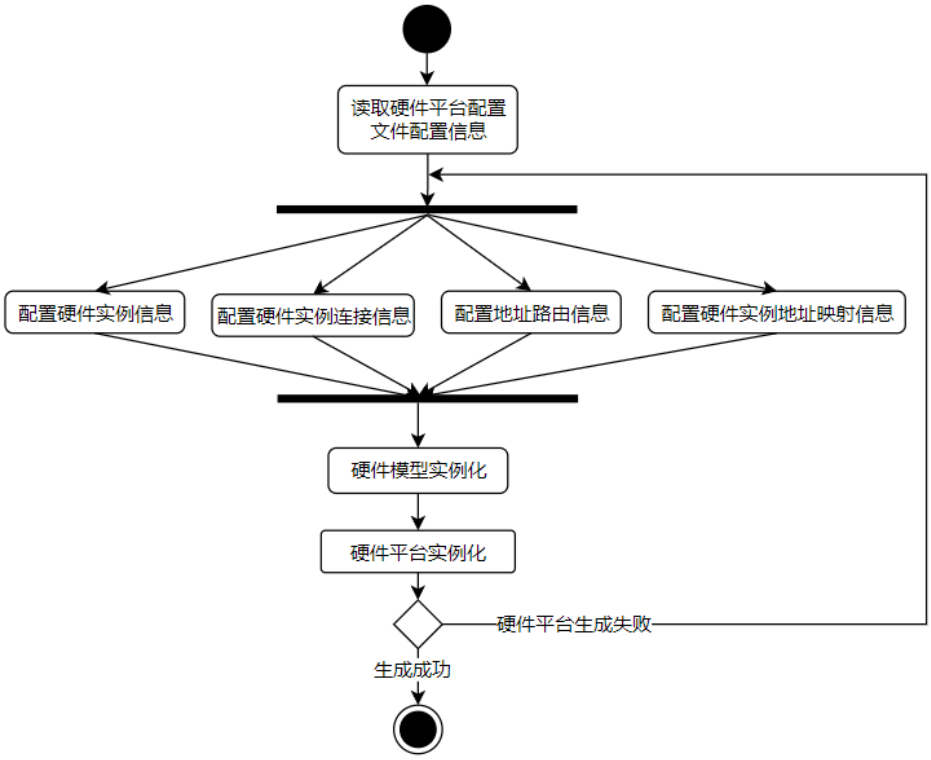

图 4.4　硬件平台生成活动图

模，包括 DSP 模型、DMA 模型以及 Memory 模型等等。在构建过程中，系统通过硬件平台配置文件中的硬件模型配置信息去修改硬件模型库中的硬件模型基类中的可配置参数去实例化具体的仿真平台所需要的硬件模型。在将建模好的硬件模型基类增加到硬件模型库中之前，通过对硬件模型的单点测试去修改和完善在设计阶段没有考虑完善的功能需求。同时由于在不同的业务背景下，仿真平台需要不同的硬件模型，我们在设计过程需要考虑到硬件模型的可扩展性。一些种类的硬件模型的具体功能虽然不同，但整体模型基础功能大致相同，所以我们在后续开发其他硬件模型时，只需基于现有的硬件模型基类去开发，而不需要从零再进行模型建模开发。硬件模型库中涉及到多种硬件模型，其中包括处理器硬件模型、调度器模型、存储器模型以及互联互通模型等。其中 Processor 模块模型、Memory 模块模型的概要设计将在后续几个小节中介绍。

4.2.2　GeneralFifo 模块概要设计

GeneralFifo 是为了满足模型处理流程需求而开发的一种通用先入先出队列（First in first out，Fifo）机制[?]，可以实现建模过程中各种前后级同步，反压等需求。它主要有以下几个方面的功能：

1. 对连续的数据流进行缓存，防止在存储操作的时候发生数据丢失；
2. 数据集中起来进行进栈和存储，可避免频繁的总线操作，减轻 CPU 的负担；
3. 允许系统进行 DMA 操作，提高数据的传输速度。

4. 作为消息队列，支撑仿真平台各个模块之间消息交互，提供事件触发机制。

在该仿真平台中，GeneralFifo 模块更多的体现的是第四条作为消息队列的功能。在硬件设计中，Fifo 一般设计为“乒乓”存储方式，在硬件仿真设计中，我们采用以先入先出队列为存储对象的存储器，GeneralFifo 还集成了 SimPy 里的 event 机制，实现仿真平台中模块与模块之间的消息交互需求。GeneralFifo 相当于 SimPy 中另一种容器的实现。GeneralFifo 模块作为一个先入先出队列，需要实现队列的基本功能，完成对于对象的存取，以及提供队列中基本信息，如队列深度，当前队中元素个数等等。

GeneralFifo 模块通过在模块实例化的时候创建向空队列写对象事件以及从满队列中取对象事件，当相对应操作出现时，便通过 SimPy 中的 event 机制去触发相对应的事件。而等待该操作的外部模块就可以即时接收到讯息，并即刻执行相对应的操作。这样就实现了各个模块之间通过 GeneralFifo 通讯时，当没有对象，进程可以一直等待其他模块传对象的情形。与此同时，为了实现整个 GeneralFifo 的事件触发机制，GeneralFifo 需要提供队列是否为空或为满、获取队列的深度以及队列中对象的数量、获取向空队列写对象事件和从满队列中取对象事件等方法和函数接口。GeneralFifo 模块实现流程如图 4.5 所示：

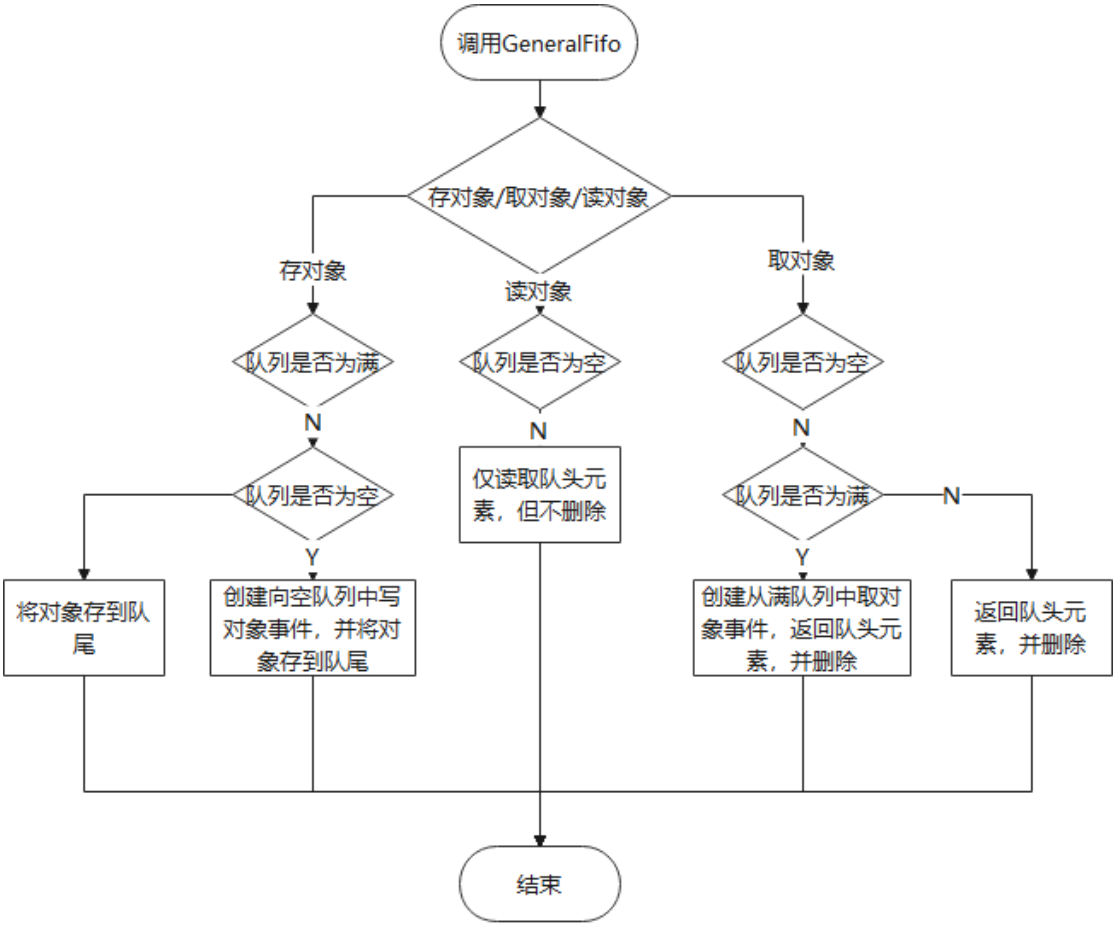

图 4.5　GeneralFifo 流程图

4.2.3　Memory 模块概要设计

Memory 模块[?]在整个硬件平台中包括 SL2（外部的公共内存池）和 PL2（在每个 processor 模块内部的内存池）两种内存池。这两种内存池通过总线模型挂载在整个硬件平台上，Memory 模块根据平台配置的 Bank 位宽在模型内部将 Memory 模块分成基于基地址偏移地址不同的各个 Bank 模块，具体的数据读写由内部 Bank 模块进行执行。

Processor 模块对数据的处理需要从 SL2 内存池中将数据取到自己内部的 PL2 内存池中，再将处理后的数据存到外部的 SL2 内存池中，Memory 模块负责数据的存取处理。Memory 模块接收从总线上传来的读写请求 Flit，并根据 Flit 中携带的数据地址以及平台中 Bank 位宽将整个数据读写请求拆分为相应 Bank 地址区间的 Bank 操作，并将操作处理符发送到到对应 Bank 上的 Fifo 中。Bank 模块在硬件平台构建时进行初始化，同时 Bank 模块创建 bank 读写操作进程，一旦有 bank 读写操作符发送到对应的 Fifo 中时，就会触发从 Fifo 中取出 bank 读写操作对象的事件，相应的 bank 读写进程开始解析 bank 读写操作符上所带的读写操作信息，开辟对应地址的内存空间并存储数据或将对应地址的数据取出并释放该地址的内存空间，并向 Memory 上的 SlavePort 发送读写响应。Memory 模块的结构图如图 4.6 所示：

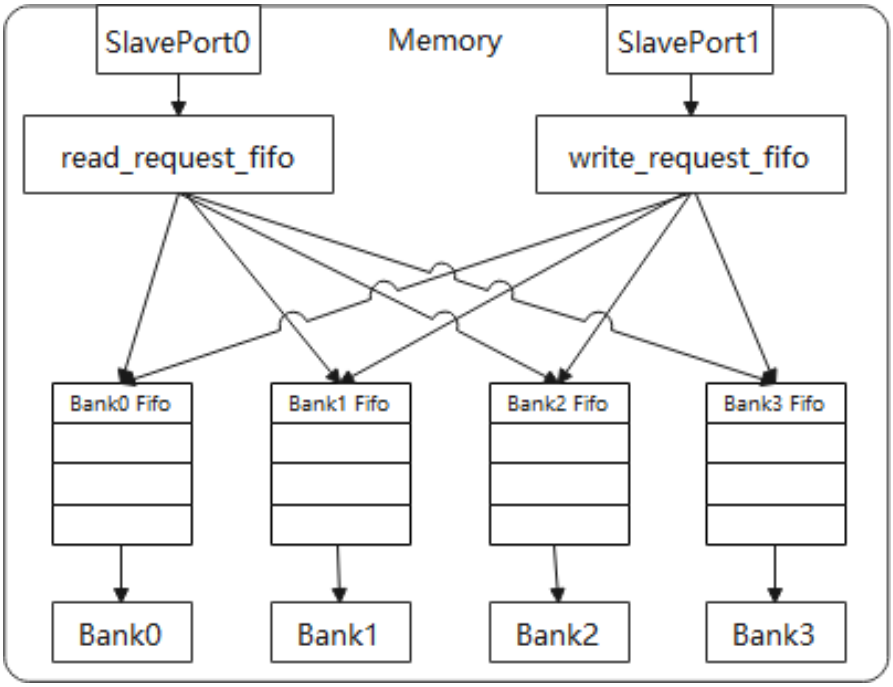

图 4.6　Memory 模块结构图

如图 4.6 所示，读写请求通过总线发送到对应地址所属 Memory 模块的 SlavePort 口，根据请求类型不同分别发到读请求 Fifo 上或写请求 Fifo 上由不同的进程进行处理，再根据基地址以及 Bank 位宽拆分，发送到不同 Bank 上的 Fifo 中

排队，最后由相对应的 Bank 进行读写请求操作。

4.2.4　Processor 模块概要设计

Processor 模块对 DSP、HAC 和 DMA 三种模型进行建模，实现模型的硬件行为、建立模型内的流水线结构并实现与其他模块的交互行为[?]。Processor 模块现在包括 ProcessorMgnt、ProcessorBase 以及具体的 DMA、DSP、HAC 模型模块。Processor 模块整体结构如图 4.7 所示：

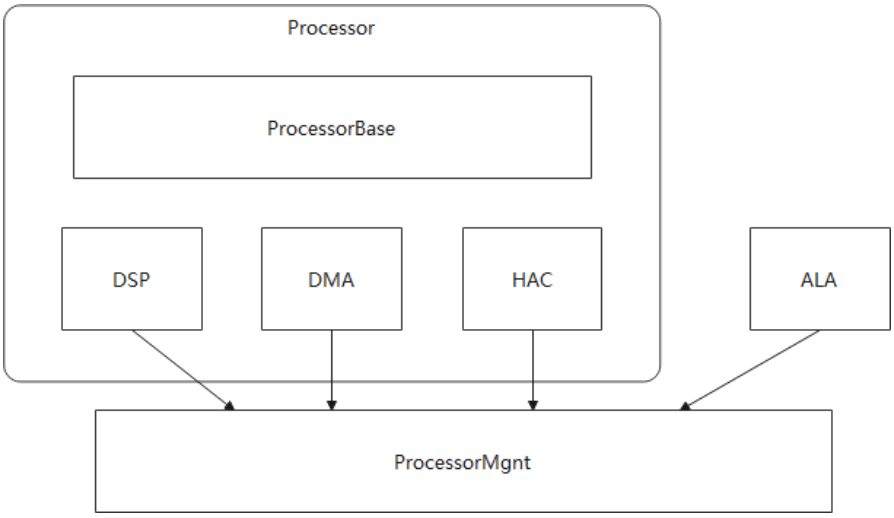

图 4.7　Processor 模块结构图

ProcessorMgnt 模块是实现具体模型的管理功能。各个模型（调度器、DSP、DMA 等等）在实例化的时候，ProcessorMgnt 记录下对应的对象，并提供各个模块通过相应的 Fifo 进行消息交互的方法，每个硬件模块都有一个唯一的硬件名，我们通过将硬件名作为 Key，将具体硬件模型作为对象存储到一个 dict 中，在交互式只需知道对方的硬件名，就可以通过 ProcessorMgnt 模块将对应的硬件模型对象取出，并直接将消息发送到相应的硬件模型的 Fifo 中。这样我们在具体模块之间进行消息交互的时候不用将消息封装成数据通过总线交互，节省了仿真时的资源浪费。由于各个模块之间消息交互十分频繁，所以这样也大大提高仿真平台的仿真效率。

ProcessorBase 模块是 DMA、DSP 及 HAC 模块的基类，实现了具体模型的基础功能，并提供相应的接口。DMA、DSP 以及 HAC 模型作为同一类硬件处理器模型，虽然各自实现的功能不同，但这几种模型的硬件结构相似，都需要进行消息交互，所以 ProcessorBase 模块实现的这几种模型相同的硬件结构以及消息交互的接口 Fifo 和消息交互的功能。

DMA(direct Memory Access, 直接存储器访问) 模块负责将数据从一块地址复制到另一块地址空间，实现数据在内存模块之间进行快速传输，在轻量级仿真平

台中主要负责 PL2 内存池和 SL2 内存池之间的数据传输。DMA 模块接收来自调度器模块 DMA 任务或整个任务链中数据传输部分，将任务解析后包装成任务描述符，通过总线模块发送到对应地址的 Memory 模块上。接收到相应的任务完成响应后再出发下一级任务。

DSP（Digital Signal Processing，数字信号处理）芯片是一种快速强大的微处理器，可以迅速的实现各种数字信号处理算法。在仿真过程中，DSP 模块负责快速完成数据的计算处理。在轻量级仿真平台中，该模块只有数据处理的时延，不负责数据搬运。只需要体现系统时延，不需要对搬运的数据进行真实的操作。处理时延的变化通过 SimPy 中 Timeout 来实现，具体的时延信息包含在任务描述符中。

HAC(Hardware Accelerator，硬件加速器) 是用来专门处理某一类执行次数很多的硬件操作的的硬件，为了加快这一类的操作，特地设计一种芯片专门处理这种操作。HAC 模型负责处理数据搬入、数据处理以及数据搬出的所有操作。HAC 模型接收到 HAC 任务后，首次解析任务信息，然后将相应数据从 SL2 内存池搬运到 HAC 模型所在区域由调度器分配的 PL2 内存池，HAC 模块对数据进行处理，最后再将处理后的数据搬运到对应的 SL2 内存池中。

4.3　设计空间构建模块及仿真预测模型概要设计

设计空间探索是指根据系统设计所关心的参数而对不需要的参数进行系统分析和修剪。设计空间探索模块实现在可配置的系统资源构成的设计空间中寻找目标优秀的设计方案。设计空间探索主要分为构建设计空间和选择优化目标、对设计空间进行剪枝以及对设计空间探索结果分析三个部分。设计空间构成由可配置的系统参数以及系统参数的可行范围构成，一般而言，设计空间的整体范围会比较大。优化目标则是我们对整个系统的优化方向，比如在本文中的设计空间探索流程中，优化目标为仿真时延、核利用率以及核一致性几个参数。设计空间探索的流程图如图 4.8 所示：

我在设计空间探索模块主要负责的是设计空间的参数提取和构建以及仿真平台预测模型的设计与实现。

仿真系统中设计空间探索设计流程如图所示，在设计探索流程的具体实现中，我们先修改仿真平台系统参数，自动化迭代运行仿真平台，并通过读取数据库信息提取出设计空间探索所有的目标值，最后将设计方案和对应的仿真数据输出到 txt 文件中。

将上个流程中提取的 6000 个训练集数据作为输入数据训练仿真平台预测模型。最后在遗传算法中探索设计空间中所有设计方案，最后输出最为优秀的设计

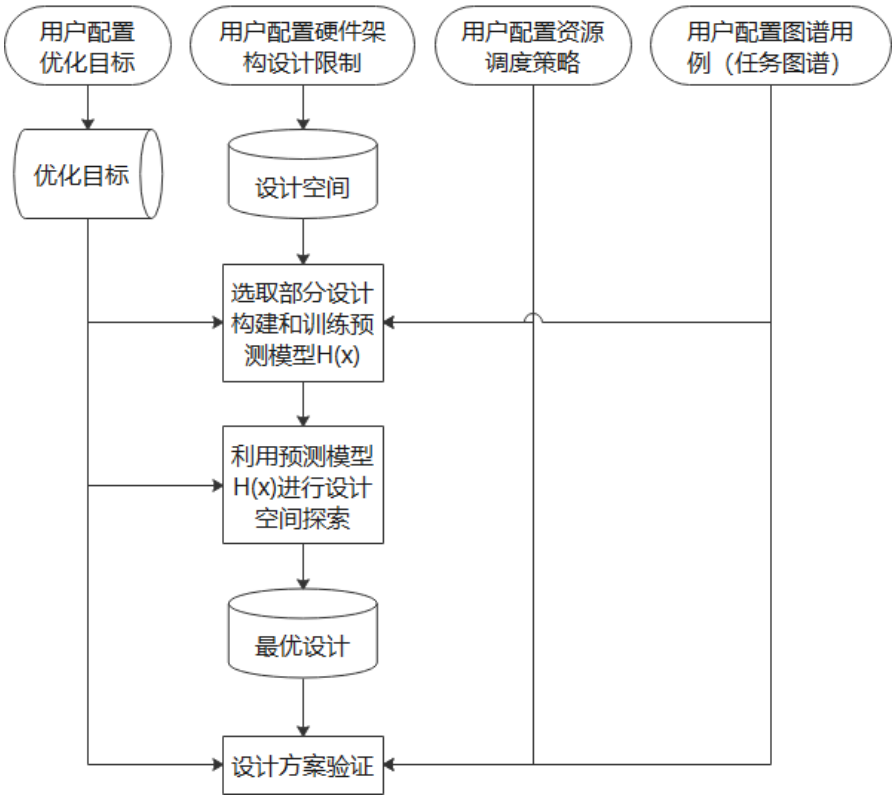

图 4.8　设计空间探索流程图

种群作为我们设计的备选方案，并对方案重新在仿真平台上运行，对仿真结果进行对比分析。

下面主要分别设计空间构建模块、探索预测模块的概要设计进行简要介绍。

4.3.1　设计空间构建模块概要设计

通过对系统设计空间探索的需求分析，找出我们所需要的系统的优化方向即优化目标，通过分析仿真系统哪几个可配置参数会对优化目标产生较大影响，提取出这几个设计参数，根据真实芯片设计的参数变化，对设计参数范围进行限制，最后形成设计空间。通过对系统设计空间探索的需求分析，我们找出我们需要对系统的优化方向即优化目标，通过分析仿真系统哪几个可配置参数会对优化目标产生较大影响，提取出几个设计参数，通过对设计参数范围的限制，最后形成设计空间。

由于后期在遗传算法进行设计空间探索的过程中进行迭代时，设计空间中配置参数值的变化要求仿真平台中进行配置参数的调整。

4.3.2　仿真平台预测模型概要设计

设计空间探索流程中遗传算法需要进行多次的仿真过程迭代，为了减少设计空间探索流程的时间开销，由于在设计空间探索的过程中，仿真平台的配置参数和目标值固定，所以将系统仿真流程设计成仿真平台预测模型，这样可以大大

减少目标值的输出时间。接下来从训练集生成和预测模型生成两个方面介绍：

训练集生成模块实现的功能包括：读取设计空间中配置并配置到硬件配置文件中、自动化读取配置文件迭代运行仿真文件、读取每次仿真结果数据库文件提取所需要的目标值、将设计方案和对应的目标值输出到文件中并最终输出训练集文件。该模块的流程图如图 4.9 所示：

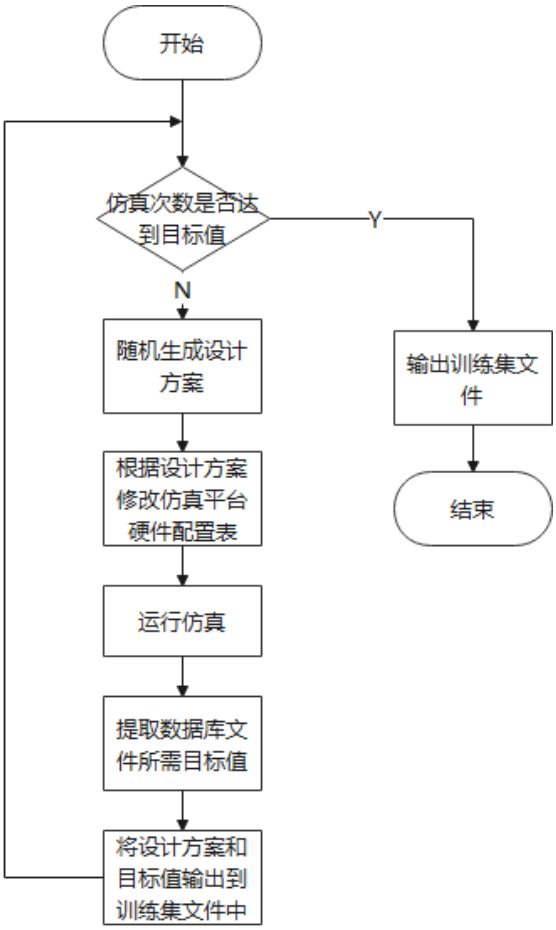

图 4.9　训练集生成模块流程图

训练集文件中的目标值即仿真数据结果，我们需要在每次通过修改硬件配置在仿真平台上进行系统仿真，在仿真结束后通过对数据库文件中数据的提取将相应的目标值统计计算到相应的文件中。这样从设计空间中采样得到样本，将样本在仿真平台上运行得出相应的目标值，从而得到训练集文件。

预测模型生成模块生成仿真平台多目标预测模型，先将上个流程中的数据集经过处理后提取出来，并分为训练集和验证集。采用现有的 sklearn 框架中的决策树回归模型[?]去训练每个目标，再通过一个多目标预测模型类将所有模型文件读取，并实现预测目标值的功能函数。这样整体每个目标的预测误差较低。

4.4　仿真信息数据库概要设计

在仿真过程中，我们在每个节点对仿真信息进行统计，最后在仿真结束时输出至数据库进行整理。从而得到一系列我们对整个仿真过程每个任务的执行过程，每个硬件的利用率，内存的申请情况，通过这些信息，我们就可以对整个仿真进行分析，并通过这些输出结果进行下一步 DSE 架构探索。

仿真系统的结果输出主要通过 log 信息和数据文件实现，log 信息主要统计任务执行的细节信息，方便寻找调优的关键节点。数据库文件统计任务的执行过程，每个硬件的利用率，内存的利用等等。仿真平台的数据库采用 python 的 SQLite 库实现。系统中只需在仿真流程中在数据库中各自独立更新数据库中各个表的数据，每个表的数据表基本独立，SimId 是所有数据表的主键，用于区别多次仿真之间结果。数据库的物理模型由下列各表所示：

表 4.1　SimRecord 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	StartTime	仿真开始时间	int	N	
3	EndTime	仿真结束时时间	int	N	
4	SimTime	仿真持续时间	int	N	
5	TestCaseName	仿真用例名字	Text	N	
补充说明					

注：一个数据库可以记录多次仿真的结果，以 SimId 作为区别。

表 4.2　HacJobTable 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	JobTypeId	任务的序号	int	N	
3	InstId	任务具体实例序号	int	N	
4	PipeLine	任务具体哪一级流水	int	N	
5	StartTime	任务开始执行时间	int	N	
6	EndTime	任务结束执行时间	int	N	
补充说明					

注：该表记录了 Hac 任务所处的流水级别和 Hac 任务在该级流水的执行细节。

表 4.3　HacJobTable 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	JobTypeId	任务的序号	int	N	
3	InstId	任务具体实例序号	int	N	
4	PipeLine	任务具体哪一级流水	int	N	
5	StartTime	任务开始执行时间	int	N	
6	EndTime	任务结束执行时间	int	N	
补充说明					

注：该表记录了 Hac 任务所处的流水级别和 Hac 任务在该级流水的执行细节。

表 4.4　AlaKernelTaskTbl 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	KernelId	Kernel 任务序号	int	N	
3	InstId	任务具体实例序号	int	N	
4	KernelType	任务类型	int	N	
5	ReadyTime	任务的就绪时间	int	N	
6	IssueTime	任务开始执行时间	int	N	
7	CompleteTime	任务结束执行时间	int	N	
8	PreMoveEnd	任务前搬移结束时间	int	N	
9	PostMoveEnd	任务后搬移结束时间	int	N	
补充说明					

注：AlaKernelTaskTbl 表记录的是在 Dsp 核执行的任务，前搬移任务和后搬移任务分别用来搬运任务的输入数据和输出数据。

表 4.5　WFEJobTable 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	JobTypeId	任务的序号	int	N	
3	InstId	任务具体实例序号	int	N	
4	StartTime	任务开始执行时间	int	N	
5	CompleteTime	任务结束执行时间	int	N	
补充说明					

WFE 任务不在核上执行，只起任务的收齐作用，当所准备的任务全部做完后，立刻执行完毕。

表 4.6　Pl2UsageRecord 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	TTI	仿真时间段	int	N	
3	SimTime	记录的时间点	int	N	
4	AlaId	调度器 Id	int	N	
5	PoolId	内存池 Id	int	N	
6	UsedSize	内存池已使用的大小	int	N	
7	AllocSize	内存池已分配的大小	int	N	
补充说明					

注：该表记录了 Pl2 内存的在每个时间段的使用情况，以内存池为单位进行记录。

表 4.7　SimUsageTable 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	ProcessorName	硬件名称	text	N	
3	TTI	仿真时间段	int	N	
4	SampleCount	时间片序号	int	N	
5	Usage	时间片内硬件利用率	int	N	
补充说明					

注：该表记录了在仿真过程中每个时间片内硬件资源的利用率。

表 4.8　DBMUsageRecord 表

序号	字段名	字段说明	类型	是否为空	说明
1	SimId	仿真序号	int	N	
2	TTI	仿真时间段	int	N	
3	SimTime	记录的时间点	int	N	
4	AlaId	调度器 Id	int	N	
5	PoolId	内存池 Id	int	N	
6	UsedSize	内存池已使用的大小	int	N	
7	AllocSize	内存池已分配的大小	int	N	
补充说明					

注：该表记录了仿真过程中以 DBM 管理的 Sl2 内存每个时间段的分配情况，以内存池为单位记录。

4.5　本章小结

本章基于仿真平台的的整体设计及设计空间探索的整体框架入手，详细介绍了仿真平台的整体流程以及相关模块的架构以及功能实现、设计空间探索模块的主要流程以及各个流程模块里面的功能实现，如何生成训练集以及预测模型的训练生成等等。

本章为系统的详细设计提供了明确的框架，表明了设计开发的重点和难点，为后续系统的具体设计提供了有力的帮助。

第 5 章　详细设计与实现

本章节是基于第四章概要设计的基础上，给出整个系统中的详细设计与实现，主要内容包括仿真平台硬件平台构建和硬件模型建模的实现以及整个平台的仿真运行流程细节、设计空间探索过程过程中训练集和多目标预测模型的生成以及对设计空间探索结果分析这些部分的具体实现。

5.1　仿真平台模块硬件平台构建模块的设计与实现

TopLevel 仿真平台是基于任务图的一个以 Simpy 引擎为仿真框架的仿真系统[?]。TopLevel 仿真平台的设计与实现是整个系统实现部分的重中之重，而对仿真系统各个模块的建模则是这部分工作的核心。整个仿真系统通过解析输入的硬件配置文件，实例化硬件模块，通过总线将各个硬件模块连接起来，形成整个仿真平台的硬件平台部分。任务图管理模块通过解析输入的任务图，并根据触发关系链式的调度任务，将任务分配到具体的硬件模型上执行。下面将详细阐述仿真平台硬件平台构建模块的功能实现及结构。

整个仿真平台的硬件平台部分从结构上可以抽象为一个自顶而下的一个树状结构，所有具体的硬件模型实例都被视为该树状结构的最底端的叶子节点，所有的叶子节点（硬件模型实例）都是由上层的节点（不同的子组件）分区管理，依次逐层向上。树的根节点即为仿真平台的硬件平台部分。仿真平台的硬件平台的树状结构示意图如图 5.1 所示：

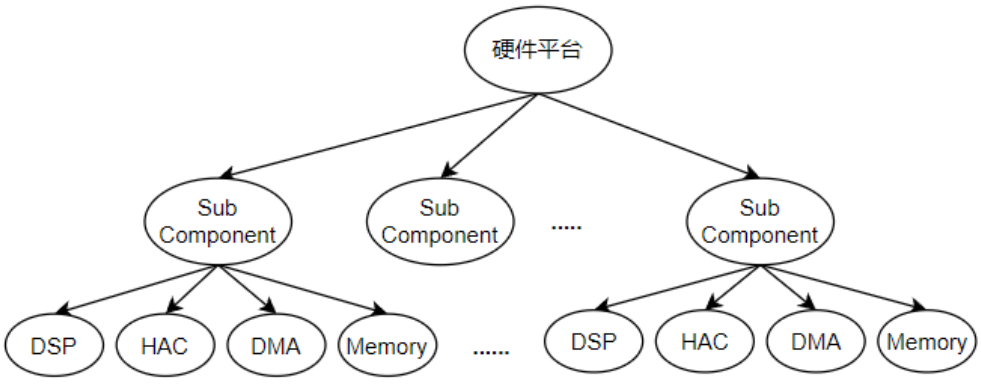

图 5.1　硬件平台树状结构示意图

用户在进行仿真之前需要根据自己的需求去修改硬件平台配置文件，硬件平台构建需要读取硬件平台配置文件并解析，系统所需的硬件平台配置文件包括：V801.xml 用来描述所有硬件模型之间的连接信息及 Port 信息、V801MemMap.xml 用来描述所有硬件模块模块与具体物理地址之间映射关系、

V801MemPt.xml 用来描述内存模块的具体信息、V801Prop.xml 用来描述所有需要实例化的模块包括 Processor 和总线模块等等在实例化的时候的所需的所有具体信息、V801Route.xml 描述了所有实例化后的硬件模块之间的路由信息，将各个硬件模型通过总线连接形成一个完整的硬件平台。在实际的用例仿真业务中，在进行用例仿真之前，仿真系统先根据硬件平台配置文件对硬件平台进行构建。PlateformBuilder 会实例化 DSP、DMA 以及 HAC 等等硬件模型，当一个子组件中所有的硬件模块全部实例化后，再继续实例化其他子组件中的硬件模块，最后当所有组件中硬件模块实例化成功后，整个仿真平台的硬件平台就构建成功了。

我们通过创建 SaeSimulator 单实例去实现整个平台初始化的功能，整个 SaeSimulator 的类图如图 5.2 所示：

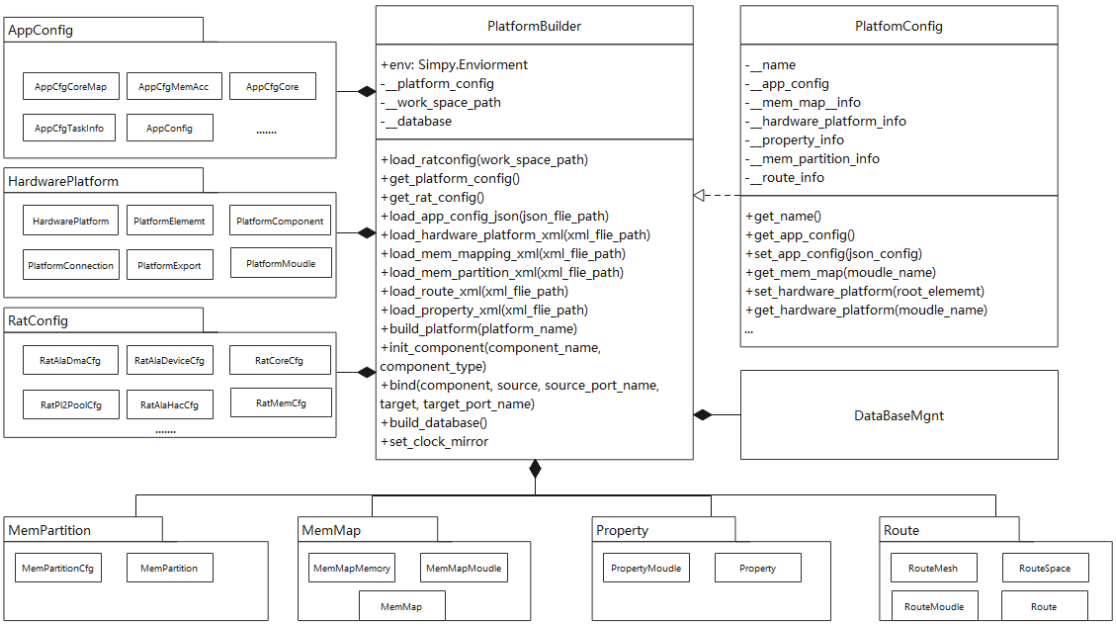

图 5.2　硬件模块实例化模块类图

图 5.2 硬件平台实例化模块需要的类以及类之间关系构造的类图，在硬件平台实例化模块中，我们在最外层通过 SaeSimulator 单实例调用 load_platform 函数实现 PlatformBuilder 类中对各个硬件配置文件的读取和解析以及实现硬件平台中各个模型的实例化，PlatformBuilder 类读取和解析硬件配置文件后，将硬件配置信息存储到 PlatformConfig 类中各个属性中，并对每个属性设置 get 和 set 函数，支持外部对类里面私有函数的设置和读取。最后，硬件模型的实例化通过调用 init_component 函数实现。

硬件模块实例化过程中，实现方式通过层次性的实例化。先按模块解析硬件配置文件读取硬件配置信息。在逐层实例化各个模块。如硬件模块实例化类图所示，每个包里是硬件的一部分，将包里的各个类根据读取的硬件配置信息实例化，在实例化整个次外层的类，如 AppConfig 里的 AppConfig 类。最后在 init_component 函数中统一，最后通过 bind 函数将各个模块通过路由信息中 Port

口连接起来。最后，通过 DatabaseMgnt 类实例化并连接数据库文件。

5.2　GeneralFifo 模块设计与实现

在整个仿真平台中，我们需要实现硬件模型能够在别的模块发送过来消息时能够及时的对发送过来的消息进行处理，以及模型外部能够接收别的模型发送过来消息的消息队列。这种消息队列主要用于 Processor 模块的流水线功能的具体实现。

我们基于 Simpy 的事件机制实现了 GeneralFifo 模块，并在整体仿真平台构建时将此模块作为基础数据结构使用，与先入先出队列的整体逻辑类似。我们在概要设计中简要介绍了 GeneralFifo 的整体执行流程。我们在这一节详细介绍 GeneralFifo 的设计。GeneralFifo 模块的类图如图 5.3 所示：

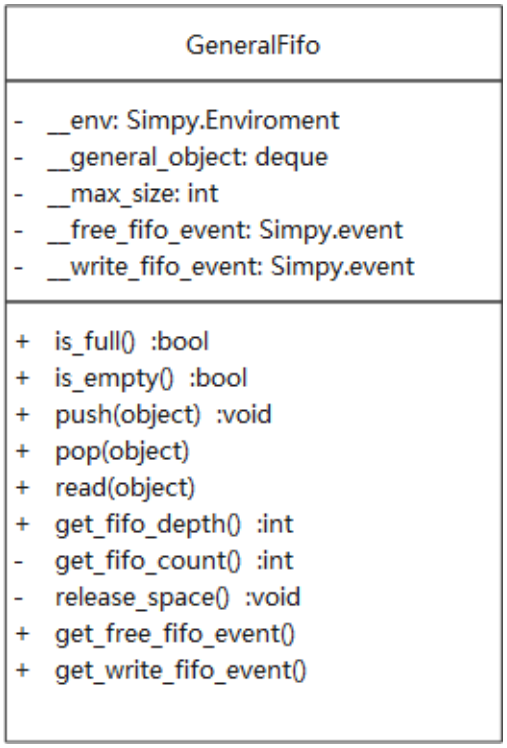

图 5.3　GeneralFifo 类图

GeneralFifo 模块的底层实现基于数据结构双向队列 deque 实现。整体实现队列先入先出，并可以设置以及查询队列深度，查询队列中现有对象数量，判断队列的状态。并基于 Simpy 的 event 机制实现了队列的 Push 和 Pop 功能。每次将对象 Push 进队列时，将触发该队列的 write_fifo_event，该 event 事件触发时，会使在其他模块等待该事件触发的进程开始继续执行，从而实现该队列一旦有对象存入，能够及时通知其他模块将对象从队列中取出进行操作。同时，当队列为满时，其他模块只需等待 free_fifo_event 事件触发，其他模块即可继续向 Fifo 中

Push 对象。这样就实现了不同模块之间基于 GeneralFifo 的信息传输的功能，以及实现一个进程或者模块等待信息的场景。在仿真平台中所有模块之间交互以及 Processor 模块的实现都是基于 GeneralFifo 实现的。

5.3　Processor 模块设计与实现

仿真平台中 Processor 模块主要功能为：接收调度器模块发送过来的任务实例并提取任务实例中中的任务信息，在具体的硬件实例上根据任务信息处理任务。Processor 模型类型主要分为 DSP 模块、DMA 模块和 HAC 模块。下面详细阐述了仿真平台 Processor 相关模块设计与实现。

Processor 模块主要实现对三种基础硬件模型（DSP、HAC 和 DMA）的建模以及 Processor 模块与调度器模块之间的交互。在实现三种基础模型的建模之前，我们实现 ProcessorBase 类，通过 ProcessorBase 类所有 Processor 模型的基础功能进行实现。

平台中所有模型都是以 GeneralModule 类为基类实现，GeneralModule 模块是统一所有已实例化的模型并将这些模型联系起来的模块。GeneralModule 模块实现了平台信息和实际硬件模型信息的联系，并为每个模型模块创建了 Log 文件，在各模块中可以调用其基类 GeneralModule 类中的 log 方法去记录 log 信息，并统一设置整个仿真过程中的 log 等级，以应对不同情况下的业务仿真；GeneralModule 模块中也将数据库模块和具体模块之间联系起来，在各模块中可以调用其基类 GeneralModule 类中的 message 方法去记录信息并存储到数据库中。

Processor 模块具体实现三种模型的功能建模，分别为 DMA 模型、DSP 模型以及 HAC 模型。DMA 模型的主要功能是要实现数据搬运功能，为 DSP 模型中的具体执行将数据从 SL2 搬运到相应 DSP 组的 PL2 中或者数据处理结束后从 PL2 搬运到 SL2，再或者仅仅只进行内存中的数据交换。DSP 模型模拟 DSP 芯片的数据处理任务执行过程的流程，主要执行数据的处理。HAC 模型模拟了从数据搬运到数据处理再到数据搬出的整体流程。HAC 主要为了执行某一种特定的执行多次的硬件任务，任务在 HAC 上执行速度较快，任务的整个执行流程均在 HAC 硬件上完成。

整个任务执行流程周期是由任务管理模块解析任务实例为开始，任务管理模块解析任务实例后，由调度器模块接收任务管理模块解析后并重新整理的任务实例，调度器模块根据接收到的任务信息去申请对应的硬件资源（包括硬件模型以及空闲的内存空间），接下来调度器模型将任务实例发送到申请到的硬件模型上，由对应的硬件模型对任务实例进行处理，硬件模型对任务处理完成后，

将任务执行完的响应消息发送回调度器，由调度器模型对该任务实例标记完成，此时一个任务的整个执行流程就结束了。任务实例的整个执行流程图如图 5.4 所示：

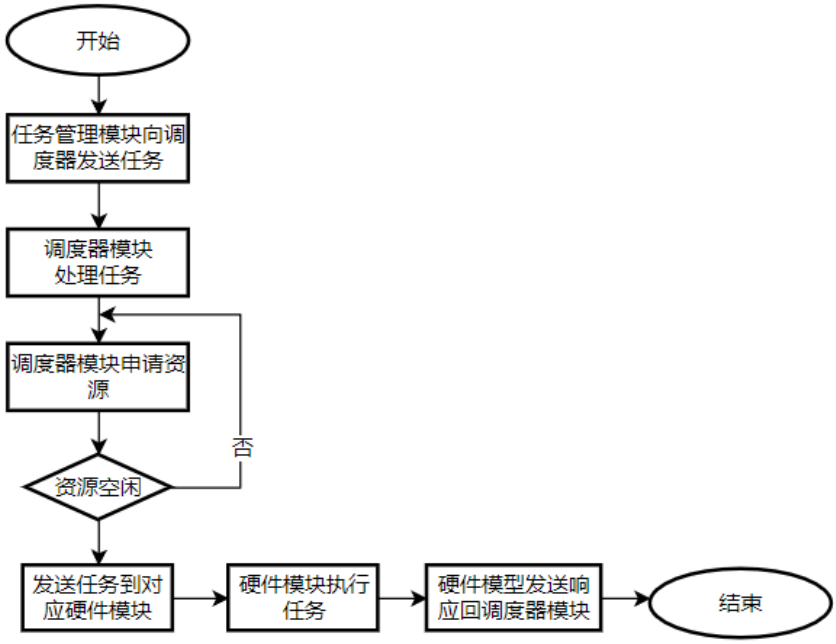

图 5.4　任务执行流程图

调度器模块需要将任务分发到调度器中资源调度模块申请的特定的硬件模型。但由于整个平台软硬件设计分离，调度器模块无法获取硬件平台整体构建信息，无法直接与硬件模块之间进行信息交互。因此我们需要一个模块实现调度器模块与具体硬件模型之间的交互，为此我们在 Processor 模块中设计了单实例 ProcessorMgnt，用于实现消息交互功能。Processor 模块以及相关模块关系如图 5.5 所示：

图 5.5 显示了 ProcessorMgnt、三种具体的 Processor 模型、所有 Processor 模型的基类 ProcessorBase 以及几者之间的关系。在 ProcessorMgnt 模块中，各个模块在实例化的时候通过调用单实例的对应方法通过相应类型的 dict 去记录，dict 以模型名为键值。当其他模块模块之间需要进行交互时，就可以通过调用 ProcessorMgnt 单实例相应方法去访问对应字典，通过模型名就可以取到相应的模型对象，从而进行交互。

三种基础的硬件模型以 ProcessorBase 为基类。ProcessorBase 类实现了几种基础硬件模型的公有方法：如实例化 Port、读数据、写数据、管理 MasterPortMgnt 等等。在介绍具体的 Processor 模型实现之前，我们先介绍在仿真平台中传输的数据类型。

在仿真平台中传输的数据主要包括 RwObject、Burst 以及 Flit 三种。在最外

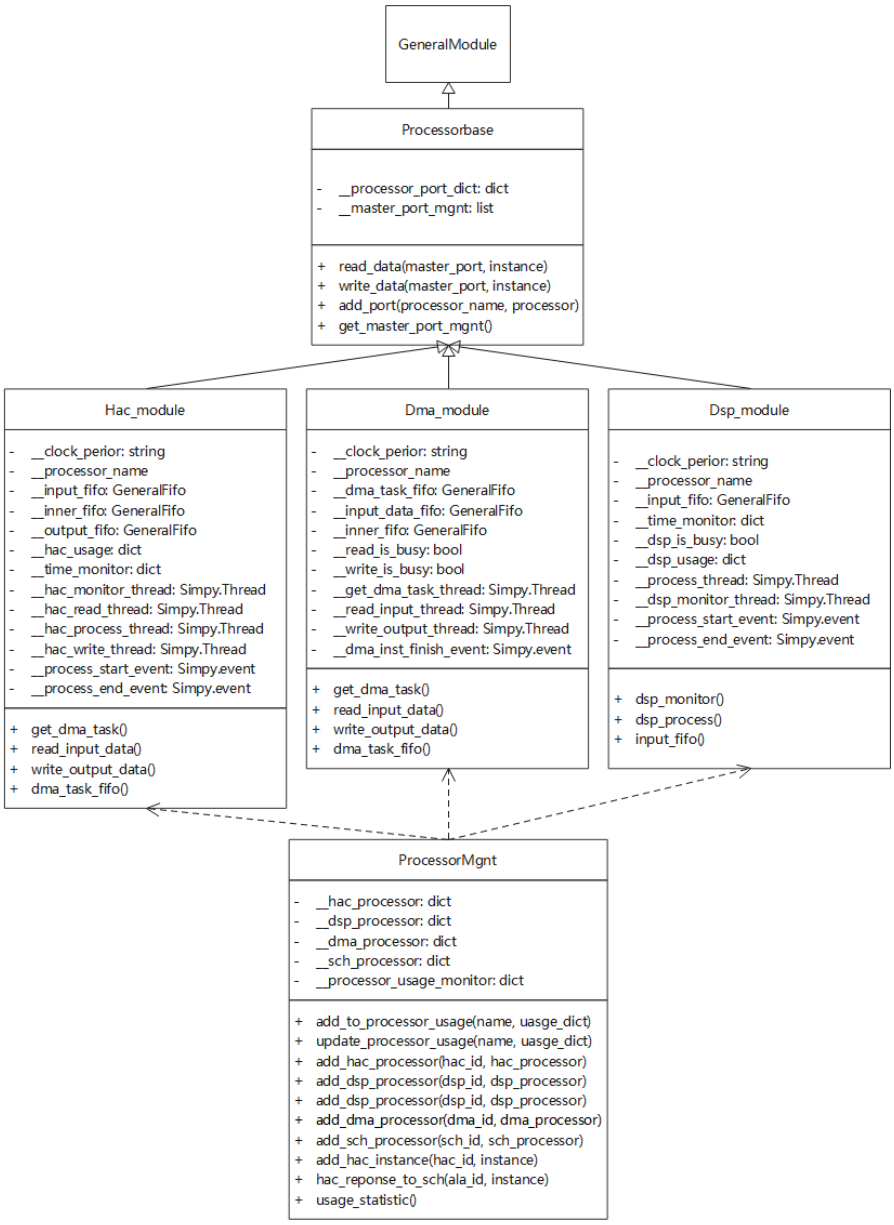

图 5.5　Processor 模块类图

层硬件模块中传输数据的数据类型为 RwObject，数据在总线上传输的数据格式为 Flit。这三种数据格式均包含着数据请求的关键信息：请求类型、目的地址，数据长度等等，还包含着每个对象独有的 id（如 ObjectId，BurstId 等等）。DMA 模块或者 HAC 模块需要进行数据搬运的时候，将数据封装成 RwObject 发送到所在硬件对应的 Port 上，由 PortMgnt 模块对 RwObject 进行拆分，将其拆封成总线位宽的 Flit 数据帧。Flit 数据帧仅在总线上可见，在仿真平台上可见的还是 RwObject。图 5.6 为三种数据类型的类图，以及图 5.7 为三种数据类型之间的关系图。

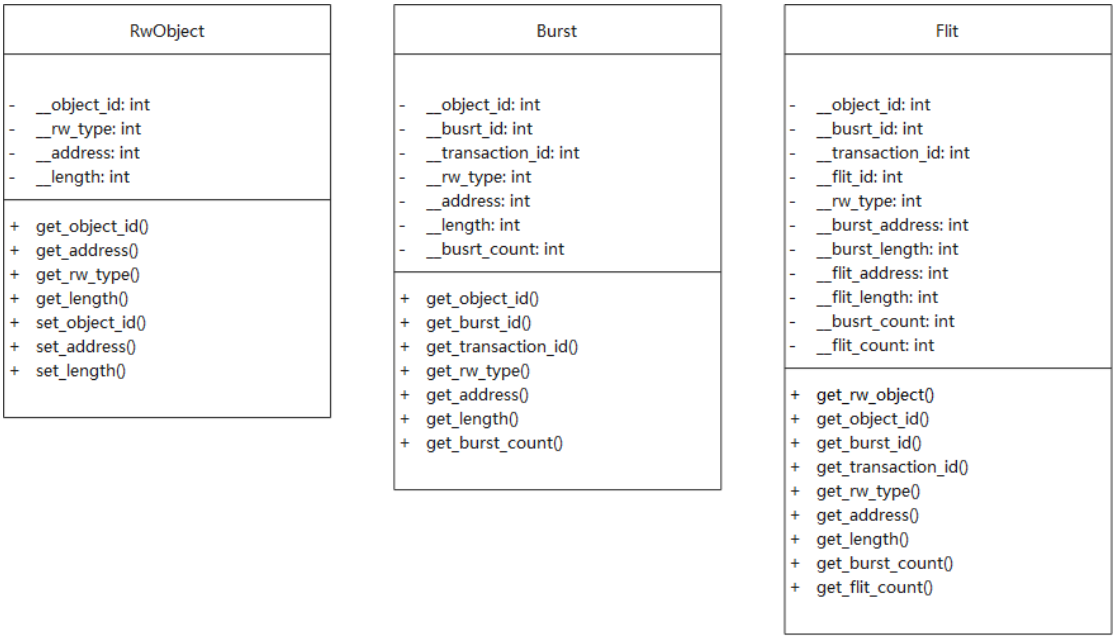

图 5.6　三种数据类型类图

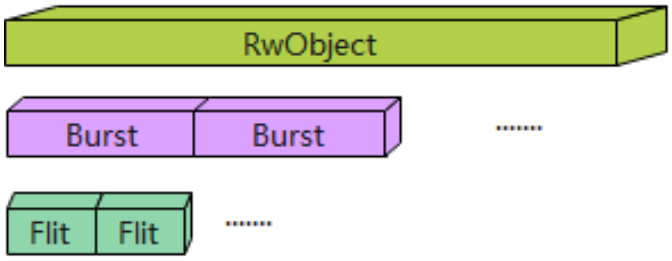

图 5.7　数据类型关系图

下面介绍三种基础硬件模型的具体实现：首先我们介绍 DSP 模块的功能实现。DSP 模块模拟的是 DSP 芯片的数据处理任务执行过程，在仿真平台的流程中只需模拟真实的处理时延即可。调度器模块将任务用例通过 ProcessorMgnt 模块发送到 DSP 模块的 input_fifo 中，在硬件模型中 process_thread 一直在等待输入，DSP 模型的 input_fifo 中一旦有对象传入，该事件被触发后，process_thread 进程就开始进行处理。处理时延在输入的任务描述符中已经存在，DSP 模型只需从任务描述符中取出时延信息，在乘以硬件的时钟频率即可得到需要等待的仿真时间。DSP 模块在任务链中作用负责完成输入数据的处理。在仿真平台中只完成处理时延的仿真，并不真实的进行数据的处理。前搬移任务、DSP 任务以及后搬移任务形成一整个任务链，三者任务信息由同一个任务实例携带，当前搬移 DMA 任务完成时，解析任务实例中的信息得到下一级 DSP 任务的硬件模型名，通过 ProcessorMgnt 将任务实例发送到对应 DSP 模型 input_fifo 中，触发相应的 DSP 任务，发回调度器的响应只用来释放 DMA 任务申请的系统资源。

接下来介绍 DMA 模块的功能实现。在仿真平台中，DMA 任务一般分为纯 DMA 任务、前搬移任务以及后搬移任务。其中前搬移任务和后搬移任务都是包含在 DSP 任务链里面的，而纯 DMA 任务只是单纯将数据从一块地址搬移到另

外一块地址。DMA 模块需要完成的对 DSP 模块需要进行处理的数据的搬入搬出，将数据从对应 SL2 内存中搬入 DSP 模块的 PL2 内存中的过程称为前搬移，将数据从 PL2 内存搬运到 SL2 的过程称为后搬移。数据的搬入搬出，DMA 模块通过任务用例中的地址和数据大小信息创建 RWObject，通过 MasterPortMgnt 经过总线进行传输，DMA 模块在实例化的过程中创建两个线程分别用来处理数据的搬入和数据的搬出，以及创建一个线程获取 DMA 任务实例和控制总体的流水线深度。DMA 模块通过各级线程以及线程之间的 Fifo 实现流水线。每一级线程均从 Fifo 中取出任务实例，执行完任务实例之后，再将任务实例发到下一级线程的 Fifo 中，从而实现流水线。DMA 模块的流水线执行时序图如图 5.8 所示：

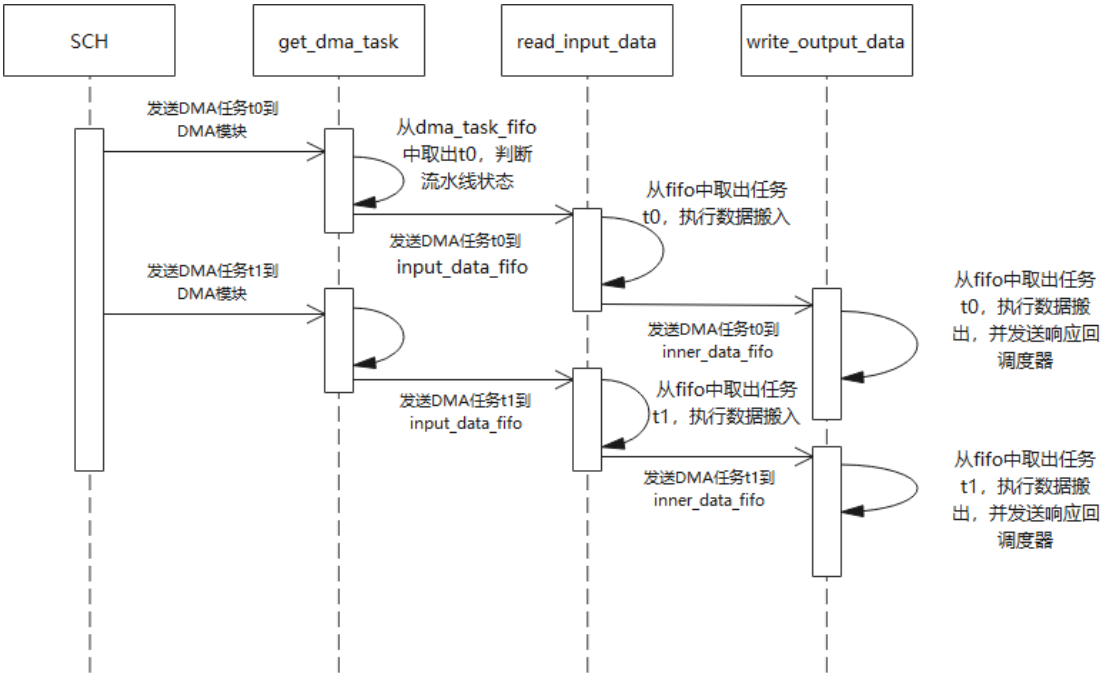

图 5.8　DMA 模块流水线时序图

HAC 模块是用来处理专用任务的硬件。一种 HAC 只用来执行一种任务，用来加速任务的处理。HAC 硬件模块中需要完成数据的搬入、处理以及搬出的一整套流程。HAC 模块接收调度器模块发送过来的 HAC 任务实例，HAC 任务实例中记录着输入数据、处理时延以及输出数据等相关信息。HAC 模块通过三级流水实现数据搬入、数据处理以及数据搬出的功能。调度器模块发送 HAC 任务实例到相应 HAC 模块的 input_fifo 中，当 input_fifo 接收到任务实例后，HAC 模块的 input_thread 等待的事件被触发，进程加锁不再接收任务实例，进程开始执行数据搬入的操作，把数据从相应的 SL2 内存中搬运到 PL2 内存中，操作执行结束后，将任务实例放入 inner_fifo 中，进程解锁。后续两个进程重复此类操作，形成 HAC 内部的三级流水的任务执行。HAC 模块的三级流水时序图如图 5.9 所示：

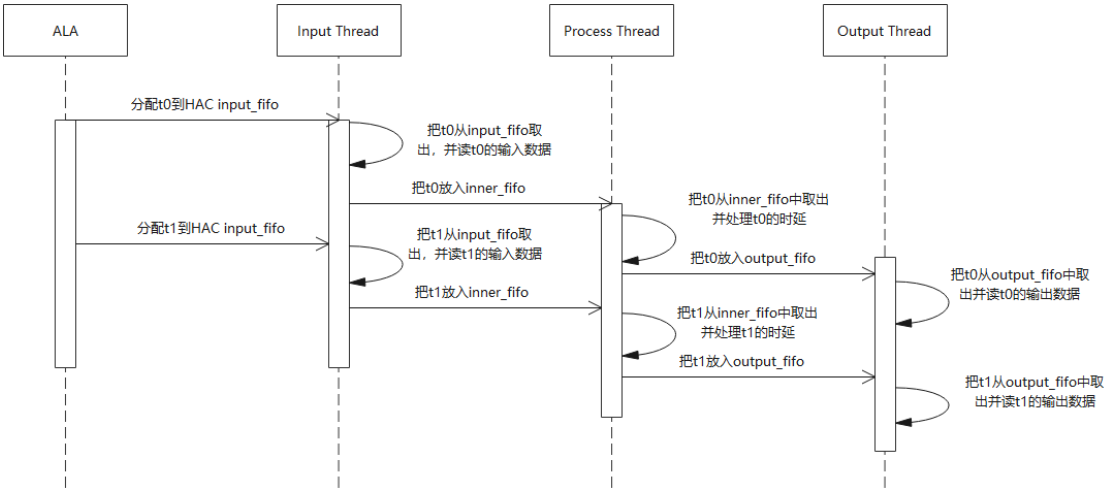

图 5.9　HAC 模块三级流水时序图

此外，Processor 模块还在 DSP 模型和 HAC 模型中实现统计每个定长的时间片内的硬件利用率。通过 dsp_monitor_thread 和 hac_monitor_thread 两个进程分别去统计 DSP 模型和 HAC 模型的执行时间片。以 DSP 模块为例，DSP 模块在从 FIFO 中取出任务实例时触发 process_start_event，处理完时延后触发 process_end_event。这两个事件都会使挂起的进程开始执行，从而统计 DSP 模型每次执行的时间片区间。同时为了统计每一段时间片内每个硬件的占用率，硬件模型中 ProcessorMgnt 模块统计了所有硬件模型的对象，ProcessorMgnt 模块在平台初始化时创建 processor_usage_monitor 字典，processor_usage_monitor 每个时间片的每个硬件模型的核利用率初始值设置为 0，用于统计每段时间片内的硬件利用率，并在对应的硬件模型中通过对应的进程去记录。在相应的进程中，先设置变量 Start 和 End 来记录任务的开始时间戳以及结束时间戳，并设置标记记录当前时间戳为任务开始的时间戳还是任务结束的时间戳。并通过每次 Start 和 End 之间时间差去计算统计每个时间片的硬件利用率，并将相应的核利用率记录到 processor_usage_monitor 中。在整个用例仿真结束后，在 ProcessorMgnt 模块中通过 usage_statistic 方法去将 processor_usage_monitor 中的内容统一的存储到数据库中。HAC 模型内部工作时间块统计与之相同。硬件模型核利用率的监控进程的伪代码如下所示：

伪代码 5.1　统计每个时间片内的核利用率

```
Data: 时间片长度 l, 任务开始时间戳 Start, 任务结束时间戳 End, 当前时间戳类型
      Flag, 核利用率字典 processor_usage_dict
Input: 进程触发时间戳 time
Result: processor_usage_dict

Start ←0;
End ←0;
/* loops                                                    */
the thread is triggered;
// Flag(0: the time is start; 1:the time is end)
if Flag is 0 then
    k ←End / l;
    interval ←(time - k * l);
    n ←interval / l;
    if n! = 0 then
        for i = k + 1 to k + n do
            processor_usage_dict(i) ←0;
    Start ←time;
    Flag ←1;
else
    k ←Start/l;
    interval ←(time − k * l);
    n ←interval / l;
    if n! = 0 then
        processor_usage_dict(k) ←(processor_usage_dict(k) + (k * l − Start)/l);
        if n! = 1 then
            for i = k + 1 to k + n − 1 do
                processor_usage_dict(k + 1) ←1;
        processor_usage_dict(k + 1) ←(time − (k + 1) * l)/l;
    else
        processor_usage_dict(k + 1) ←(time − Start)/l;
    End ←time;
    Flag ←0;
```

5.4　仿真平台 Memory 模块设计与实现

仿真平台的 Memory 模块主要实现接收 Master 设备通过总线互联模块发送过来的读写请求 flit，根据 Bank 位宽拆成对应的 Bank 操作发送到对应地址的 Bank 的 FIFO 上。

Memory 模块提供交织的 Bank。这些 Bank 模块根据地址相互交织，每个 Bank 的偏移地址与 Bank 位宽相同。仿真平台中 Memory 模块 Bank 模块设计为 4 个 Bank，Bank 位宽为 256 字节的设计方案。Memory 模块在实例化的时候创建 4 个 Bank 模块，Memory 模块可以同时处理不同的 Bank 模块上的 Bank 操作，但每个 Bank 模块同时只能处理一个 Bank 操作。Bank 模块的交织方案如图 5.10 所示:

Bank0	Bank1	Bank2	Bank3
0x0000	0x0020	0x0040	0x0060
0x0080	0x00A0	0x00C0	0x00E0
0x0100	0x0120	0x0140	0x0160
0x0180	0x01A0	0x01C0	0x01E0
...	...	...	...

图 5.10　Bank 模块交织方案

Bank 模块在初始化时创建读写 Bank 线程。读写 Bank 操作完成向对应的 SlavePort 发送读写响应 flit。Memory 模块接收 SlavePort 发送过来的 flit，因为对读写请求的操作不同，Memory 模块会根据读写请求的区别分到不同的 FIFO 中。Memory 对发送过的 flit 的处理流程图如图 5.11 所示：

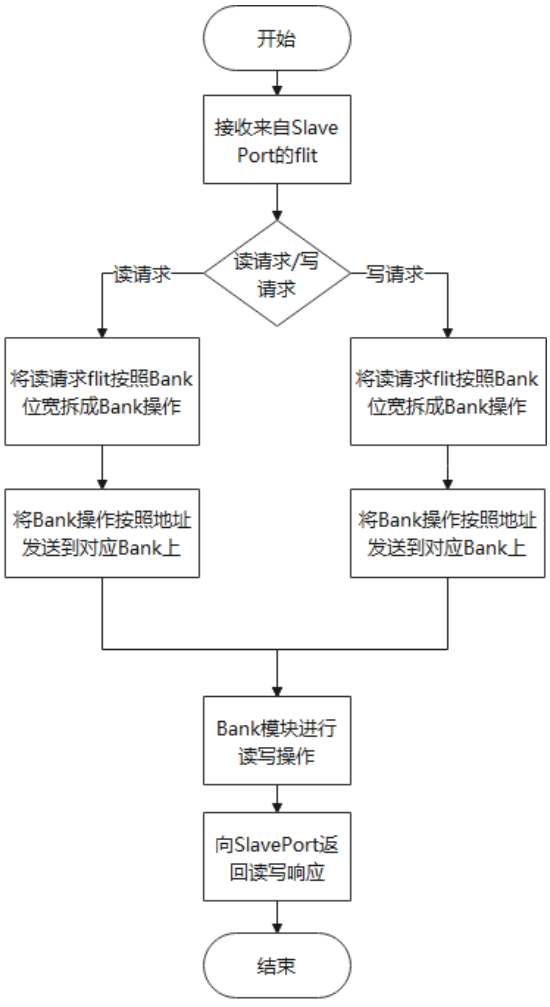

图 5.11　Memory 模块处理流程图

Bank 写操作结束时，Memory 模块会统计一个 Burst 完成 flit 的数量，当最

后一个 flit 操作完成后会统一发送一个写响应回 SlavePort。而 Bank 读操作则时完成一个 flit 就发送一个 flit 的读响应到 SlavePort 上。最后由总线互联模块发送回发送读写请求的 Master 设备，完成一整个读写操作。

5.5　设计空间探索相关模块的设计与实现

设计空间探索模块实现的功能是对我们所需要探索的设计目标在设计空间中寻找最优或者是较优设计。设计空间模块主要分为三个模块：训练集生成模块、预测模型生成模块以及遗传算法探索模块。设计空间探索模块的整体流程图如图 5.12 所示：

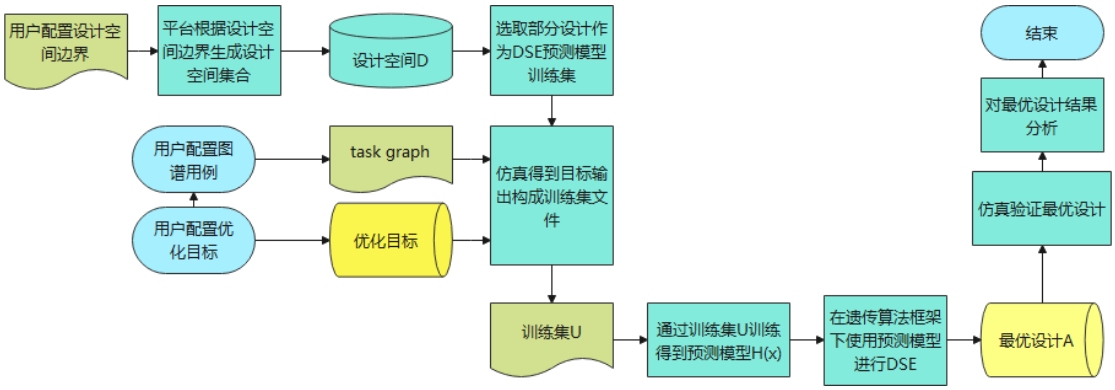

图 5.12　设计空间探索模块整体流程图

接下来主要介绍训练集生成模块和预测模型生成模块的具体设计与实现以及设计空间探索结果分析。

5.5.1　训练集生成模块的设计与实现

训练集生成模块主要包括设计参数、设计边界与目标值的选取、设计空间的生成以及根据设计空间以及目标值生成下一步预测模型所需要的训练集文件。在仿真平台中我们选取可配置参数作为我们的设计参数，并根据实际情况设置各个设计参数的设计边界，具体设计参数及设计边界如表 5.1 所示：

我们选取了七个参数作为我们设计空间探索的设计参数值，设计参数分别为：PL2 内存池大小、PL2 每个内存单元大小、SL2 内存池大小、SL2 每个内存单元大小、Cu 数量、Dsp 数量以及总线位宽。这些参数均会影响最后仿真结果。这些设计参数的组合最后构成的设计空间包括了超过 56 万个不同的设计配置。

我们设计空间探索主要探索的方向主要是时延以及核的使用方面。我们最终选取了仿真时延、每个 TTI 核的利用率、每个 TTI 核的一致性为我们设计空间探索的目标值。接下来我们说明每个目标值的统计标准。仿真时延在数据库文件中有着直接的记录，我们只需直接提取数据库中的信息即可，核的利用率在数

表 5.1　仿真平台设计空间

设计参数	设计参数取值	数量
PL2 Pool Size	100000-190000: 10000+	19
PL2 Unit Size	10-50: 10+	5
SL2 Pool Size	1500000-1850000: 50000+	7
SL2 Pool Size	10-50: 10+	5
Cu Num	1-5: 1+	5
Dsp Num	4-16: 2+	7
BUS Bit Width	64-512: 64+	8

据库中记录了每个时间片（0.002TTI）的核的利用率，记录方法已在 5.1.5 节介绍。我们只需将数据库中信息整合每个 TTI 的核的利用率即可。核的一致性指的是在同个 TTI 内所有核的利用率的方差，我们需要将之前统计的核的利用率进行计算即可。这就是我们设计空间探索的优化目标值。

我们生成训练集的流程图如图 4.9 所示。我们将 RatConfig 表中需要改变的参数值单独设置一页 Config 表，并将这一页的值与整张表中的相关的值相关联，通过 openpyxl 插件去修改 Config 表中的参数。这样我们就可以通过脚本自动改变设计参数去运行仿真，仿真结束后提取数据库中目标值信息，处理后将设计参数和目标值一同输出到训练集文件中。我们训练集文件大小为 6000，生成训练集模块一共运行了 6000 次仿真。训练集文件格式如图 5.13 所示：

```
200000, 10, 1500000, 10, 1, 4, 64, 5.73917183, 0.998, 0.979620728, 0.2, 0.0004153310310958721 5, 0.0004153310310958721 5, 0.0004153310310958721 5
200000, 10, 1500000, 10, 1, 12, 256, 4.832061332, 0.7640275805, 0, 0, 0.0015473600005001652, 0, 0
200000, 10, 1500000, 10, 2, 6, 448, 4.446532458, 0.998, 0.7132356186666666, 0.028387046, 0.0001314863741721313 4, 0.0001314863741721313 4, 0.0
200000, 10, 1500000, 10, 2, 16, 128, 3.072661964, 0.6052714880000001, 0.107360282 25, 0, 0.0044861808911912 73, 0.0044861808911912 73, 0
200000, 10, 1500000, 10, 3, 10, 320, 3.86860705, 0.9851735564, 0.014920831999999997, 0.0689428194333333 3, 0.0013062035885172 325, 0.1606521823078554 4, 0.1545393433920319 6
200000, 10, 1500000, 10, 4, 4, 512, 5.396966198, 0.998, 1.0, 0.2, 0.0, 0.0, 0.0
200000, 10, 1500000, 10, 4, 14, 192, 3.166264162, 0.6952896460000001, 0.11233229752380952, 0, 0.007889619597958473, 0.15989079143644913, 0
200000, 10, 1500000, 10, 5, 8, 384, 3.882946648, 0.998, 0.1856045694999998, 0.08745586983333334, 0.006864377893324 76, 0.4108291925402181, 0.4088312148587 21
200000, 10, 1500000, 20, 1, 4, 64, 5.688281258, 0.998, 0.980221328, 0.2, 0.0003912048945382652 7, 0.0003912048945382652 7, 0.0003912048945382652 7
200000, 10, 1500000, 20, 1, 12, 256, 4.770970076, 0.7640275805, 0, 0, 0.0015473600005001652, 0, 0
```

图 5.13　训练集文件格式示例

5.5.2　预测模型生成模块设计与实现

我们在遗传算法进行设计空间探索过程中，在不断迭代每一代种群时如果每一次都要进行仿真从而得到对应设计参数的目标值，整体设计空间探索的时间就会十分的漫长。于是，我们使用机器学习生成的预测模型去代替实际的仿真平台去进行遗传算法设计空间探索的过程，这样能够大大减少设计空间探索的时间。

我们多目标预测模型采用先对不同目标值分别进行机器学习的过程，得到对应 7 个目标值的 7 个预测模型。我们将这 7 个预测模型通过一个 multi_model 类集成到一起，通过 multi_model 类的方法我们可以通过输入设计参数从而得到相应的目标值。多目标预测模型集成的各个目标值的准确度（以 R2 确定系数为标准）如表 5-2 所示：

表 5.2　多目标预测模型误差值

时延	TTI0 核利用率	TTI1 核利用率	TTI2 核利用率
0.99824568	0.9999998	0.9982341	0.9873023
TTI0 核一致性	**TTI1 核一致性**	**TTI2 核一致性**	
0.8638929	0.9831229	0.8559540	

从表中我们可以清楚的看出对于多个目标值而言，每个目标值的误差值不超过 0.15。整个预测模型可以比较高效精准的预测出目标值，与真实仿真平台的仿真结果相差不大，为了更加精准的看出预测模型与仿真平台的结果差距，我们将 1200 次仿真结果作为验证集与预测模型进行对比，仿真结果与预测结果对比如图 5.14 所示：

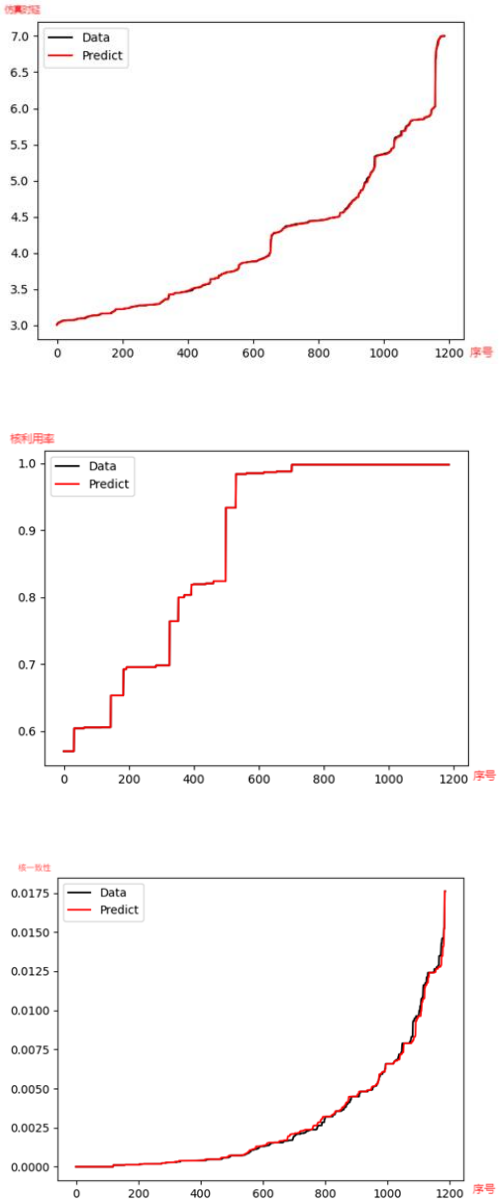

图 5.14　仿真结果与预测结果对比图

5.5.3　设计空间探索结果分析

在得到多目标预测模型之后，我们将使用多目标预测模型使用遗传算法进行设计空间探索。遗传算法执行过程中我们使用 NGSA-II，我们通过交叉变异等操作产生子代，将父代和子代放到一起进行快速非支配排序，构造出不同等级的非支配解集。按照需求计算每一代所有个体的拥挤距离，并根据拥挤度比较构造下一代种群。

遗传算法我们遗传代数设置为 200 代，每代种群数量为 50，变异概率为 0.1。最终演化种群中的每个基因型经过仿真平台仿真运行得出相应的目标值。最终我们得到了 50 个设计方案，并且将这 50 个设计方案在仿真平台上进行仿真，得到了相应的目标值。我们根据这 50 个设计方案以及对应的目标值进行探索，得到了以下结论。

我们先将三个方面的 50 个目标值与原始设计相对比得到图 5.15。

Index	PL2_Pool_Size	PL2_Unit_Size	SL2_Pool_Size	SL2_Unit_Size	CU_Num	BUF_Num	BUS_Bit_Width	Sim_Time	Core_Usage_TTI_0	Core_Usage_TTI_1	Core_Usage_TTI_2	Core_Consistency_TTI_0	Core_Consistency_TTI_1	Core_Consistency_TTI_2
0	120000	20	1850000	50	4	16	64	3.022076718	0.605447012	0.096327824	0	0.003915966	0.003915966	0
1	190000	30	1850000	40	3	10	320	4.920066942	0.985173556	0.014828998	0	0.000437207	5.229458-05	0
2	120000	40	1600000	20	5	8	64	3.817027952	0.998	0.18276169	0.130821223	0.001433045	0.478704279	0.478704279
3	100000	30	1600000	40	4	8	512	4.751218006	0.998	0.254562749	0	0.000510049	0.000510049	0
4	100000	30	1550000	10	2	8	512	3.798726924	0.998	0.232395405	0.2	0.004499009	0.611189939	0.004499009
5	100000	50	1650000	20	4	14	512	3.093965084	0.695289646	0.135804823	0	0.006603266	0.078948567	0
6	100000	40	1650000	20	5	8	64	3.68966223	0.998	0.2243591 75	0.2	0.002034939	0.628112996	0.002034939
7	170000	50	1800000	20	2	6	64	5.988512764	0.998	0.680380094	0	6.5843E-05	6.5843E-05	0
8	110000	50	1650000	10	1	12	512	4.511086836	0.764027581	0	0	0.00154736	0	0
9	120000	50	1800000	50	2	16	448	3.049215444	0.605271488	0.097869434	0	0.004486181	0.051108696	0
10	110000	30	1650000	20	2	6	448	4.396264016	0.998	0.713235619	0.018930386	0.000131486	0.000131486	9.167888-05
11	190000	10	1550000	10	2	6	64	4.463999624	0.998	0.713235619	0	0.000131486	0.000131486	0
12	100000	50	1650000	40	5	10	384	4.468461704	0.983904218	0.008338177	0	0.000183757	5.87936E-05	0
13	140000	30	1650000	20	3	16	512	2.997748338	0.603916318	0.107360282	0	0.005057244	0.005057244	0
14	160000	30	1650000	20	5	8	192	3.66104802	0.998	0.258356711	0.198168254	0.007545162	0.598563005	0.007545162
15	170000	10	1500000	40	1	16	64	3.180344358	0.5695896	0.154687811	0	0.001302076	0.001302076	0
16	170000	50	1650000	20	2	14	512	5.078451844	0.680071136	0	0	0.000127124	0	0
17	120000	40	1800000	40	2	10	320	3.55032117	0.986754304	0.191497268	0.021550416	0.001033283	0.382349463	0.000216481
18	120000	10	1650000	50	2	16	448	3.072754072	0.605271488	0.109535591	0	0.009632083	0.157429926	0
19	110000	50	1550000	30	4	8	384	3.715893916	0.998	0.263095937	0.2	0.000720885	0.560321757	0.000720885
20	130000	40	1850000	50	4	8	448	3.674879242	0.998	0.2164848	0.177070253	0.0067421	0.585009518	0.0067421
21	150000	30	1650000	20	2	14	384	3.13492486	0.695339301	0.129698856	0	0.005262509	0.121858982	0
22	160000	50	1700000	40	2	8	448	3.843645406	0.998	0.207358936	0.188326543	0.003318983	0.580449324	0.003318983
23	180000	40	1550000	30	5	14	64	3.144283298	0.697960713	0.118742828	0	0.006016822	0.122588787	0
24	110000	50	1600000	40	4	8	448	4.85706892	0.998	0.254562749	0	0.000510049	0.000510049	0
25	190000	50	1500000	30	5	8	320	3.642313918	0.998	0.330211407	0.183491292	0.003197721	0.568841863	0.003197721
26	130000	50	1650000	20	3	12	192	3.279768796	0.823859754	0.16244356	0	0.005735915	0.195528618	0
27	110000	50	1850000	20	3	8	448	3.946079098	0.998	0.155319363	0.079361876	0.004448324	0.44243132	0.442182265
28	150000	40	1650000	30	5	16	512	3.0259983	0.60746804	0.102589043	0	0.007486961	0.053788072	0
29	110000	50	1850000	40	1	14	128	3.107720994	0.653088574	0.13808911	0	0.005185113	0.005185113	0
30	120000	20	1650000	50	4	16	448	3.346987546	0.605447012	0.130667455	0	0.004860429	0.005557607	0
31	120000	10	1850000	10	5	6	64	4.21007044	0.998	0.604039755	0.025618852	0.001425215	0.341398023	8.33345E-06
32	120000	50	1500000	30	5	12	320	4.46405535	0.814711217	0	0	0.000357644	0	0
33	120000	30	1500000	50	2	12	448	3.298892278	0.821642342	0.177653355	0	0.002433192	0.16987736	0
34	100000	50	1750000	30	4	4	448	5.496027674	0.998	1	0.2	0	0	0
35	170000	30	1500000	40	2	10	192	3.44982299	0.986754304	0.209929812	0.002188476	0.001301131	0.362171449	0.00070134
36	150000	30	1800000	30	2	8	384	4.018038859	0.998	0.177653473	0.078502537	0.003579728	0.405468892	0.450140129
37	140000	50	1500000	20	3	10	448	3.402110818	0.985173556	0.184372247	0	0.000545396	0.379840872	0
38	100000	50	1550000	20	4	4	448	5.49750607	0.998	1	0.2	0	0	0
39	140000	50	1650000	40	3	10	448	3.330658056	0.986559798	0.212863093	0	0.004882176	0.308869927	0
40	140000	50	1750000	20	3	10	320	3.330658056	0.986559798	0.212863093	0	0.004882176	0.308869927	0

图 5.15　设计空间探索结果仿真目标值结果对比图

根据图 5.15 针对仿真时延、核利用率以及核一致性三个目标值，我们可以得到相对于这三个目标值各自方向的最优解。下面我们对三个目标值单独进行分析并将设计空间探索所得到的最优解集放在一起进行比较。我们基于优化目标将所有方案的结果以及原始设计方案放到一起进行比较。比较结果如图 5.16 至 5.18 所示：

图 5.16 中横坐标为设计方案序号，纵坐标为仿真时延，红色虚线为初始设计方案及其对应的仿真时延。由图 5.16 及对应序号的设计方案可以看出序号为 35、43、20、27、14、7、48、12、13、23 的方案较好。并且通过各个设计方案值之间的结果对比可以得出影响时延结果的主要因素为核的数量以及总线位宽等等。

由图 5.17 及对应序号的设计方案可以看出序号 33、28、42、29、47、39、36、38、4、26 的方案较好。并且通过各个设计方案之间的结果对比可以得出影响核

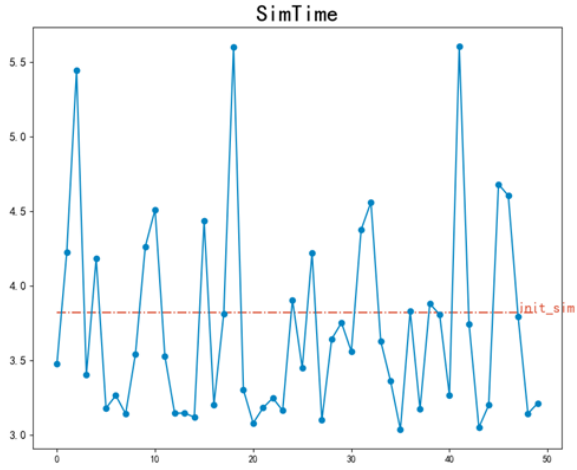

图 5.16　50 组设计方案时延结果对比

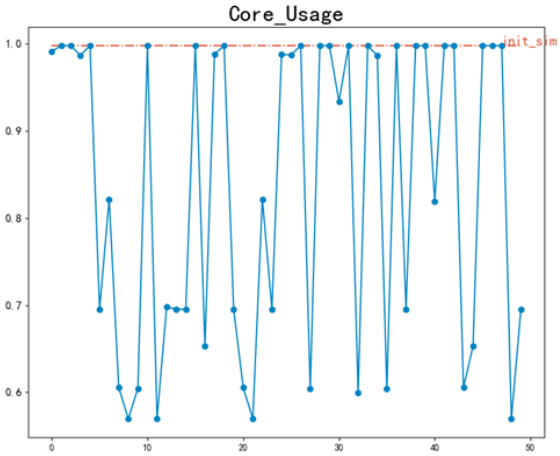

图 5.17　50 组设计方案核利用率结果对比

利用率的主要因素为核的数量及 CU 数量等等。

由图 5.18 及对应序号的设计方案可以看出序号 2、18、41、4、31、15、10、26、9、32 的方案较好。并且通过各个设计方案之间的结果对比可以得出影响核利用率的主要因素为核的数量及 CU 数量等等。而且对比图 5.16 和图 5.17 可以发现核利用率和核一致性的优化方向和影响因素大致相同。

但针对多个目标值同时考虑而言，我们需要找到多个目标值均衡的几个设计方案。从图 5.16 至图 5.18 可以看出时延和核的利用率优化方向相反。仿真时延低的几种方案，设计参数中 Dsp 数量较大，但由于 Dsp 数量较大，导致核的利用率和一致性效果较差。而核的利用率和核的一致性优化方向相同，最优的几个设计方案设计参数大致相同。于是我们就核利用率和仿真时延两个方面进行考虑，结果图如图 5.19 所示：

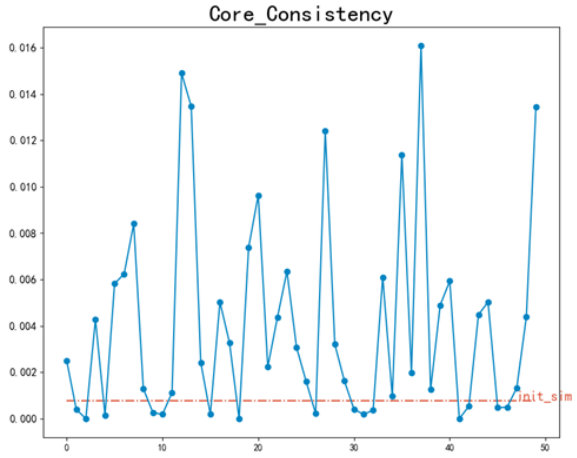

图 5.18　50 组设计方案核一致性结果对比

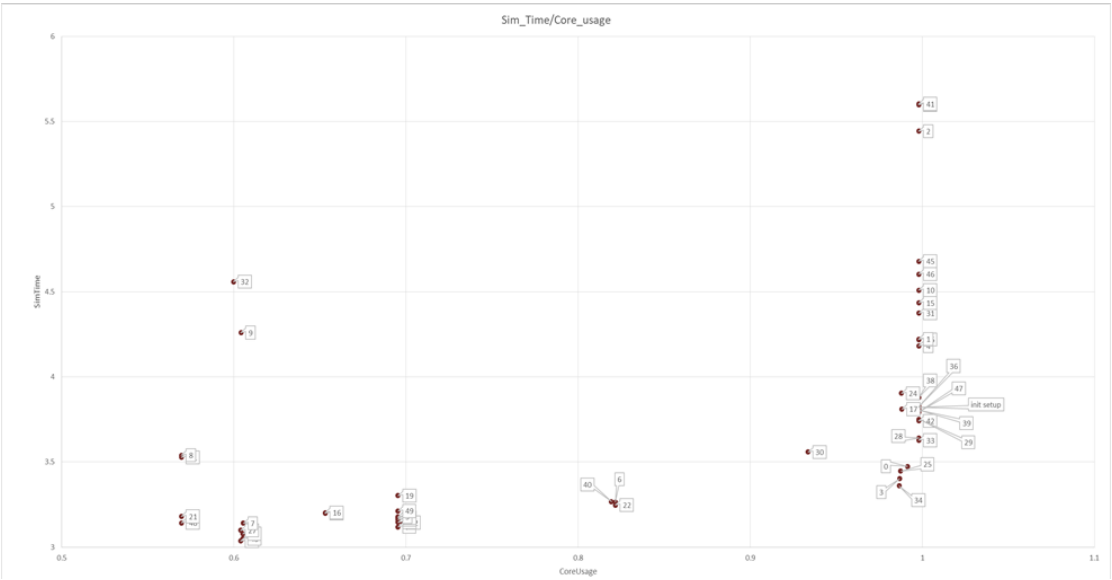

图 5.19　仿真平台结果分析图

从图 5.19 中我们可以找到 6 种方案在仿真时延和核利用率两个方面表现都很优秀的设计方案如表 5.3 所示:

表 5.3　最优设计方案设计参数

Index	PL2_Pool_Size	PL2_Unit_Size	SL2_Pool_Size	SL2_Unit_Size	Cu_Num	Dsp_Num	BUS_Bit_Width
33	140000	30	1550000	20	4	8	448
25	170000	40	1650000	10	5	10	320
3	170000	20	1800000	50	2	10	512
42	100000	20	1800000	20	4	8	512
0	100000	10	1750000	50	4	10	128
25	170000	30	1650000	10	5	10	320

从整体设计参数以及目标值的变化，我们可以看出增加 Cu 数量可以增快仿真速度，提高总线带宽可以使核的利用率提高同时还可以增快仿真速度。

5.6　本章小结

本章的工作主要是在第四章概要设计的基础上，详细的阐述了仿真平台中硬件平台的构建、Processor 模块以及 Memory 模块的具体实现，介绍了设计空间探索流程的具体设计和实现。实现整个仿真平台的搭建以及基于仿真平台的设计空间探索流程，最后得到了设计空间探索结果种群，并对最终得到的设计空间探索结果进行了分析，得到了最终的最优设计方案以及设计方案优化的目标

在本章基础上，下一章节我们将完成对仿真平台的测试。

第 6 章　系统测试与结果分析

本章主要针对系统对仿真平台的仿真平台的需求，对整个仿真平台进行功能性和非功能性测试。

6.1　仿真平台测试

6.1.1　测试环境

因为公司项目的保密性，所有测试环节均在公司电脑上完成，测试电脑的配置如下所示：

CPU:：Intel(R) Core(TM) i7-8700 CPU @3.2GHz

内存：32G

磁盘：1T

操作系统：Windows10

系统类型：64 位操作系统

6.1.2　测试工具和方法

针对仿真平台，测试主要包括功能测试和非功能测试两个部分。功能测试主要从两个方面进行测试，一方面，在整个平台构建之前，我们需要对建模好的各个硬件实例进行功能测试，保证其功能正确。另一方面，在所有模块单独测试通过后，在整个仿真平台构建时，我们从程序的输入输出为角度，对仿真平台的各个模块进行测试。这里主要对硬件平台构建模块和仿真运行模块这两个模块的功能进行测试。针对仿真平台而言，我们需要得到有效的仿真结果，包括时延、任务的开始结束信息、内存信息以及核的使用情况等等。且仿真平台需要保证任务仿真执行时序与任务用例图的时序一致。

6.1.3　硬件模块功能测试

在对各个建模好的硬件模块进行功能测试主要测试的是实例化后的硬件模块时候能够接收来自其他模块的消息并即时触发执行任务的流程、自身的硬件功能以及能够正常与其他模块进行消息交互。在这个测试过程中，我们主要对 GeneralFifo 模块、Processor 模块以及 Memory 模块这三个模块进行测试。针对需求分析中对这三个模块提出的测试指标进行测试，测试结果如表 6.1、表 6.2 和表 6.3 所示：

如表 6.1 所示，我们对 GeneralFifo 模块进行了功能性测试，主要分为三个

表 6.1　GeneralFifo 模块测试表

序号	测试项	操作流程	预测结果	实际结果
1	能否正确的 存取对象	调用 GeneralFifo 的方法 函数来存取对象	通过调用函数方法 成功存取对象	正确
2	存取函数能否 触发事件	调用该模块的事件接口 去执行计数任务，存取对象后看 每一步计数值是否正确	每一步计数值正确	正确
3	该模块外部 接口方法是否正确	通过存取对象，调用方法函数， 观察结果是否正确	结果正确	正确

部分，测试项 1 测试 GeneralFifo 模块作为一个先入先出队列的存取对象的功能，通过该模块提供的外部接口方法，调用函数进行对象的存取，测试结果表示该功能正确。

测试项 2 测试 GeneralFifo 的事件机制是否能够被正确触发，GeneralFifo 模块提供内部的 event 事件接口，其他模块可以通过 Simpy 的 yield 机制去等待该 event 被触发。我们通过设置计数任务，每次 event 被触发，计数任务被触发且打印计数值，我们通过像队列中存取对象来触发对象，通过观察计数值，即可测试出 GeneralFifo 模块的事件机制是否正确。最后测试结果表明 GeneralFifo 模块的事件机制正确。

测试项 3 测试 GeneralFifo 模块提供外部接口和方法是否正确，这部分我们只需通过调用模块的外部接口即可测试，通过这部分测试表明 GeneralFifo 模块的外部接口正确。

表 6.2　Processor 模块测试表

序号	测试项	操作流程	预测结果	实际结果
1	DSP 模块处理能否 即时并准确的 接收任务实例	向 DSP 模块入口 Fifo 发 送任务实例，观察 DSP 模块 对任务实例的解析	DSP 模块即时接 收并正确解析 任务实例	正确
2	DSP 任务处理时延 是否与任务实例 中一致	DSP 接收并处理 任务结束，查看 仿真环境的时延	仿真环境的时延 变化与任务实例 中一致	正确
3	DMA 模块处理能否 即时并准确的 接收任务实例	向 DMA 模块入口 Fifo 发 送任务实例，观察 DMA 模块 对任务的解析	DMA 模块即时接 收并正确解析 任务实例	正确
4	DMA 模块能否根据 任务信息正确 搬入搬出数据	DMA 执行完数据搬 运任务，查看源地址与 目的地址的内存变化	源地址与目的地址 内存接收数据请求	正确
5	DMA 模块与 DSP 模块能否流水 线化执行任务	向 DMA 模块发送任务实 例观察整个任务能够 完整执行	整个任务链完整 且正确的执行	正确
6	HAC 模块处理能否 即时并准确的 接收任务实例	向 HAC 模块入口 Fifo 发 送任务实例，观察 HAC 模块 对任务实例的解析	HAC 模块即时接 收并准确解析任 务实例	正确
7	HAC 模块能否正 确执行任务实例	HAC 执行完整个任务实例 ，从对应 HAC 的日志中提 取任务执行信息	任务执行信息与任 务实例中信息一致	正确

如表 6.2 所示，我们对 Processor 模块的三种硬件模型进行了功能性测试。主

要按照不同的硬件模型分为三部分：

测试项 1 和测试项 2 主要测试 DSP 模块的硬件功能是否正确。测试项 1 通过构造测试用的任务实例，并将测试用例发送到 DSP 模块的入口 Fifo 上，这部分测试当 DSP 模块的入口 Fifo 接收到任务实例时，能不能立即处理任务实例，并能够正确的将任务实例中的任务信息解析出来；测试项 2 在测试项 1 的基础上，测试了 DSP 模块处理任务实例，任务是否能够被正确执行，DSP 任务只体现时延的变化，只要保证在任务执行结束后，仿真环境的仿真时延变化与任务实例中信息一致即可。从两项测试的结果来看，DSP 模块的硬件功能符合要求。

测试项 3~5 主要测试了 DMA 模块的硬件功能以及 DMA 模块和 DSP 模块流水线执行任务的能力。测试项 3 通过构造测试用的任务实例，将测试用例发送到 DMA 模块的入口 Fifo 上，DMA 模块能够立即取出任务实例并正确的解析任务实例中的任务信息；测试项 4 在测试项 3 的基础上对 DMA 模块对任务实例的处理能力进行测试，DMA 模块负责数据的搬入搬出，我们只需要查看源地址与目的地址对应的内存模块查看是否收到来自 DMA 的数据请求；测试项 5 是测试 DMA 模块和 DSP 模块的流水线执行任务链的能力，在业务场景中常常会出现数据搬运后立即对数据进行处理的场景，为了满足这种业务场景，测试项 5 通过构造带有数据搬运以及数据处理的任务链级的任务实例，观察 DMA 模块执行完数据搬运能否通知目标 DSP 实例去执行数据处理任务。根据测试结果显示，DMA 模块的硬件功能以及 DMA 模块和 DSP 模块流水线执行任务的能力符合要求。

测试项 6 和测试项 7 主要测试 HAC 模块对任务的处理能力，测试项 6 通过构造在 HAC 模块上处理的特定任务实例，HAC 模块能能够立即接收任务实例并且正确的解析出任务实例中的任务信息；测试项 7 通过查看对应 HAC 的日志去提取 HAC 模块在执行测试用例时的具体任务信息，通过任务的执行信息，我们从时延变化、数据搬运等方面可以看出任务有没有正确执行。从两个测试项的测试结果可以看出 HAC 模块对任务实例的处理能力符合要求。

表 6.3　Memory 模块测试表

序号	测试项	操作流程	预测结果	实际结果
1	能否正确的接收读写数据请求	发送读写数据请求到入口 Fifo 上，观察其解析结果	解析结果正确	正确
2	能否正确的处理读写数据请求	查看对应模块日志文件中的数据处理信息	数据处理信息与数据请求中信息一致	一致
3	能否与其他模块正确的消息交互	查看处理完任务后，模块的返回信息	返回信息发送到对应模块	正确

如表 6.3 所示我们对 Memory 模块对数据请求的接收、处理以及消息交互的能力。测试项 1 通过构造读写数据请求发送到 Memory 模块的入口 Fifo 上，

Memory 模块需要区分为读数据请求以及写数据请求，再解析出数据处理的任务信息。测试结果表明 Memory 模块能够及时接收数据请求并对数据请求进行正确解析。

测试项 2 在测试项 1 的基础上测试 Memory 模块对数据处理请求的执行能力。Memory 模块在解析完数据处理请求之后，需要根据请求的地址来将请求分为更小一级的 Bank 操作符，读写数据均按照 Bank 级别进行操作，我们在对应 Memory 模块的日志文件中可以查看 Bank 读写操作的执行情况。测试结果为数据处理请求的执行信息与数据处理请求所携带的任务信息一致，表明 Memory 模块能正确执行数据处理请求。

测试项 3 测试 Memory 模块在数据处理结束后即时返回数据处理完成信息的能力。Memory 模块在完成数据处理请求之后，需向发送请求的模块返回请求完成的消息，我们只需查看消息内容以及是否正确时间发送即可。测试结果表明 Memory 模块在请求完成之后返回的消息正确且返回消息事件正确。

6.1.4　硬件平台构建模块测试

硬件平台构建模块的功能主要是解析硬件配置文件、实例化硬件模型以及构建硬件平台。这三个功能将硬件配置文件中的硬件平台有关信息解析出来实例化硬件模型，并通过总线模块将各个模型连接起来构建成为一个完整的硬件平台。硬件平台构建模块的测试结果如表 6.4 所示：

表 6.4　硬件平台构建模块测试表

序号	测试项	操作流程	预期结果	实际结果
1	是否能够解析硬件配置文件的信息	运行 Task 类中 Load 函数	硬件配置信息包含在类里	正确
2	能否正确实例化硬件模型	运行 PlatformBuilder 类中 init 函数	所有模型类成功创建	正确
3	能否正确构建硬件平台	运行 SaeSimulator 类中 build_platform 函数	平台正确构建	正确

如表 6.4 所示我们对硬件平台构建模块进行了测试，测试项 1 通过运行 PlatformBuilder 类中 Load 函数对硬件配置文件进行解析，我们可以调试过程中在类中找到所有硬件配置信息，通过验证所有信息均已解析。

测试项 2 通过运行 PlatformBuilder 类中 init 函数根据测试 1 项中解析并保存的配置信息将硬件平台中的所有硬件模型一一实例化，我们会通过对实例化的模型与硬件配置文件中的模型类型与个数进行对比，来判断是否所有模型均已正确实例化。

测试项 3 中通过运行 SaeSimulator 类中的 build_platform 函数将测试项 2 中实例化的所有硬件模块根据硬件配置文件中的路由信息连接起来，构建形成完整的仿真平台。仿真平台的功能实现主要通过后续仿真运行结果来进行测试，仿

真平台的构建过程的输出信息如图 6-1 所示：

```
SAESimulator.V801.R0_SUB.DSPS1.DSS5.DSP21
SAESimulator.V801.R0_SUB.DSPS1.DSS5.DSP22
SAESimulator.V801.R0_SUB.DSPS1.DSS5.DSP23
SAESimulator.V801.R0_SUB.DSPS1.DSS6.DSP24
SAESimulator.V801.R0_SUB.DSPS1.DSS6.DSP25
SAESimulator.V801.R0_SUB.DSPS1.DSS6.DSP26
SAESimulator.V801.R0_SUB.DSPS1.DSS6.DSP27
SAESimulator.V801.R0_SUB.DSPS1.DSS7.DSP28
SAESimulator.V801.R0_SUB.DSPS1.DSS7.DSP29
SAESimulator.V801.R0_SUB.DSPS1.DSS7.DSP30
SAESimulator.V801.R0_SUB.DSPS1.DSS7.DSP31
SAESimulator.V801.R1_SUB.DSPS2.DSS8.DSP32
SAESimulator.V801.R1_SUB.DSPS2.DSS8.DSP33
SAESimulator.V801.R1_SUB.DSPS2.DSS8.DSP34
SAESimulator.V801.R1_SUB.DSPS2.DSS8.DSP35
SAESimulator.V801.R1_SUB.DSPS2.DSS9.DSP36
SAESimulator.V801.R1_SUB.DSPS2.DSS9.DSP37
SAESimulator.V801.R1_SUB.DSPS2.DSS9.DSP38
SAESimulator.V801.R1_SUB.DSPS2.DSS9.DSP39
SAESimulator.V801.R1_SUB.DSPS2.DSS10.DSP40
SAESimulator.V801.R1_SUB.DSPS2.DSS10.DSP41
SAESimulator.V801.R1_SUB.DSPS2.DSS10.DSP42
SAESimulator.V801.R1_SUB.DSPS2.DSS10.DSP43
SAESimulator.V801.R1_SUB.DSPS2.DSS11.DSP44
SAESimulator.V801.R1_SUB.DSPS2.DSS11.DSP45
SAESimulator.V801.R1_SUB.DSPS2.DSS11.DSP46
SAESimulator.V801.R1_SUB.DSPS2.DSS11.DSP47
SAESimulator.V801.R2_SUB.TPS0.TP0.TP0
SAESimulator.V801.R2_SUB.TPS0.TP1.TP1
SAESimulator.V801.R2_SUB.TPS0.TP2.TP2
SAESimulator.V801.R2_SUB.TPS0.TP3.TP3
PlatForm Build Complete

Process finished with exit code 0
```

图 6.1　硬件平台构建输出消息

6.1.5　仿真平台运行模块测试

仿真平台运行模块功能主要要保证输入的任务用例能够正确的解析任务图信息、调度模块能够按照任务执行顺序正确读取任务、仿真平台能够将整个任务图按序执行完毕。这些功能将任务图中信息解析出来并由调度模块读取分配执行，使整个任务用例在仿真平台上完整的运行完毕。同时我们也需要保证任务执行时序以及关键信息的准确性。仿真平台运行模块的测试结果如表 6.5 所示:

如表 6.5 我们对仿真平台仿真运行模块进行了测试，测试项 1 通过运行 TaskGraphMgnt 模块的 Load 函数去解析输入的用例文件。最后在调试过程中检测任务图信息是否完整包含在一个 Graph 类中，测试结果显示任务图信息完整。

测试项 2 通过运行 Schedule 类中 load 函数来读取 Graph 类中的任务信息，我们通过写测试函数，将任务图信息按序全部打印出来，测试结果显示所有任务按序输出显示在终端上，结果正确。

测试 3~5 是为了测试各个硬件模块的功能是否正确。我们通过手动构造各种类型的任务用例到硬件模块的接口 FIFO 上来模拟调度器模块分配任务的流程，看各个硬件模块能不能正确执行任务用例，且任务执行结果是否和任务描述是否一致。测试结果显示各个硬件模块均能正确执行调度器分配的任务。

测试项 6 通过运行项目主函数中的 SimRun 函数运行仿真，函数参数包括任务用例名、任务图执行次数、输出数据库位置、Log 文件是否记录等等。我们在

表 6.5　仿真平台运行模块测试表

序号	测试项	操作流程	预期结果	实际结果
1	是否能够完整解析任务图信息	运行 TaskGraphMgnt 类中 Load 函数	任务图信息完整包含在类中	正确
2	能否正确按序读取任务实例	运行 Schedule 类中 load 函数	将任务图中任务按序读取	正确
3	DMA 模块能否正确执行任务	发送 DMA 任务到硬件模块上执行	数据能够正确搬运到相应位置	正确
4	DSP 模块能否正确执行任务	发送 DSP 任务到硬件模块上执行	任务执行时延与任务信息一致	正确
5	HAC 模块能否正确执行任务	发送 HAC 任务到硬件模块上执行	任务执行结果与任务信息一致	正确
6	仿真平台能否完整运行整个任务图	运行主函数中的 SimRun 函数	任务图中所有任务正确执行	正确
7	仿真运行结果时序是否正确	对比数据库仿真信息与任务用例信息	两者信息一致	正确

硬件平台搭建完成的基础上，使用任务用例输入执行仿真，仿真运行输出信息如图 6-2 所示。且在数据库中将任务执行信息与任务用例中任务信息相对比，执行时序正确无误。

```
1855790701 : HacTaskInst: ('HAC_SYNC', 6) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 7) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 8) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 9) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 10) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 11) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 12) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 13) times: 1  is finished in ALA0
1855790701 : HacTaskInst: ('HAC_SYNC', 14) times: 1  is finished in ALA0
1855848451 : HacTaskInst: ('LCDP', 60) times: 1 is finished in  LCDP1
1855906201 : HacTaskInst: ('LCDP', 61) times: 1 is finished in  LCDP2
1855963951 : HacTaskInst: ('LCDP', 62) times: 1 is finished in  LCDP3
1860715757 : HacTaskInst: ('LCDP', 63) times: 1 is finished in  LCDP0
1860715757 : HacTaskInst: ('HAC_SYNC', 15) times: 1  is finished in ALA0
1905006701 : KernelTaskInst: ('CRCRpt', 0) times: 0 is finished in dsp 8
1905006701 : KernelTaskInst: ('CRCRpt', 0) times: 2 is finished in dsp 9
1906577505 : KernelTaskInst: ('CRCRpt', 0) times: 2 is finished in DMA1
1906589055 : KernelTaskInst: ('CRCRpt', 0) times: 0 is finished in DMA0
1909931757 : KernelTaskInst: ('CRCRpt', 0) times: 1 is finished in dsp 10
1910717161 : KernelTaskInst: ('CRCRpt', 0) times: 1 is finished in DMA2
simulator finished in 1910717161 us

Process finished with exit code 0
```

图 6.2　仿真运行输出消息

由上面两个方面的验证能够保证仿真平台的功能的正确性。能够满足业务需求以及后续设计空间探索的需求。

6.1.6　仿真平台性能测试

在对仿真平台功能方面进行测试的同时，我们需要对仿真平台的性能方面进行测试。我们在需求分析阶段提出了几个对于仿真平台以及设计空间探索流程的性能测试指标，我们针对需求分析中表 3-2 中指标进行测试，测试结果如表 6.6 所示：

表 6.6　仿真平台性能测试表

序号	测试项	预期结果	实际结果
1	仿真硬件平台搭建时间	<=5s	4.76s
2	单次单小区仿真执行时间	<=15s	13.56s
3	仿真结果时序是否与用例一致	True	True
4	单次一代遗传算法执行时间	<=500ms	398ms
5	设计空间探索结果是否全局最优	True	True

如表 6.3 所示，通过对仿真平台以及设计空间探索流程进行多次运行，得到的测试结果如上表所示。由测试结果可以看出仿真平台的单次单小区的执行时间低于预期值，满足后续设计空间探索流程的需求。针对需求分析阶段提出的性能需求的几大指标而言，测试结果都满足性能测试指标需求。

6.2　本章小结

本章主要对仿真平台做了功能上的测试，仿真平台能够正确的构建硬件平台，并能够在硬件平台上运行任务用例，保证任务仿真的时序正确。同时也对设计空间探索的结果进行了分析，并得到了一些结论和在所有目标值方面比较优秀的设计方案。

第 7 章　结果与展望

本章主要用来总结本文的工作，总结本文的创新点和不足，并给出在未来工作中的展望和希望研究的部分。

7.1　论文总结

目前部门内的仿真建模平台是基于 SystemC 的，模型建模做的十分精密，但因此整个仿真平台也十分复杂，仿真运行比较慢，运行一次用例仿真时间较长。因此，为了解决仿真时间长和仿真建模繁琐的同时，也为了满足后续设计空间探索的要求，本文设计并实现了基于 Simpy 的轻量级仿真平台，并基于轻量级仿真平台打通了设计空间探索流程。

在前期的调研过程中，我们最终确定了以 Simpy 为仿真环境，以 python 为开发语言，这样大大减少了开发时间。以之前基于 SystemC 的 SAE 仿真平台各个模块的建模为原型，重新实现了各个模块的建模。实现了硬件模块建模、硬件平台搭建、用例文件解析、任务调度以及数据库输出的仿真平台功能。并兼容原有的 SAE 仿真平台，与原有的仿真平台用例输入和硬件配置文件输入一致，可以实现一个用例在两个仿真平台上运行，这样了降低了后续开发人员的工作量。

在设计空间探索流程中，我们通过调研现在业内设计空间探索以及多目标优化的案例，最终确定了以进化算法为设计空间探索的方法。为了简化进化算法的每一代种群的计算过程，我们采用将仿真过程通过机器学习拟化为一个预测模型，这样将执行一次设计空间探索流程的时间从 4 天缩短到 2 分钟，这样研究人员可以将更多的时间却研究设计空间探索的结果。

基于以上方面，本文实现的轻量级仿真平台可以实现业务仿真并大大减少了仿真时间，并在后续一些方案设计中体现了良好的性能。设计空间探索流程的打通为部门内部方案设计提供了新思路和新方法。

7.2　问题和展望

本文实现的仿真平台虽然能够对比原平台能够实现大部分功能，但调度器部分在原型设计过程中仍十分简陋，资源调度方面不够灵活。有些硬件模块没有实现，定时器功能没有实现，这些都有待后续完善。

在设计空间探索方面，整个流程虽然打通，但是整个流程的精度并没有很高，只是对现有的仿真平台做出一个针对性的探索，并没有对更多的平台进行适配，甚至不仅对仿真平台，还可以对直接从单板上提取的输入输出的格式进行适

配。在接下来的完善过程中，可以改进预测模型的部分，可以通过这方面去适配更多的输入格式，通过调整进化算法去改善整个设计空间探索流程的精度

附录 A　缩略词对照表

缩略词	英文全称	中文对照
SoC	System on Chip	片上系统
ESL	Electronic System Level	电子系统级
RTL	Register transfer Level	寄存器传输级
TLM	Transaction Level Modeling	事务级建模
HLS	High Level Synthesis	高层次综合
DSE	Design Space Exploratiom	设计空间探索
EDA	Electronic Design Automation	电子设计自动化
TLM	Transaction Level Modeling	事务级建模
MPSoC	Multi-core System on Chip	片上多处理器系统
Fifo	first in first out	先入先出
DSP	Digital Signal Processing	数字信号处理
DMA	Direct Memory Access	直接存储器访问
HAC	Hardware Accelerator	硬件加速器