Specification Platform for Library IP Development

Jung Kyu Chae

Université Pierre et Marie Curie

Ecole doctorale informatique, télécommunications et électronique

Laboratoire d'informatique de Paris 6 / Equipe Circuits Intégrés Numériques et Analogiques

Plateforme de spécification pour le développement de bibliothèques de cellules et d'IPs

Par Jung Kyu CHAE

Thèse de doctorat d'informatique

Dirigée par Habib MEHREZ et Roselyne CHOTIN-AVOT

Présentée et soutenue publiquement le 9 Juillet 2014

Le jury est composé de :

M. Philippe COUSSY	Maître de conférence	Rapporteur
M. Naohiko SHIMIZU	Professeur	Rapporteur
Mme. Emmanuelle ENCRENAZ	Maître de conférence	Examinateur
M. Habib MEHREZ	Professeur	Directeur de thèse
Mme. Roselyne CHOTIN-AVOT	Maître de conférence	Co-directrice de thèse
M. Jean-Arnaud FRANCOIS	Manager de l'équipe HW/SW	Co-directeur de thèse

Acknowledgements

I would like to express my gratitude, appreciation and sincere thanks to my supervisors Prof. Habib MEHREZ and Assoc. Prof. Roselyne CHOTIN-AVOT for their excellent guidance, helpful and useful discussions, and continuous encouragement.

I would like to express my thanks to my ST supervisor Dr. Jean-Arnaud François for giving me the opportunity to carry out my thesis within Central CAD & Design Solutions Department at STMicroelectronics. Moreover, I also sincerely thank him for his support, guidance, encouragement and advice.

I am really grateful to Paul MOUGEAT, who provided valuable expertise, insightful comments and constructive critiques. His cool personality and sense of humor always inspired me. I have learnt a lot from him.

I wish to express my deepest gratitude to Dr. Marc QUAST for the helpful and valuable discussions, constructive remarks and encouragement which helped me carry out the PhD thesis work in an efficient manner.

I am especially grateful to my colleagues Severine BERTRAND, Sylvain LANDELLE, Pierre-Francois OLLAGNONG, Syed Shahkar KAKAKHAIL, Brice SERVAIS, Frederic ESTEBAN, Lahcen HAMOUCHE, Sophie RABADAN, Lionel MAIAUX, Pierre MONNOT, Aymen ALBOUCHI, Christophe BOURELY, Sami ABBADI, Mohamed Karim KSONTINI and Arnaud MAGRY for their support both in terms of resources and encouragement.

My special thanks go to my family members who have been extremely understanding and supportive of my studies. This thesis is dedicated to my parents.

Abstract

A design platform (DP) is a total solution to build a System-On-Chip (SOC). DP consists of a set of libraries/IPs, CAD tools and design kits in conformity with the supported design flows and methodologies. The DP specifications provide a wide range of information from technology parameters like Process-Voltage-Temperature (PVT) corners to CAD tools' information for library/IP development. However, the library/IP developers have difficulties in obtaining the desired data from the existing specifications due to their informality and complexity.

In this thesis, we propose methodologies, flows and tools to formalize the DP specifications for their unification and to deal with it. The proposed description is targeting to be used as a reference to generate and validate libraries (standard cells, I/O, memory) as well as complex IPs (PLL, Serdes, etc.).

First, we build a data model to represent all required information for library/IP development and then propose a specification language named Library Development Specification based on XML (LDSpecX). Furthermore, we introduce a reference-based method to create a reliable specification in LDSpecX and task-based keywords to efficiently extract data from it. On the basis of the proposed solutions, we develop a specification platform. This platform allows not only creating a complete and consistent specification with a huge amount of data but also rapidly and precisely extracting data depending on the task.

We develop a standard cell library from the specification creation to library validation by using the specification platform. We show that our approach enables to create a complete and consistent specification with a considerable reduction in time. It also bridges the gap between the specification and current automatic system for rapid library/IP development.

Keywords: Design Platform, Design Methodology, Unified specification, Library/IP development, Specification creation, Data extraction

Résumé

Une plateforme de conception est une solution totale qui permet à une équipe de conception de développer un système sur puce. Une telle plateforme se compose d'un ensemble de bibliothèques et de circuits réutilisables (IPs), d'outils de CAO et de kits de conception en conformité avec les flots de conception et les méthodologies supportés. Les spécifications de ce type de plateforme offrent un large éventail d'informations, depuis des paramètres de technologie, jusqu'aux informations sur les outils de CAO pour le développement des bibliothèques/IPs. En outre, les développeurs de bibliothèque/IP ont des difficultés à obtenir les données nécessaires à partir des spécifications existantes en raison du fait qu'elles ne soient pas formellement spécifiées et de leur complexité.

Dans cette thèse, nous proposons des méthodologies, des flots et des outils pour formaliser les spécifications d'une plateforme de conception pour leurs unification et les traiter. Cette description proposée vise à être utilisée comme une référence pour générer et valider les bibliothèques (cellules standard, entrée/sortie et mémoire) ainsi que les IPs complexes (PLL, Serdes, etc.).

Premièrement, nous construisons un modèle de données pour représenter toutes les informations nécessaires pour le développement de bibliothèques et proposons un langage de spécification pour le développement de bibliothèques basé sur XML (LDSpecX). De plus, nous présentons une méthode basée sur des références pour créer une spécification fiable en LDSpecX et des mots-clés basés sur des tâches pour en extraire les données. A l'aide des solutions proposées, nous développons une plateforme de spécification qui fournit une interface utilisateur graphique (GUI) et une interface de programmation (API). Cette plateforme permet non seulement la création de la spécification, mais aussi l'extraction rapide des données en fonction de la tâche.

Nous développons une bibliothèque de cellules standard depuis la création de la spécification jusqu'à la validation de la bibliothèque en utilisant cette plateforme de spécification. Nous montrons ainsi que notre approche permet de créer une spécification complète et cohérente avec une réduction considérable du temps. Cette proposition comble également l'écart entre les spécifications et le système automatique existant pour le développement rapide de bibliothèques/IPs.

Mots-clés : Plateforme de conception, Méthodologie de conception, Spécification unifiée, Développement des bibliothèques/IPs, Création de la spécification, Extraction des données

Table of Contents

List of Figures

List of Tables

1. Introduction

Contents

Gordon E. Moore presented the well-known Moore's law in an article published in 1965 [1]. This law describes a trend that the number of transistors on a chip doubles every 24 months. Until now, the development of new process technologies enables to follow Moore's law as shown in Figure 1.1 [2]. In addition to the increasing number of transistors on a chip, the performance of System-on-Chip (SoC) (speed, functionality, etc.) considerably increases but the cost decreases by 30% per year [3].

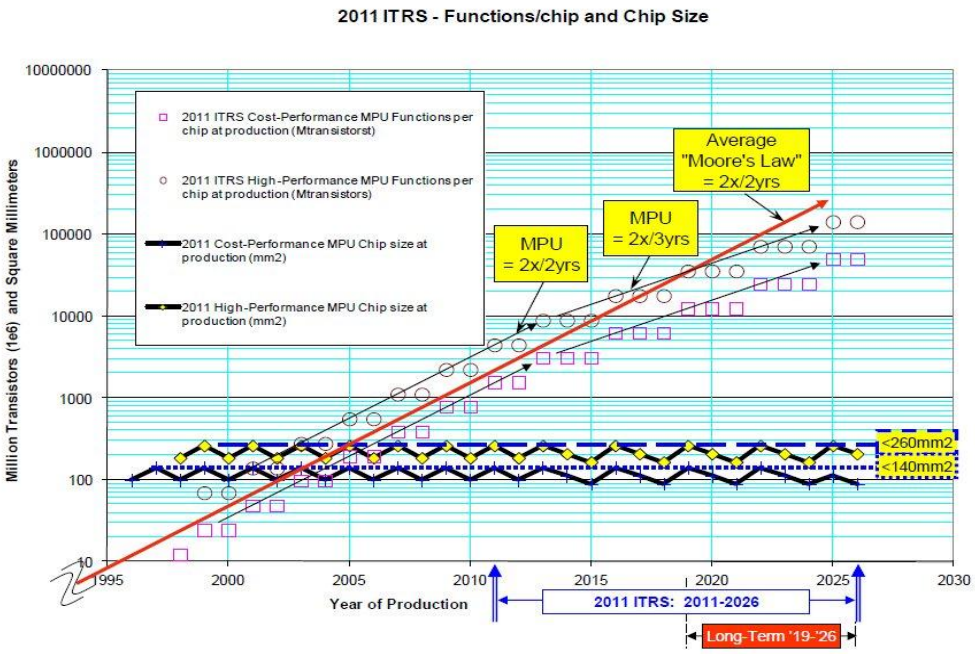

Figure 1.1 Product Technology Trends: MPU Product Functions/Chip and Industry Average "Moore's Law" and Chip Size Trends (source: ITRS)

Recently, a new trend called “More than Moore” for the functional diversification of semiconductor-based devices is also remarked. This “More than Moore” trend gives more possibilities to diversify the applications of SoC by interacting with the outside world through an appropriate transduction such as RF communication, passive component, sensor and actuators [4]. The International Technology Roadmap for Semiconductors (ITRS) represents the overview of these two trends toward the miniaturization and diversification as described in Figure 1.2. These trends allow various applications of SoC such as automotive, set-top box and telecommunication so that the market is growing rapidly [3].

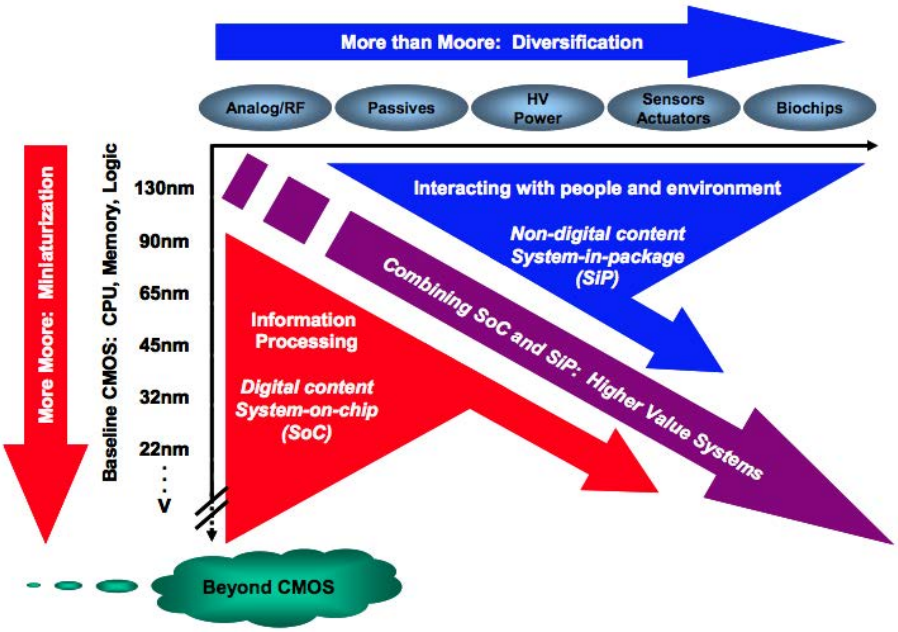

Figure 1.2 ‘Moore’s law’ and ‘More than Moore’ trends in the semi-conductor industry (source: ITRS)

However, there are more and more challenges with the new advanced technologies in the manufacturing process, circuit design, system design, and CAD technology [5]. In the manufacturing process, to achieve the downscaling of the devices, both front- and back-end processing must overcome major technological challenges: lithography, device isolation, gate stack, shallow junctions, device engineering, high- and low-k dielectrics, and interconnect schemes [6], [7], [8]. For circuit design, technology scaling brings some advantages such as high speed and density of integration. Nevertheless, circuit designers have to face significant issues such as increased leakage, variability, reducing power supply voltage, signal integrity problems, etc. Some of these issues were never encountered before. In addition, new types of devices and new materials may increase the complexity of the circuit design [9], [10], [11],

[12]. To illustrate an example, for sub-20nm IC technologies, Ultra-Thin Body Silicon on Insulator (UTB-SOI) and FinFET were proposed in order to address the problems derived from the short-channel effects of Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) like decreased sub-threshold voltage (V_t) and increased device variations by eliminating the leakage paths far from the gate [13], [14]. Hence, more efforts are needed for circuit design with such new types of devices. Likewise, system designers must deal with the issues and challenges for system design in nanometer-scale technologies (inferior to 100nm). Chang [15] remarked three challenges in system design: escalating cost, design complexity and time to market. For example, one of the key challenges is to reduce the power dissipated by the clock tree up to 40% of the total power consumption because of the increased number of flip-flops in the system [16]. For that purpose, the clock gating method [17], [18] and flip-flop banking method [19] were proposed to save power consumption of the clock tree. On the other hand, the challenges for CAD technology are strictly related to the aforementioned manufacturing and design challenges. Thus, CAD vendors endeavor to provide users with an efficient tool to assist them in order to overcome their challenges as well as to improve the productivity [20], [21], [22].

Foundries and Integrated Device Manufacturer (IDM) like STMicroelectronics fabricate chips for their customers. They must therefore provide customers with a design platform containing all necessary components such as libraries and Intellectual Property (IP) for chip design. However, the aforementioned issues complicate the development of components. In this thesis, such issues are considered from the perspective of the library development.

1.1 Motivation

In order to rapidly and efficiently bridge increasing design productivity gap with advanced technology, a new paradigm is highly required for system designers as well as library providers.

In the system design, the top-down approach is recently more significant than the bottom-up to efficiently improve the productivity. Thus, many recent researches focused on this approach, for example, a new level of abstraction so-called Electronic System Level (ESL) above the familiar Register-Transfer Level (RTL) and design verification based on the formal specifications [23].

From this aspect, we intend to improve the library development in a top-down approach from the specification because it is the most important starting point for library development. On the other hand, the specifications for library development consist of a cell-dependent part and a cell-independent one. The former represents functional and structural descriptions of the cell, whereas the latter represents all other information such as process parameters. Especially, the design platform specifications, which we need to deal with, cover only the cell-independent information such as Process-Voltage-Temperature (PVT) corners, CAD tool information, etc. Since such information has an impact on the cell design, we need to discuss it in order to efficiently provide this information for cell design.

However, the design platform specifications are usually written in natural language or tabular form. Thus, they represent a collection of heterogeneous elements. Despite their excessive complexity, library developers must extract the desired data from them either manually or through their own scripts. On top of that, since the current specifications may lack a part of the required information for library development, they should also complete missing information. The lack of information for library development in the current specifications can be caused by two reasons. The first reason is the difference of the view points of the system designers and library developers. For example, the references for the cell characterization such as time threshold and unit information are not defined in these specifications because such information is required for library development but not for SoC design. The second one is the current specification creation method. The specification creation is a manual work. For this reason, it often results in missing information. For example, the specification developer may omit to mention the version of CAD tools. Such missing information leads to increase library development time because we must get it in order to accomplish the library development. Thus, dealing with the specifications depends highly on the expertise of the library developer and manual interventions. It becomes a significant bottleneck in library/IP development.

1.2 Thesis Objectives

The main goal of this thesis is to precisely and rapidly provide library developers with all required information for library development from the specification in order to improve the productivity and quality of the library.

Firstly, we propose a unified specification for library development. Secondly, this thesis describes how to efficiently cope with it by developing new methods and tools which provide library developers with a powerful way to smoothly accomplish their tasks for library development.

1.3 Structure of this Manuscript

Chapter 2 introduces the system design flow as well as its corresponding design platform. Since this design flow is based on predefined libraries/IPs, we explain how to develop them. Then, an example of library development is described. With the help of these observations and experiments, we present crucial issues related to the specification itself and its current processing methods.

Chapter 3 reviews the state of the art with two traditional specifications in natural language and tabular form. In addition, Unified Modeling Language (UML) and eXtensible Markup Language (XML)-based specifications are presented as emerging ones in SoC design. STMicroelectronics' design platform specifications are also described. Finally, we evaluate them with respect to their advantages and disadvantages.

Chapter 4 proposes the identification of all required data for library development and their classification. This survey helps to propose a suitable data model to represent our specification. To deal with it in an efficient way, we propose some methods to create a reliable specification and to efficiently extract data from it. Furthermore, how to verify both the specification and library is also explained at the end of this chapter.

Chapter 5 describes a specification language in XML named LDSpecX (Library Development Specification based on XML) based on the data model proposed in the previous chapter. Then, we present a specification platform that consists of two parts. The first one is a user-friendly GUI to create a complete and consistent specification in LDSpecX. The second one is an API to precisely and rapidly extract desired data from the LDSpecX-based specification database.

Chapter 6 presents the evaluation of our solutions. We show the creation of specification in LDSpecX and the development of a standard cell library from it on the proposed platform.

Chapter 7 gives the conclusion and proposes relevant directions for future work based on the researches presented in this thesis.

2. Library Development from the Design Platform Specifications

Contents

2.1 Introduction

The system design requires more and more predefined libraries/IPs due to the increasing complexity of the system. Thus, foundries provide system designers with a design platform containing all necessary libraries/IPs in order to support their system design.

The library includes a collection of cells. Its main goal is to offer a wide range of information about the cells to system designer for integrating them into his system. For example, STMicroelectronics's standard cell library for CMOS 28nm FD-SOI technology contains several hundreds of basic cells like combinational and sequential cells. This library package provides various library views to transmit the required cell information to all CAD tools for system design. This library package is produced according to the given library development flow with the help of an automatic system. This system permits to automatically execute more and more tasks instead of the manual work in order to improve the productivity.

Consequently, to achieve the objectives of this thesis, a good understanding of such library development flow and automatic system is indispensable. Additionally, the analysis of the extremely complex relationship between the library development environment and the specification is also required because it may give a key idea to address the issues that arise in library development at the specification level.

In the following section, we present the design flow and design platform for SoC design. Then, we introduce how to develop libraries from the design platform specifications. In addition, an example of the development of a standard cell library is described. From these observations, we discuss the issues related to the specification.

2.2 System-on-Chip Design Flow

As mentioned in the previous chapter, technology scaling brings not only attractive advantages but also significant issues and challenges. Thus, many attempts were made to address them from different angles. For system designers, the most crucial challenge is how to rapidly and reliably design the complex system which meets the system requirements. For this purpose, there are several major approaches: custom design, Field Programmable Gate Array (FPGA), standard-cell based design and platform/structured design [24]. Among them, the cell-based design technique is widely adopted to reduce the design cycle as well as to guarantee the functionality and performance of the complex system although the full custom design usually provides a high-performance system than the cell-based design [24], [25], [26], [27], [28], [29], [116], [117]. The richness or optimization of cells in the library may have a great influence on the improvement of the performance of system, for example, in terms of area, delay, and power [60], [61]. Hence, it helps to reduce the performance gap. For instance,

Hashimoto *et al.* [30] demonstrated that a rich library including various driving strength cells improved circuit performance close to transistor-level optimized circuits.

The cell library now plays a central role in SoC design. Various library categories are required, for example, standard-cell library and macro cells such as I/Os and memories. Like these cell libraries, the macro-cell devoted to a specific function, also known as IP has recently become very important. In other words, the library offers a set of cells of the same category, whereas IP deals with only one block such as Pulse-Locked Loop (PLL). They have the same goal: to provide pre-designed cells for system design. As shown in Figure 2.1, the library cells and IPs account for a large part of the recent SoC up to almost 75% [82].

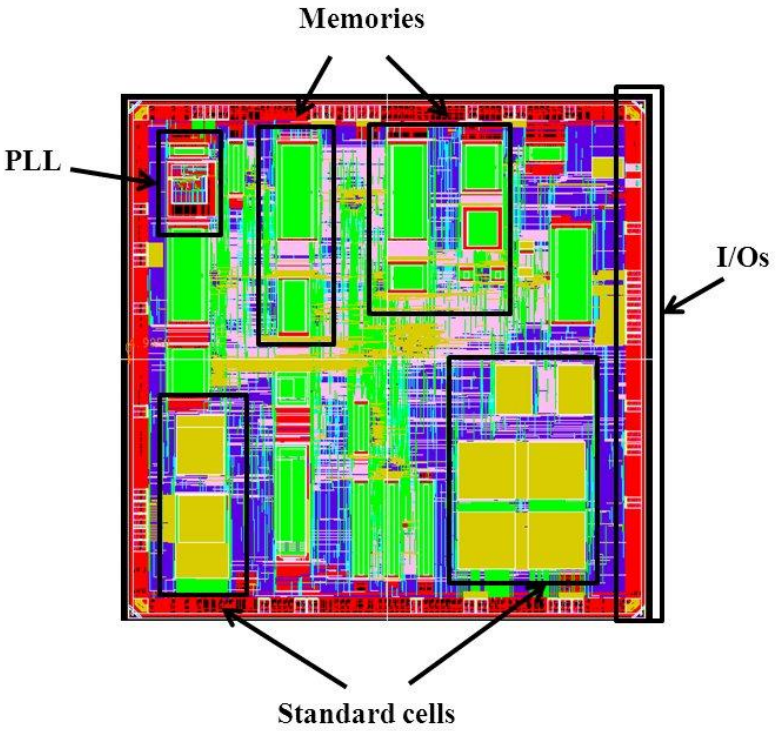

Figure 2.1 Example of a SoC (source: ST)

Figure 2.2 represents the cell-based design flow. This design flow splits into three principal steps: Front-End (FE) design, Back-End (BE) design and Sign-Off phases [24], [31]. For the integration of cells, the given library and IP packages provide the required views by CAD tools at all design steps to build a complete SoC.

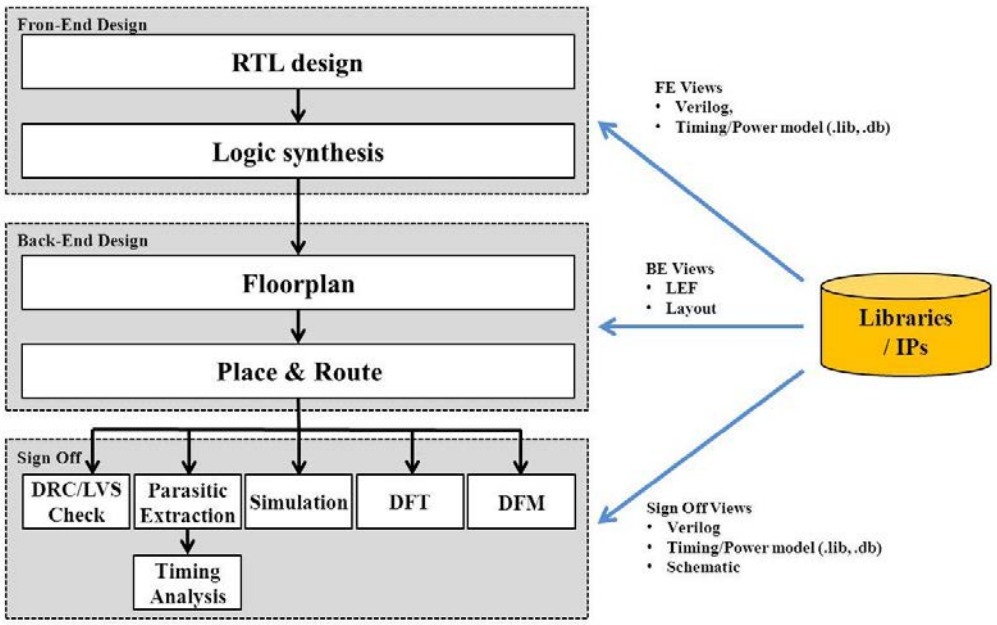

Figure 2.2 Cell-based design flow

Firstly, the FE design for logical creation can begin with RTL design. This step describes the functionality of the system in Hardware Description Language (HDL) according to its specification. After the RTL design, the logical synthesis must be executed for translating an abstract form of desired circuit behavior (RTL model) into a design implementation in terms of logic gates by using logical views provided by the library/IP package.

After FE design, the Back-End (BE) design for physical creation must be performed to create a physical view of the system so that the floor-planning and place-and-route activities should be accomplished using physical views of the library/IP package. The floor planning activity is the first step in physical design. The process of floor-planning allows analyzing the die size, selecting the packaging, placing I/Os, placing the macro cells, planning the distribution of power and clocks, and partitioning the hierarchy. Then, the placement process determines where each individual cell is physically located on the chip and the routing process permits to create the physical wire connections for the signal and power nets.

Finally, the Sign-Off phase should be carried out to check the final physical layout by several criteria such as design rules and timing constraints. For this purpose, library views such as Verilog view and timing/power models are needed. This phase is absolutely critical to catch any remaining error before tape-out. The created and validated layout of a system can be delivered to fab for manufacturing.

2.3 Design Platform

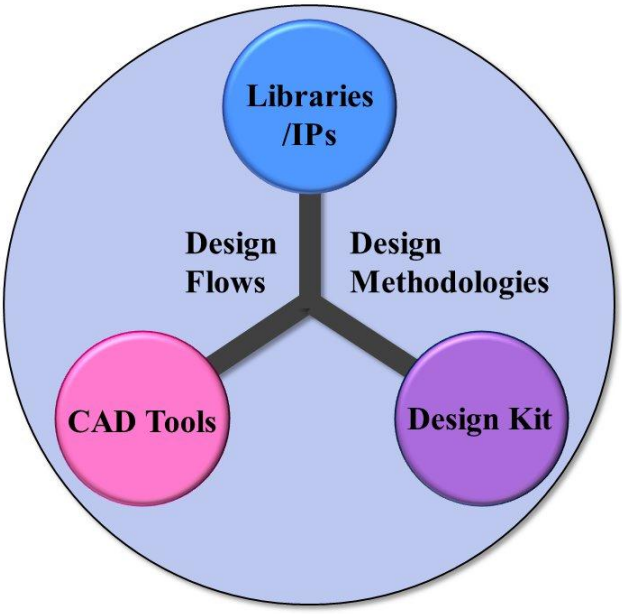

Figure 2.3 Design platform

The design platform is a total solution to achieve the SoC design [32], [33], [34]. Since the system designer needs a complete set of libraries and IPs, CAD tools and design kits in conformity with his design flows and methodologies, the design platform consists of all such elements as described in Figure 2.3.

In order to develop a design platform, its specification must first be defined. This specification contains all information related to the target design platform. For each manufacturing technology, a suitable design platform should be developed according to its given specification. Except for commercial CAD tools, which are selected but not developed by definition, all the other components have to be created: libraries, IPs, and design kits including a collection of target manufacturing technology data files such as SPICE models, tech files, and DRC/LVS rule files [35].

In this thesis, we concentrate on the library development from the design platform specifications but this study can also cover IP development because both can be developed by the same activities even if the used methods and tools are slightly different.

2.4 Library Development Flow

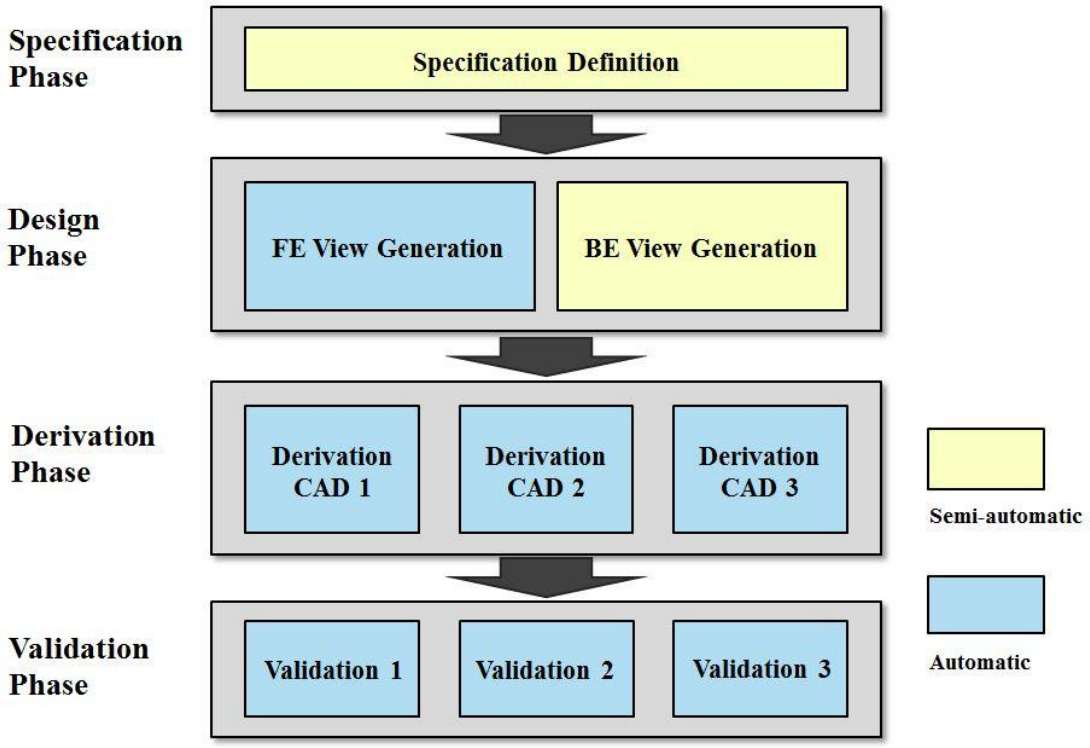

Figure 2.4 Foundry's library development flow (source: ST)

The traditional library development flow presented in the literature focus only on the design phase including FE and BE designs [36], [37], [38], [39], [40]. However, as shown in Figure 2.4, the library development flow in industry is more complex than that. In general, it is composed of four major phases. The first phase is to define a specification for the target technology. The second phase is to design all library cells. The third phase is to derive from the fundamental library views (e.g. layout and schematic) developed during the design phase to various CAD views for supporting various CAD flows. The last phase is to validate the library package by several criteria before their delivery to customers.

The library development requires the specification which can be divided into the cell-dependent part and cell-independent one. The cell-dependent part states the structural and functional description of the cell such as its pin name and logical functionality. On the contrary, the cell-independent part specifies all other information that are required for cell design but independent of cell such as process parameters and CAD tools' information. This specification data become increasingly complex. To give an example, each library category

needs a set of PVT corners for its cell design. The number of required PVT corners increases with the evolution of manufacturing technology. Figure 2.5 shows the increasing number of PVT corners for standard cell library according to the manufacturing technology. In addition, the required PVT corners may be multiplied by the necessary library categories.

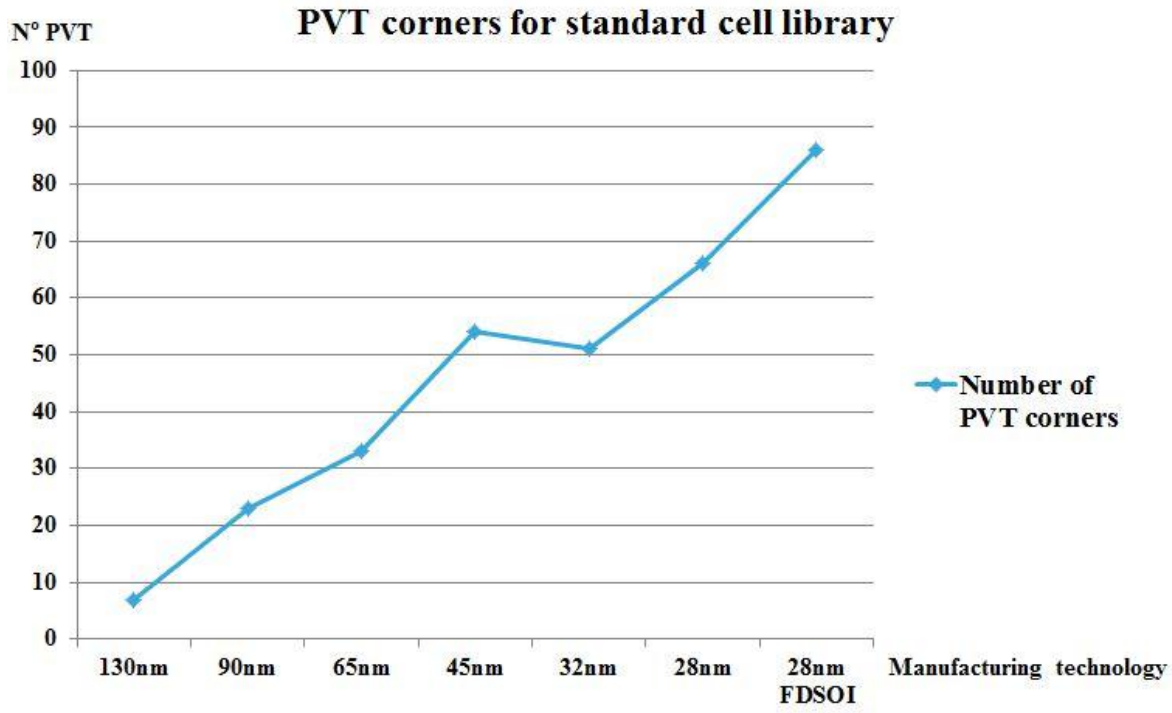

Figure 2.5 PVT corners for standard cell library (source: ST)

Where can the library developers obtain such cell-independent information?

Usually, they can get this information from the design platform specification because it covers most of the cell-independent information. As depicted in Figure 2.6, its scope is larger than that of the library development specification. In other words, some information of the design platform specifications is not necessary to develop libraries. For example, a list of approvers about the information may be required for specification definition but not for library development. On the contrary, the complete list of library/IP packages can be given after finishing their development. Therefore, in this thesis, we limited the scope of the specification to cover only the required information for library development because there are difficulties to deal with the entire design platform specifications under limited conditions. Taking account of it, the following subsections present each phase of the library development flow.

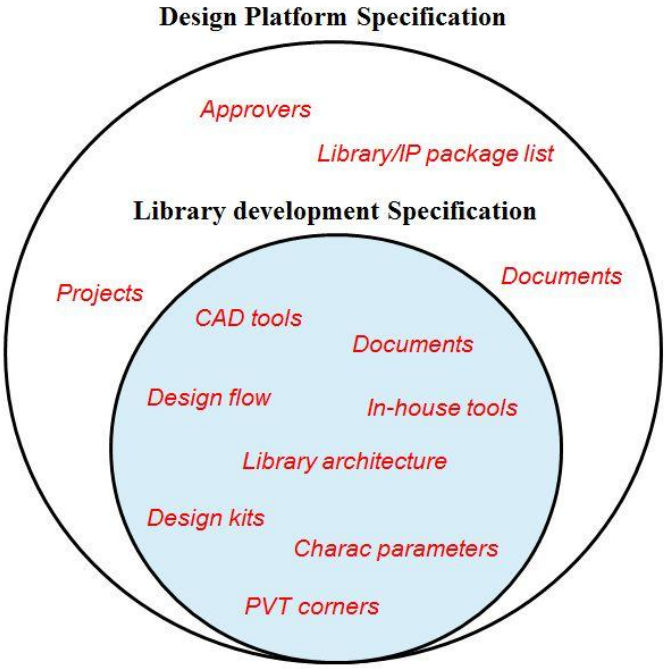

Figure 2.6 Design platform specifications vs. library development specification (source: ST)

2.4.1 Specification Phase

The specification phase is significantly important for library development because it aims at collecting all required information. Yuan [41] highlighted the interaction with various functional groups such as process engineering, reliability engineering, CAD tool developers, and the marketing group for gathering the environmental considerations for library development as summarized in Figure 2.7.

	Considerations	Related Functional Group
1.	Customer requirement and competitive analysis	Marketing/Design Engineering
2.	Process limitations and reliabilitiy considerations - latchup, metal migration, ESD, hot electrons, and associated layout constraints	Process, Reliability, and Design Engineering
3.	Performance limitations based on above	Design Engineering
4.	Place and Route tool capabilities	CAD/Design Engineering
5.	Chip statistics of interconnect capacitance, routability, process variation, voltage and temperature sensitivity	CAD, Process, and Design Engineering
6.	Packaging limitations and modeling	Packaging and Design Engineering
7.	Modeling restrictions and workarounds	CAD/Design Engineering
8.	Tool development for the library	CAD/Design Engineering

Figure 2.7 Environmental considerations for library development [41]

Baltus [42] remarked the essential relationship between the process parameters obtained from the fabrication process development and the cell layout generation in library development as shown in Figure 2.8

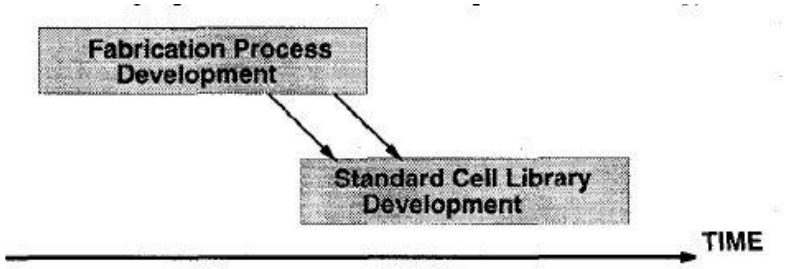

Figure 2.8 Layout development flow with manual layout [42]

As stated in these studies, the specification for library development must cover a wide range of information. Moreover, the information must be collected from several information owners. Thus, the specification phase is a collaborative work. It consists of collecting the information and creating a specification with the collected information. For example, we need a list of library views and CAD tools' information for library development. The library architect defines the contents of the library package for each library category. The design kit developer determines necessary CAD tools in order to develop a design kit for supporting them. Thus, the information relevant to library views and CAD tools can be obtained from them respectively. Then, the gathered information is defined in the specification.

2.4.2 Design Phase

The design phase is to design all library cells. This phase permits to generate all fundamental views for FE and BE designs of the system. Specifically, in accordance with the cell specification, the cell is designed and then its physical view is created by using a layout editor. However, much effort and time should be devoted to layout design because it determines the performance of the cells. For example, Bittlestone [43] stated that cell height must be considered for layout of standard cells because increasing track height allows reducing total resistance and improving performance. Additionally, power rail also influences the performance of the standard cells. Thus, layout design has to be done by taking into consideration the physical architecture.

After finishing physical design, library views such as behavioral model for FE design must be produced. This design step involves generating behavioral model and timing model. Behavioral model represents the structural description and the functionality of the cell in Hardware Description Language (HDL) such as Verilog. On the contrary, the timing model containing the timing characteristics of the cell can be generated by two steps. The first step is to characterize the cell in order to extract its timing information such as delay and constraints. The second step is to create a timing model by encapsulating the obtained timing information in Liberty format which is an open source industry standard for library modeling [44]. Since it is very important to provide an accurate timing model for verifying the functionality of the system, various characterization and timing modeling methods were proposed in order to create a more accurate timing model [45], [46], [47], [48], [49], [50]. Traditionally, the timing characterization of the standard cells is based on SPICE simulations to obtain the most accurate timing information by using manufacturer-provided SPICE models although the simulations requires a lot of time. Furthermore, for timing model, Non-Linear Delay Model (NLDM) based on Look Up Table (LUT) is widely used in industry. LUT aims at providing corresponding timing information according to input slope and output load because the timing characteristics rely on them. Thus, various input slope and output load values should be applied to the simulation in order to create a LUT. Recently, for 90nm designs and below, Synopsys's Composite Current Source (CCS) timing model based on the current instead of the voltage threshold for NLDM was proposed because it takes into account some physical effects such as Miller effects, high interconnect impedance, and noise propagation. However, it has several major inconveniences, high runtime being one of them [51], [52]. In addition, power and noise models may also be provided for analyzing the system in terms of power and noise [53], [54], [55], [56], [57]. Moreover, such characteristic models must be made for each PVT corner. It means that they will be multiplied by the number of PVT corners.

2.4.3 Derivation Phase

The derivation phase allows generating various CAD views from the fundamental views in order to completely support customers' CAD flows because some of them require their own semantics due to the lack of standardization efforts [58]. For instance, Cadence and Synopsys tools are based on different database platform OpenAccess and MilkyWay respectively [59]. For this reason, we must create these two physical databases for supporting

both CAD implementation flows. Consequently, all required CAD views should be created by the derivation phase to satisfy all customers' design flows.

2.4.4 Validation Phase

The quality of the libraries and IPs is directly related to not only their reuse and integration but also the design efficiency of SoC [62]. So they must be verified before their delivery to customers. For that purpose, Lin [63] first discussed the requirements of the high-quality cell library such as the correct functionality of the cell, its accurate timing performance and its layout having no design rule violation. Then, he classified possible errors into five types: incompleteness, inconsistency, functional errors, design rule violation and inaccuracy. As he remarked, errors are easily made during library development process. Therefore, this phase is greatly needed for generating high quality library.

2.5 Existing Automatic System for Library Development

The design automation is an inevitable trend in the SoC design due to the increased complexity of systems [64], [65], [66]. Likewise, it is also indispensable for library development [67] because it is almost impossible to manually perform all related activities. Therefore, an automatic system is needed to facilitate the library development execution as well as to reduce its development time. Indeed, many efforts were made to develop this kind of system in industry but only some of them were presented in literature [68], [69] because all know-how relevant to the library development is related to the competitiveness of the company. In addition, several academic researches were introduced in literature [70], [71], [72], [73] even if there were some limitations to entirely deal with the issues concerning library development due to the restricted accessibility of information. For instance, Onodera [70] developed an automatic generation system of process-portable library named P2Lib which represents the fundamental process as shown in Figure 2.9. This system allows library designers to generate process-portable libraries from a core library containing process-independent information like functional and structural information and process-dependent information. Specifically, the timing/power analyzer generates timing and power information of each cell from process-independent and process-dependent information and then produces an intermediate library containing them. From the intermediate library, the format converter produces tool-specific library views. In addition, two other tools create data sheets and

physical library views. Finally, the proposed automatic system produces a complete set of libraries.

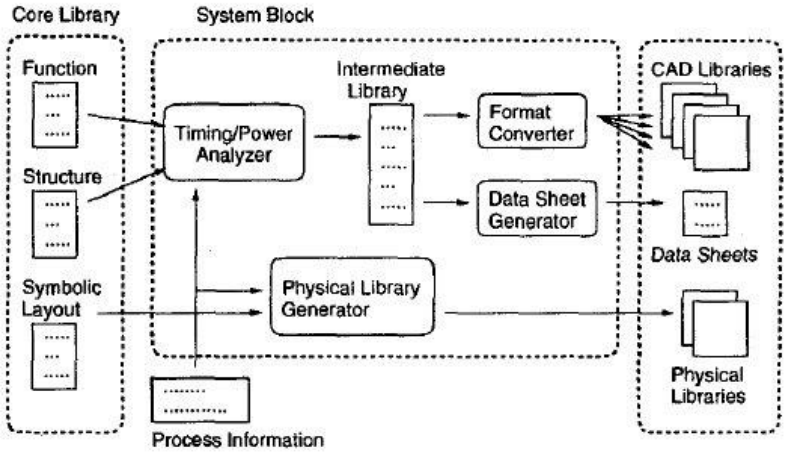

Figure 2.9 Overview of P2Lib [70]

The automatic library development system often consists of several in-house tools developed in programming languages like TCL, Scheme and Perl. Figure 2.10 describes the overview of STMicroelectronics' automatic library development system with only principal in-house tools. These tools which enable to perform the activities for generating library views or verifying them cover most of the phases in Figure 2.4 except semi-automatic steps. In the following sections, we introduce how to accomplish each phase by using these specific in-house tools.

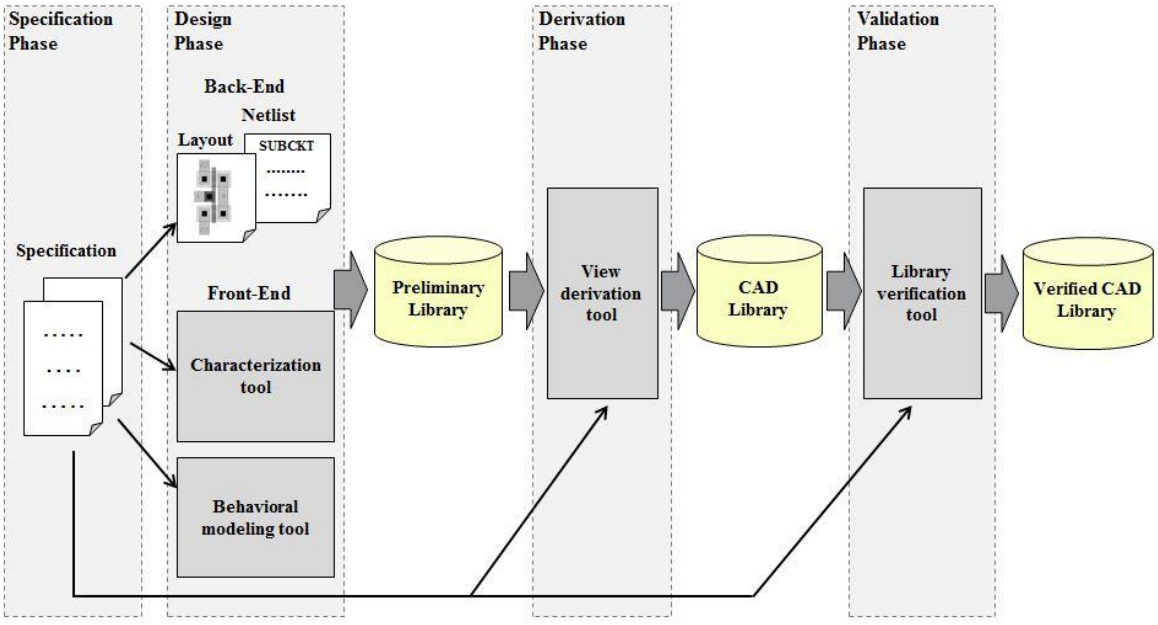

Figure 2.10 Overview of an automatic library development system (source: ST)

2.5.1 Specification Creation Tools

Traditionally, the specifications for library development are based on natural language and tables. Thus, according to the characteristics of the information, the specification developer defines data in natural language or tabular form so that he can store them in a file by using office tools like Word and Excel. To illustrate an example, the person in charge of the information relevant to the cell characterization has to enter manually all process parameters such as capacitive loads in the tables. Figure 2.11 shows an example of the definition of maximum capacitance in table form by using Excel. In this manner, other process parameters for cell characterization can also be defined. However, each group of process parameters must be given in different tables or different spreadsheets.

	A	B
1	DRIVE STRENTH	MAX CAPACITANCE
2	X2LOAD	0,02235
3	X3LOAD	0,03278
4	X4LOAD	0,0447
5	X5LOAD	0,05513
6	X6LOAD	0,06705
7	X7LOAD	0,07748
8	X8LOAD	0,0894
9	X9LOAD	0,10132
10	X10LOAD	0,11175
11	X11LOAD	0,12218
12	X12LOAD	0,1341
13	X13LOAD	0,14602
14	X14LOAD	0,149
15	X15LOAD	0,1639
16	X16LOAD	0,1788
17	X17LOAD	0,1937
18	X18LOAD	0,2086
19	X19LOAD	0,2086
20	X20LOAD	0,2235

Figure 2.11 Maximum capacitance (source: ST)

2.5.2 Circuit Design Tools

The library designer must design all involved cells in the library according to their functional and structural description. For circuit design, he also needs the correct information about CAD tools, design kit, and various technology-dependent parameters. All such information should be obtained from the given design platform specifications. If all fundamental information is prepared, the circuit design can be performed by using CAD tools. Firstly, in order to create BE views such as schematic, symbol, and layout, a CAD tool suite for physical creation is required. The cell design is often carried out manually with expertise in designing the high performance cell in terms of area and delay. On the contrary, FE view

generation can be accomplished automatically by using in-house tools such as characterization and behavioral modeling tools depicted in Figure 2.10. As a result, logical models in HDL and timing/power models can be produced for logical synthesis and timing analysis [74], [75], [76], [77], [78], [79], [80]. The logical model can be derived from only functional and structural information by using a behavioral modeling tool because it is independent of technology. On the contrary, the timing/power modeling is highly dependent of technology so that its generation requires lots of information related to the target technology for cell characterization as shown in Figure 2.12. To make accurate timing/power models, we must extract the timing/power data from the simulations under various conditions which is made of PVT corners, input slopes and output loads. The characterization tool first create SPICE input file (.cir) including a SPICE netlist, stimuli and statements for the data extraction with the given characterization conditions. Then, the obtained characteristics of the cells can usually be encapsulated in Liberty library file (.lib).

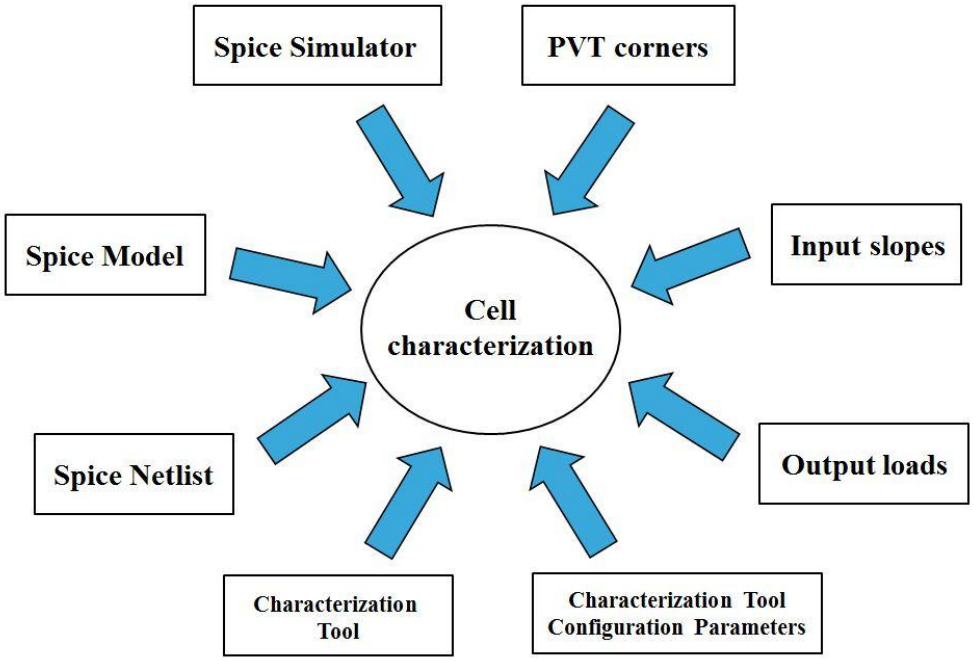

Figure 2.12 Cell characterization

2.5.3 View Derivation Tool

The library package must contain the appropriate CAD views for customers' CAD tools supported by the design platform. The required CAD view can be created by transforming a reference view into a desired CAD view by using the view derivation tool. For

example, Library Exchange Format (LEF) containing physical layout information of the cells is required for place-and-route step in system design. This physical abstract view can be derived from GDSII view depending on the type of library cells. There are two possible methods to generate this view from GDSII view as described in Figure 2.13. The first method is to use Cadence tool through several steps from GDSII view to LEF view. The second method is to directly produce LEF view from GDSII view by using the design rule of Mentor Calibre [118].

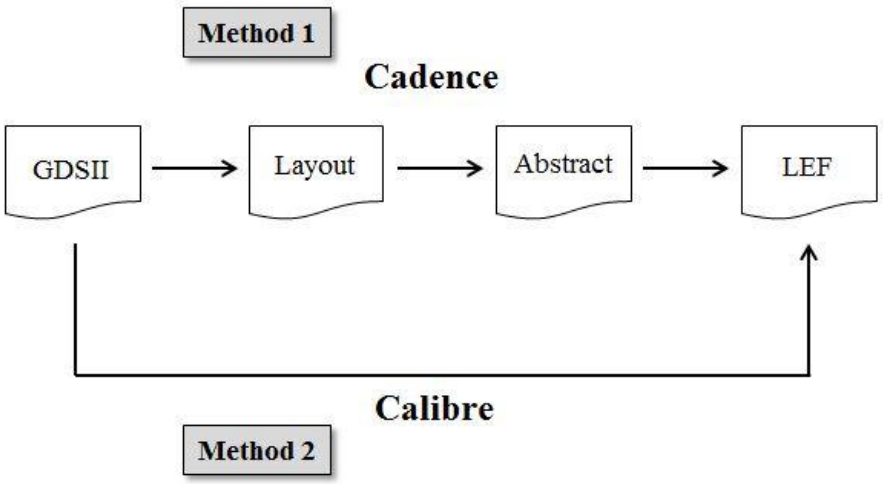

Figure 2.13 LEF generation methods (source: ST)

[59] presents two principal programs named **oa2mw** and **mw2oa** to translate between Cadence OpenAccess (OA) [119] and Synopsys Milkyway (MW) [120] databases. Figure 2.14 briefly describes the proposed method and tool for view transformation. The **oa2mw** program permits to transform Cadence OpenAccess database into Synopsys Milkyway database via the following files: Design Exchange Format (DEF), GDSII, LEF and Verilog netlist. To give an example of generating a place and route abstract (FRAM) view (Synopsys MW) from an abstract view (Cadence OA), an in-house tool first generates an abstract view. By using **oa2mw** program, the abstract view is translated into LEF view and then, LEF view can be translated into FRAM view.

As shown previously, the view derivation tool permits to generate the desired CAD views from the reference views by applying appropriate methods for each CAD view.

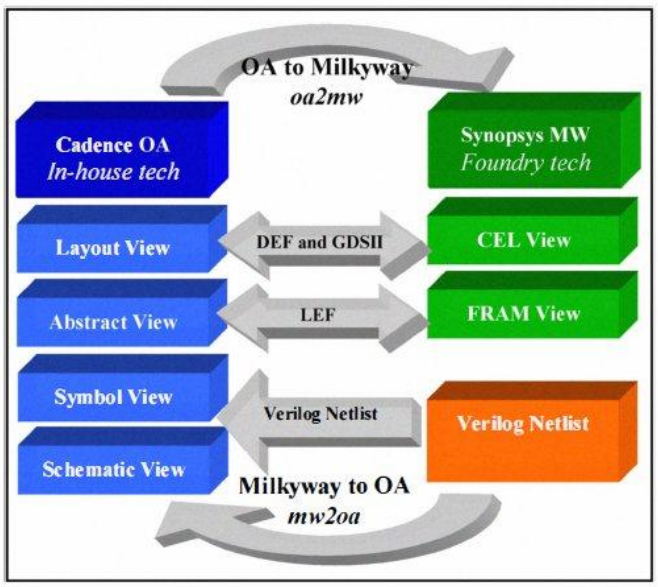

Figure 2.14 Methodology of the transformation between Cadence OA and Synopsys MW databases [59]

2.5.4 Library Verification Tool

The current library verification tool provides several categories of check: library view list, syntax, consistency, modelization, etc. It helps to verify the completeness and correctness of the library for its high quality. For instance, the library should be verified in accordance with a list of library views to make sure if the library contains all necessary views. For that, the library verification tool reads a specification file containing this list, and then compares the generated library views with the given list of library views. In addition, the correctness of library view is also verified. For instance, an attempt to read Verilog model by synthesis tool allows checking its correctness in terms of syntax. Consequently, if the library verification tool provides more checks, the quality of the library would be more guaranteed.

2.6 Example of the Development of a Standard-cell Library

In this section, we describe how to develop this standard-cell library through the library development process by using in-house tools because developing a library can be very beneficial to effectively understand the library development process with its automatic system. A standard-cell library including sequential circuits named ‘multi-bit flip-flop bank’ has been developed from BE views with the specification for 90nm CMOS technology. This

library contains 2-, 4- and 8- bit flip-flop banks. The schematic of a 4-bit flip-flop bank is given in Figure 2.15.

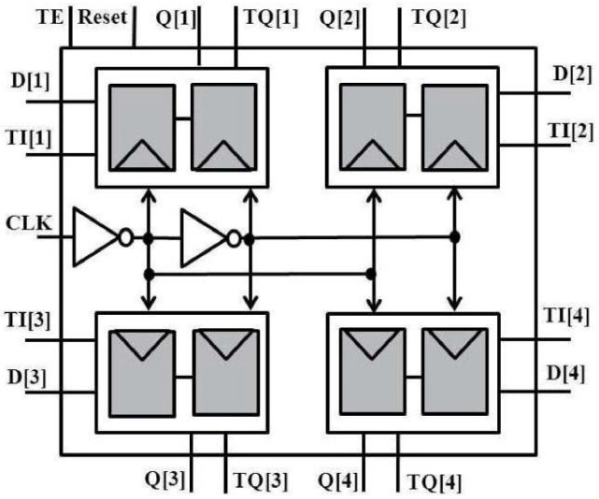

Figure 2.15 Schematic of a 4-bit Flip-Flop Bank

- **Specification Data Preparation**

As mentioned above, the library development requires a large number of data that must first be gathered from the specifications. Firstly, we need a list of library views to produce a complete library package. Secondly, a list of PVT corners and a set of process parameters must be obtained from the characterization specification. Thirdly, the correct design kit and tools' information like tool version are needed to execute tasks for library view generation and validation. Finally, for in-house tool use, the configuration of tool should be correctly fitted by defining the specific tool configuration parameters. A large part of these parameters can be obtained from the current specifications by human inspection or scripts of the library developer. The extracted specification data have been stored in several machine-readable files for in-house tools.

In this observation, we needed to accomplish the FE and CAD view generation and to perform library validation. For these steps, all necessary data have been manually collected from several scattered files containing specification data in natural language or tabular form.

- **Front-End View Generation**

FE view generation is divided into two major tasks: timing/power models and behavioral model generations. For creating timing/power models, all library cells were

characterized under three PVT corners (for best, typical and worst process corners) with various process parameters such as input slope by using SPICE simulations. The corresponding timing and power characteristics could be obtained from the simulation results for each PVT corner. As multi-bit flip-flop bank is a sequential circuit, delay, setup, hold and min pulse width must be extracted as timing information of the cell. In addition, the worst value is used to make a timing model. In contrast, leakage power and internal energy must be extracted to provide power information of the cell [107]. The obtained timing and power information was built in LUT form according to the corresponding input slope and output load values and then these characteristic information and the structural and functional description of the cell were written in Liberty library format (.lib) [44]. Furthermore, 90nm and below, CCS timing model is required so CCS timing models were also produced. Unlike timing/power model, which aims at providing the characteristics of the cells, the behavioral model gives the description of the circuit and its function. Such information could be extracted from the cell specification and then written in Verilog by using behavioral modeling tool.

- **CAD View Generation from the Fundamental Views**

After generating fundamental views, supplementary CAD views must be produced for the supported CAD flows. By using the view derivation tool, Liberty library files (.lib) were first compiled by Library Compiler and then, LEF file was generated from GDSII by Method 2 presented in Figure 2.13. Additionally, OpenAccess and Milkyway databases were also generated from the fundamental physical view for supporting Cadence and Synopsys CAD flows respectively.

- **Library View Validation**

By using the library verification tool, the multi-bit flip-flop bank library obtained from the previous phases has been verified according to the checklist below:

  - ✓ Completeness check (library view list)
  - ✓ Syntax check (Verilog, Liberty library)
  - ✓ Consistency check of library views

Finally, multi-bit flip-flop bank library package was obtained through the aforementioned library development process.

2.7 Problems

In the context of this thesis, we focused on the cell-independent specification for library development. Thanks to the aforementioned observations and experiments, we remarked several crucial issues related to the specification from various view-points. These issues can be divided into two parts depending on whether they relate to the specification contents or specification processing. Many individuals like design platform developers, library architects and library developers deal with our target specifications. They can be regrouped into two groups according to their role: specification developer and specification user. The specification developer is the person who participates in creating the specifications. On the contrary, the specification user is the person who uses data obtained from the specifications. According to these two roles, the specification processing can be divided into two activities: specification creation and data extraction from the specification. Thus, the issues regarding the specification processing are also classified by these activities.

All remarked issues are summarized as follows:

❖ **Specification:**

- ***Informality***: No unified formalism is currently available to define a wide range of information in the specification. As a consequence, information is often written in different ways like natural language or table form and stored in different files according to the characteristics of information and author's ability.
- ***Inconsistency***: The current specifications are created by several individuals and stored in scattered files having duplicate data so that the specification may often have consistency problems like mismatching CAD tool version and naming convention.
- ***Missing information***: The specification developer defines a specification from the different point of view with the library developer. Thus, it is difficult to know precisely which information is required to develop libraries. In other words, there is always a gap between the information provided by specification developers and the required information for specification users which results in missing information.
- ***Increasing specification data volume*:** Technology scaling leads to continuously increase the process variation which considerable influences the functionality of

the system. For instance, PVT variability causes fluctuation in timing and power for SoC designs, hence more and more PVT corners are required as shown in Figure 2.5 [43], [81], [82], [83]. In addition to technology scaling, supported library categories and new design flows also may increase the volume of specification data like the number of the necessary library views.

- ***Ambiguity***: The specification information is interpreted by specification developers and specification users. In order to define obvious information in the specification, the interaction between them is significantly important. However, it is not sufficiently done and thus, some information may provoke ambiguity problems.

❖ **Specification creation:**

- ***Lack of a unique reference database*:** To define a specification requires a set of golden references such as library naming convention, tool list and library structure. However, there is no unique repository containing all references. In addition, since the existing references represent the documents in natural language and tabular form, it is difficult to centralize them. For these reasons, their accessibility can be low. It results in increasing specification creation time.
- ***Absence of a specification assistant tool*:** The specification developer enters all specification data manually with the help of office tools. Furthermore, they prefer to copy the old specification rather than refer to the references in order to create a new specification. As a result, such tedious way may cause specification errors.
- ***Absence of specification checks***: The verification of specification relies on the specification developers so that the specification errors can be detected by human inspection. It is not sufficient to ensure the high quality of the specification as well as cover increasing specification data volume.

❖ **Specification data extraction:**

- ***Need for efficient methods and tools for the specification data extraction*:** One of the important issues in library development is to extract necessary data from the specification by human inspections and individual scripts. Thus, current specification data extraction method can significantly degrade library development time. The average time consumption for the data extraction from the current

specifications is obtained from library developers. It is about 4 hours because most of data should be extracted manually. For example, the cell characterization requires a lot of process parameters but their extraction is now carried out manually.

2.8 Conclusion

In this chapter, we have introduced the cell-based design flow and design platform for supporting this design flow. In order to produce a design platform, all of the required libraries/IPs must be developed. Firstly, the library development flow has been presented phase by phase. Then, we explained how to perform each phase in the existing automatic system. Additionally, we also developed a standard cell library containing several multi-bit flip-flop banks throughout the development flow with the help of the existing automatic system.

In summary, the library developers need to get an amount of required data from the specification for successfully accomplishing the library development. It enables to build a complete design platform. Furthermore, efficiently dealing with the specification data is one of the most important challenges in library development. Thus, all crucial issues relevant to the specification have been remarked in the previous section. By addressing these issues, we may considerably improve the productivity of the library by quickly and precisely providing all required data to the automatic library development system. For this purpose, the following chapter first reviews various kinds of the specifications presented in the literature as well as the current specifications employed by STMicroelectronics.

3. State-of-the-art

Contents

3.1 Introduction

The specification represents a collection of information to carry out an activity and is considered as its starting point. For instance, the component specification provides all the information relevant to cells such as their functionality and structural description for cell design. Likewise, the specification is required for library development as described in the previous chapter. However, it is highly important to determine which information must be defined in the specification and how to deal with it. We need a unified specification instead of current ones because it may facilitate to deal with its information.

There are various kinds of specifications from the natural language-based specification to the specification based on XML in order to represent the information. In this chapter, we first discuss the traditional specifications in natural language and tabular form, and the emerging specifications based on UML and XML. Furthermore, the current specifications employed at STMicroelectronics are also introduced.

3.2 Natural Language-based Specification

The specification has often been written in natural language until now although the development of computer engineering enables to efficiently manage information. It is so because the natural language is preferred as the initial and simple way to describe the desired information. Furthermore, the component specification and system specification are also defined by designers with the same method. Thus, in system design, some previous works focused on translating the natural language-based specification into a formal description, e.g. HDL or ESL by using a semantic analysis that allows extracting the information from the given textual specifications. For example, [84] presents a translator named Semantic Parser, which is based on a syntactic parser, the Natural Language ToolKit and a semantic grammar. This parser enables to translate the natural language description of a component into a simulatable Verilog model by extracting the key information from the given sentences and rewriting it in Verilog. Similarly, [85] introduces the translation of informal textbook specifications into a formal ESL implementation such as SystemC. For this purpose, Formal Specification Level (FSL) has been proposed to bridge them. Specifically, this intermediate level allows decomposing the sentences in terms of noun, verb, and adjective by syntactic and grammatical analysis and then extracting the structure behavior and properties of the system from the decomposed sentences. Finally, the obtained information is used to create a SystemC model.

Despite these previous attempts of formalization, a more precise syntactic and grammatical analysis is still highly required to obtain the desired information from the complex textual specifications. If it is not available, the specification authors must use simple sentences and restricted words as well as pay an attention to create a simple specification. Nevertheless, since the natural language-based method offers authors a great freedom in defining a specification, there are always possibilities to cause errors such as inconsistency and misunderstanding. Furthermore, as is well known, dealing with this specification is a tedious and time-consuming job.

3.3 Table-based Specification

The table-based method is as popular as the natural language-based method to represent the specification thanks to the facility of entering data in a table and capturing

desired data from that. However, by using tables, we must refine the information in comparison of natural language. It means that only the necessary data should be stored in tabular form after removing the redundant one. Hence, the table-based specification gradually replaces a large part of the natural language-based specification to describe the specification of a component or system as a more efficient method. The following subsections introduce two table-based specifications.

3.3.1 System Requirements Specification in SCR

In the late 1970s, Software Cost Reduction (SCR) method, which is based on "user-friendly" tabular notation, was applied to design software systems developed by David Parnas and researchers of the U.S. Naval Research Laboratory (NRL) [86]. The requirements specification in SCR tabular notation refers to a repository for all information required by developers to construct the software for a computer system. During the 1980s and 1990s, it has been used in many industrial organizations including Lockheed, Grumman, Bell laboratory and so on thanks to its scalability and cost-effectiveness. On top of that, a complete tool set for 1) creating, 2) debugging, 3) validating and 4) verifying this specification has been developed over a decade or more. By using this tool set, constructing the requirements specification for an automatic Cruise Control System (CCS) is introduced in [87], [88]. Figure 3.1 specifies this automatic system in SCR.

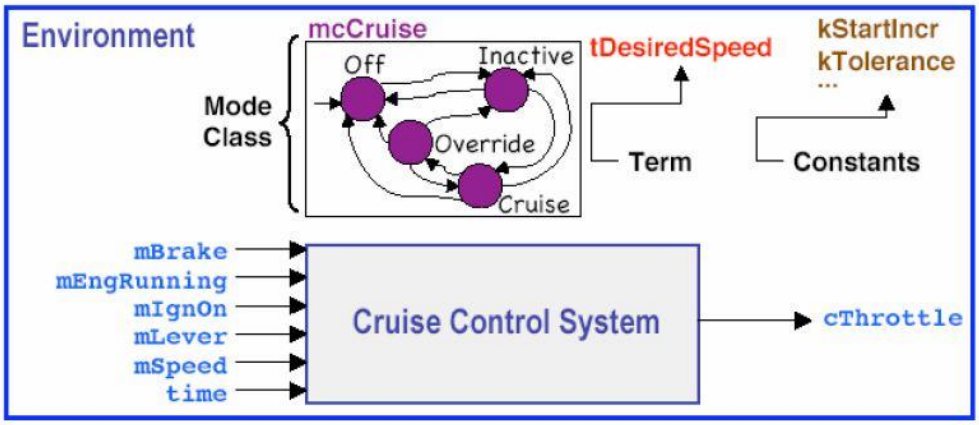

Figure 3.1 The cruise control system in SCR [88]

Creating the requirements specification in SCR notations for the system in Figure 3.1 requires two kinds of tables: function and dictionary. Function table gives the change of the value of the mode by the event of variables. For example, Figure 3.2 gives a function table to describe the mode transition of **mcCruise** depicted in Figure 3.1. This table represents a finite

state machine to give the new mode (**destination mode**) of the system as a function of current mode (**source mode**) and the monitored variables (**events**).

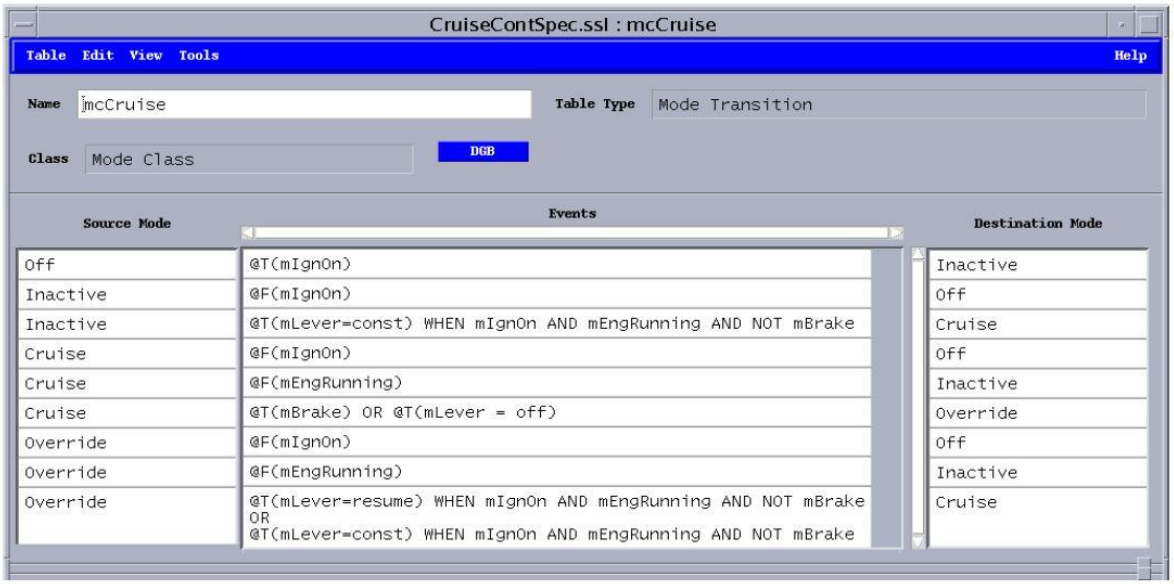

Figure 3.2 Mode transition table defining the mode class mcCruise [88]

On the contrary, dictionary table contains the fixed information such as variable declarations, environmental assumptions, and type definitions. To give an example of the dictionary table, Figure 3.3 shows a variable dictionary that defines the types, initial values and accuracy requirements of all variables. The monitored variables like **mIgnOn** for the CCS are also given in this table. These variables are used for the event in the above mode transition table.

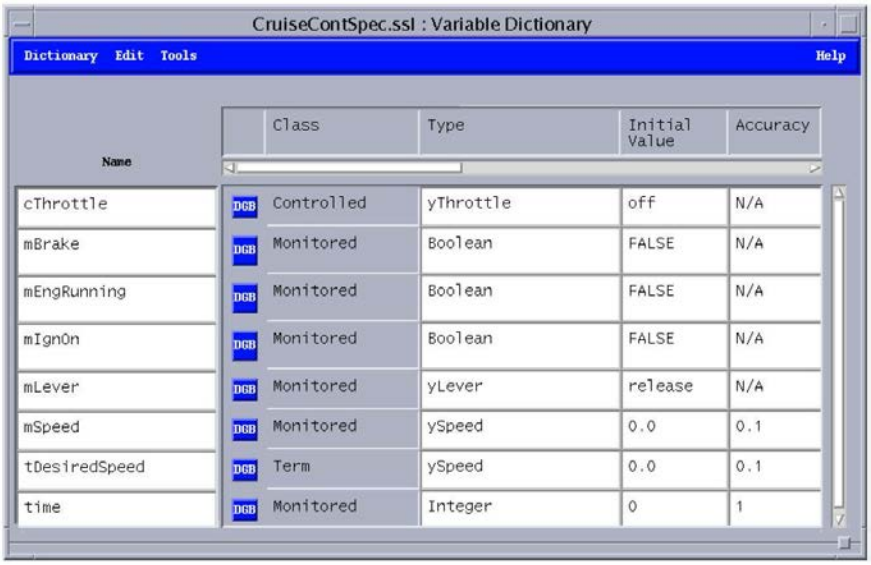

Figure 3.3 Variable dictionary table for the CCS [88]

By using a specification editor, all necessary tables can be created. After the creation of this specification, several checkers permit to check its well-formedness by detecting various errors such as consistency, property, and dependency. Additionally, the checked specification is also validated to ensure that it meets the customers' needs and analyzed to verify the properties of the application.

3.3.2 ADeVA

Instead of using large natural language documents to represent the specification of the system, Advanced Design and Verification of Abstract Systems (ADeVA) has been developed on the basis of the concept of SCR notation explained above. It also provides two types of tables: Mode Transition Table (MTT) and Data Transformation Table (DTT) which are function and dictionary tables respectively [89]. Some specification tools based on ADeVA have been introduced in the literature. Alcatel-Lucent Technologies developed a specification editor named SpecEdit using ADeVA for the specification of a complex telecommunication system [90]. Figure 3.4 shows its Graphic User Interface (GUI) which enables users to enter data in MTT and DTT to describe the behavior of the system.

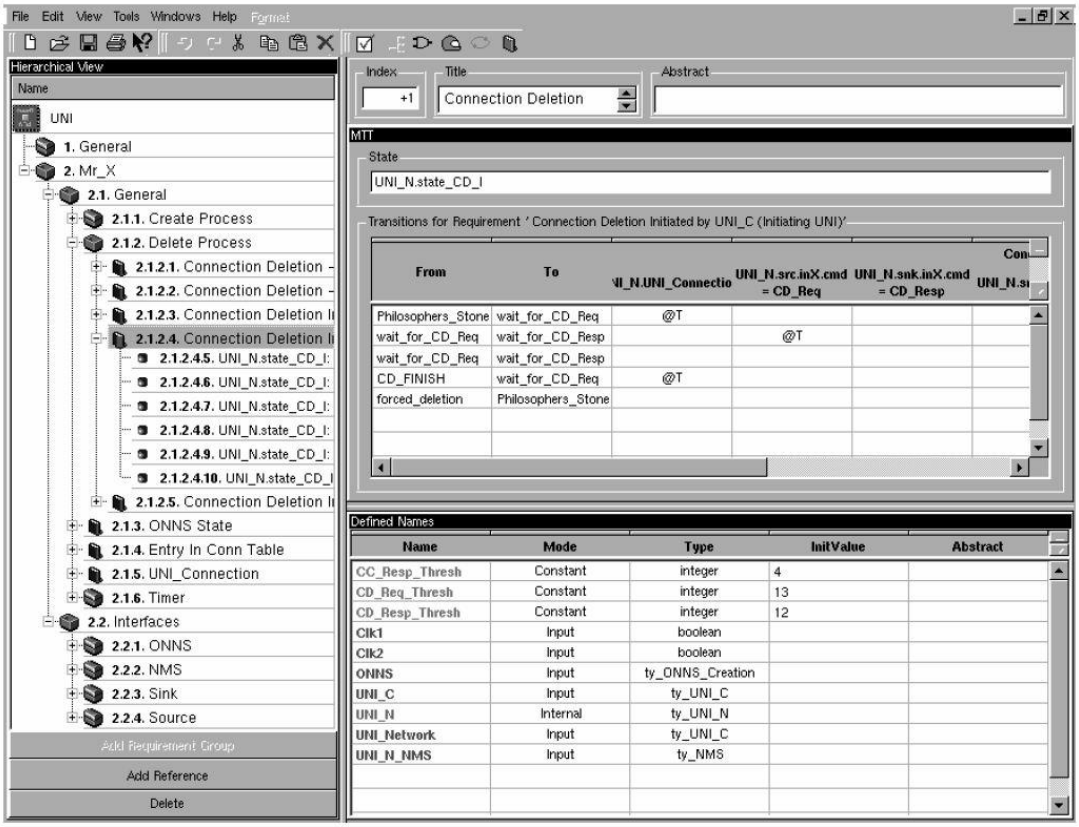

Figure 3.4 SpecEdit GUI [89]

Figure 3.5 shows a MTT that gives the symbol to indicate an event (T/F, @T, -, etc.) under the given condition when a state transition occurs. Figure 3.6 illustrates a DTT defining the event symbol under different conditions according to the given value of the signal. Similarly, [91] also presents a specification platform based on ADeVA notations for representing the requirements specification of the embedded system.

From	To	UNI_C cmd = CC_Req	UNI_C quality	UNI_N cmd = CC_Resp	UNI_N quality
Wait_Req	Check_Req	@T	-	-	-
Check_Req	Wait_Req	-	T	-	-
Check_Req	Wait_Resp	-	F	-	-
Wait_Resp	Wait_ONNS	-	-	@T	T

Figure 3.5 Example of MTT [89]

Output	Value	UNI_Network State = wait_for_Resp	UNI_Network State = wait_for_ONNS	UNI_Network CC_Table AND Ins = 2
UNI_N.cmd	CC_Req	@T	F	-
UNI_N.cmd	CC_Resp	-	@T	T

Figure 3.6 Example of DTT [89]

In summary, the table-based specification provides some important advantages. One of the interesting advantages is to easily extend the specification by simply adding supplementary columns or rows. Another is to rapidly get the desired data from the corresponding table by matching each column with the given condition in comparison with the natural language-based specification that mainly relies on the human inspection or syntactic analysis. In addition, the specification creation and data extraction can be accelerated by using the predefined form. However, this kind of specification also has several important limitations. First, since a table represents a set of data of the same type, the specification containing a variety of data should be defined by using many tables. For instance, to cover the system requirements specification of CCS described previously, three function tables and six dictionary tables are needed. Second, it is difficult to understand the relationship between tables. For this reason, an additional specific tool must be developed to visualize the dependency of data in different tables as shown in Figure 3.7 [88]. It means that we need an additional work to identify the relationship of the tables.

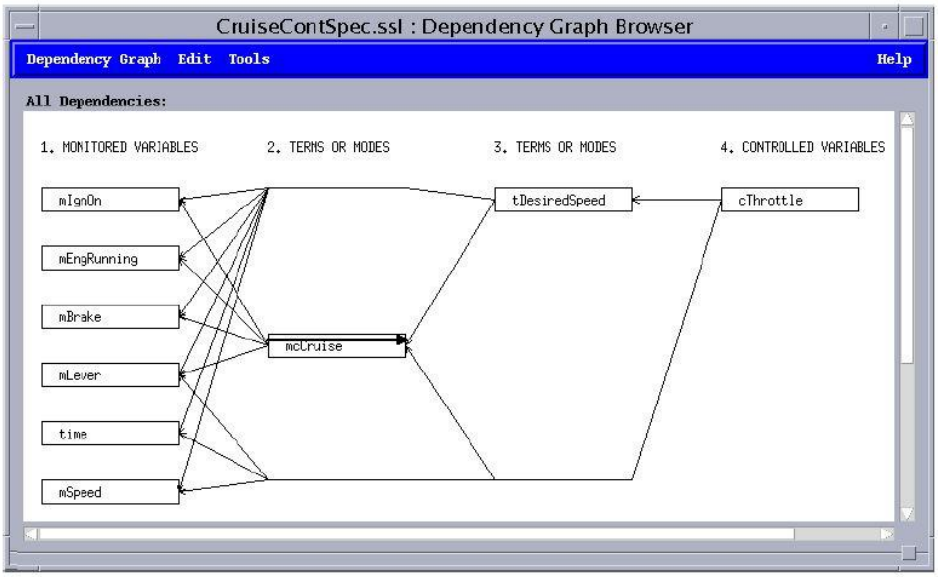

Figure 3.7 Dependency graph tool for the CCS [88]

3.4 UML-based Specification

UML is a modeling language using graphical notations for creating visual models of the system in software domain [92]. This language provides a set of diagrams (e.g. class diagram, sequence diagram, use case diagram, activity diagram, etc.). These diagrams can be classified into two groups depending on the characteristic of the model: structure and behavior diagrams as depicted in Figure 3.8 [112].

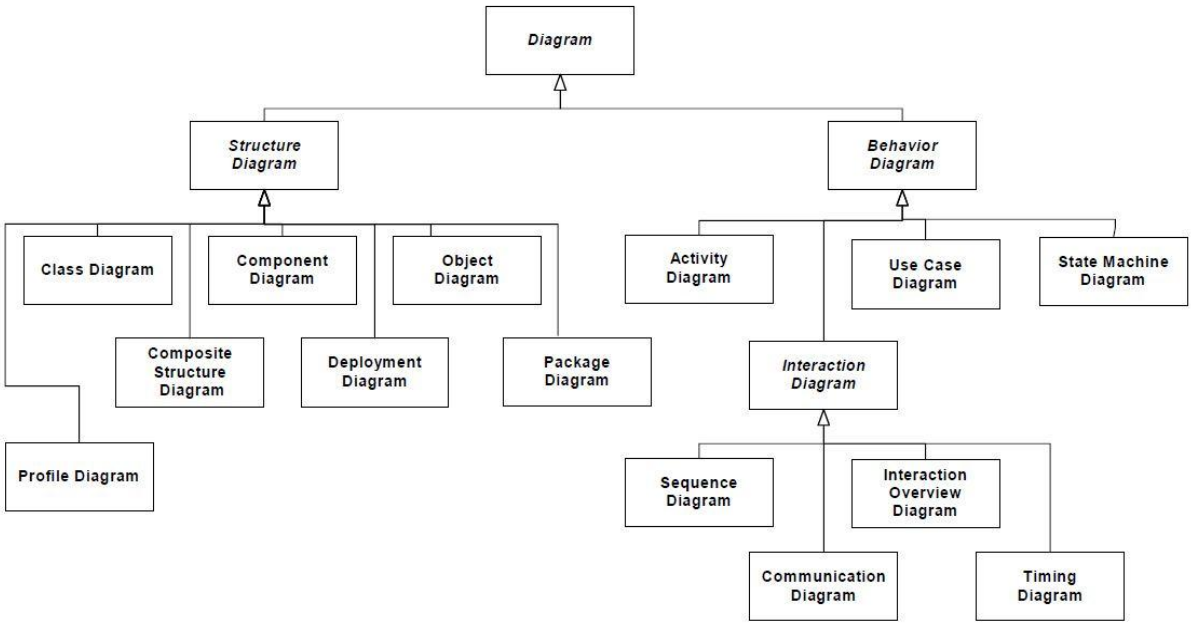

Figure 3.8 Taxonomy of structure and behavior diagrams [112]

UML allows rich expressive capabilities by using diagrams so that it can be quickly applied to various other domains including SoC. For instance, the authors of [93] proposed a SoC validation method based on UML and Component Wrapper Languages (CWL) for modeling the component specification and the interface protocol specification respectively. Figure 3.9 describes the specification based verification process which consists of two steps: specification validation and SoC verification. First, the specification in natural language is validated by checking its completeness and consistency with the help of UML models and CWL. Second, the functionality of the target system can be verified by using test scenarios derived from the UML models of the system.

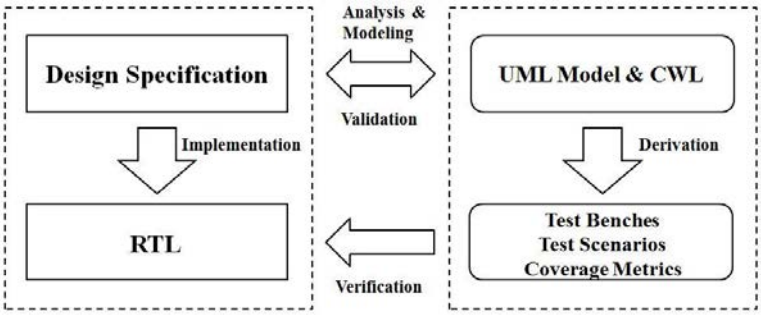

Figure 3.9 Specification based verification process [93]

[94] presents how to create SoC specification by using UML models and actor-oriented modeling. The authors emphasized the efficiency of modeling the static and dynamic nature of a system in SoC design by using a number of UML diagrams. For example, UML class diagram provides a static view of the system by describing blocks of the system using the class and the relationship among them by the connection lines as shown in Figure 3.10.

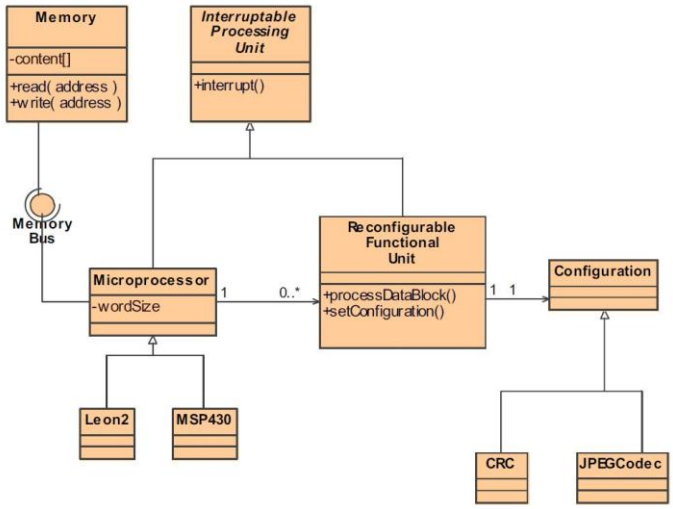

Figure 3.10 UML class diagram [94]

On the contrary, UML sequence diagram represents the interaction among classes over the time. Figure 3.11 shows the sequence of the communication between a sender and a receiver to describe the behavior view of the system.

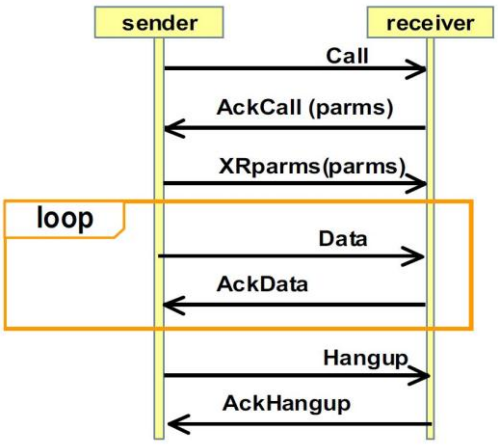

Figure 3.11 UML sequence diagram [94]

To summarize, the UML-based specification enables to describe the structural and behavioral system specification more comprehensively by means of graphical notations than the aforementioned traditional specifications. However, UML diagrams are considered as conceptual models. Thus, extracting the desired information from the UML-based specification requires the interpretation of the related diagram.

3.5 XML-based Specification

The XML is a subset of Standard Generalized Markup Language (SGML) [95]. It becomes a W3C recommendation language to store many different kinds of data in textual form with markup tags on Web [96]. The main contribution of this language is to represent metadata in order to exchange information across different systems. Additionally, it provides several important benefits such as extensibility, flexibility, availability of various data types, portability, and machine-readability. Thanks to its advantages, XML is widely applied to various areas. Recently, it is used as an emerging method in SoC design to represent the specification. For example, IP-XACT has been proposed by SPIRIT consortium [98] as an XML-based specification language to describe the specification of digital IPs. Similarly, Analog Specification Description in XML (ASDeX) for analog IPs has been introduced in the literature [106]. The following subsections present these XML-based specification languages in detail.

3.5.1 IP-XACT for Digital IPs

IP-XACT standard has been developed for packaging, integrating, and reusing IPs provided by different vendors within tool flows [98]. This standard provides an efficient way to describe and interpret metadata for IP integration and configuration requirements with the help of its well-formed XML schema that describes the syntax and semantic rules [97]. Furthermore, the IP-XACT schema provides seven top-level schema definitions: bus definition, abstraction definition, component, design, abstractor, generator chain, and design configuration. Each schema definition can be used to encapsulate the object description. For example, the bus definition object defines the information of a bus. Its schema contains a set of elements and attributes to encapsulate the appropriate information. Figure 3.12 illustrates all objects and their interactions. As described in this figure, the interaction between objects is denoted by the arrow. This arrow (A → B) represents a reference relationship of one object to another (e.g. reference of object B from object A).

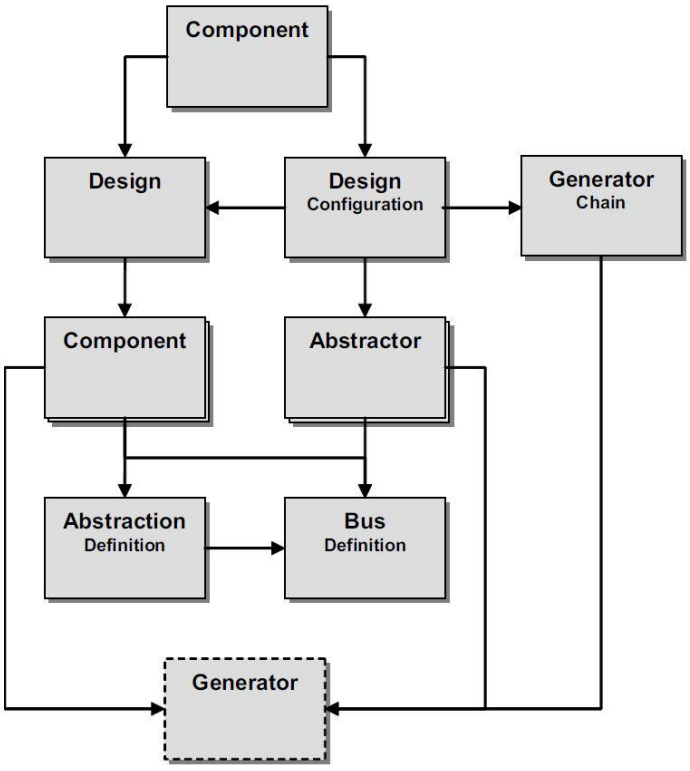

Figure 3.12 IP-XACT data object interactions [98]

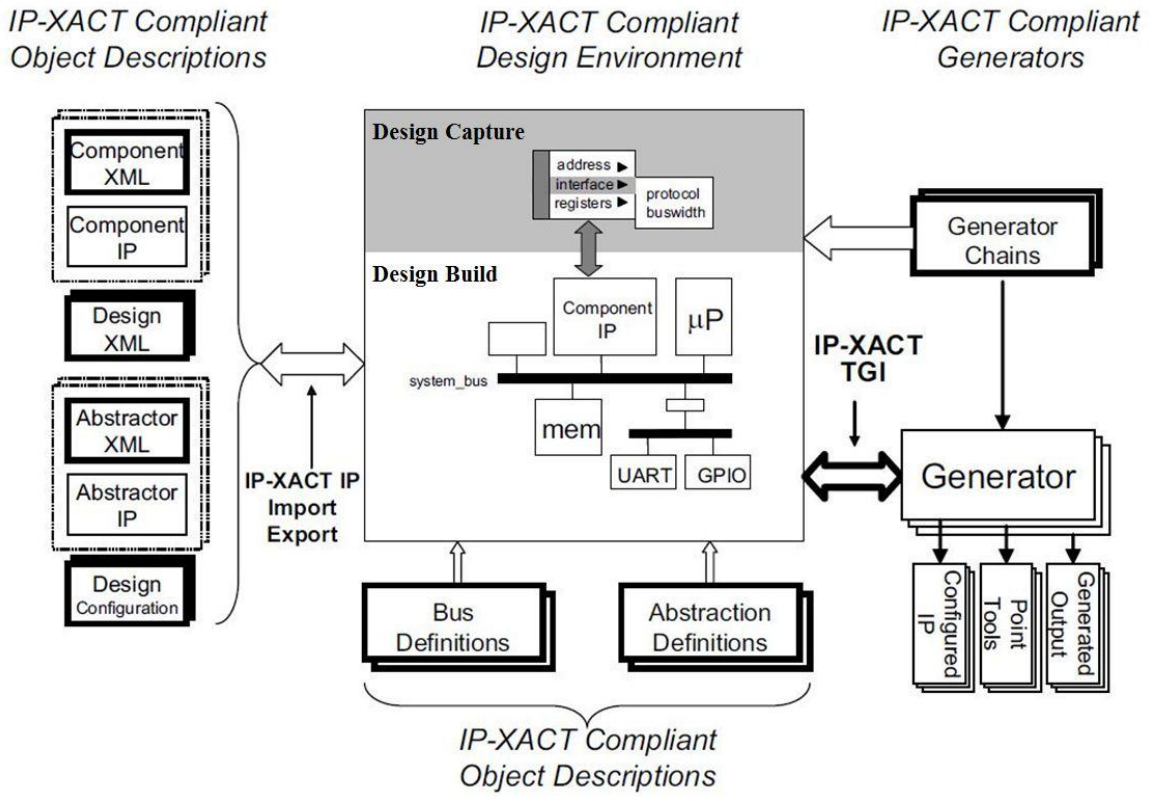

Figure 3.13 IP-XACT design environment [98]

Figure 3.13 describes the IP-XACT design environment. This design environment enables the system designer to create and manage the top-level meta-description of a system by using IP-XACT design IP. For this purpose, it provides two types of services: design capture and design build. The design capture is to capture the structure and configuration information of imported IP by a system design tool. The design build is to achieve the integration and configuration of required IP by using generators provided by a system design tool. Indeed, IP-XACT specifications are being broadly adopted by IP and EDA vendors through their own IP-XACT enabled design environment [99], [100], [101], [102], [103]. To give an example, an environment named Kactus2 for embedded product development is presented in [101]. This system uses IP-XACT to describe HW components. To search component, the authors introduced the component capture method based on a tuple {Vendor, Library, Name, Version} (VLNV) that is given as a unique identifier by the IP-XACT object description. Furthermore, to improve this VLNV-based method, [102] and [103] introduce the extended IP selection method and an unambiguous VLNV naming convention respectively. The former paper highlights that nonfunctional information such as power dissipation should also be considered to find and select functionally suitable ones among many different predesigned components in IP libraries. The authors thus proposed to describe components by

using an extended IP-XACT with important additional information and to filter them by criteria based on keywords and nonfunctional information. As a result, the proposed method aids to select the best-suited components. The latter paper introduces unambiguous VLNV naming conventions by using keywords which are based on detailed taxonomy of components. Hence, the VLNV name composed of keywords helps to find the appropriate ones by removing its ambiguity.

3.5.2 ASDeX for Analog IPs

IP-XACT is a specification language for digital IPs but unfortunately it is not suitable for analog IPs. For this reason, ASDeX has been proposed as a specification language in XML to describe the specification of analog circuits in [104], [105], [106]. This language covers all the required information for the automatic design validation with it. Figure 3.14 shows the schema of ASDeX which consists of the identifier block *design*, the main block *specification* and the three optional blocks *function*, *testbench* and *task*. These blocks and their interactions are described in Figure 3.15. As stated in the previous subsection, the arrow represents the reference relationship between blocks. For instance, the specification description is referenced by tasks and testbenches.

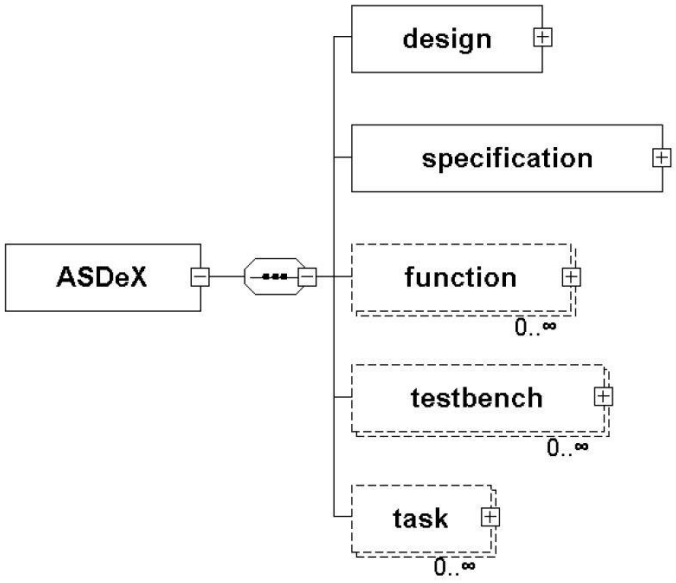

Figure 3.14 Schema of ASDeX [104]

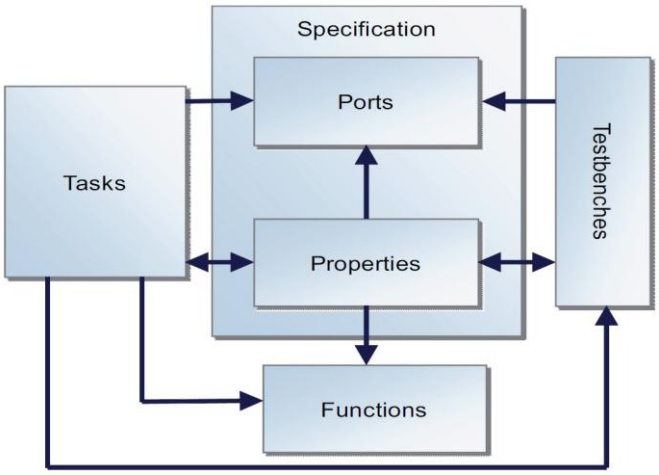

Figure 3.15 ASDeX data object interactions [104]

In addition, a validation work flow is introduced to validate analog circuits by using ASDeX as described in Figure 3.16. This work flow is based on the Essence Framework and the meta-simulator. Especially, the Essence Framework transforms ASDeX specification into an execute task for validating circuits. To achieve this validation, all the required information for building up a testbench, simulation setup and result calculation could be extracted from ASDeX documents by using the templates derived from the template library. The same authors also presented an Application Programming Interface (API) named AXEI (ASDeX Engine Interface) to derive the desired information from the specification more easily than using the templates in [106].

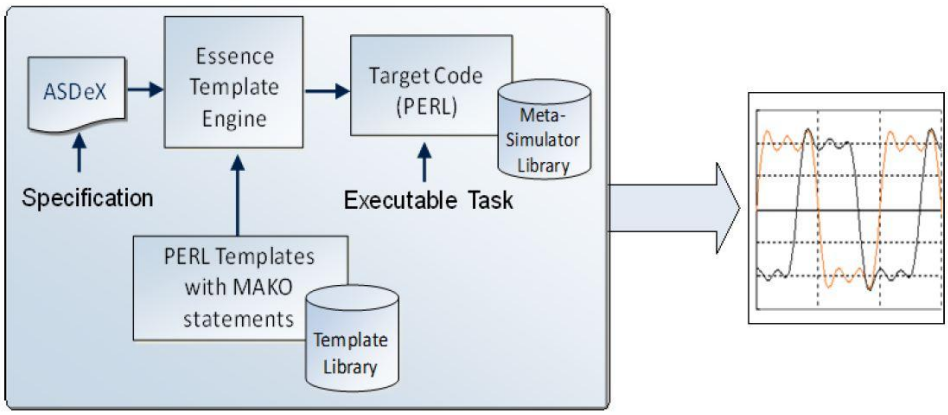

Figure 3.16 Validation work flow based on ASDeX [104]

As shown above, the XML-based specification represents a tree data structure that allows encapsulating various data sets. As a consequence, it enables to store all necessary data in an XML format file in accordance with the predesigned XML schema. The previous

studies show well the facility of capturing data from the XML documents by using templates thanks to its machine-readability. Furthermore, several software supports for XML and XML-specific language like eXtensible Stylesheet Language Transformations (XSLT) [110] greatly help to deal with XML documents. However, both IP-XACT and ASDeX propose an appropriate data model for the component description.

3.6 STMicroelectronics' Design Platform Specifications

As mentioned in the previous chapter, the current design platform specifications at STMicroelectronics are a collection of heterogeneous information from technology-dependent parameters to CAD tools' information. Thus, these specifications are made by natural language-based and table-based documents as given in Figure 3.17. From the perspective of end-users like library developer, these types of the specifications become more and more difficult to extract the desired data from them due to their increasing complexity with technology scaling and different forms. In addition, the data extraction depends on human inspection or scripts of the library developer. Thus, we need a unified form to cover all of the specifications as well as novel methods to cope with it.

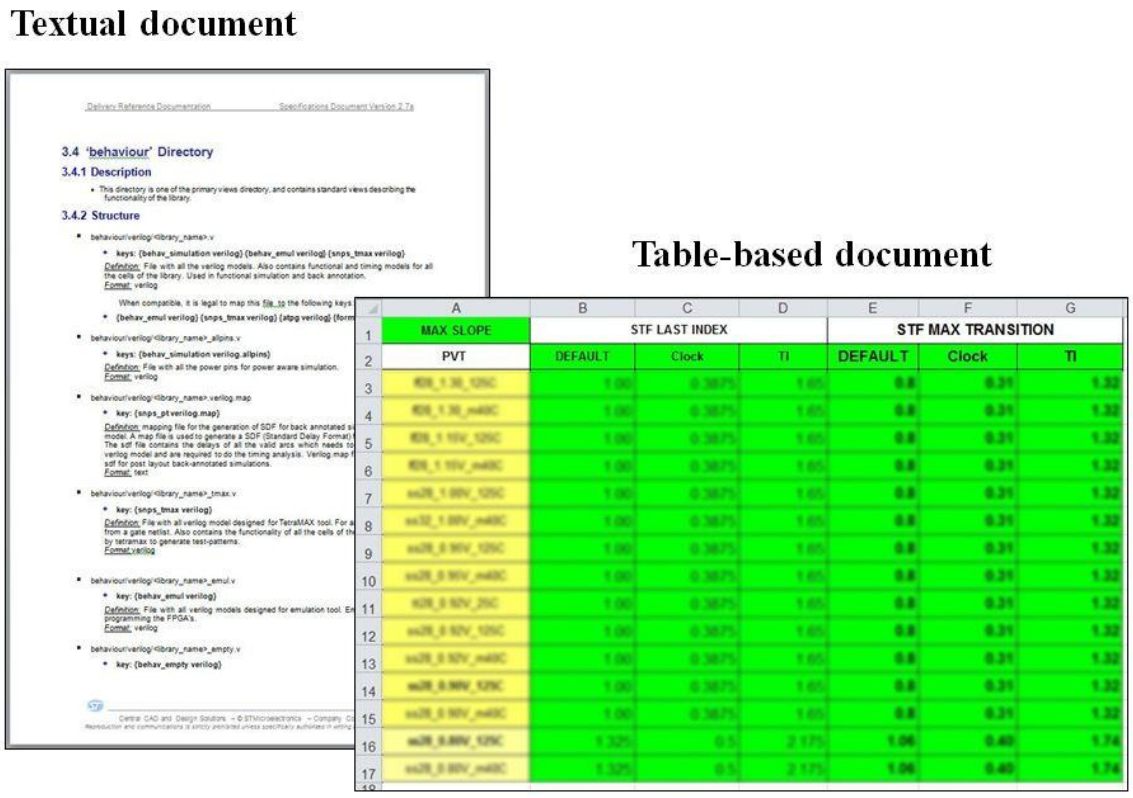

Figure 3.17 STMicroelectroincs' design platform specifications (source : ST)

3.7 Discussion

We have previously seen four kinds of the specification. Table 3.1 summarizes their advantages and disadvantages. As given in this table, there are six and four criteria for data modeling and data processing respectively. According to these criteria, each specification has been estimated. In the table, the circle symbol represents “good” and the X symbol represents “bad”. To give an example, for data modeling, the natural language-based specification provides good extensibility, flexibility, expressiveness and availability of various data types for data modeling, whereas it has bad formality and data relationship. In addition, for data processing, it gives good facility of entering data and human readability but bad facility of capturing data and machine-readability. Likewise, the advantages and disadvantages of other specifications are also given. On the contrary, STMicroelectronics’ design platform specifications have both the features of the natural language-based and table-based specifications.

Table 3.1: Summary of the specifications

Criteria		**Natural language-** based specification	**Table**-based specification	**UML**-based specification	**XML**-based specification
Data Modeling	Formality	✗	○	○	○
	Extensibility	○	○	○	○
	Flexibility	○	○	○	○
	Expressiveness	○	○	○	○
	Availability of various data types	○	○	○	○
	Data relationship	✗	✗	○	○
Data Processing	Facility of entering data	○	○	✗	○
	Facility of capturing data	✗	○	✗	○
	Human readability	○	○	○	✗
	Machine readability	✗	✗	✗	○

3.8 Conclusion

In this chapter, we presented natural language-based and table-based specifications. In addition, the UML-based and XML-based specifications were introduced. The current specification used at STMicroelectronics was also discussed. The advantages and disadvantages of these specifications were compared and then the comparison results were given.

Consequently, we first need a unified specification to represent all required information independent of cells for library development instead of the current ones. The above table shows that the best form of the specification is the XML-based specification over all criteria. Thus, we aim at proposing an XML-based specification thanks to its attractive benefits and software supports. However, we must propose a suitable one because of the absence of the XML-based specification for our target information. On top of that, we also need efficient methods to create a reliable specification and to extract desired data from the target specification.

In the subsequent chapter, we try to provide answers to the following question:

- How can we propose an XML-based specification for the specifications of library development?
- How can we create a reliable specification?
- How can we precisely and rapidly extract the required information from the specification?

4. Methodology for Library Development Specifications

Contents

4.1 Introduction

One of the most crucial issues for dealing with our target specifications is to create a unified specification, which could contain a wide variety of information independent of cells.

In addition, for efficient data processing, we also have two important challenges. The first is how to create a reliable specification containing a huge amount of various data. The second is how to precisely and rapidly extract the desired data from it in order to accomplish the library development process.

In this chapter, we first introduce an appropriate data model to represent the specification for library development. Furthermore, two key methods are presented for specification creation and specification data extraction. Additionally, some approaches to verify the specification and library are also discussed.

4.2 Formalism of the Specification

Before discussing the formalism of specification, we have to clarify its requirements. Then, the identification and classification of specification data are required, i.e. we need to collect all required information for library development and analyze it. These studies may help to propose a suitable data model for our target information.

4.2.1 Requirements of the Specification

The objective of the specification is to provide the library developers with all required information for successfully accomplishing their work. From this perspective, the most important requirements of the specification were given as follows:

- **Formality**: The specification covers various categories of information. The data in each category must be formalized by determining their essential elements to give complete information. This formality enables to define all necessary specification data in a unified way.
- **Consistency**: The information given in the specification should be consistent to avoid mismatch errors which can greatly impact the quality of the library.
- **Completeness**: Since the current specifications do not provide complete information for library development, library developers have difficulties in finding missing information. Thus, if all required information is defined in the specification, they are able to get all necessary information from a unique source quickly.

- **Extensibility**: The technology evolution leads to increase the volume of specification data. For this reason, the specification data model has to address increasing data volume.
- **Unambiguousness**: The information in the specification must be comprehensible and unambiguous for all library developers. It aids to reduce specification errors like misunderstanding.

4.2.2 Specification Data Analysis

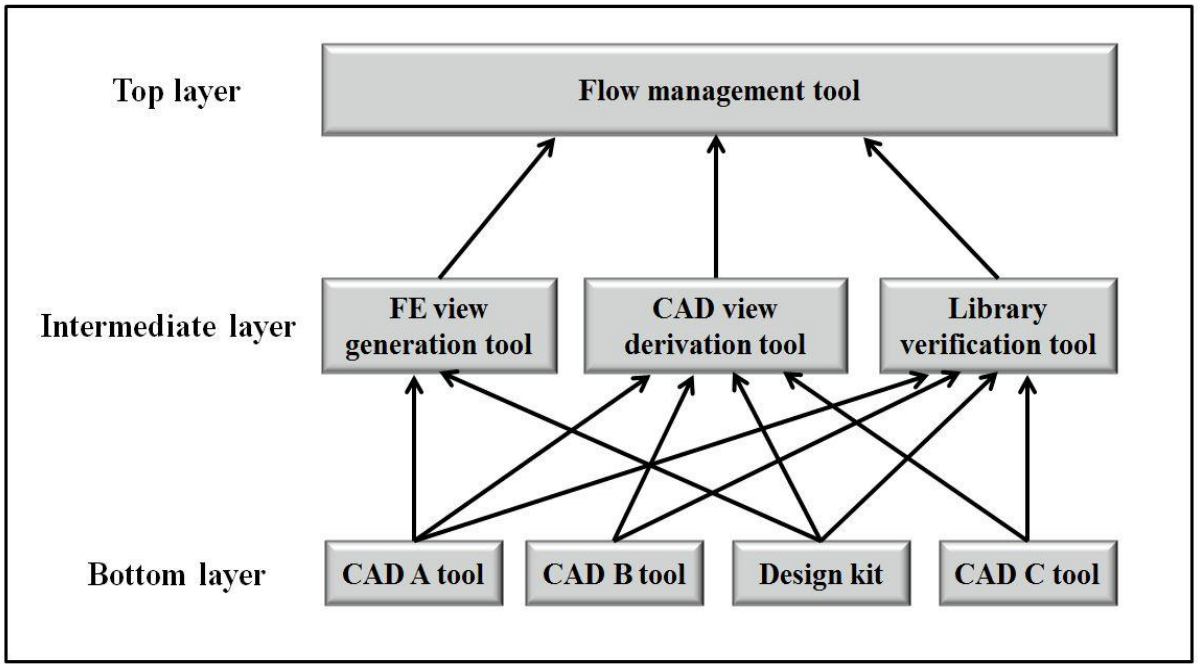

Figure 4.1 Automatic library development system (source: ST)

At STMicroelectronics, the library development is carried out by using an automatic library development system based on in-house and commercial CAD tools as described in Figure 2.10. This automatic system covers aforementioned three main phases for library development: FE view generation, CAD view derivation and validation phases. Figure 4.1 describes the hierarchical layers of the system and the interoperability between integrated tools at different layers. At the top layer, a flow management tool controls the whole process by using the specific in-house tools. These in-house tools at the intermediate layer execute all tasks for generating, deriving, and validating the library views with the help of CAD tools and a design kit.

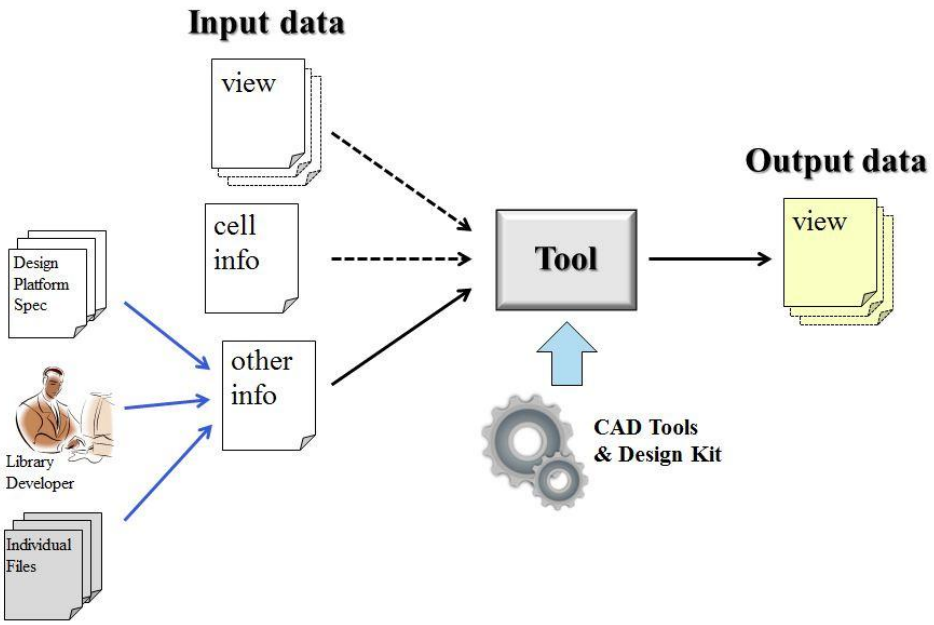

Figure 4.2 A task for library development (source: ST)

Definition	**Task for library development** A task permits to generate more than one view or to verify it using the given input data and the corresponding tool.

Figure 4.2 describes how the in-house tool executes a task for library development. This tool requires a set of input data to produce output data. As shown in this figure, the input data splits into three groups. The first group includes the view obtained from the previous task. The second group represents the information concerning the cells such as their functional and structural description. The third group represents all other information. A large part of the other information can be extracted from the current design platform specifications, whereas the remaining information may be gathered from several sources such as library developer's knowledge and individual documents.

The necessary input data for each task can be determined by the characteristics of the target task. For example, the task to produce a behavioral view requires only cell information because the behavioral view such as Verilog model mainly includes the structural and functional description of cells which can be used for the RTL design of the system. On the contrary, generating a timing model of the cell is highly dependent on technology and thus this task needs not only a SPICE netlist file and cell information but also a number of technology parameters as input data.

In this thesis, we focus on input data of the third group, i.e. data that are not related to the cell but to the design platform, library developer's knowledge and individual documents. Firstly, all input data for cell characterization, behavioral modeling, and CAD view generation were identified and classified according to the data categories. For example, the characterization tool for timing model generation performs the cell timing characterization with various data. The required input data for the timing characterization of a non-scan D flip-flop named DFPQX2 are given in Figure 4.3. First, Spice simulator performs the simulation. Second, design kit provides Spice models. Third, Spice netlist of this flip-flop can be obtained from the Spice file (.spi). Finally, the information about technology parameters such as PVT corner, slopes of pin D and of pin Clock for the given PVT corner, and capacitive loads for drive strength (X2) are needed. By using all these parameters, the characterization tool generates a circuit file (.cir) including the condition of the simulation, the spice netlist, the statements of data extraction and so on. The simulation can be accomplished with this circuit file and the desired timing information can be measured. As a consequence of the timing model generation, a Liberty file (.lib) containing the timing information of the cell can be produced.

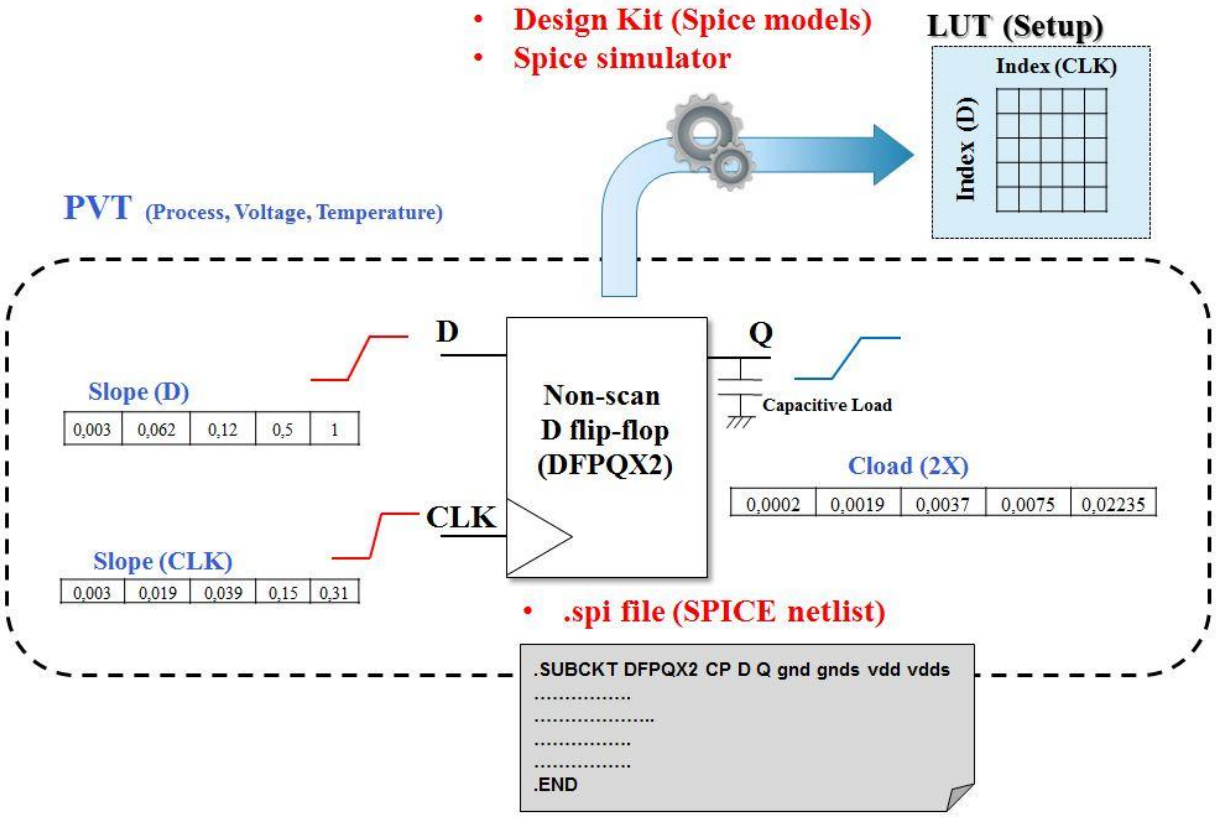

Figure 4.3 Timing characterization of a non-scan D flip-flop (source: ST)

By the same method, the input data for other tools could be analyzed. All required data for each tool are divided into three data groups: **tool**, **technology** and **view**. According to this, the identified parameters could be classified and counted. Figure 4.4 gives the tool input data analysis results. The results show that the dominant parameters rely upon the characteristics of the activity of the tool. For instance, as depicted in this figure, the characterization tool needs a set of technology-dependent parameters as well as a large amount of tool configuration parameters that are employed in running the accurate simulation.

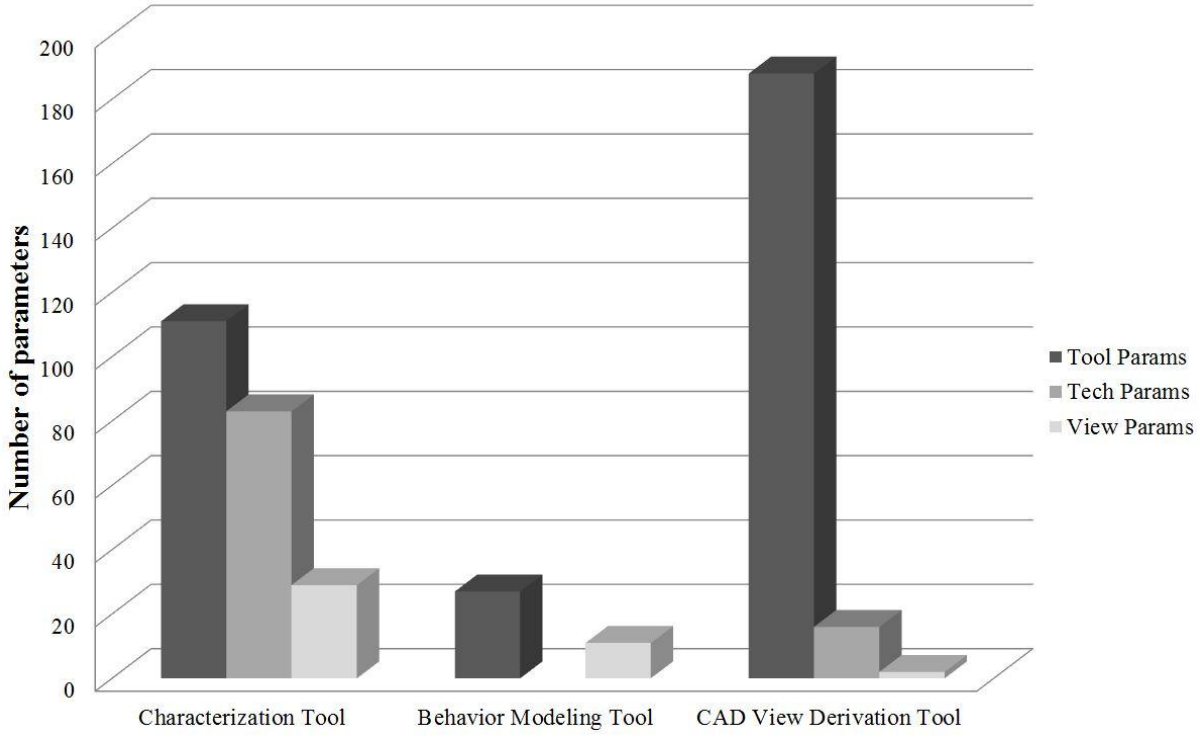

Figure 4.4 Tool input data analysis results (source: ST)

If all of these parameters collected from different sources are defined in a specification, the library developer can easily get complete information from a unique source. But a supplementary data group needs to be added to provide the information about the design flows supported by the library. This is because the supported design flows determine which library view must be produced and which tools are needed for system design and library development. It can provide the bridge between tool and view information. Thus, we propose four main data groups: **technology**, **tools**, **views** and **design flows**.

4.2.3 Specification Data Classification

Classifying the specification data in detail is significantly important for data modeling. In the previous subsection, we present four main data group. Further, we also propose subgroups for each data group to present the detailed taxonomy of data as shown in Figure 4.5.

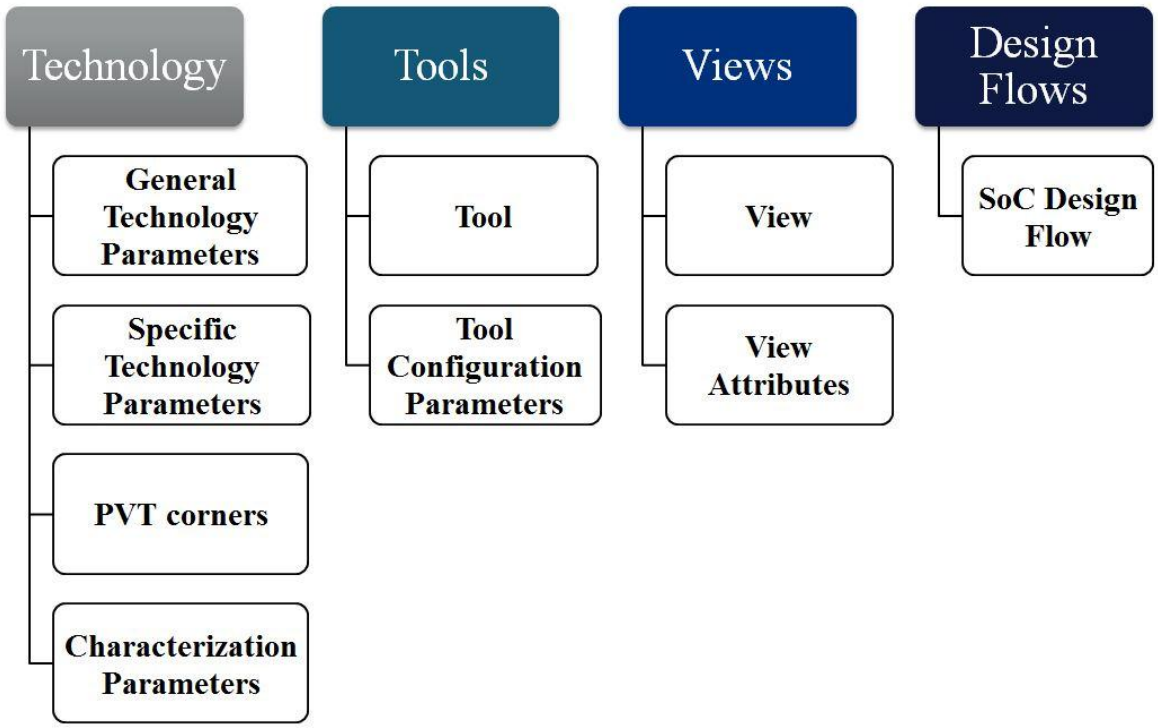

Figure 4.5 Specification data taxonomy

Now, we present which data can be included in each data group as follows:

- **Technology**

The technology data group must cover all technology-dependent parameters from fundamental technology information to process parameters for characterization. The concerned information was regrouped into four data subgroups by the characteristics of data as described in Figure 4.5.

- **General Technology Parameters**: represents the fundamental technology information independent of library category. For example, the target technology node can be regarded as a general technology parameter.
- **Specific Technology Parameters**: represents the fundamental technology information dependent of library category. For instance, the library architecture may be regarded

as a specific technology parameter because there are several different library architectures by the number of tracks for standard cell library and by the type of frame for I/O library. The values of this parameter rely on the target technology.

- **PVT corners**: represents PVT corner lists. The PVT corner list should be given for each library category because the required PVT corners or the essential elements of a PVT corner can be different. For instance, for the operating condition, one supply voltage is needed for standard cells, whereas memories may need two supply voltages.
- **Characterization Parameters**: provides all necessary process parameters for cell characterization. Specifically, characterization references, slopes, and capacitive loads can be included in this data subgroup.

- **Tools**

The main goal of this data group is to provide the user with a complete list of all necessary tools and design kit. Additionally, this data group can contain tool configuration parameters.

- **Tool**: provides the underlying information about a tool such as tool name and tool version.
- **Tool Configuration Parameters**: provide the specific configuration parameters according to the purpose of the tool. These parameters are missing in the current specifications in spite of their importance and needs. Moreover the tool configuration is one of the most time-consuming steps in library development. Thus, this data subgroup aims at automatically setting up the correct tool configuration.

- **Views**

The main goal of this data group is to give a list of library views to produce a complete library package. In addition, this data group may also include view attributes.

- **View**: provides the underlying information about a view such as its view path that specifies its physical location in the directory structure of the library. In addition, it determines whether the view should be generated by the library category. For example, IP-XACT library view is required for memories and IPs but not for standard cell library.
- **View Attributes**: provide the specific attributes of a library view. Generally, the definition of view attributes totally depends on the tool that generates the view. Unfortunately, it does not ensure whether they are well defined in the view. For this

reason, to provide these specific view attributes is proposed to correctly generate a view with them. For example, Liberty library has its own specific attributes, for example, *direction* attribute for specifying the direction of a pin [44]. The view attributes must be defined at different levels such as library, cell and pin depending on which level relates to the information given by it.

- **Design Flows**
  - **SoC Design Flow**: provides the information about the supported design flows by the library.

Finally, most of the identified data were classified according to the data taxonomy mentioned above. Table 4.1 gives the classified specification data. For example, the general technology parameters data subgroup contains all technology parameters independent of library category: technology name, technology node, process, metal option, voltage option, Resistor and Capacitor (RC) type, maximum temperature, minimum temperature, and Design Rule Manual (DRM). Likewise, other parameters are also classified in the corresponding data group. If there are new parameters, the specification developer can easily add them to one of the related data group.

Table 4.1: Specification data

<table>
<tr><th colspan="6">Technology</th></tr>
<tr><th>General Technology Parameters</th><th>Specific Technology Parameters</th><th>PVTs</th><th colspan="3">Characterization Parameters</th></tr>
<tr><td rowspan="2">Technology name,
Technology node,
Process,
Voltage option,
Metal option,
Resistor and Capacitance type,
Max. temperature,
Min. temperature,
Design rule manual</td><td rowspan="2">Library architecture,
Library type,
Multivolt supplies,
Multivolt grounds</td><td rowspan="2">Process,
Voltage,
GND,
Temperature,
Age</td><td>Character-ization Reference</td><td>Slope</td><td>Capacitive load</td></tr>
<tr><td>Time,
threshold,
Slope threshold,
Cap threshold
Time unit,
Load unit,
Load unit,
Power unit,
Voltage unit,
Energy unit,
Glitch tolerance,
Peak tolerance</td><td>Max transition(Clk),
Max transition (Data),
Max transition (TestIn),
Last index (Clk),
Last index (Data),
Last index (TestIn),
Intermediate values constraint (Clk),
Intermediate values (Data),
Intermediate values (TestIn)</td><td>Max capacitive load (worst corner),
Max capacitive load (best, typical corners),
Index timing,
Index power</td></tr>
<tr><th colspan="2">Tools</th><th colspan="2">Views</th><th colspan="2">Design Flows</th></tr>
<tr><th>Tool</th><th>Tool Configuration Parameters</th><th>View</th><th>View Attributes</th><th colspan="2">SoC Design Flow</th></tr>
<tr><td>Commercial CAD tools
(Layout editor, synthesis tool, SPICE simulator, etc.)
In-house tools
(Characterization tool, Behavioral modeling tool, CAD derivation tool, validation tool)
Design kit</td><td>Characterization tool configuration parameters,
Behavioral modeling tool configuration parameters,
View derivation tool configuration parameters</td><td>Documents,
IP-XACT,
Verilog,
VHDL,
Symbol library,
Liberty library,
CCS library,
ECSM library,
LEF,
DEF,
CDL,
OpenAccess,
Milkyway,
ATPG</td><td>Liberty library attributes</td><td colspan="2">FrontEnd,
BackEnd,
SignOff</td></tr>
</table>

4.2.4 Specification Data Modeling

Data model describes the data objects with their properties and the relationship between objects in order to represent the real world. From this aspect, we created an appropriate data model to describe the specification for library development on the basis of the observations about the specification data presented previously.

The specification data model consists of seven main data objects encapsulating the categorized data given in Table 4.1. Figure 4.6 represents all data objects as well as the reference relationship between them.

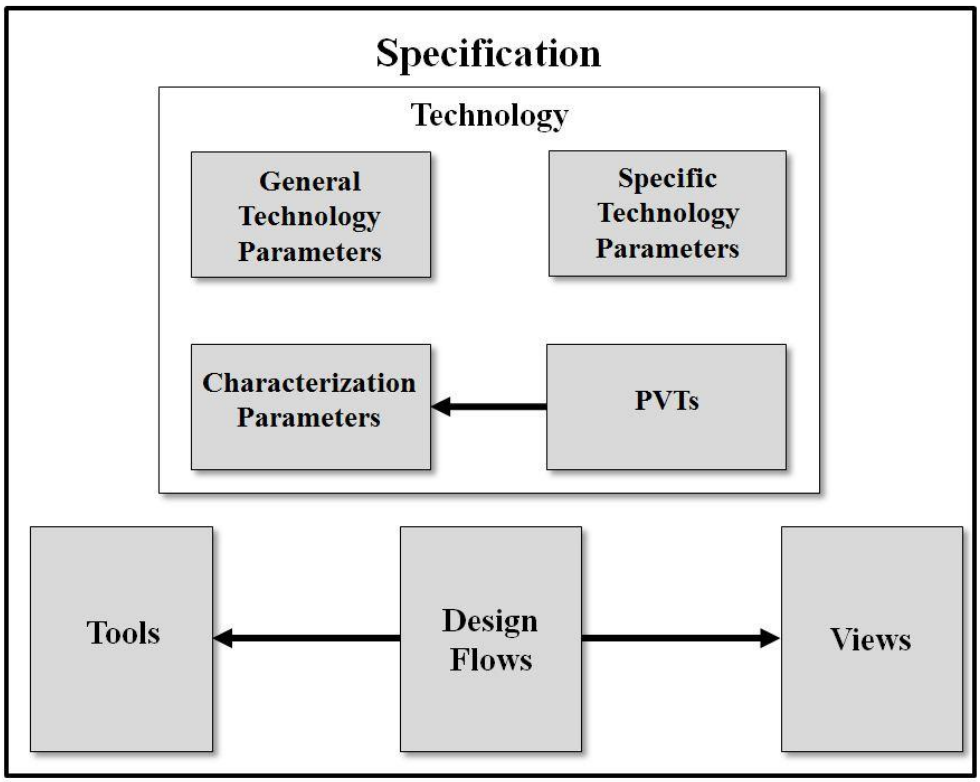

Figure 4.6 Specification data objects and their reference relationship

a) Data object

Definition	**Data object** A data object (indicated by a gray box) represents one or more parameter sets.

One of the most important issues is to represent a variety of data objects in a unified way. For this reason, we need a generic and simple data model that enables to represent the parameter set. We propose a basic data model for the data object as shown in Figure 4.7 by

using UML class diagram notation [112]. The parameter set may represent more than one parameter. The parameter information can be given with its name and value elements. This data model is used to create the specification data model over all data objects.

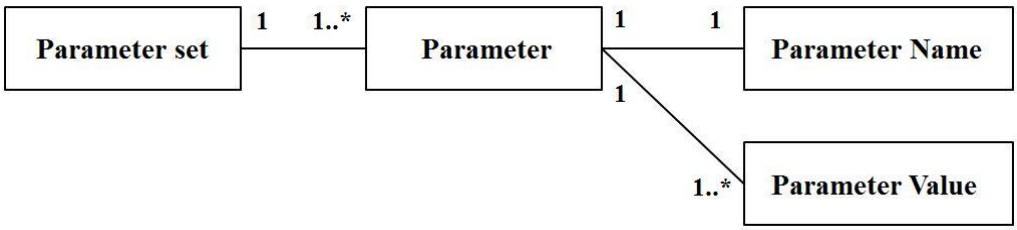

Figure 4.7 Basic data model of the data object

There are two types of data objects depending on whether their parameters are sub-categorized or not. Figure 4.8 illustrates the example of these different types. As shown in Figure 4.8 (a), since **General Technology Parameters** data object contains a set of general technology parameters regardless of the library category, it may be referred as the first type. In contrast, **Specific Technology Parameters** data object may be referred as the second type because it may contain several subsets (so-called List) of technology parameters sub-categorized by the library category as described in Figure 4.8 (b).

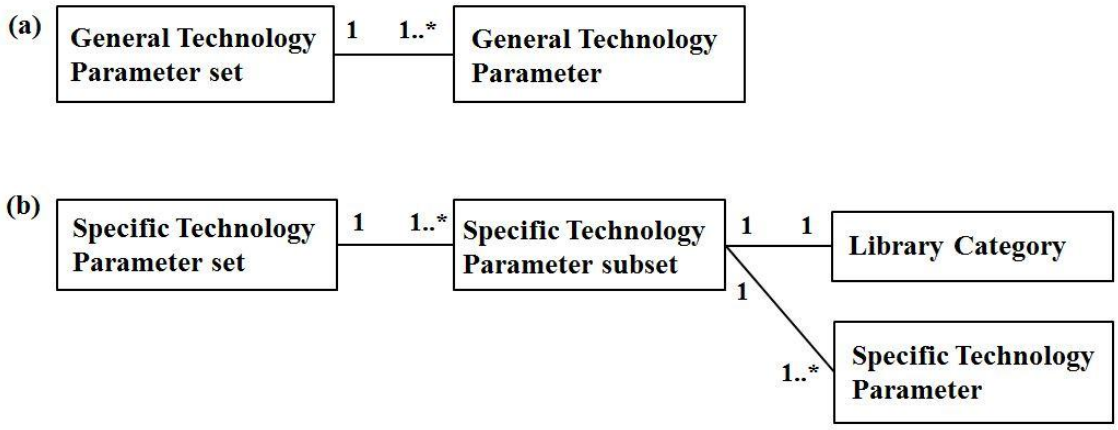

Figure 4.8 Examples of (a) non-subcategorized parameter set (b) sub-categorized parameter set

As described in Figure 4.7, in order to define the parameter information, we propose a simple unified way to give its name and values. In addition to these fundamental elements, the key attribute is used as an identifier at STMicroelectronics to identify the tool and view instead of its name. However, there is no common agreement for naming the key attribute. For this reason, we propose a naming convention for the key attribute as follows:

key = <identifier>_<purpose>

The key is composed of two fields: *identifier* and *purpose*. The first field *identifier* is a mandatory field to give identification information. The second field *purpose* is an optional field to give the specific purpose of the tool or view that may relate to the activity of the system design or library development. In addition, this attribute can also be used to link its parameter with other parameters. For instance, a tool and its suitable configuration parameters are coupled by the tool key. As given in Table 4.1, both the characterization tool and its configuration parameters may be identified by their same tool key. Likewise, a view and its view attributes can be coupled via the view key.

Consequently, there are two kinds of parameter data model according to the presence or absence of the key attribute.

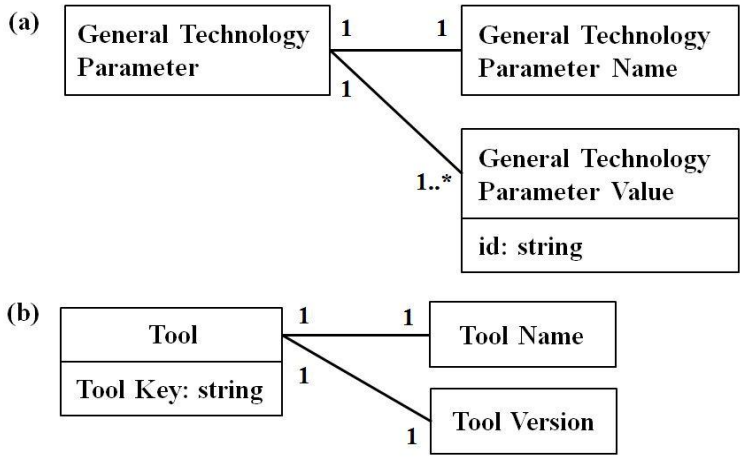

Figure 4.9 Examples of (a) parameter without key attribute (b) parameter with key attribute

Figure 4.9 (a) represents the first kind of parameter. As shown in this figure, the general technology parameter can be defined by a parameter name and values. For instance, a technology node can be expressed as follows:

General Technology Parameter Name = "**TechnoNode**"

General Technology Parameter Value = "**28**"

Additionally, when the parameter has more than one value, we can assign a value identifier attribute named **id** to each value. For instance, the information about process can be given as below:

General Technology Parameter Name = "**Process**"

General Technology Parameter Value = "**ff**", **id** = "**best**"

General Technology Parameter Value = "**tt**", **id** = "**typical**"

General Technology Parameter Value = "**ss**", **id** = "**worst**"

Figure 4.9 (b) represents the second kind of parameter. As described in this figure, the tool information consists of a tool name, version and key attribute. As explained previously, the tool key attribute is used to identify the desired tool. Especially, in the case that CAD vendor offers a tool suite containing several tools, this attribute is very useful to identify the desired tool instead of the name of tool suite. For instance, since Synopsys provides a tool suite named **synthesis** providing Design Compiler, Library Compiler, etc., we have to load **synthesis** to use Design Compiler. Thus, the tool information for Design Compiler can be given as follows:

Tool Key = "**designcompiler_synthesis**"
Tool Name = "**synthesis**"
Tool Version = "**i-2013.12**"

b) Relationship of data objects

We defined the data objects of the specification above. Now, we note the relationships between them expressed by the arrow as shown in Figure 4.6. Even if the data objects are structurally independent of each other, there are relationships between their information, which can be identified by the common element of data objects.

Firstly, 'Characterization Parameters' data object depends on 'PVTs' ones because the process parameters for cell characterization, like slopes, change by the PVT corner. For this reason, the number of required slope values depends on the number of the given PVT corners. The relationship of these data objects can be identified by PVT corner's name.

Secondly, in order to perform design activities, we need CAD tools and library views at each step. It means that the design flow has an important influence on determining which tools and library views should be required so that 'Tools' and 'Views' data objects are related to 'Design Flows' data object. We propose to use the key attribute in order to define the relationship between these three data objects. As mentioned above, the key attribute contains a purpose field. In order to link the design flow with tools and views, the purpose field of the key attribute should be based on the supported design flow. For example, the information about Front-End design flow can be defined in 'Design Flows' data object as follows:

SoC Design Flow Name = "**FrontEnd**"
SoC Design Flow Value = "**synthesis**"

As is well known, we need a synthesis tool like Design Compiler and a Verilog model for the logical synthesis and thus the key attributes relevant to these elements can be expressed as follows:

ToolKey = "**designcompiler_synthesis**"

ViewKey = "**verilog_synthesis**"

Consequently, Figure 4.10 represents a complete data model of the specification which is made up of basic data objects described in Figure 4.6. This data model covers the data given in Table 4.1. As shown in this figure, the class name is simplified instead of a being long name. For example, 'General Technology Parameters' data object is expressed by a class element named **GeneralTechParams**. This class is able to define all general technology parameters in Table 4.1 (Technology name, Technology node, Process, Voltage option, Metal option, Resistor capacitance type, Maximum temperature, Minimum temperature, and Design rule manual).

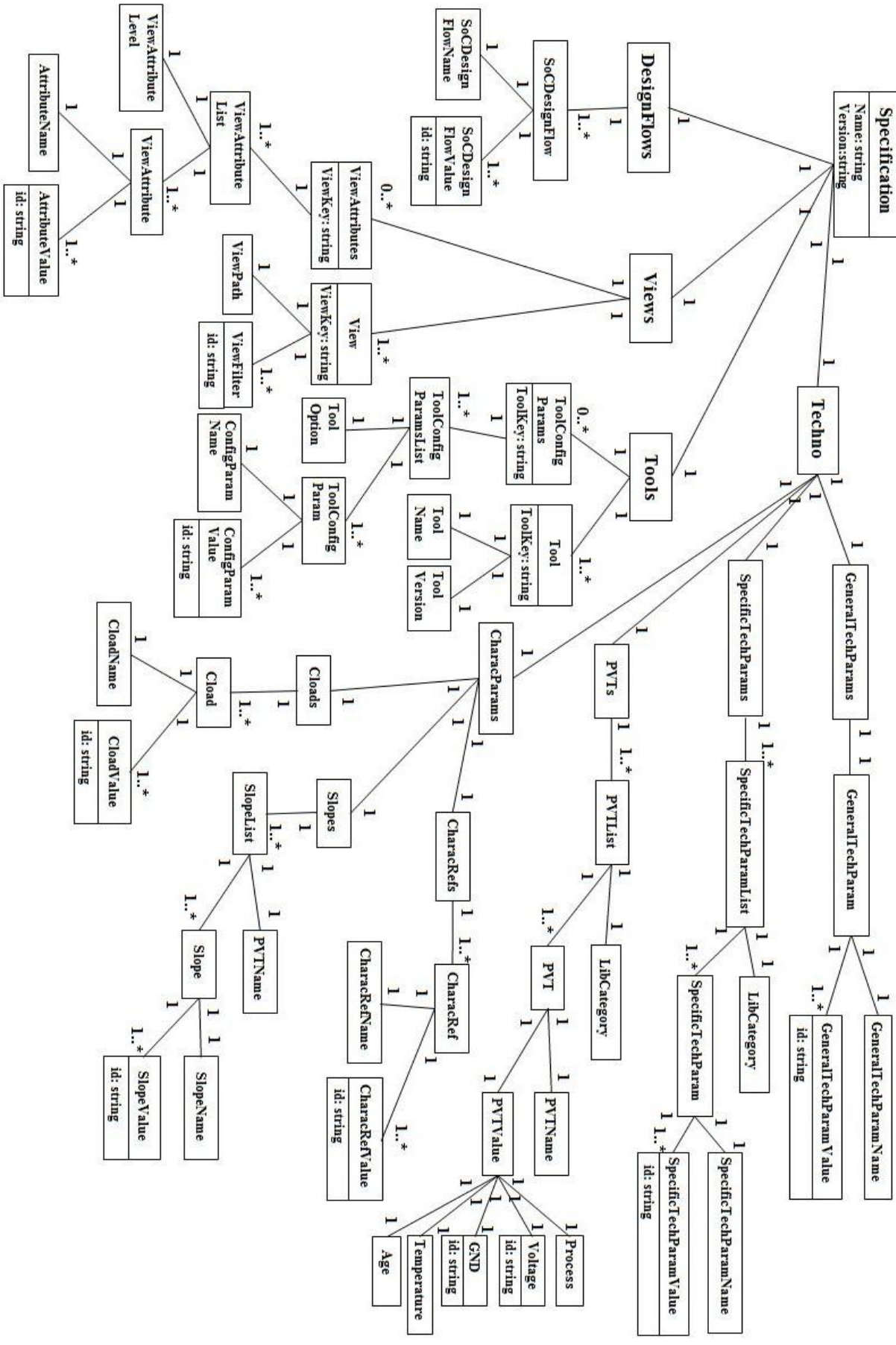

Figure 4.10 Data model of the specification

4.3 Reliable Specification Creation Method

The specification for library development is a collection of information obtained from several information owners and existing references. This specification is traditionally created depending only on the expertise of the specification developers. In addition, a sequence of activities which gathers the information and defines it in the specification is a totally manual work. Unfortunately, no efficient unified method is available to achieve this significantly important work. Moreover, the share-ability and usability of the existing references are very low due to the complex data acquisition from them. For these reasons, the specification reliability highly depends on the attention the specification developers pay to remove specification errors such as inconsistency and incompleteness.

Thus, we propose a reference-based method to create a reliable specification by addressing the critical issues of the traditional manual methods for specification creation. Its main ideas are to provide as much predefined information as possible from a centralized reference database and to increase the share-ability and usability of the references. Generally, the specification data can be divided into two parts: one is the information such as PVT corners, slope and capacitive load values which must be given by its owner. Another is the information that can be derived from the predefined documents. Our reference-based method focuses mainly on the second one.

4.3.1 Reference Database

In Figure 4.9, two types of parameters were introduced. In order to cover them, we propose a reference database that consists of two kinds of references: dictionary and list reference. The dictionary provides the reference relevant to the first type of simple parameters without a key attribute. The list reference provides the reference relevant to the second type of parameters with a key attribute.

a) Dictionary

The dictionary is a collection of terms. Like a semantic dictionary providing all meanings of a term, the proposed dictionary also aims at providing all possible values of the predefined term. Figure 4.11 shows its data model. As shown in this figure, a term enables to encapsulate any parameter information with a name and more than one value.

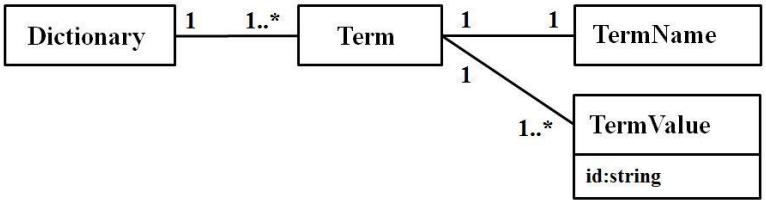

Figure 4.11 Dictionary

By using this simple and formal representation, most of the shareable data for specification creation can be predefined in the dictionary. Especially, this dictionary permits to cover parameters corresponding to the data model in Figure 4.9 (a) thanks to its similarity. In order to support such data, the list of parameters and all available values of each parameter can be derived from this reference. For example, to define the general technology parameters, their complete list and desired parameter values are required. First, **GeneralTechParam** term having all parameter names as term values (**TechnoNode**, **Process**, **VtOption**, **MetalOption**, and so on) may be defined in the dictionary to give a list of general technology parameters. Second, all available values of each parameter may also be given by using this reference. To give an example, **TechnoNode** term with all its possible values (**130nm**, **90nm**, **65nm**, **45nm**, **32nm** and **28nm**) can be defined to provide the information about technology node. In consequence, the specification developers are capable to obtain the complete parameter list from the dictionary. Once having a parameter list, they may retrieve the information about each parameter and then to select desired parameter values to finally define it in the specification. Furthermore, if we need a new parameter or a new parameter value, it is sufficient to add a new term or a new value to the corresponding term. Therefore, the updated information can easily be shared with all users through the dictionary.

b) List reference

The list reference is proposed to provide a complete parameter set at a time unlike the dictionary based on the repetitive retrieval of the term. This reference covers the tool and view information having the key attribute which corresponds to the data model in Figure 4.9 (b). It offers a complete list of predefined keys with its identification information as described in Figure 4.12.

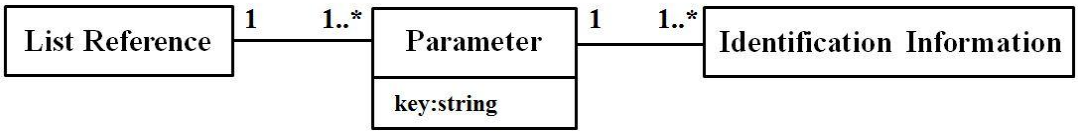

Figure 4.12 List reference

As shown previously, the tool information includes a tool name, version and key attribute. Among them, a couple of tool identification information, which is tool key and its corresponding tool name, is regarded as fixed information. In contrast, the tool version is no fixed information because it must be given by the specification developer. For this reason, the tool list reference must provide only the fixed information as shown in Figure 4.13. As a result, to complete the information for all required tools, it is sufficient to give only the correct tool version for each tool provided by the tool list reference.

To define the information about Design Compiler tool, its tool key "**designcompiler_synthesis**" and tool name "**synthesis**" can be obtained from the tool list reference, whereas the tool version (e.g. "**i-2013.12**") must be given by the specification developer.

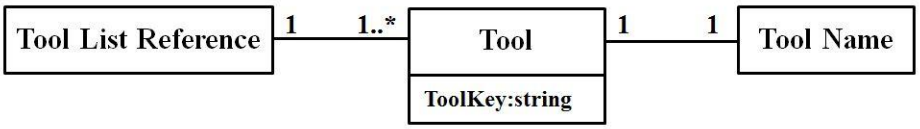

Figure 4.13 Tool list reference

Using the same method as to update the dictionary, defining a new tool or a new purpose of an existing tool can be simply added as a new tool element with its specific tool key and corresponding tool name in this reference. Likewise, a view list reference can be made. Finally, these references help to easily and rapidly define complete tool and view information.

To summarize this subsection, the aforementioned reference database enables the specification developer to create a complete and consistent specification by providing rich predefined information relevant to parameter list, values and identification information. In addition, it also provides several significant benefits. One of them is that enforcing and centralizing the references leads to improve their share-ability and reusability. Another is to provide a unified method to easily extend the specification data via the concerning reference and rapidly apply the updated information to the specification creation. However, when defining a reference, the author must not make an error because it may have a direct impact on all specifications related with this error.

4.3.2 Specification Creation using a Reference Database

As mentioned above, the specification data can be created from the dictionary or list references according to the parameter type. Figure 4.14 shows how to gather specification data from the reference database. As shown in this figure, the dictionary-based flow is carried out by parameter list acquisition and parameter value selection based on the repetitive retrieval. On the contrary, the list reference-based flow is accomplished by parameter list acquisition and parameter value entry. The specification developers can easily define all specification data to produce a final specification through these flows.

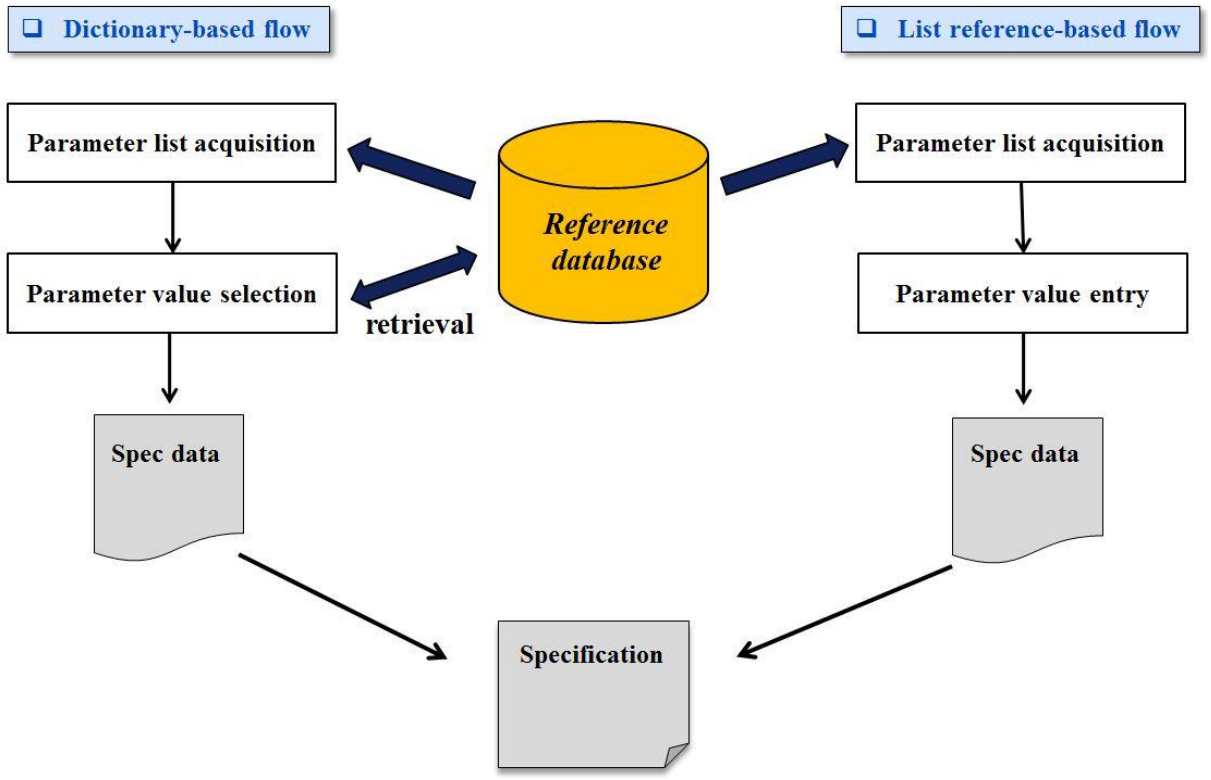

Figure 4.14 Specification creation using a reference database

4.4 Efficient Method for Data Extraction from the Specification

For library development, the library developers must extract the necessary data from the specification. However, there is unfortunately no effective data extraction method to meet their needs due to the absence of a complete specification. One of the most important needs is to precisely get the desired data from the specification. Another is to obtain a complete set of data for performing the library development task.

To address these needs, we propose the keyword for precisely identifying the data and task-based keywords for selectively extracting all necessary data for the target task. These proposed methods are introduced in the following subsections.

4.4.1 Keyword for Precise Data Identification

As described in Figure 4.10, the specification represents a hierarchical structure to cover a variety of data. To extract data from this specification, we must first find which branch contains the desired data in the tree structure and then identify it. The library developer usually needs a part or whole set of data from the corresponding data group. For example, for the characterization of standard cells, we need a SPICE simulator, design kit and characterization tool as a part of tools and a whole set of PVT corners for the standard cell library. In order to satisfy such needs, we propose to use a keyword that consists of tuple of elements given by:

$$Keyword = (Parameter, Category, Name\ or\ Key, Id) \quad (4.1)$$

Where:

Parameter: identifies the target parameter.

Category: identifies the category of parameter such as library category. It is required only for the categorized parameter set.

Name or Key: identifies one parameter. According to the kind of parameter, either name or key should be given.

Id: identifies the parameter value. When the parameter has more than one value, each value can have its specific id to differentiate one from another.

This keyword provides the key information to search data in the specification. Specifically, **Parameter** and **Category** elements permit to select a set of parameters. In addition, **Name** or **Key** element permits to identify one parameter from the selected parameter set. If selecting one parameter with its specific value, **Id** element must also be given.

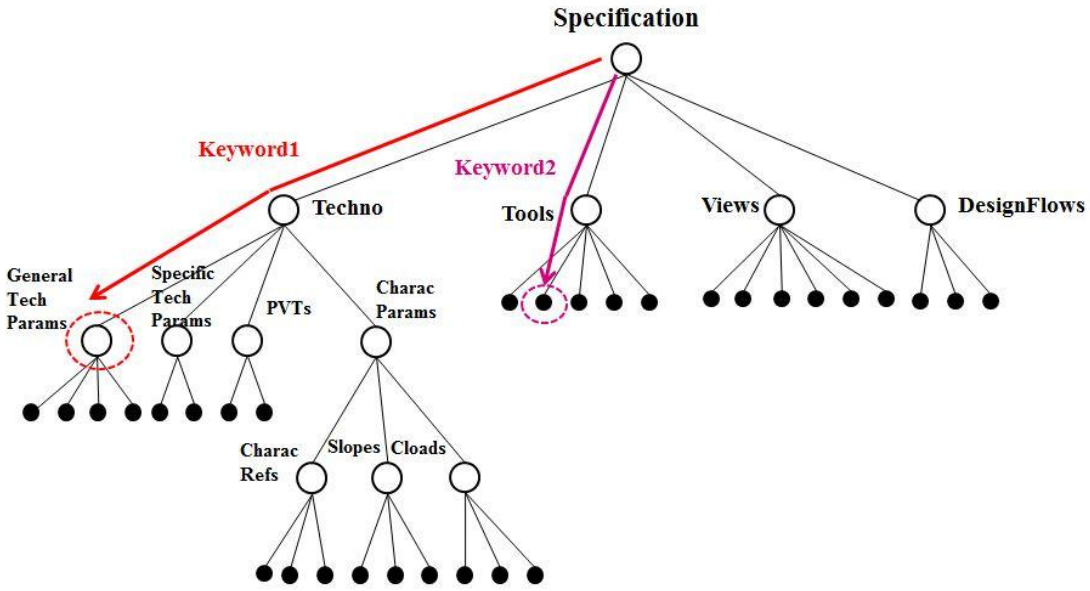

Figure 4.15 Data extraction with keyword

For example, two keywords are given as follows:

1) **Keyword1** = (Parameter = **GeneralTechParams**)
2) **Keyword2** = (Parameter = **Tools**, Key = **designcompiler_synthesis**)

The first keywords give only **GeneralTechParams** as **Parameter** element value. It means that the given element identifies the total set of general technology parameters as described in Figure 4.15. The second keyword gives two elements' values: **Tools** and **designcompiler_synthesis**. The first one represents the **Parameter** element and the second one the **Key**. These values indicate Design Compiler tool in the tool set as depicted in Figure 4.15. As a consequence, the keywords enable to know where the desired data are located in the specification to selectively access them.

4.4.2 Task-based Keywords for Efficient Data Extraction

The tasks for library development necessitate various data from the specification. The library developer must collect them before executing each task. However, this data preparation is a bottleneck in library development because he has difficulties in collecting only data relevant to the target task. To address these difficulties, we propose task-based keywords that enable to get the required data for executing a task without having knowledge about the relationship between the task and specification data. For this purpose, we must first

explicitly define this relationship. However, no way is available to efficiently represent it so far. The aforementioned keywords are therefore used for the relationship definition.

In order to identify all data related to a task, several keywords can be given by:

$$Keywords_{Task} = \bigcup_{i=1}^{m} Keyword_i = \bigcup_{i=1}^{m} (Parameter_i, Category_i, Name_i \text{ or } Key_i, Id_i) \quad (4.2)$$

Where: *m* is the number of necessary keywords

These keywords permit to link the task with specification data. For instance, to define the relationship between the timing model generation for standard cells and a collection of specification data including a set of PVT corners, design kit, etc., the keywords can be given as follows:

$$\begin{aligned} Keywords_{TIMING\_MODEL\_GENERATION} &= \bigcup_{i=1}^{m} Keyword_i \\ &= (Parameter = \boldsymbol{PVT}, Category = \boldsymbol{STDCELL}) \\ &\cup (Parameter = \boldsymbol{Tool}, Key = \boldsymbol{designkit}) \\ &\cup \quad \ldots\ldots \\ &\cup (Parameter = \boldsymbol{View}, Key = \boldsymbol{timinglib\_timinganalysis}) \\ &\cup (Parameter = \boldsymbol{View}, Key = \boldsymbol{timingdb\_timinganalysis}) \\ &\cup (Parameter = \boldsymbol{SoCDesignFlow}, Name = \boldsymbol{FrontEnd}) \\ &\cup (Parameter = \boldsymbol{SoCDesignFlow}, Nmae = \boldsymbol{SignOff}) \end{aligned}$$

The first keyword represents a set of PVT corners for standard cells. The second one represents a design kit. These data are needed to perform the timing model generation. Unlike these keywords, the four last keywords are also given to specify the views and design flows related to the task. Specifically, the timing model generation task relates to the creation of the Liberty library (.lib) and the compiled library (.db). In addition, these views are required for Front-End and Sign Off stages in the SoC design flow. Consequently, we can easily define the relationship between the tasks and specification data by using keywords.

4.5 Validation of the Specification

The reference-based method may help to reduce possible specification errors by providing the predefined information such as parameter list and available parameter values

during the creation of the specification. However, it is difficult to guarantee the correctness of the information determined by specification developers. For this reason, after its creation, a post-validation is highly required to ensure the reliability of the information. Thus, two methods are discussed in this subsection.

1. **Using the relationship of data objects**

   ✓ Consistency check of specification data

As mentioned above, the relation between the independent data objects were created by their common element so that the check can be carried out by verifing this element. For instance, we can verify if the slopes are well defined for all PVT corners by comparing the given PVT names in **PVTs** data object with those of **Characterization Parameters** data object for the consistency check. To illustrate another important example, **Design Flows** data object can be linked to **Tools** and **Views** data objects via **ToolKey** and **ViewKey** respectively as described in Figure 4.6. Performing a design step requires at least one view and one tool. Thus, more than one concerned **ToolKey** and **ViewKey** for each step must be defined. It permits to check the consistency of the tool and view information from the point of view of the design flow. In other words, we can verify if all tool and view information is well defined in the specification to support all required design flows.

2. **Using the rule**

   ✓ Correctness check of identifier naming

We presented the name element and key attribute to identify a parameter. To assign them in a unified way, the naming rule must be determined. For example, the PVT name is a concatenation of all essential elements of PVT corner like '**Process_Voltage_GND_Temperature_Age**'. According to this naming rule, a PVT name should be given for each PVT corner. After finishing the definition of all PVT corners, their name must be verified according to this rule.

   ✓ Correctness check of the parameter value

The parameter value verification is a delicate and complex task because it inevitably requires the deep analysis on the characteristics of parameter value. Firstly, we should check the data type of parameter value. For instance, the data type of slopes and capacitive loads should be checked as a positive float. Secondly, the given parameter value can be verified by using the rule to determine its value. For example, the tool version is differently assigned according to the strategy of the tool developer as shown in Figure 4.16. Since the tool version given by the specification developer may create compatibility problems due to wrong versioning, we need to verify it to provide the correct one. One of the solutions is to compare the tool version of the new specification with that of the previous one as a reference to detect an error. It can be performed by checking if it is greater than or equal to the previous one because the tool version tends to increase numerically or alphabetically when updating the specification.

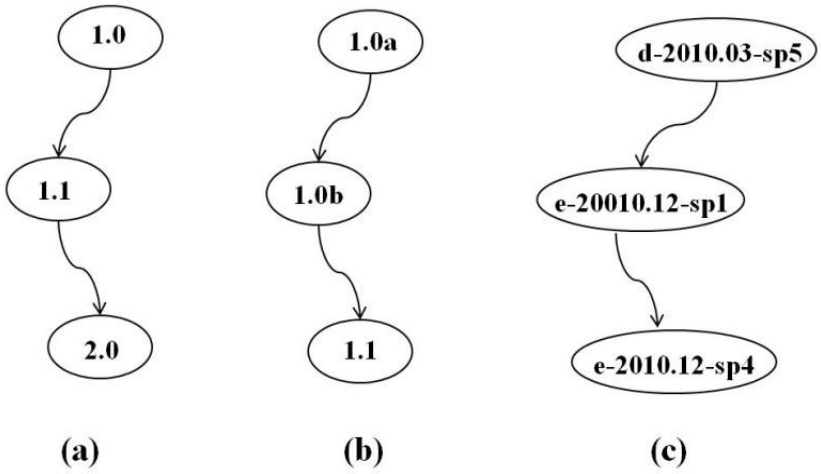

Figure 4.16 (a) simple tool versioning with numerical increment (b) simple tool versioning with numerical and alphabetical increments (c) complex tool versioning

4.6 Validation of the Library against the Specification

The generated library package must be verified before its delivery to customers in order to ensure its quality. The most efficient and effective way to validate the library is to check it against the specification given for its production.

Table 4.2: Library checklist

Library Check List	Specification data	Library view
Completeness	Technology Views	All library views
Compatibility	Tools	All library views
Correctness	View attributes	Liberty library view

As given in Table 4.2, the library checklist consists of three checks: completeness, compatibility and correctness.

- Completeness of the library

The library views should be created according to the lists of library views and PVT corners provided by the specification. Thus, the completeness of the library can be achieved by simply verifying if all required library views exist on the given view path.

- Compatibility between the library view and CAD tool

The library package must provide system designers with a complete set of library views to support their design flow. When using library files, they should not have compatibility problems with CAD tools. It means that their syntax must be correct to be readable by the tools. Such problems can be detected by reading each library file with CAD tool given in the specification.

- Correctness of the library view

The correctness check verifies the library view with respect to its syntax but not contents. One of the ways to verify the correctness of the library view is to check if its specific attributes provided by the specification are correctly defined. Now, there are only view attributes for Liberty library.

4.7 Conclusion

In this chapter, we have proposed an appropriate data model to represent the specification containing all target information from technology-dependent parameter to design flow information. To efficiently cope with these specification data, we also introduced

two approaches from two different perspectives: specification developers and library developers. One is the reference-based method to create a complete and consistent specification having a huge amount of data. Another is the task-based keywords to quickly and precisely extract the desired data from the proposed specification according to the target task. Finally, the validations for the specification and the library have also been discussed. In the next chapter, we present an XML-based specification language to represent our target specification as well as a specification platform to efficiently deal with it by the proposed methods.

5. Specification Platform

Contents

5.1 Introduction

The previous chapter was dedicated to analyze the specification data for library development and then to propose a suitable data model for representing it. Additionally, two methods were introduced to create a complete and consistent specification and to precisely extract data from it according to the target task. In addition to these methods, the verifications of the specification and library were also presented.

In this chapter, we first present an XML-based specification language to encapsulate all required information for library development by using the proposed data model. Then, we introduce a specification platform that consists of several tools for creating specifications, extracting data and verifying the library.

5.2 LDSpecX: Library Development Specification based on XML

In the previous chapter, we presented a hierarchical data model to encapsulate a variety of data. By using it, we would like to propose an XML-based specification. One of the most interesting features of XML is machine-readability. It enables to transmit data to the library development system. In other words, the automation of data processing is easily achievable. Furthermore, the specification language facilitates building a database and dealing with it thanks to the unified format. From the conceptual data model in the previous chapter, Library Development Specification based on XML (LDSpecX) was proposed [108]. This specification language allows defining all necessary information in a unique XML file. Figure 5.1 shows its schema which corresponds to that in Figure 4.10. This XML schema represents the structure of our XML-based language and details the syntax and semantic rules. Specifically, LDSpecX provides seven top level elements that correspond to the data objects in Figure 4.6 respectively: **GeneralTechParams**, **SpecificTechParams**, **PVTs**, **CharacParams**, **Tools**, **Views** and **DesignFlows**. In order to create a complete specification, the specification developers must define all indispensable parameters of every top level element using LDSpecX. Additionally, the identifier attributes of the specification namely **LDSpecXName** and **LDSpecXversion** should also be given.

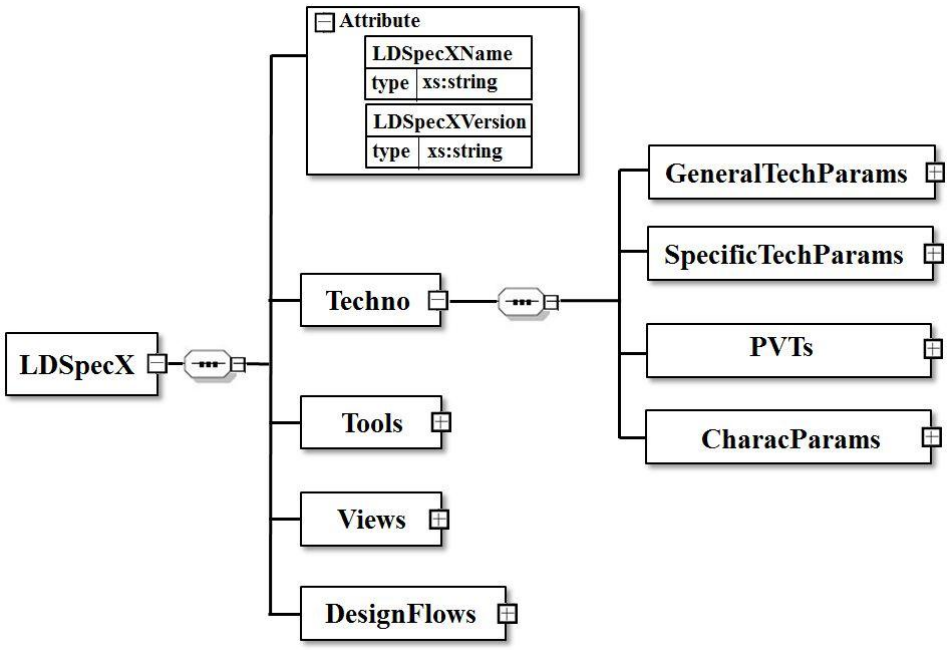

Figure 5.1 Schema of LDSpecX

To illustrate an example of the top level elements, the schema of **GeneralTechParams** element is described in Figure 5.2 This schema details the organization of the sub-elements and attributes. In addition, it can define the data type and the number of element.

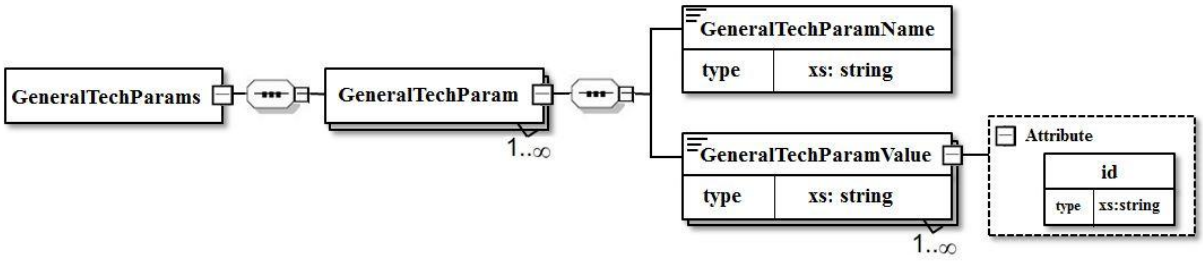

Figure 5.2 Schema of GeneralTechParams element

Figure 5.3 shows an example of defining a technology node named **TechnoNode** in the **GeneralTechParams** element. It should be in accordance with its syntax given in Figure 5.2.

```
<GeneralTechParams>
    <GeneralTechParam>
        <GeneralTechParamName>TechnoNode</GeneralTechParamName>
        <GeneralTechParamValue>28</GeneralTechParamValue>
    </GeneralTechParam>
        :
</GeneralTechParams>
```

Figure 5.3 Example of GeneralTechParams element

In the same way, the whole schema of LDSpecX was defined in detail. It becomes the foundation of the proposed XML-based specification language. Furthermore, the well-defined XML schema can be used to validate the resultant XML document with Document Object Model (DOM) parser or Simple API for XML (SAX) parser with respect to the syntax. As a result, it helps to verify the completeness and correctness of the specification in LDSpecX.

5.3 Specification Creation Tool

Now, the specifications for advanced technology such as 28nm Fully Depleted Silicon-on-Insulator (FDSOI) CMOS contain several thousand parameters. Especially, the development of manufacturing and design methodologies may lead to increase the number of

such parameters. We have to define all these parameters using our XML-based specification language. However, although there are some useful editors to create XML documents like Oxygen XML editor [114], they are not suitable to enter a large amount of data because all data should be entered manually. Thus, we highly need an appropriate tool support to efficiently create an LDSpecX-based specification.

This section describes a specification creation tool based on the reference-based approach to create a complete and consistent specification [115]. It splits into two parts: XML-based reference database construction and user-friendly graphic user interface (GUI) development. This GUI enables specification developers to create specifications in LDSpecX with the help of the reference database.

5.3.1 XML-based Reference Database

In the previous chapter, we presented two types of references to provide as much predefined information as possible for specification creation: dictionary and list reference. First of all, the reference database containing these references should be constructed. In this thesis, in order to deal with data in a unified way, this database is also created by using XML.

Firstly, the schema of dictionary was made from its data model described in Figure 4.11. The resultant schema is represented in Figure 5.4.

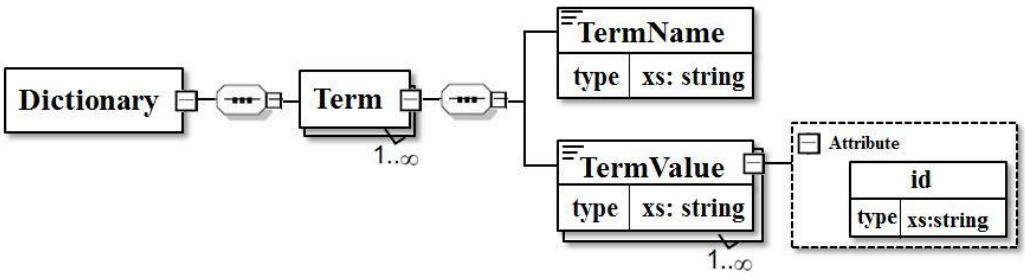

Figure 5.4 Schema of dictionary

To illustrate how to define terms in the dictionary, Figure 5.5 gives the definition of two terms. The first one represents a list of general technology parameters. The second one gives all available values of the technology node as a general technology parameter. As explained previously, we can obtain the information about the parameter list as well as parameter values from this reference.

```
<Dictionary>
    <Term>
        <TermName>GeneralTechParam</TermName>
        <TermValue>TechnoNode</Termalue>
        <TermValue>Process</TermValue>
        <TermValue>VtOption</TermValue>
        <TermValue>MetalOption</TermValue>
        <TermValue>RCType</TermValue>
        <TermValue>MaxTemp</TermValue>
        <TermValue>MinTemp</TermValue>
        <TermValue>DRM</TemrValue>
    </Term>
    <Term>
        <TermName>TechnoNode</TermName>
        <TermValue>28</TermValue>
        <TermValue>32</TermValue>
        <TermValue>45</TermValue>
        <TermValue>65</TermValue>
        <TermValue>90</TermValue>
        <TermValue>130</TermValue>
    </Term>
      :
</Dictionary>
```

Figure 5.5 Example of a dictionary

Secondly, the schemas of two list references for the tool and view have been made from their own data model. Figure 5.6 describes the schema of tool list reference. As depicted in this figure, the tool list reference can have more than one tool element having a tool key attribute and a tool name.

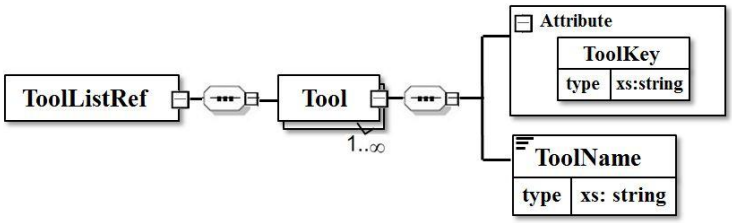

Figure 5.6 Schema of tool list reference

Figure 5.7 gives an example of the tool list reference. This example includes two tool elements for Design Compiler and an in-house tool named alto for the cell characterization.

```
<ToolListRef>
    <Tool ToolKey="designcompiler_synthesis">
        <ToolName>sythensis</ToolName>
    </Tool>
    <Tool ToolKey="alto_charac">
        <ToolName>alto</ToolName>
    </Tool>
      :
      :
</ToolListRef>
```

Figure 5.7 Example of a tool list reference

Finally, all necessary references were created by using XML according to their proposed syntax. As a consequence, the XML-based reference database could be made up of these references.

5.3.2 User-friendly GUI for Specification Creation

As mentioned above, most of XML documents are created entirely manually with text editors. However, in our case, it is very difficult to use such existing tools for creating our XML-based specification due to its high complexity. Thus, we intend to develop an appropriate tool to create an LDSpecX-based specification. For this purpose, a systematic and unified flow has first been proposed by using the reference database and then a user-friendly GUI was developed.

a) Specification creation flow

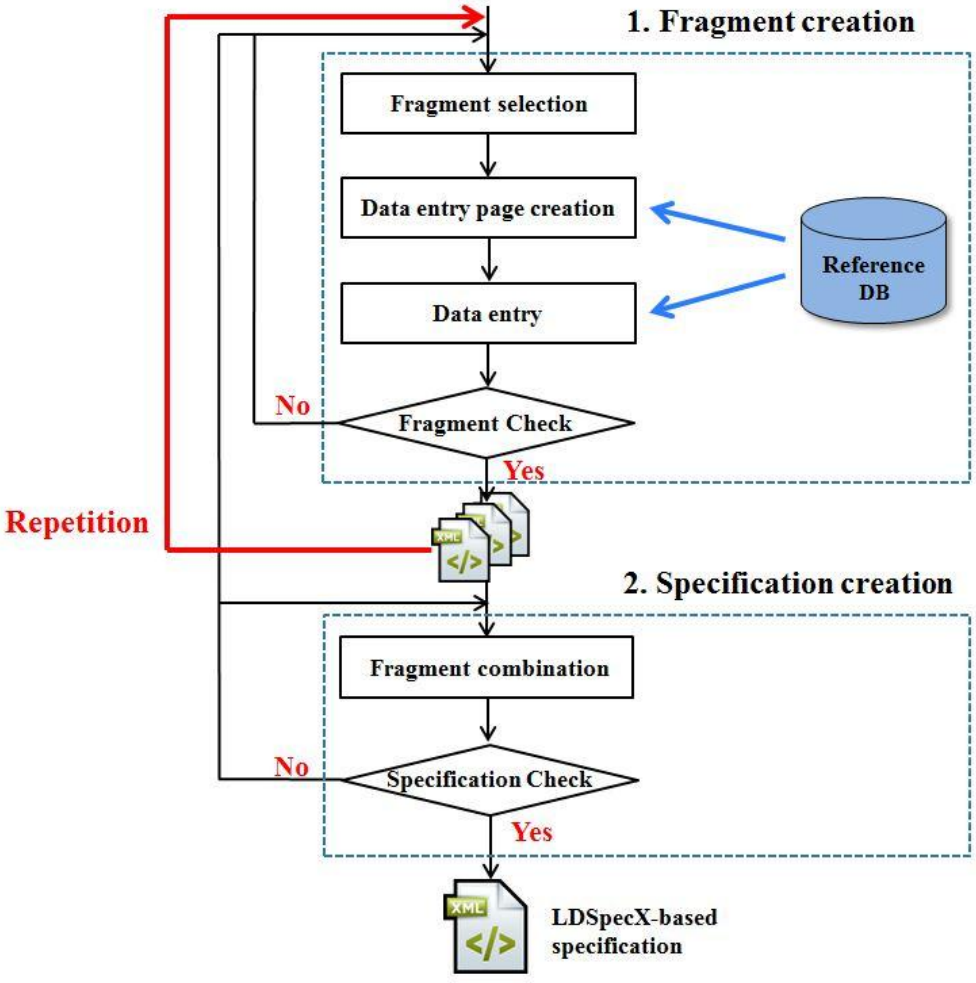

Figure 5.8 Reference-based specification creation flow

To begin with, the reference-based specification creation flow is described in Figure 5.8. This flow is divided into two main steps: fragment creation and specification creation.

Definition	**Fragment** A fragment represents a section of the specification. This fragment corresponds to the top-level element containing parameters of the same category.

❖ **Fragment creation**

The fragment creation step can be accomplished by four sub-steps: fragment selection, data entry page creation, data entry and fragment check.

- **Fragment selection**: The specification must split into several fragments because the specification data should be gathered from several persons according to the data category or different schedules. In other words, the specification creation is a collaborative work. Thus, the specification developers should select one or more concerning fragments to define the desired parameters.
- **Data entry page creation**: If the fragment is selected, the specification tool acquires its parameter list from the reference database and then the data entry page is dynamically created by the obtained parameter list.
- **Data entry**: The specification developer enters data on the data entry page with the help of the reference database.
- **Fragment check**: The entered data are stored in a split XML file for each fragment. The fragment file must be checked for the syntax and completeness. The syntax check validates the created XML file against the schema of the corresponding fragment by using DOM parser. The completeness check verifies if all parameters have at least one value.

❖ **Specification creation**

To start the specification creation step, the fragment creation step should be repeated until all required fragments are created. Then, this step can be carried out by fragment combination and specification check.

- **Fragment combination**: In this step, all created fragment files are merged into an LDSpecX-based specification file by using an XSLT template and XSLT processor [110].
- **Specification check**: After the fragment combination, we need to verify the created specification. The specification check covers not only the syntax and completeness but also the consistency and correctness. The syntax check can be performed by validating the specification against the schema of LDSpecX with DOM parser. The completeness check verifies if all fragments are given in the specification. On top of that, the consistency and correctness check are also added to verify the information in the specification as proposed in the previous chapter. Finally, the LDSpecX-based specification can be validated thanks to these checks. It significantly aids to guarantee the specification reliability by comparison with its verification dependent only on the specification developers.

b) User-friendly GUI

In order to support the aforementioned flow, a user-friendly GUI was developed in Python [109]. The GUI facilitates the creation of the LDSpecX-based specification.

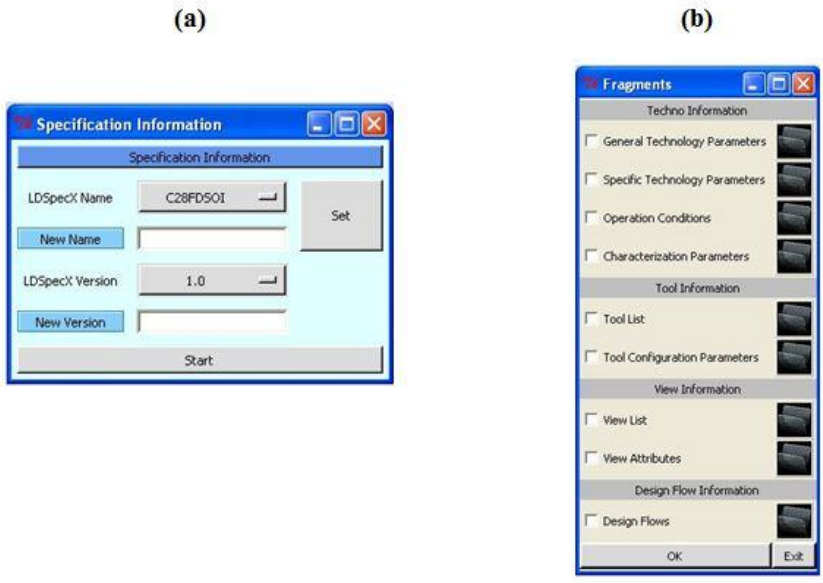

Figure 5.9 (a) Specification information entry window (b) Fragment selection window

Firstly, it can start with its name and version definition on the specification information window in Figure 5.9 (a). Then, the proposed GUI provides a fragment selection window as

shown in Figure 5.9 (b). Each specification developer is able to choose the desired section of the specification by simply selecting one or more target fragments. According to the selected fragments, the corresponding data entry pages having its parameter list obtained from the reference database will be created. The data entry pages provide facilities to access the references and to enter data. As we introduced the aforementioned specification creation flows based on the dictionary and list references in Figure 4.14, there are two types of fragments according to these flows. The data page of each type of fragment can be built by using its related reference.

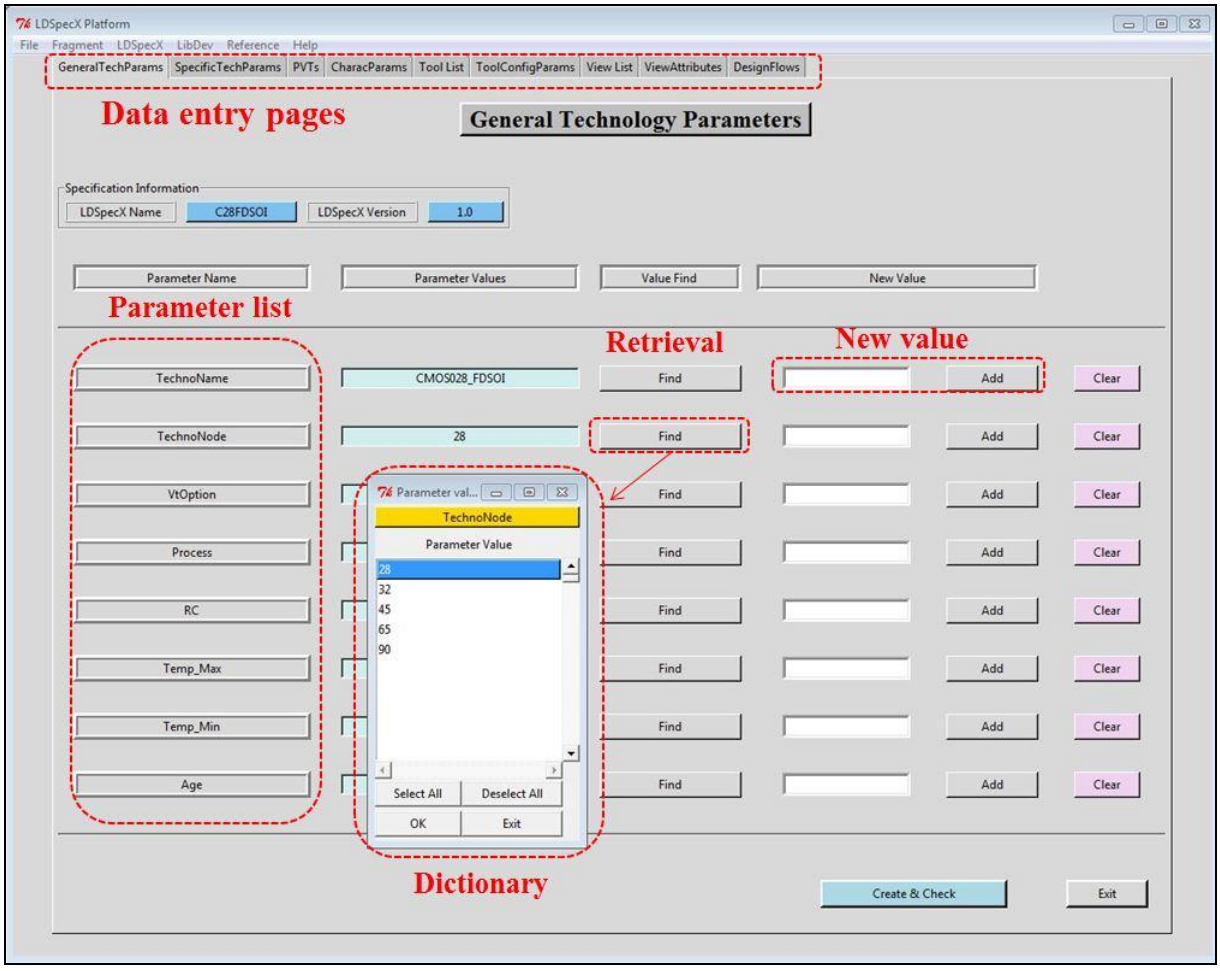

Figure 5.10 Data entry page of GeneralTechParams fragment

To illustrate an example of the first type of fragment, the data entry page of General Technology Parameters fragment is shown in Figure 5.10. The list of general technology parameters can be obatined from the dictionary and then given on the data page. To define each parameter, there are two ways. First, the ***find*** button allows accessing the dictionary to get all possible parameter values from it by retrieval. The specification developer can select

the desired values among them. For instance, all possible technology nodes can be displyed on the dictionary window by clicking the ***find*** button as described in this figure. Then, we can select one of them to give a technology node. Second, if the dictionary does not contain the desired parameter value, a new value must directly be entered in the entry. If all paramters are defined with these methods, the ***create&check*** button permits to create the fragment file and to check it with respect to the syntax and completeness.

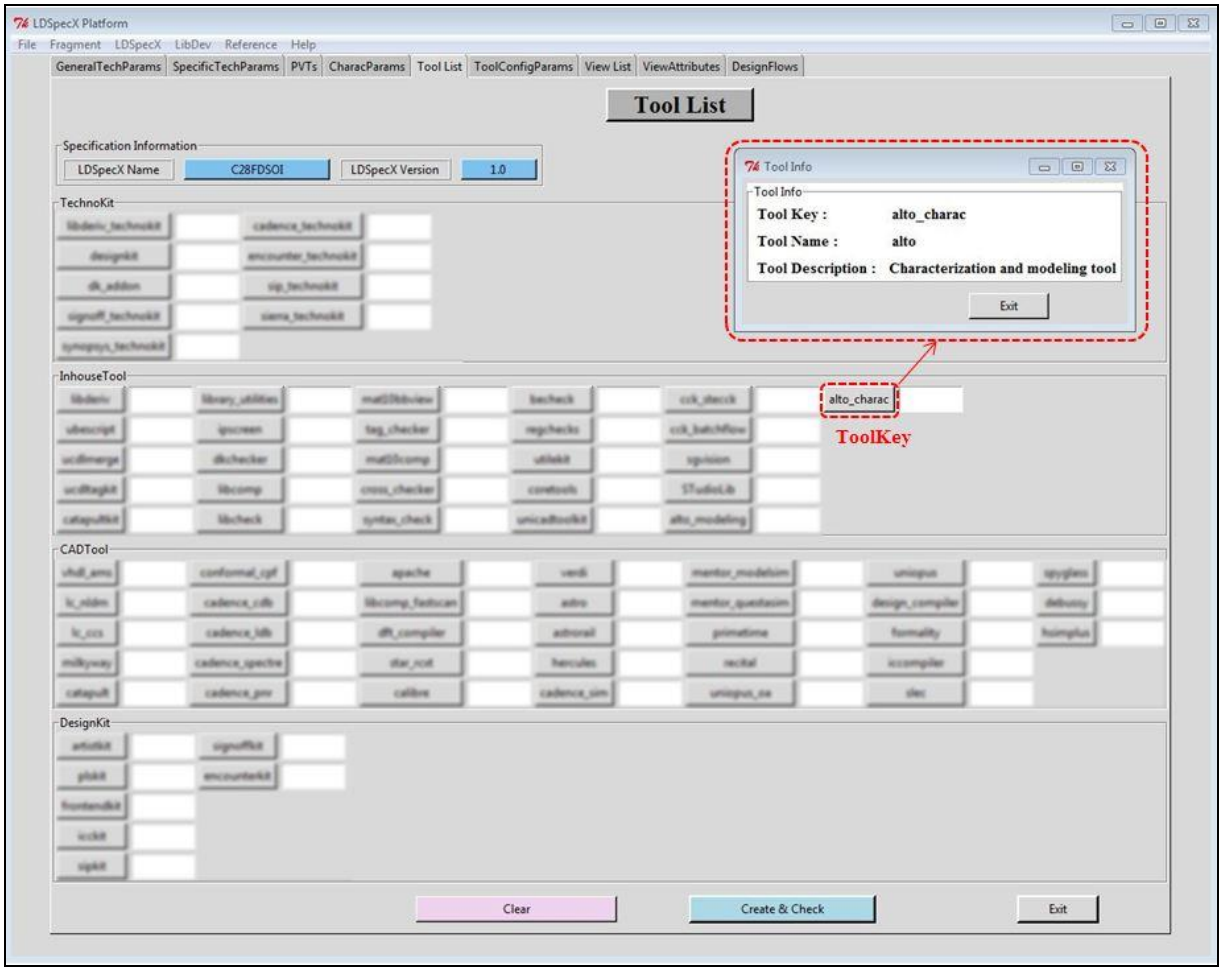

Figure 5.11 Data entry page of Tool List fragment

Figure 5.11 shows the data entry page of tool list fragment as the second type. This fragment requires a complete list of tools. Thus, this list must be derived from the tool list reference in order to construct its data entry page. Unfortuantely, due to its limited space, only tool keys are displayed. However, clicking each tool key button allows showing its identification information such as tool name. To complete this fragment, the appropriate tool version for all tool keys must be given. Likewise, the ***create&check*** button creates the XML file and verifies the resultant fragment file.

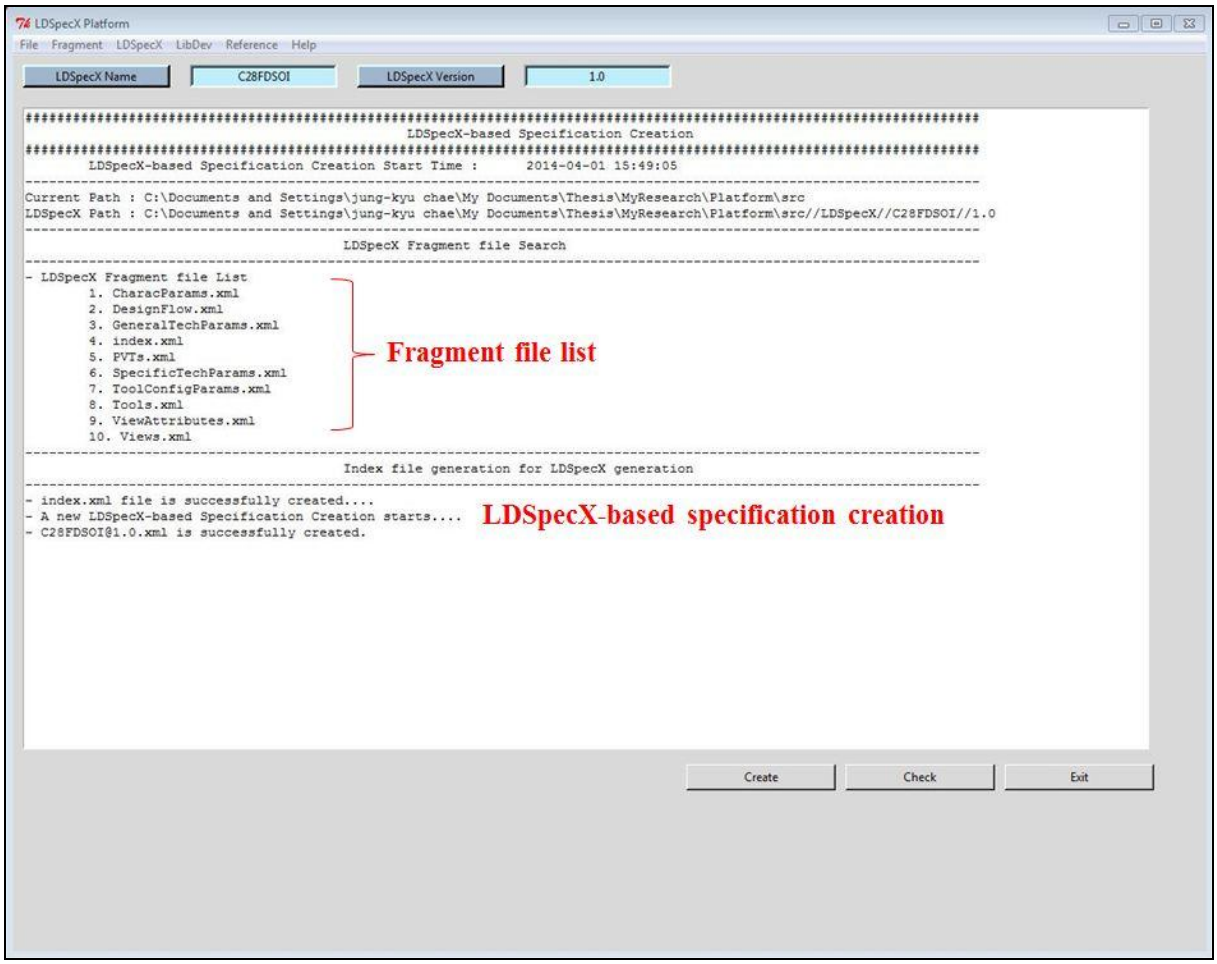

Figure 5.12 Specification creation page

Figure 5.12 descirbes the specification creation page. The ***create*** button at the bottom allows finding all created fragment files and creating an LDSpecX-based specification by merging them. After the specification creation, the ***check*** button permits to check the specification file for its syntax, completeness, consistency and correctness.

As described above, the parameters to define in the specification rely greatly on the references because the data entry page provided by the GUI is flexibly established depending on which parameters are derived from the reference database. In other words, the proposed GUI can be customized by the specification developers by simply updating the references. For this purpose, it also offers the end-users the ability to access the reference and to update it. For instance, Figure 5.13 shows the dictionary edit page. As shown in this figure, all terms of the dictionary are listed in the left box. After selecting the desired term name, all possible term values will be displayed in the right box by clicking the *Get Value* button. The entry

boxes at the bottom of the page are provided to add a new term or a new term value to the dictionary. It enables to enrich the dictionary.

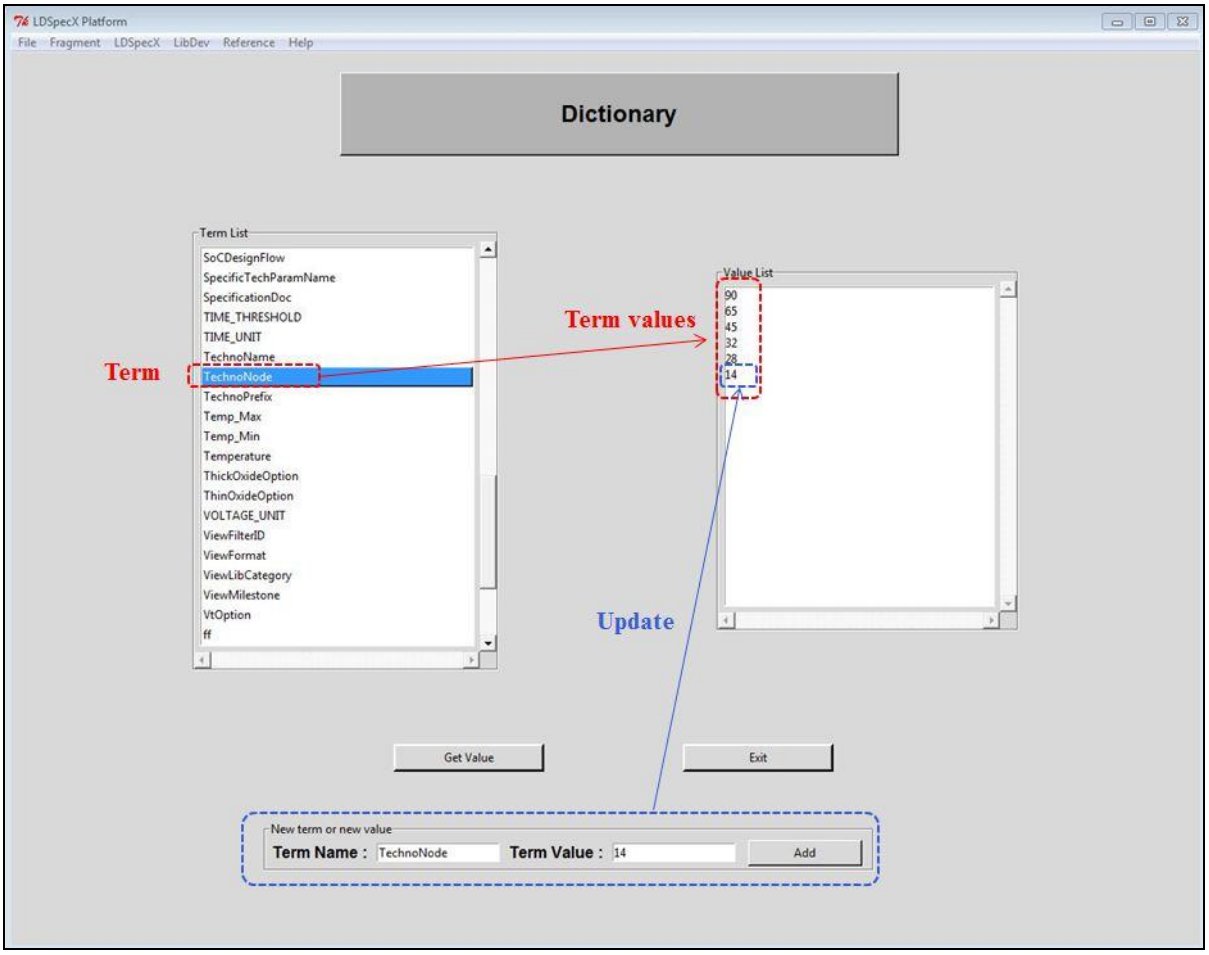

Figure 5.13 Dictionary edit page

5.4 API for Specification Data Extraction

This section introduces a simple API for data extraction from the LDSpecX-based specification. It depends on the keywords to identify specification data as proposed in the previous chapter.

5.4.1 Library Development Task Definition

The specification creation was presented previously by using GUI. It corresponds to the specification phase in the library development process. Since the specification is now available, the library developers are able to perform the subsequent phases from design to validation. Each phase can be divided into several steps that represent a sequence of tasks. For

example, the design phase contains Front-End (FE) and Back-End (BE) view generation steps. The FE view generation step contains a timing model generation task. This task can be performed by timing characterization and timing modeling. However, the granularity of the task may be different according to the library category because the methods to perform the task depend on the features of the library cell. For instance, the cell characterization method is not the same for standard cells as that for memories. As a result, the required specification data depends on the library category. Thus, we need to define the library development tasks for each library category separately. Figure 5.14 (a) visualizes the schema of the library development flow. As shown in this schema, we can differentiate a set of steps by the library category. Furthermore, as mentioned earlier, a wide range of specification data is required for executing the task and thus, the task-based keywords were proposed in the previous chapter. Figure 5.14 (b) shows how to define the relationship between task and data by using these keywords. This element may have more than one selector element, which permits to encapsulate task-based keywords.

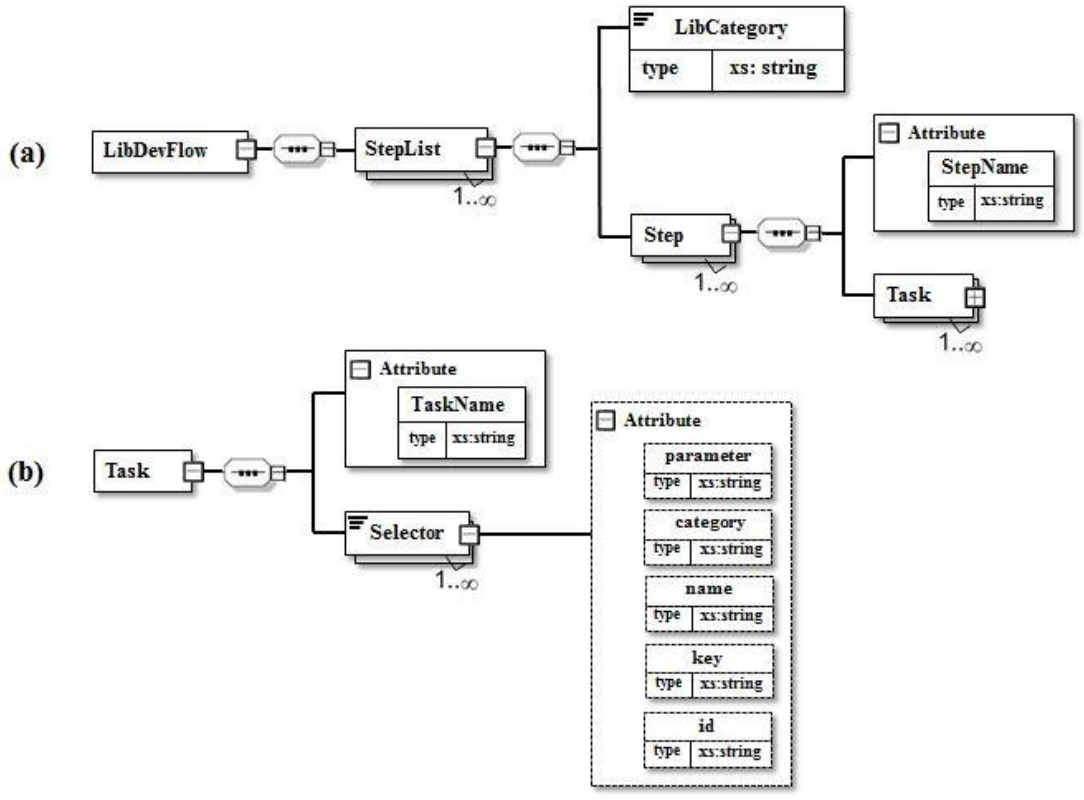

Figure 5.14 (a) Library development flow (b) Task

To give an example of the task definition, Figure 5.15 represents how to define a timing model generation task for standard cell library by using selectors to identify the specification data. In this example, the first selector gives keywords to identify a set of PVT

corners for standard cell library. The second and third selectors give keywords to identify a characterization tool and tool configuration parameters for timing characterization respectively. In the same way, all library development tasks over library categories can be predefined as a function of keywords and stored in an XML file.

```
<LibDevFlow>
  <StepList>
    <LibCategory>STDCELL</LibCategory>
      <Step StepName="FE_VIEW_GENERATION">
        <Task TaskName="TIMING_VIEW_GENERATION">
          <Selector parameter="PVT" category="STDCELL"/>
          <Selector parameter="Tool" key="alto_charac"/>
          <Selector parameter="ToolConfigParam" category="TIMING_CHARAC" key="alto_charac"/>
                :
        </Task>
          :
      <Step>
          :
    </StepList>
      :
</LibDevFlow>
```

Figure 5.15 Example of a library development flow for standard cell library

5.4.2 Specification Data API

The specification data API was developed in Perl [111]. This API allows accessing the LDSpecX-based specification database to obtain data from it as described in Figure 5.16. In order to apply the proposed method, the aforementioned library development flow was written in an XML file.

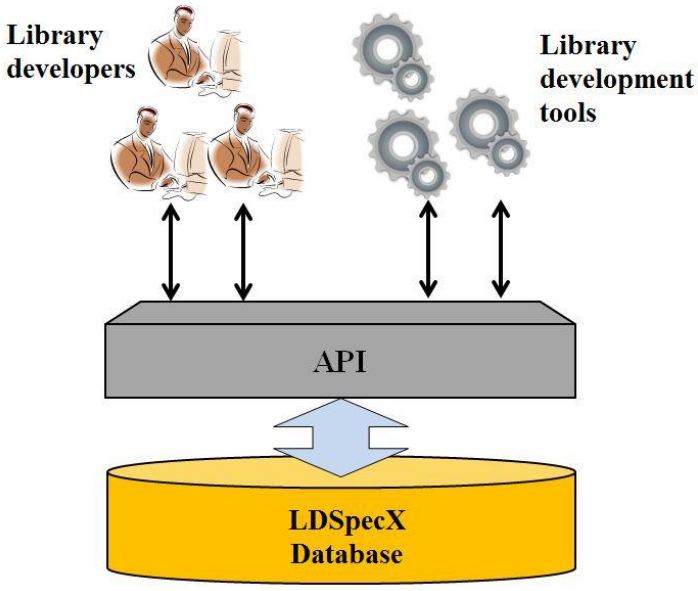

Figure 5.16 Overview of the API

The API provides two fundamental functions as follows:

1) **getSpecData** (specification, keyword): returns data (list of strings) derived from the selected LDSpecX-based specification (XML file) by using the given keyword (string).
2) **getFlowInfo** (library category, task): returns keywords (list of string) from the library development flow according to the given library category (string) and task (string).

Additionally, by using these functions, a function to extract data depending on the desired task is also provided as below:

3) **getSpecDataViaFlow** (specification, library category, task): returns all necessary data (list of string) from the LDSpecX-based specification (XML file) for the target task (string).

Figure 5.17 describes the data extraction flow by using the task-based keywords. First, the keywords are gathered from the flow file. Second, by using these keywords, the data are derived from the specification database.

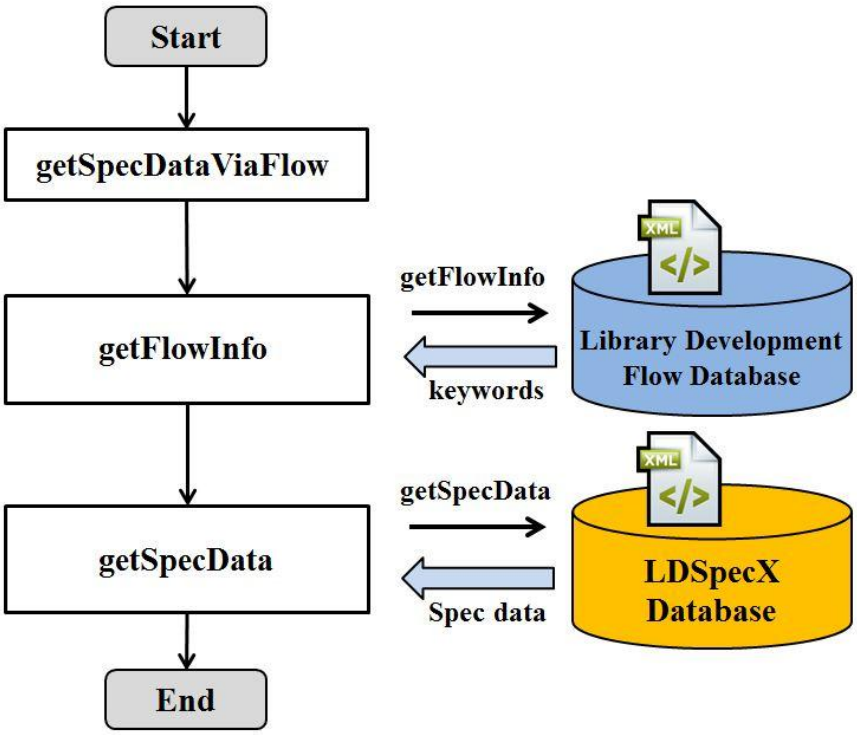

Figure 5.17 Data extraction flow by using task-based keywords

To give an example of using the task-based keywords, the library developer needs the tool information for executing a task. At STMicroelectronics, tools' name and version should be written in a specific file named ***.ucdprod*** that is employed to load them. However, when such information is collected manually without knowing the required tools for a target task, redundant tools can be given in this file. As a result, it may cause a compatibility problem when loading them at the same time. On the contrary, by using the proposed method, we can get only minimal necessary tools. Thus, it not only gets tool information in an efficient way but also addresses the compatibility problems.

In summary, the API offers efficient functions for data extraction according to the needs of library developers. Fundamentally, it permits any user to selectively extract the desired data by giving a keyword. In addition, the task-based keywords permit to precisely identify only the data needed for executing the chosen task and quickly get them. Therefore, it is very useful for library developer to accomplish his tasks without having a good understanding of the relationship between the specification data and library development task.

Furthermore, the data extracted from the given specification for task execution often need to be rewritten in accordance with the specific syntax of the tool. In order to automate this process, we also developed specific generators in Perl integrating the functions provided by the API into them. These generators facilitate to produce the machine-readable files where specification data are given. Thus, they enable library developer to accomplish his target task more smoothly. For example, in order to automatically generate the aforementioned file (***.ucdprod***), a generator named ***genucdprod*** was developed. It allows extracting the tool information by using the API according to the selected target and rewriting it by following its syntax.

5.4.3 Library Verification Tool using the API

As stated in the previous chapter, three basic checks against the specification were remarked: completeness, compatibility and correctness.

The compatibility check can be performed using the existing library verification tool and the tool information obtained by the API. On the contrary, for completeness and correctness checks, we have to develop a verification tool because the existing one is not able to cover them. Thus, a supplementary verification tool was developed in Perl. It aims at

verifying the resultant library against the specification with respect to the completeness and correctness. The verification can be achieved by deriving the specification data related to each check as given in Table 4.2 via the API. The completeness check needs a library view list and PVT corners for the related library category. The correctness check requires attributes of the generated library view. Finally, this tool may improve the quality of the library by enforcing the library check against the specification. Additionally, this tool also shows how to apply the capabilities provided by the API.

5.5 Conclusion

In this chapter, we have presented an XML-based specification language named LDSpecX to represent the specification for library development. Furthermore, an LDSpecX-based specification platform was introduced by using the methods proposed in the previous chapter. This prototype consists of two main components: a specification creation tool with user-friendly GUI and a specification data API. In the next chapter, we demonstrate the development of a standard cell library from an LDSpecX-based specification by using the developed specification platform.

6. Experiments

Contents

6.1 Introduction

We have presented in the previous chapter a XML-based specification language named LDSpecX. This specification language enables specification developers to define all necessary information for library development. Furthermore, a specification platform was developed to collaboratively create a complete and consistent specification as well as to efficiently extract specification data from it.

This chapter introduces the development of a standard cell library by using the proposed specification platform. Experimentally, a specification with 28nm FDSOI CMOS technology is first created in LDSpecX through the GUI. Then, we show the development of a standard cell library from this specification by using the API. Thus, a standard cell library is developed according to the library development process from the specification creation to the library validation presented in the chapter 2.

6.2 Library Development from the Specification

6.2.1 Specification Creation

The specification must provide every library developer with complete information to develop his target libraries or IPs. The specification data may increase by the supported

library categories. For example, we need a set of PVT corners for each supported library category such as standard cell library and I/O library. In these experiments, we focus only on the standard cell library to reduce the quantity of data. The specification for developing a standard cell library with 28nm FDSOI CMOS technology was created by using LDSpecX with the help of the GUI of the specification platform according to its process presented earlier.

The specification creation phase can be achieved by the fragment creation and fragment combination. Firstly, we create all fragments of the specification. To create each fragment, the information about its parameters is gathered from the reference database as well as information owners. After consecutively creating all necessary fragments, they are merged into a specification in LDSpecX. Then, the created specification is verified.

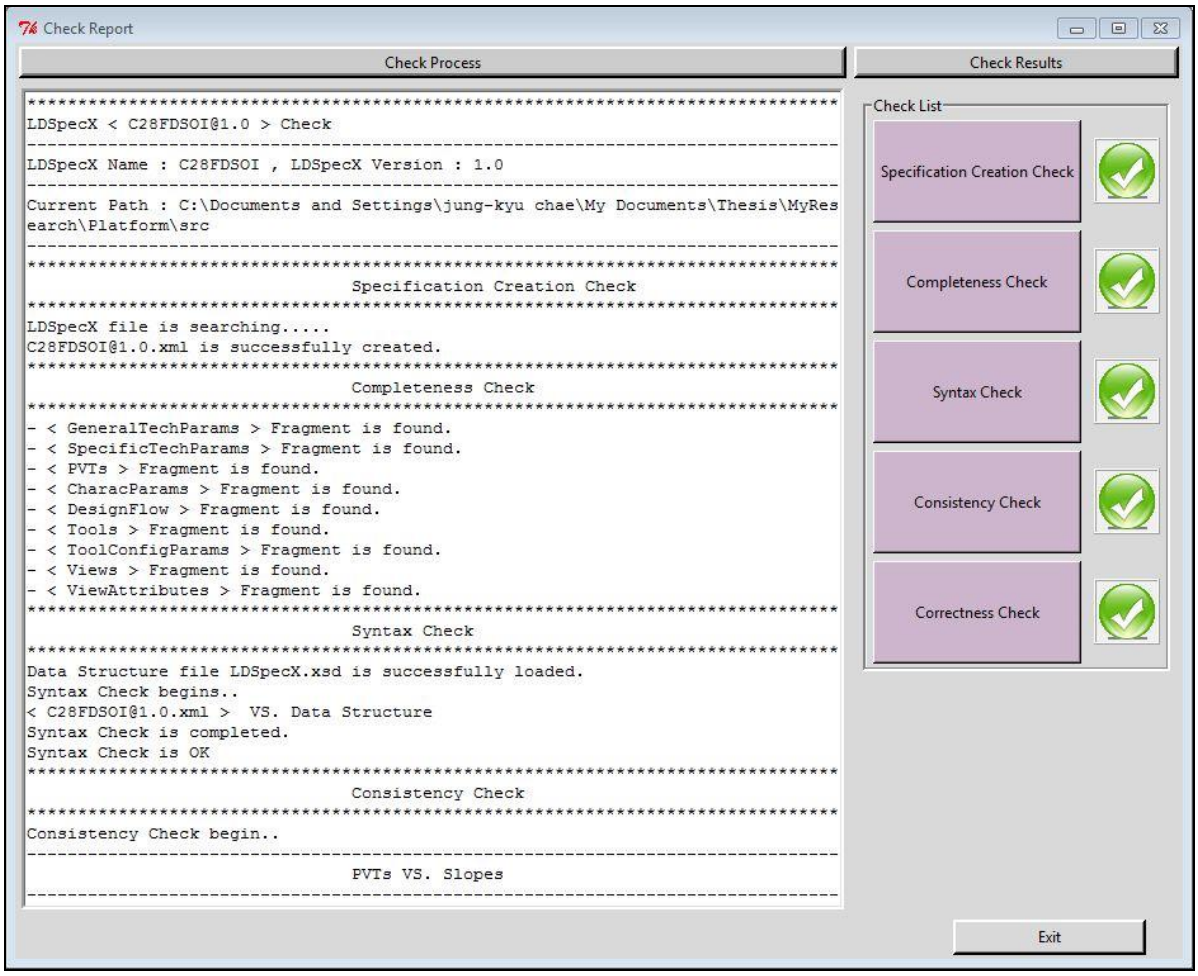

Figure 6.1 Screenshot of specification check report

In conclusion, the target specification was created and checked via the GUI. Figure 6.1 shows the obtained check report for five checks: specification creation, completeness, syntax, consistency and correctness. The first check has verified the existence of the specification file. The completeness check has identified all included fragments. The syntax check has validated the LDSpecX file against its schema by using DOM parser. The consistency check has verified the consistency between **PVTs** fragment and **CharacParams** fragment via PVT name and that between **Tools**, **Views** and **DesignFlows** fragments via key attributes. Finally, the correctness check has verified the PVT name and tool version by using their rule.

The resultant specification is compared with the traditional specifications that were indeed written in natural language and tabular form by design platform developers. The comparison results are given in Table 6.1. These results show the effectiveness of the proposed GUI.

First, the total time consumed by the proposed method for the specification creation is around 42 minutes. It is about 3 times faster than by traditional manual method even if much more specification data were dealt with. This is the reason why the traditional methods to define specifications depends entirely on human intervention. On the contrary, our user-friendly GUI allows a high reduction of human interventions by providing many facilities to enter data and to get the information from the reference database.

Second, these results also show that LDSpecX permits to provide richer information than the traditional methods having missing parameters that are individually stored outside the specification or are in the knowledge of library developers. These missing parameters such as tool configuration parameters may provide abilities to allow the automatic tool configuration without the help of tool developers.

Finally, even if it is difficult to quantitatively estimate the human-errors, the reduction of potential errors may also be expected thanks to a unified specification instead of several scattered ones and a centralized reference database. In addition, the specification verification tool support also aids to reduce error like inconsistency.

Table 6.1: Comparison of LDSpecX-based specification and traditional specifications

Information Category	Parameter		Number of parameters	
			Traditional specifications	LDSpecX
Technology	General Technology Parameter		6	9
	Specific Technology Parameter		6	6
	PVT		20	20
	Characterization Parameters	Characterization reference	-	13
		Slope	240	240
		Capacitive load	1136	1136
Tools	Tool		73	76
	Tool Configuration Parameter		-	449
Views	View		154	154
	View Attribute		10	10
Design Flows	SoC Design Flow		3	3
Total number of parameters			**1648**	**2116**
Number of files			**2 excel files (5 spreadsheets), 3 word files**	**1 xml file**
Creation time			**2 hours**	**42 minutes**

6.2.2 Library Development from the Specification

With the created specification, we can perform the next steps for library development. We develop a standard cell library by using the developed API. In addition, as mentioned at the beginning of the thesis, we focus only on the cell-independent information so that the number of cells is not meaningful. For this reason, even though a standard cell library generally contains several hundreds of cells, we used a small set of cells that consist of 11 combinational and sequential ones for our experiment. Likewise, only three of the twenty PVT corners in the given specification are used to minimize the repetition of tasks. Furthermore, we aim at generating all required library views to support the entire design flow for Cadence and Synopsys. In these experiments, the library development starts with FE view generation by using BE database of the library including layout, Spice netlist and so on. Table 6.2 summarizes the target tasks with the specification data as input and the library views as

output for developing our test library. In consequence, it can be considered as a scenario of these experiments.

Table 6.2: Summary of input and output for library development tasks

Library development flow	Task	Input		Output (library views)
		Data group	Specification data	
Design phase	NLDM timing/power model generation	Technology	• General technology parameters • Specific technology parameters • PVT corner list • Characterization references • Slopes • Capacitive loads	.lib files
		Tools	• Characterization tool • Spice simulator • Design kit • Tool configuration parameters for timing/power characterization	
		Views	• Liberty view attributes	
	CCS timing model generation	Technology	• General technology parameters • Specific technology parameters • PVT corner list • Characterization References • Slopes • Capacitive loads	.lib.gz files
		Tools	• Characterization tool • Spice simulator • Design kit • Tool configuration parameters for CCS timing/power characterization	
		Views	• Liberty view attributes	

	Verilog model generation	Tools	• Behavioral modeling tool • Tool configuration parameters for Verilog modeling	.v file
Derivation phase	Library compilation	Technology	• PVT corner list	.db files
		Tools	• View derivation tool • Library compiler • Design kit • Tool configuration parameters for library compilation	
	LEF generation	Tools	• View derivation tool • Calibre • Design kit • Tool configuration parameter for lef generation	.lef file
	Cadence CAD view generation	Tools	• View derivation tool • Calibre • EDI • ETS • Virtuoso • Conformal • Incisive • Design kit • Tool configuration parameter for Cadence view generation	OA layout, symbol, abstract, schematic
	Synopsys CAD view generation	Tools	• View derivation tool • Milkyway • IC compiler • Design compiler • Design kit	Cell view, FRAM view files
Library verification phase	Completeness check	Technology	• PVT corner list	reports[1]
		Tools	• Library verification tool	
		Views	• View list	
	Correctness check	Tools	• Library verification tool	
		Views	• Liberty view attributes	

[1] These reports give the check results of the library but are not included in the library package.

According to the flow described in Table 6.2, all tasks were executed to produce the rest of the contents of the library. All necessary specification data for each task are given in this table. By using the aforementioned generators, the necessary input data were derived from the given specification and the tool input files could be produced with obtained data. For example, NLDM timing/power model generation requires technology parameters, tool parameters and view attributes as shown in Table 6.2. To execute this task, two tool input files are required. The first file (***.ucdprod***) gives a characterization tool, Spice simulator and a design kit with their proper version to load them correctly. The second file includes all technology parameters, tool configuration parameters and view attributes so as to provide process parameters for cell characterization and to set up the configuration of the characterization tool. These input files could be generated by using their specific generator.

During design and derivation phases, 86 library views were obtained to produce a complete library package. This library package must be verified. Thus, the library verification tool based on the functions of the API is used to achieve this objective. This verification phase consists of the following:

- Firstly, the library package is checked if it contains all mandatory library views. Using PVT corner list and view list derived from the specification, the complete view list could be reestablished. The library is checked by comparison with this view list. For instance, Liberty library containing both timing and power models has to be made for each PVT corners. As mentioned above, we use three PVT corners (ff28_1.30V_0.00V_125C, tt28_0.92V_0.00V_25C, ss28_0.80V_0.00V_125C). As a result, the final library package must be checked if it includes as many views as the number of PVT corners as described below:

    a) C28SOI_SC_12_CORE_LR_ff28_1.30V_0.00V_125C.lib
    b) C28SOI_SC_12_CORE_LR_tt28_0.92V_0.00V_25C.lib
    c) C28SOI_SC_12_CORE_LR_ss28_0.80V_0.00V_125C.lib

- Likewise, the correctness check is carried out by using the view attributes obtained from the given specification. In these experiments, only Liberty view attributes were dealt with. However, according to the availability of the view attributes in the specification, this check may be extended to cover other views.

Finally, the standard cell library including a complete set of views is developed and checked with the LDSpecX-based specification. By using our method, the total time consumption to extract data for all tasks is approximately inferior to 3 minutes compared to 4 hours by using the traditional (semi-manual) method. In addition to the time saving, the proposed method allows to interactively extract the desired data whereas the traditional one does not permit it.

6.3 Evaluation

A standard cell library was experimentally developed to evaluate the LDSpecX-based specification and its platform.

6.3.1 Specification Evaluation against Five Requirements

In the chapter 4, the requirements of the specification for library development were discussed to propose a novel specification instead of the current ones. Now, we need to verify whether the LDSpecX-based specification coincides with them.

- **Formality**: LDSpecX as an XML-based language allows to unify forms of the current specifications that represent the combination of natural language and tabular forms. In addition, this data model enables to represent the specification encapsulating a large variety of data in a unified way. We are thus able to define more than two thousands parameters by using LDSpecX.
- **Consistency**: In the data model of LDSpecX, the duplication of data is minimized except for the common elements like PVT name to represent the relationship between data. At the end of the specification creation, we have also checked the consistency of these common elements by the help of the checks provided by the platform. In addition, since the specification platform provides every specification developer with the same information from a unique reference database, it may help to ensure the consistency of the specifications.
- **Completeness**: The completeness can be evaluated from two points of view: missing parameter and missing parameter value. Firstly, since we get a complete list of parameters from the reference database, there are few possibilities to have missing parameters. Secondly, the prototype provides the completeness check at the end of

each step (fragment creation and specification creation) while creating a specification. Fundamentally, this check verifies whether each parameter has at least one parameter value. Consequently, a standard cell library has been successfully developed with the given specification. It shows that the created specification in LDSpecX is complete to provide all information for library development.

- **Extensibility**: The extensibility must be guaranteed because it is one of the most important requirements for increasing data of the specification. The prototype proposes two ways to extend specification data. One is to enrich the references so that we can get more parameter information from them. Another is to provide possibilities to define as many parameters as you need. In the case of PVT corner, it tends to increase with the evolution of manufacturing technology as shown in the chapter 2. Thus, these parameters can be totally given by the specification developer.
- **Unambiguousness**: The information defined in the traditional specification may be ambiguous for library developers because each part can be made by different person with his own method and knowledge. For this reason, different users could interpret it differently. To address this issue, all references are centralized in a unique repository shared with all specification developers. In addition, XML language encapsulates data in tags which can also help to easily comprehend the contained information.

6.3.2 Specification Data Processing

The main goal of the prototype is to improve the existing automatic system for library development by the top-down approach that smoothly translates data from the specification to the system throughout the entire process. Indeed, a complete and consistent specification for standard cell library with 28nm FDSOI CMOS technology was created using LDSpecX via the GUI of the prototype. Then, a sequence of all necessary tasks for library development was successfully accomplished with the help of the generators based on the API. These tools allowed us to quickly and automatically generate tool input files containing necessary specification data for the target task. Consequently, the experiments show that the API can significantly help library developers to achieve their goal.

6.4 Conclusion

In this chapter, we have introduced the library development process by using the LDSpecX-based specification platform. A specification with 28nm FDSOI CMOS technology was defined for a standard cell library development. Even if these experiments dealt with only standard cell library, LDSpecX can cover other library categories by supplementing data for each library category. In the afore-described experiments, we saw how to rapidly develop a standard cell library from the LDSpecX-based specification with minimal human interventions in comparison with the traditional manual methods. As a consequence, the main contributions of the proposed prototype are not only to bridge the gap between the specification and the system, but also to improve the productivity of the library. In the next chapter, we conclude this thesis and give some perspectives for future work.

7. Conclusion and Perspectives

Contents

7.1 Conclusion

In this thesis, we presented a first attempt to deal with these inevitable problems relevant to the specification representing the cell-independent information because it becomes a significant bottleneck in library/IP development due to various factors; increasing needs of system designers, technology scaling, developing design methodology, and so on. Especially, the quantity of cell-independent information like PVT corners continuously increases with technology scaling. It can be a pressing problem for library/IP providers to produce their libraries with such information. Furthermore, the current specifications have some limitations such as informality and inconsistency to use their information. Thus, the formalism of the specification is firstly required because the well-formalized specification can enable end-users to explicitly identify the desired information and to manage it.

Before formalizing the specification, several existing specifications have been discussed with respect to the advantages of each form and the effectiveness of its data extraction method. Two well-known traditional specifications in natural language and tabular form were presented. Indeed, the current specifications of STMicroelectronics are also made up of these forms. However, these specifications are not an adapted form to allow to cover extending data because they require frequent modifications. For this reason, a novel approach is greatly required. Two rising methods based on UML and XML that are employed representing the system specification were introduced. The UML-based specification represents visual models by using various graphical notations. On the contrary, the XML-

based specifications like IP-XACT represents physical data model for modeling their information by using an XML language. However, they deal only with the description of systems or components until now so that it is difficult to apply them directly to our target information. Thus, we need a novel specification to address the issues of the current ones.

In order to clarify the issues to address in this thesis, three key points needed to be fully addressed. The first one relates to the formality and contents of the specification. The remaining two are related to the management of the specification from the perspectives of specification developer and library developer: specification creation and specification data extraction. These questions were addressed and the main contributions were oriented towards them:

- **Unified specification for library development**

We intended to propose an XML-based specification representing our target information. Thus, we conducted the analysis of data that are required by library development tools. That was the reason why it was necessary to identify all required data and to classify them for constructing a complete unified specification. An appropriate data model behind these observations was presented. By using this data model, a specification language based on XML named LDSpecX was proposed. This language allows encapsulating complete information for library development in a specification file.

- **Efficient management of the specification**

We introduced a reference-based method and task-based keywords to collaboratively and rapidly create reliable specifications as well as to precisely and rapidly obtain the specification data. Then, LDSpecX-based specification platform was built to implement the proposed methods for dealing with the specifications. This platform provides a user-friendly GUI and an API for creating specification and extracting data respectively. Figure 7.1 describes the developed specification platform. Additionally, a library verification tool based on the API was developed for verifying the library.

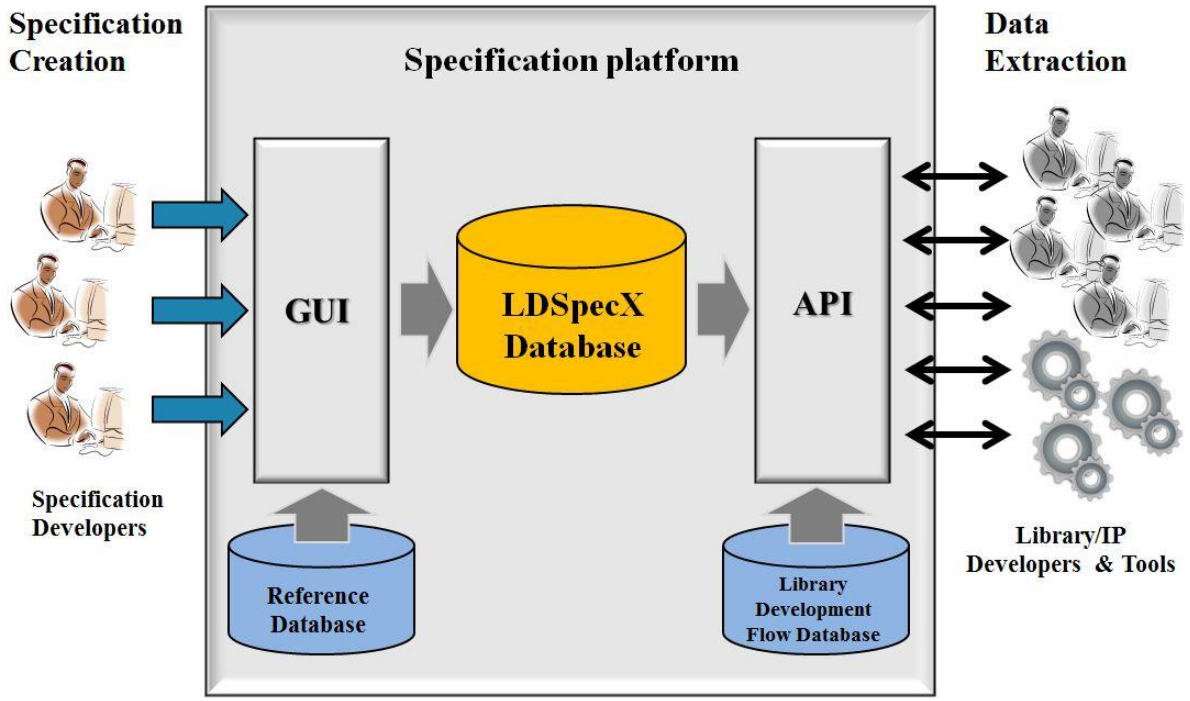

Figure 7.1 Overview of the specification platform

Finally, we detailed the experiments to show the entire development process by using the developed prototype. An appropriate specification for standard cell library with 28nm FDSOI CMOS technology was first created using LDSpecX through the GUI. Then, we presented the development of a standard cell library including 11 combinational and sequential cells using this specification. This library package contains 86 library views to support all required design flows. These experiments show that the prototype significantly facilitates the entire library development process. As a result, we can reduce time and effort for library development. In addition, the proposed specification platform helps to remove potential errors by providing a unique reference database and facilities for dealing with specification data. Although only a standard cell library was dealt with experimentally, we can use our proposal for all library categories to support all required libraries and IPs for the design platform.

7.2 Perspectives

First, one of the imminent tasks to be done is the specification-driven automatic tool configuration. Even if we showed with a simple example, the automatic generation of tool input files by using the API, it is highly required to extend this automation because the library/IP developer often faces obstacles in adjusting the configuration of the system without the awareness of the tool configuration parameters. Specifically, since the tool often has more than one hundred parameters, it is very difficult for every end-user to perfectly take account of them in order to execute his tasks. The specification-driven automatic configuration may therefore ensure that the end-users are able to successfully attain their goals. Moreover it leads to efficiently improve the productivity of the library/IP.

Second, we need to allow the traceability of the specification. After producing the first version of the specification, it can usually be updated when receiving additional requests of customers, releasing a new version of tool, and so on. The changed information must be given in the new version of the specification. As a result of the update of the specification, the contents of all related libraries and IPs should be reproduced completely or partially depending on the impact of the changed information. In this case, generating only the impacted library views helps to save time and effort. We can obtain the information about the tasks to be executed in order to reproduce only the related views by reversely capturing the task information with the help of the task-based keywords. This future work would allow increasing the reusability of the existing library/IP package.

Third, we also presented our prototype to design platform developers, library developers and in-house tool developers. They gave us some important feedbacks concerning this prototype in order to improve it. The most important one is to reuse the existing traditional specifications in natural language or tabular form. It can be very useful to create a specification reusing the existing one. However, we need an efficient method that allows capturing the information from these traditional documents to translate them into the LDSpecX-based specification.

Finally, in this thesis, we focused on the specification for library/IP development. The proposed specification and its platform can further be extended for SoC design by covering the additional information such as library/IP package list that is only required to design a SoC. For this purpose, we need to add a new branch corresponding to the added data category in

the data structure of the specification without the modification of the existing one. The specification platform can be supported to the extended specification by enriching the reference and flow databases. This future work may greatly facilitate both library/IP developers as well as system designers to get the desired data from the specification.

Bibliography

[1] G. E. Moore, "Cramming more components onto integrated circuit," *Electronics,* pp. 114-117, Apr. 1965.

[2] "International Technology Roadmap for Semiconductors, Executive summary," 2011. [Online]. Available: http://www.itrs.net.

[3] J. P. Kent and J. Prasad, "Microelectronics for the real world: "Moore" versus "More than Moore"," in *IEEE Custom Integrated Circuit Conference*, 2008.

[4] W. Arden, M. Brillouet, P. Cogez, M. Graef, B. Huizing and R. Mahnkopf, "More-than-Moore," 2010. [Online]. Available: http://www.itrs.net.

[5] S. Borkar, "Design perspectives on 22nm CMOS and beyond," in *Design Automation Conference*, 2009.

[6] C. Claeys, "Technological challenges of advanced CMOS processing and their impact on design aspects," in *International Conference on VLSI Design*, 2004.

[7] S. Kawanaka, A. Hokazono, N. Yashutake, K. Tatsumura, M. Koyama and Y. Toyoshima, "Advanced CMOS technology beyond 45nm node," in *International Symposium on VLSI Technology*, 2007.

[8] S. H. Yang and et al., "28nm Metal-gate High-K CMOS SoC technology for high-performance mobile applications," in *IEEE Custome Integrated Circuits Conference*, 2011.

[9] D. Sylvester and H. Kaul, "Future performance challenges in nanometer design," in *Design Automation Conference*, 2001.

[10] B. Grundamann, R. Galivanche and S. Kundu, "Circuit and platform design challenges in technologies beyond 90nm," in *Design, Automation and Test in Europe Conference and Exhibition*, 2003.

[11] G. G. Gielen and W. Dehaene, "Analog and digital circuit design in 65nm CMOS: end of the road?," in *Design, Automation and Test in Europe Conference and Exhibition*, 2005.

[12] G. E. Gielen, "Design methodologies and tools for circuit design in CMOS nanometer technologies," in *European Solid-State Circuits Conference*, 2006.

[13] C. Hu, "New sub-20nm transistors - Why and How," in *Design Automation Conference*, 2011.

[14] J. Warnock, "Circuit design challenges at the 14nm technology node," in *Design Automation Conference*, 2011.

[15] A. Chang, "Case study of a 65-nm SoC design," *IEEE Design & Test of Computers,* vol. 56, no. 2, pp. 14-19, Apr. 2009.

[16] P. E. Grnonowski, W. J. Bowhill, R. P. Preston, M. K. Gowan and R. L. Allmon, "High-performance microprocessor design," *IEEE Journal of Solid-State Circuits,* vol. 33, no. 5, pp. 676-686, May 1998.

[17] T. Kitahara, "A clock-gating method for low-power LSI design," in *Asia and South Pacific Design Automation Conference*, 1998.

[18] M. O. Sharker and M. A. Bayoumi, "A clock gated flip-fop for low power applications in 90nm cmos," in *IEEE International Symposium on Circuits and Systems*, 2011.

[19] M. H. Lin, C. C. Hsu and Y. T. Chang, "Recent research in clock power saving with multi-bit flip-flops," in *IEEE International Midwest Symposium on Circuits and Systems*, 2011.

[20] T. Karnik, S. Borkar and V. De, "Sub-90nm technologies-challenges and opportunities for CAD," in *IEEE International Conference on Computer Aided Design*, 2002.

[21] D. J. Frank, R. Puri and D. Toma, "Design and CAD challenges in 45nm CMOS and beyond," in *IEEE International Conference on Computer Aided Design*, 2006.

[22] T. Kam, S. Rawat, D. Kirkpatrick, R. Roy, G. S. Spirakis, N. Sherwani and C. Peterson, "EDA challenges facing future microprocessor design," *IEEE Transaction of Computer Aided Design of Integrated Circuits and Systems,* vol. 19, no. 12, pp. 1498-1506, Dec. 2000.

[23] "International Technology Roadmap for Semiconductors, Design," 2009. [Online]. Available: http://www.itrs.net.

[24] L. Xiu, VLSI Circuit Design Methodology Demystified: A Conceptual Taxonomy, Willey, 2007.

[25] J. Borel, "System on a Chip (SoC) and design methodology challenges," *Microelectronic Engineering,* vol. 54, no. 1-2, pp. 15-22, Dec. 2000.

[26] H. Onodera, M. Hashimoto and T. Hashimoto, "ASIC design methodology with on-demand library generation," in *Symposium on VLSI Circuits Digest of Technical Papers*, 2001.

[27] H. Eriksson and P. Larsson-Edefors, "Full-custom vs. standard-cell design flow - an adder case study," in *Asia and South Pacific Design Automation Conference*, 2003.

[28] R. Saleh, G. Lim, T. Kadowaki and K. Uchiyama, "Trends in low power digital system-on-chip designs," in *IEEE International Symposium on Quality Electronic Design*, 2002.

[29] P. Magarshack and P. G. Paulin, "System-on-Chip beyond the nanometer wall," in *Design Automation Conference*, 2003.

[30] M. Hashimoto, K. Fujimori and H. Onodera, "Standard cell libraries with various driving strengh cells for 0.13, 0.18 and 0.35 μm technologies," in *Asia and South Pacific Design Automation Conference*, 2003.

[31] B. Ren, A. Wang, J. Bakshi, K. Liu, W. Li and W. Dai, "A domain-specific cell based ASIC design methodology for digital signal processing applications," in *Design, Automation and Test in Europe Conference and Exhibition*, 2004.

[32] S. Wu, F. Wang and L. S. Juang, "Foundry's perspective of system integration: quality design and time-to-volume," in *IEEE International Symposium on Quality Electronic Design*, 2001.

[33] P. Royannez and et al., "A design platform for 90-nm leakage reduction techniques," in *Design Automation Conference*, 2005.

[34] K. Kranen, "Electronic Design Automation (EDA) flow for development of an ARM processor-based silicon-on-insulator (SOI) SoC," in *IEEE International SOI Conference*, 2009.

[35] A. Chenouf, A. Slimane, M. L. Berrandjia, A. K. Oudjida, A. Smatti and L. Akak, "Design-kit development based upon ISIT's CMOS 1μm process technology," in *International Multi-Conference on Systems, Signal, Devices*, 2010.

[36] J. Dreesen, "Standard cell development flow," in *Euro ASIC*, 1990.

[37] J. W. Chong and R. G. Forbes, "Design of basic CMOS cell library," *IEE Processings-*

G Circuits, Devices and Systems, vol. 139, no. 2, pp. 256-260, Apr. 1992.

[38] A. B. Jambeck, A. R. Mohd Noor Beg and M. R. Ahmad, "Standard cell library development," in *International Conference on Microelectronics*, 1999.

[39] J. D. Djigbenou, T. V. Nguyen, C. W. Ren and D. S. Ha, "Development of TSMC 0.25μm standard cell library," in *IEEE SoutheastCon*, 2007.

[40] D. Bekiaris, A. Papanikolaou, G. Stamelos, D. Soudris, G. Economakos and K. Pekmestzi, "A standard-cell library suite for deep-deep sub-micron CMOS technologies," in *International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, 2011.

[41] J. K. Yuan, "Key to a successful cell library development: design methodology," in *IEEE ASIC Seminar and Exhibit*, 1989.

[42] D. G. Baltus, T. Varga, R. C. Armstrong, J. Duh and T. G. Matheson, "Developing a concurrent methodology for standard-cell library generation," in *Design Automation Conference*, 1997.

[43] C. Bittlestone, A. Hill, V. Singhal and A. N.V., "Architecting ASIC libraries and flows in nanometer era," in *Design Automation Conference*, 2003.

[44] "Liberty User Guides and Reference Manual Suite Version 2012.06," [Online]. Available: http://www.opensourceliberty.org.

[45] M. A. Cirit, "Characterizing a VLSI standard cell library," in *Custom Integrated Circuits Conference*, 1991.

[46] B. Ackalloor and D. Gaitonde, "An overview of library characterization in semi-custom design," in *Custom Integrated Circuits Conference*, 1998.

[47] J. B. Sulistyo and D. S. Ha, "A new characterization method for delay and power dissipation of standard library cell," *VLSI Design,* vol. 15, no. 3, pp. 667-678, 2002.

[48] S. Sundareswaran, J. A. Abraham, R. Panda, Y. Zhang and A. Mittal, "Characterizing of sequential cells for constraint sensitivities," in *IEEE International Symposium on Quality Electronic Design*, 2009.

[49] U. Doddannagari, S. Hu and W. Shi, "Fast characterization of parameterized cell library," in *IEEE International Symposium on Quality Electronic Design*, 2009.

[50] S. Miryala, B. Kaur, B. Anand and S. Manhas, "Efficient nanoscale VLSI standard cell library characterization using a novel delay model," in *IEEE International Symposium on Quality Electronic Design*, 2011.

[51] "CCS Timing Version 2.0 Technical White Paper," 2006. [Online]. Available: http://www.opensourceliberty.org.

[52] T. E. Motassadeq, "CCS vs NLDM comparison based on a complete automated correlation flow between primetime and hspice," in *Saudi International Electronics, Communications and Photonics Conference*, 2011.

[53] J. Y. Lin, W. Z. Shen and J. Y. Jou, "A power modeling and characterization method for the CMOS standard cell library," in *International Conference on Computer-Aided Design*, 1996.

[54] R. Ramanarayanan, N. Vijaykrishnan and M. J. Irwin, "Characterizing dynamic and leakage power behavior in flip-flops," in *Annual IEEE International ASIC/SOC Conference*, 2002.

[55] W. Roethig, "Library characterization and modeling for 130nm and 90nm SoC design," in *International SOC Conference*, 2003.

[56] S. Chandrasekar and G. K. Varshney, "A comprehensive methodology for noise

characterization of ASIC cell libraries," in *IEEE International Symposium on Quality Electronic Design*, 2005.

[57] A. Wielgus and W. A. Pleskacz, "Characterization of CMOS sequential standard cells for defect based voltage testing," in *East-West Design & Test Symposium*, 2008.

[58] J. P. Moreau, J. Borel and D. Samani, "European trends in library development," *IEEE Mirco,* vol. 12, no. 4, pp. 43-53, Aug. 1992.

[59] Y. H. Poh, C. Y. Chew, K. L. Hoi and W. P. Soo, "Interoperable physical design database between OpenAccess and Milkyway," in *IEEE Asia Pacific Conference on Circuits and Systems*, 2010.

[60] B. Guan and C. Sechen, "Large standad cell libraries and their impact on layout area and circuit performance," in *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1996.

[61] M. Rahman, R. Afonso, H. Tennakoon and C. Sechen, "Design automation tools and libraries for low power digital design," in *IEEE Dallas Circuits and Systems Workshop*, 2010.

[62] L. W. Wang and H. W. Luo, "Automated IP quality qualification for efficient system-on-chip design," in *International Conference on Electronic Packaging Technology & High Density Packaging*, 2012.

[63] R. B. Lin, I. H. Chou and C. M. Tsai, "Benchmark circuits improve the quality of a standard cell library," in *Asia and South Pacific Design Automation Conference*, 1999.

[64] J. Darringer and et al., "EDA in IBM: past, present and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 19, no. 12, pp. 1476-1497, Dec. 2000.

[65] A. Sangiovanni-Vincentelli, "The tides of EDA," *IEEE Design & Test of Computers,* vol. 20, no. 6, pp. 59-75, 2003.

[66] A. Hemani, "Charting the EDA roadmap," *IEEE Circuits and Devices Magazine,* vol. 20, no. 6, pp. 5-10, 2004.

[67] D. MacMillen, M. Butts, R. Camposano, D. Hill and T. W. Williams, "An industrial view of electronic design automation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 19, no. 12, pp. 1428-1448, Dec. 2000.

[68] R. D. Kilmoyer, D. J. Hathaway and A. M. Chu, "A reduced circuit library design system," in *IEEE Custom Integrated Circuits Conference*, 1988.

[69] R. N. Rao, "An open environment for standard cell and gate array library development," in *Euro ASIC*, 1992.

[70] H. Onodera, A. Hirata, T. Kitamura and K. Tamaru, "P2Lib: process-portable library and its generation system," in *IEEE Custom Integrated Circuits Conference*, 1997.

[71] J. Togni, F. R. Shneider, V. P. Correia, R. P. Ribas and A. I. Reis, "Automatic generation of digital cell libraries," in *Symposium on Integrated Circuits and Systems Design*, 2002.

[72] M. Hashimoto, K. Fujimori and H. Onodera, "Automatic generation of standard cell library in VDSM technologies," in *IEEE International Symposium on Quality Electronic Design*, 2004.

[73] Y. Jung, "Automated standard cell library generation & study of cell library functional content," [Online]. Available: http://vlsicad.eecs.umich.edu.

[74] T. H. McFaul and K. Perrey, "Characterizing a cell library using ICCS," in *IEEE ASIC Seminar and Exhibit*, 1990.

[75] D. Patel, "CHARMS: characterization and modeling system for accurate delay prediction of ASIC design," in *IEEE Custom Integrated Circuits Conference*, 1990.

[76] J. C. Herbert, "An integrated design and characterization environment for the development of a standard cell library," in *IEEE Custom Integrated Circuits Conference*, 1991.

[77] K. Anshumali, "ACC: automatic cell characterization," in *Euro ASIC*, 1991.

[78] C. S. Wu, L. C. Lin, D. Chou and K. C. Ting, "An automatic cell characterization environment for cell-based design methodology," in *IEEE International ASIC Conference and Exhibit*, 1993.

[79] A. J. Daga, L. Mize, S. Sripada, C. Wolff and Q. Wu, "Automated timing model generation," in *Design Automation Conference*, 2002.

[80] R. I. K. and M. S. Bhat, "AutoLibGen: an open source tool for standard cell library characterization at 65nm technology," in *International Conference Electronic Design*, 2008.

[81] S. Borkar, T. Karnik and V. De, "Design and reliability challenges in nanometer technologies," in *Design Automation Conference*, 2004.

[82] S. Basu, P. Thakore and R. Vemuri, "Process variation tolerant standard cell lbirary development using reduced dimension statistical modeling and optimization techniques," in *IEEE International Symposium on Quality Electronic Design*, 2007.

[83] S. Pasricha, Y. H. Park, F. J. Kurdahi and N. Dutt, "Incorporating PVT variation in system-level power exploration of on-chip communication architectures," in *International Conference on VLSI Design*, 2008.

[84] I. G. Harris, "Extracting design information from natural language specifications," in *Design Automation Conference*, 2012.

[85] R. Drechsler, M. Soeken and R. Wille, "Formal specification level: towards verification-driven based on natural language processing," in *Forum on Specification and Design Language*, 2012.

[86] K. Heninger, D. L. Parnas, J. E. Shore and J. W. Kallander, "Software requirements for the A-7E aircraft," Naval Research Lab, 1978.

[87] C. Heitmeyer, J. Kirby and B. Labaw, "Tools for formal specification, verification, and validation of requirements," in *Conference on Computer Assurance*, 1997.

[88] C. Heitmeyer, M. Archer, R. Bharadwaj and R. Jeffords, "Tools for constructing requirements specifications: the SCR toolset at the age of ten," *International Journal of Computer Systems Science & Engineering,* vol. 20, no. 1, pp. 19-35, 2005.

[89] W. Haas, S. Gossens and U. Heinkel, "Integration of formal specification into the standard ASIC design flow," in *International Symposium on High Assurance Systems Engineering*, 2002.

[90] A. Schneider, T. Bluhm and T. Renner, "Formal verification of abstract system and protocol specifications," in *IEEE/NASA Software Engineering Workshop*, 2006.

[91] U. Pross, E. Markert, J. Langer, A. Richter, C. Drechsler and U. Heinkel, "A platform for requirement based fomal specification," in *Forum on Specification and Design Languages*, 2008.

[92] J. Rumbaugh, I. Jacobson and G. Booch, The Unified Modeling Language Reference 2nd ed., Addison Wesley, 2004.

[93] Q. Zhu, R. Oishi, T. Hasegawa and T. Nakata, "System-on-Chip validation using UML and CWL," in *International Conference on Hardware/Software Codesign and System*

Synthesis, 2004.

[94] L. S. Indrusiak and M. Glesner, "SoC specification using UML and actor-oriented modeling," in *International Baltic Electronics Conference*, 2006.

[95] "ISO 8879:1986 Information processing - Text and office systems - Standard Generalized Markup Language (SGML)," [Online]. Available: http://www.iso.org.

[96] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau and J. Cowan, "Extensible Markup Language (XML) 1.1 2nd ed.," W3C Consortium, 2006.

[97] "XML Schema 1.0," W3 Consortium, [Online]. Available: http://www.w3.org/XML/Schema.

[98] "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integration, and Reusing IP within Tool Flow," 2009. [Online]. Available: http://standards.ieee.org.

[99] C. K. Lennard and et al., "Industrially proving the SPIRIT consortium specifications for design chain integration," in *Design, Automation and Test in Europe Conference and Exhibition*, 2006.

[100] M. Zys, E. Vaumorin and I. Sobanski, "Straightforward IP integration with IP-XACT RTL-TLM switching," 2008.

[101] A. Kamppi and et al., "Kactus2: environment for embedded product development using IP-XACT and MCAPI," in *Euromicro Conference on Digital System Design*, 2011.

[102] C. Trummer and et al., "Searching extended IP-XACT components for SoC design based on requirements similarity," *IEEE System Journal,* vol. 5, no. 1, pp. 70-79, 2011.

[103] E. Salminen, T. D. Hamalainen and M. Hannikainen, "Applying IP-XACT in product data management," in *International Symposium on System-on-Chip*, 2011.

[104] M. Ma, L. Hedrich and C. Sporrer, "A machine-readable specification of analog circuits for integration into a validation flow," in *Forum on Specification and Design Language*, 2011.

[105] M. Ma, M. Meissner and L. Hedrich, "A case study: automatic topology synthesis for analog circuit from an ASDeX specification," in *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, 2012.

[106] M. Ma, L. Hedrich and C. Sporrer, "ASDeX: a formal specification for analog circuit enabling a full automated design validation," *Design Automation for Embedded Systems,* Jun. 2012.

[107] J. K. Chae, S. Bertrand, P.-. F. Ollagnon, P. Mougeat, J.-. A. François, R. Chotin-Avot and H. Mehrez, "Efficient State-Dependent Power Model for Multi-bit Flip-Flop Banks," in *IEEE International Midwest Circuits and Systems Conference*, 2013.

[108] J. K. Chae, P. Mougeat, J.-. A. Francois, R. Chotin-Avot and H. Mehrez, "A formalism of the specifications for library development," in *IEEE International System-on-Chip*, 2013.

[109] "Python," [Online]. Available: http://www.python.org.

[110] "XSLT Version 1.0," W3 Consortium, [Online]. Available: http://www.w3.org/TR/xslt.

[111] L. Wall, T. Christiansen and J. Orwant, Programming Perl, 3nd ed., O'reilly, 2009.

[112] "OMG Unified Modeling Language (OMG UML) Superstructure Version 2.4.1," [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Superstructure.

[113] W. Kruijtzer and et al., "Industrial IP Integration Flows based on IP-XACT Standards," in *Design, Automation and Test in Europe Conference and Exhibition*, 2008.

[114] "Oxygen XML Editor," [Online]. Available: http://www.oxygenxml.com.

[115] J. K. Chae, P. Mougeat, J.-A. Francois, R. Chotin-Avot and H. Mehrez, "A reference-based specification tool for creating reliable library development specifications," in *IEEE International New Circuits and Systems Conference*, 2014.

[116] X. D. Xie, P. Li., A. W. Ruan, W. C. Li and W. Li, "Design and implementation of a standard cell based programmable logic core," in *International Conference on Communications, Circuits, and Systems*, 2009.

[117] P. Meinerzhagen, S. Y. Sherazi, A. Burg and J. N. Rodrigues, "Benchmarking of standard-cell based memories in the sub-Vt domain in 65-nm CMOS technology," *IEEE Transactions on Emerging and Selected Topics in Circuits and Systems,* vol. 1, no. 2, pp. 173-182, June 2011.

[118] "Mentor Calibre," [Online]. Available: http://www.mentor.com.

[119] "Cadence OpenAccess," [Online]. Available: http://www.cadence.com.

[120] "Synopsys Milkyway," [Online]. Available: http://www.synopsys.com.

Publications

❖ INTERNATIONAL CONFERENCES

[C1] J.K. CHAE, S. BERTRAND, P.-F. OLLAGNON , P. MOUGEAT, J.-A. FRANCOIS, R. CHOTIN-AVOT and H. MEHREZ, "Efficient state-dependent power model for multi-bit flip-flop banks," IEEE International Midwest Symposium on Circuits and Systems, Columbus, USA, Aug. 2013.

[C2] J.K. CHAE, P. MOUGEAT, J.-A. FRANCOIS, R. CHOTIN-AVOT and H. MEHREZ, "A formalism of the specifications for library development," IEEE International System-on-Chip Conference, Erlangen, Germany, Sept. 2013.

[C3] J.K. CHAE, P. MOUGEAT, J.-A. FRANCOIS, R. CHOTIN-AVOT and H. MEHREZ, "A reference-based specification tool for creating reliable library development specifications, " IEEE International New Circuits and Systems Conference, Trois-Rivieres, Canada, Jun. 2014. (accepted)

❖ NATIONAL CONFERENCE

[C4] J.K. CHAE, P. MOUGEAT, J.-A. FRANCOIS, R. CHOTIN-AVOT and H. MEHREZ, "Formalisme de la spécification de la plateforme de conception pour le développement de la bibliothèque," 15ème Journées Nationales du Réseau Doctoral en Micro-nanoélectronique, Grenoble, France, Jun. 2013.

Résumé

1. Introduction

Gordon E. Moore a présenté la fameuse loi de Moore dans un article publié en 1965[1]. Cette loi prédit que le nombre de transistors des processeurs doublerait à coût constant tous les deux ans. De nos jours, une autre tendance nommée ‘More than Moore’ pour une diversification fonctionnelle, s’applique également. La figure 1.1 présente les diverses applications des systèmes sur puce (*System-on-Chip* SoC) vis-à-vis de ces différentes tendances.

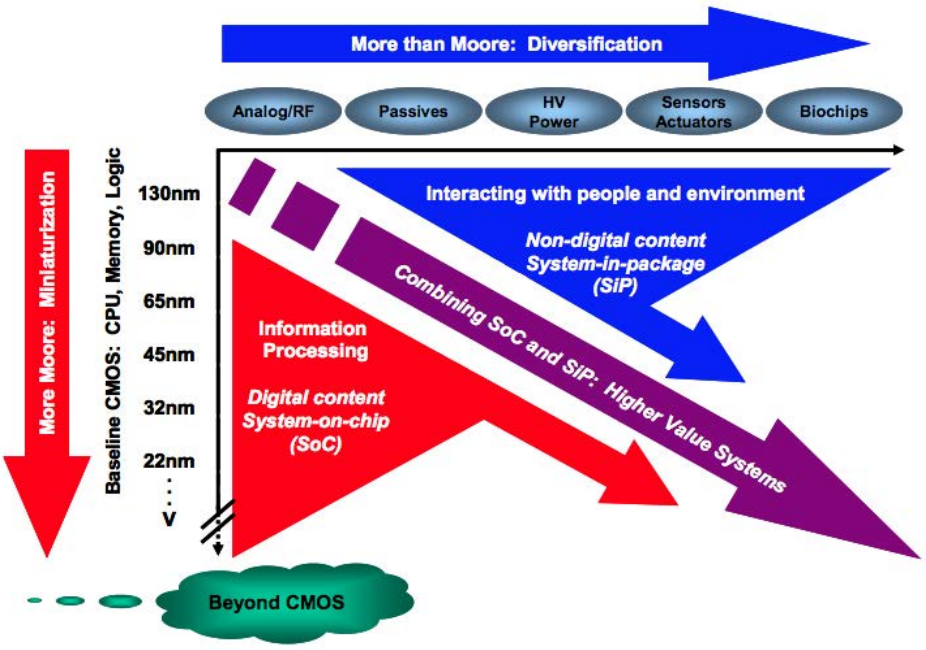

Figure 1.1 ‘Moore’s law’ et ‘More than Moore’ dans l’industrie des semi-conducteurs (source : ITRS)

Pour la conception de systèmes complexes, les fabricants de semi-conducteurs comme STMicroelectronics, fournissent à leurs clients une plateforme de conception (*Design Platform* DP) contenant tous les composants nécessaires comme des bibliothèques de cellules et des blocs plus complexes (*Intellectual Property* IP). Afin d’améliorer la productivité, en automatisant et sécurisant les processus de conception, de nouvelles approches sont

nécessaires pour les concepteurs de systèmes et également pour les fournisseurs de bibliothèques.

Dans cette thèse, nous proposons d'améliorer le développement de bibliothèques par une meilleure formalisation des spécifications.

2. Développement d'une bibliothèque de cellules et d'IPs à partir de la spécification de la plateforme de conception

La DP est une solution totale qui permet à une équipe de conception de produire un système sur puce. La spécification de DP doit définir une large gamme d'informations depuis les paramètres technologiques comme la nature du process, jusqu'aux conditions d'utilisation telles que la tension et la température, en passant par l'information sur les outils de conception (CAO) pour le développement de bibliothèques de cellules et d'IPs.

Le développement d'une bibliothèque est effectué en quatre phases comme l'illustre la figure 2.1 : spécification, conception, dérivation et validation.

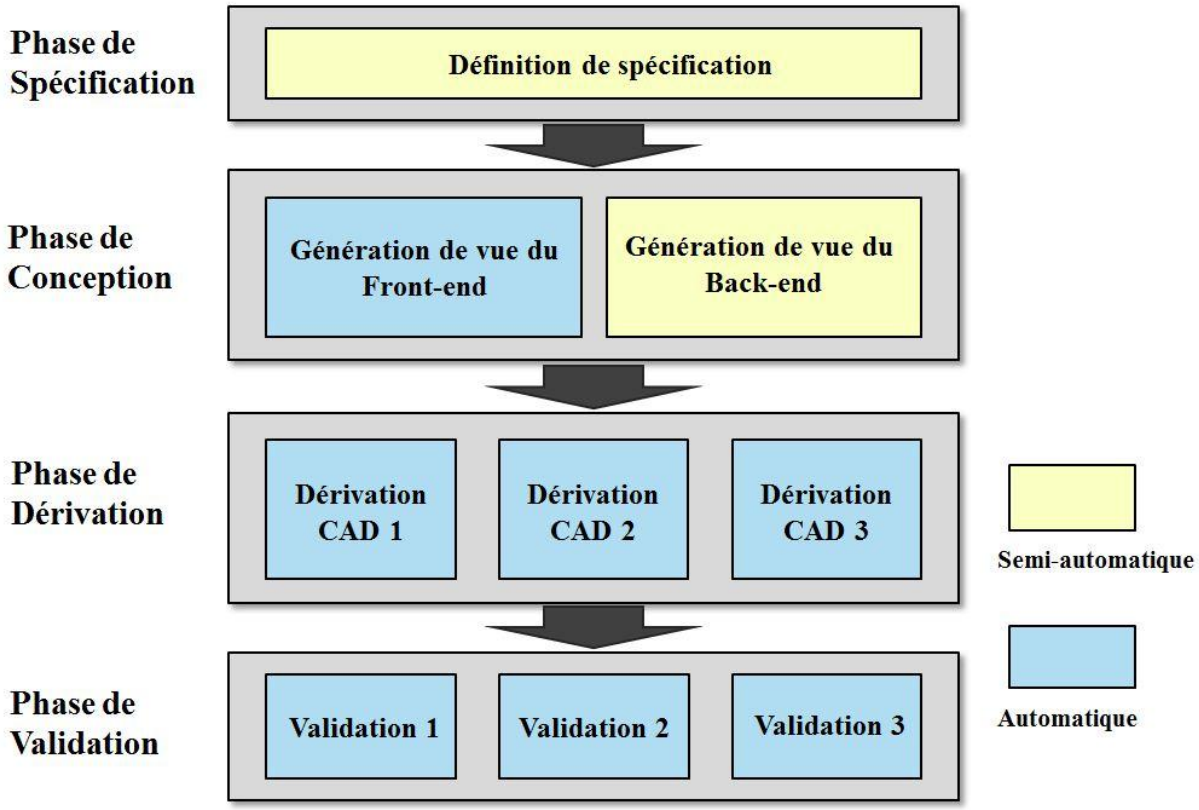

Figure 2.1 Flot de développement de bibliothèques (source : ST)

Pour automatiser ce flot, STMicroelectronics a un système automatique qui est composé d'outils internes comme le montre la figure 2.2. Ces outils permettent d'exécuter des tâches de générer des vues nécessaires et de les vérifier avec des données extraites des spécifications.

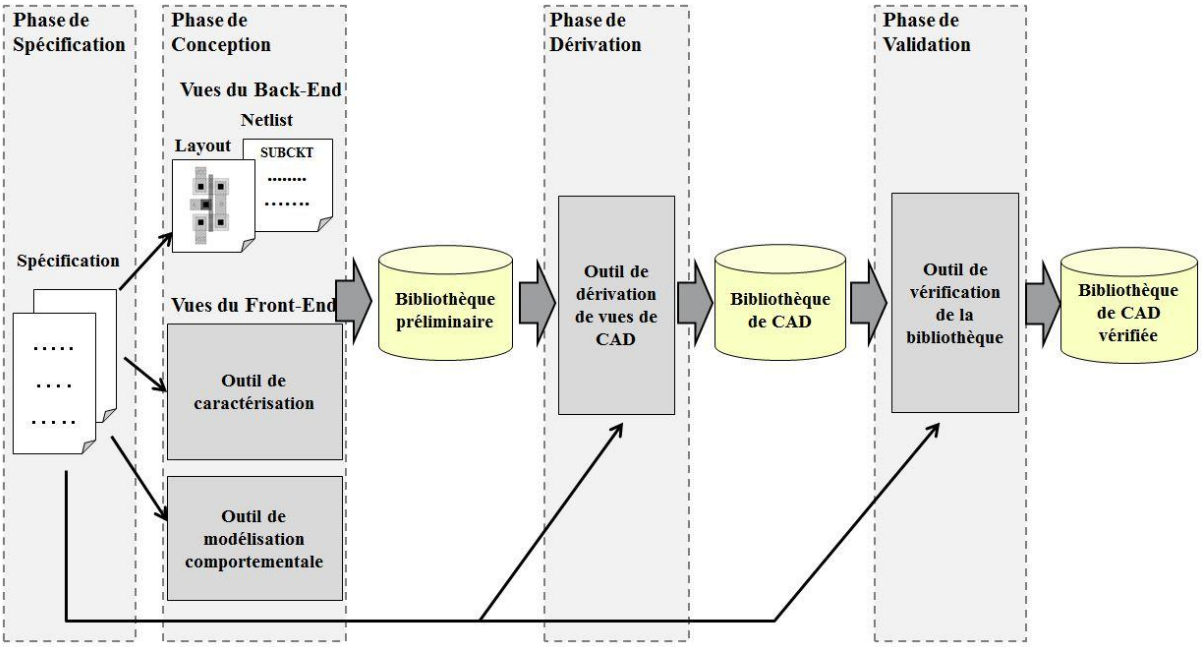

Figure 2.2 Système automatique de développement de bibliothèques (source: ST)

D'autre part, les développeurs de bibliothèques/IPs ont de plus en plus de difficultés à obtenir les données désirées à partir des spécifications existantes en raison de leur manque de formalisme et leur complexité. Nous avons d'abord identifié les problèmes concernant le contenu de la spécification et le traitement des spécifications par :

- **Spécification**
  - **Manque de formalisme**
  - **Inconsistance**
  - **Ambiguïté**
  - **Information manquante**
  - **Augmentation du volume de données**
- **Création de la spécification**
  - **Manque d'une base de données de référence**
  - **Manque d'un outil assistant de la spécification**
  - **Manque de vérification de la spécification**

- **Extraction des données de spécification**
  - **Nécessité de méthodes efficaces et d'un outil pour l'extraction des données de spécification**

3. Etat de l'art

La spécification représente une collection d'informations pour effectuer une activité. Plusieurs formes peuvent être utilisées pour définir cette spécification. Traditionnellement, les spécifications sont écrites en langage naturel ou sous forme de tableaux. La méthode basée sur le langage naturel est une méthode très courante pour définir l'information mais pour avoir une forme non-ambiguë, il faut utiliser des phrases très simples et un vocabulaire limité. D'ailleurs, pour effectuer l'extraction de l'information à partir de ce type de spécification, il faut une bonne analyse sémantique [84], [85]. Par contre, la méthode basée sur des tableaux permet de facilement étendre l'information et de rapidement l'extraire en identifiant les colonnes et les lignes ou en utilisant des modèles. Cependant, ce type de spécification a également quelques limitations comme l'indépendance des données [87], [88], [89], [90], [91]. Récemment, deux spécifications basées sur le langage de modélisation unifié (*Unified Modeling Language* UML) et le langage de balisage extensible (*eXtensible Markup Language* XML) ont aussi été utilisées dans le domaine du système. L'UML basé sur les notations graphiques fournit beaucoup d'expressions pour décrire la spécification d'un système mais il représente un modèle conceptuel [94], [95]. L'XML permet d'enregistrer de nombreux types de données en forme textuelle avec des balises. Donc, il est largement appliqué à divers domaines grâce à ses nombreux avantages comme l'extensibilité, la portabilité et la lisibilité par une machine. Par exemple, IP-XACT [98] et ASDeX [106] sont proposés pour représenter la spécification d'IPs numériques et celle d'IP analogique respectivement à l'aide de XML. Malheureusement, ces spécifications existantes ne peuvent représenter l'ensemble de nos informations. Enfin, les spécifications actuelles de STMicroelectronics font appel au langage naturel et aux tableaux selon les caractéristiques des données comme le montre la figure 3.1.

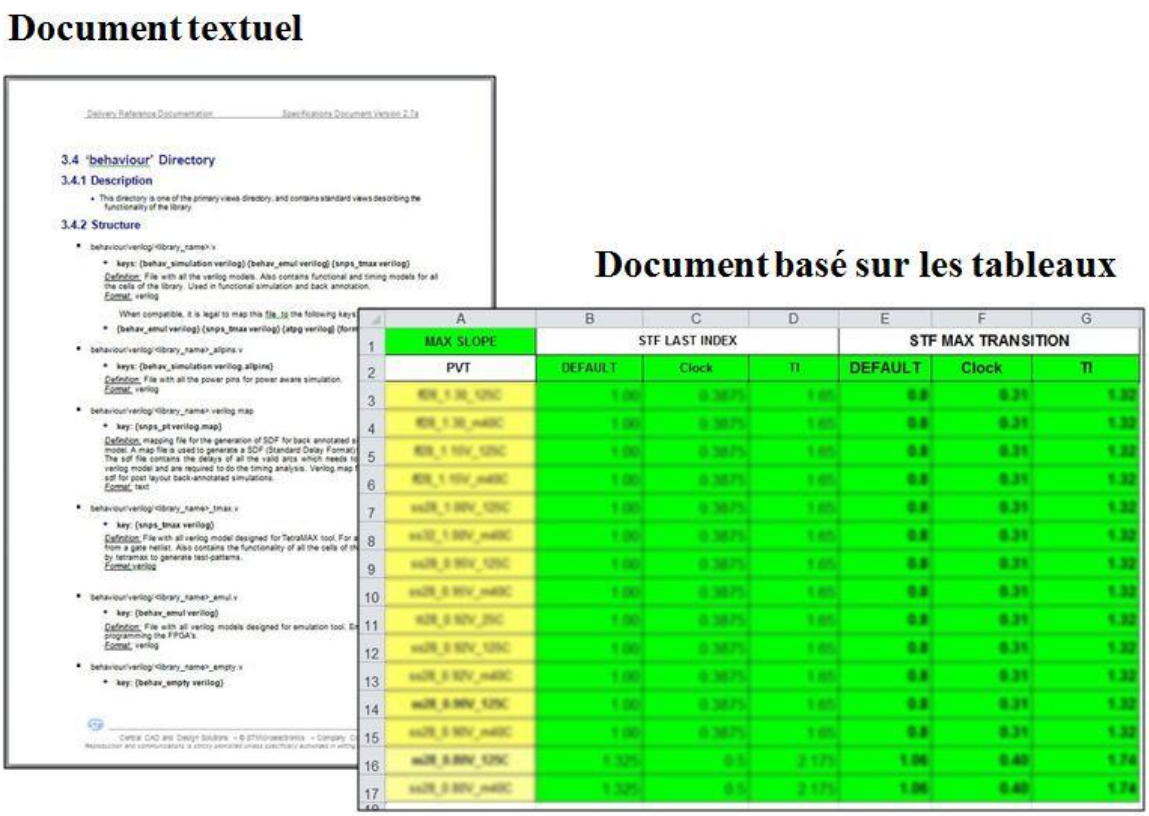

Figure 3.1 Extrait des spécifications de plateforme de conception (source: ST)

En résumé, les avantages et les inconvénients de tous les types de spécifications sont donnés dans le tableau 3.1 du point de vue de la modélisation des données et du traitement de ces données. Dans ce tableau, un cercle vert indique que le type de spécification est adapté au critère au contraire d'une croix rouge. D'après ce tableau, les spécifications de STMicroelectronics basées sur le langage naturel et des tableaux, ont des avantages comme l'extensibilité et la flexibilité pour la modélisation des données, ainsi que la facilité pour les entrer et la lisibilité humaine pour les traiter. En revanche, elles ont des inconvénients comme la manque de formalisme pour la modélisation et la difficulté d'extraire des données pour leur traitement. C'est pourquoi nous proposons une spécification basée sur XML pour le développement de bibliothèques car XML fournit le plus d'avantages pour encapsuler une large gamme d'informations et facilement traiter des données.

Table 3.1: Résumé des spécifications

Critères		Spécification basée sur le **langage naturel**	Spécification basée sur les **tableaux**	Spécification basée sur l'**UML**	Spécification basée sur l'**XML**
Modélisation des données	Formalisme	✗	○	○	○
	Extensibilité	○	○	○	○
	Flexibilité	○	○	○	○
	Expressivité	○	○	○	○
	Disponibilité de divers types de données	○	○	○	○
	Relation entre les données	✗	✗	○	○
Traitement des données	Facilité d'entrer des données	○	○	✗	○
	Facilité d'extraire des données	✗	○	✗	○
	Lisibilité humaine	○	○	○	✗
	Lisibilité par machine	✗	✗	✗	○

4. Méthodologies pour la spécification du développement de bibliothèques

Nous proposons un formalisme de la spécification pour le développement de bibliothèques. De plus, nous présentons deux méthodes pour la création de la spécification et l'extraction des données.

- **Formalisme de spécification**

Pour formaliser les spécifications pour le développement de bibliothèques, il faut d'abord établir un modèle de données approprié. C'est pour cette raison que nous avons identifié et classifié toutes les données utilisées par les outils internes pour le développement

d'une bibliothèque. La figure 4.1 représente les résultats d'analyse des données d'entrée des outils. D'après ces résultats, toutes les données identifiées peuvent être classifiées en trois groupes : paramètres d'outil, paramètres de vue et paramètres de technologie.

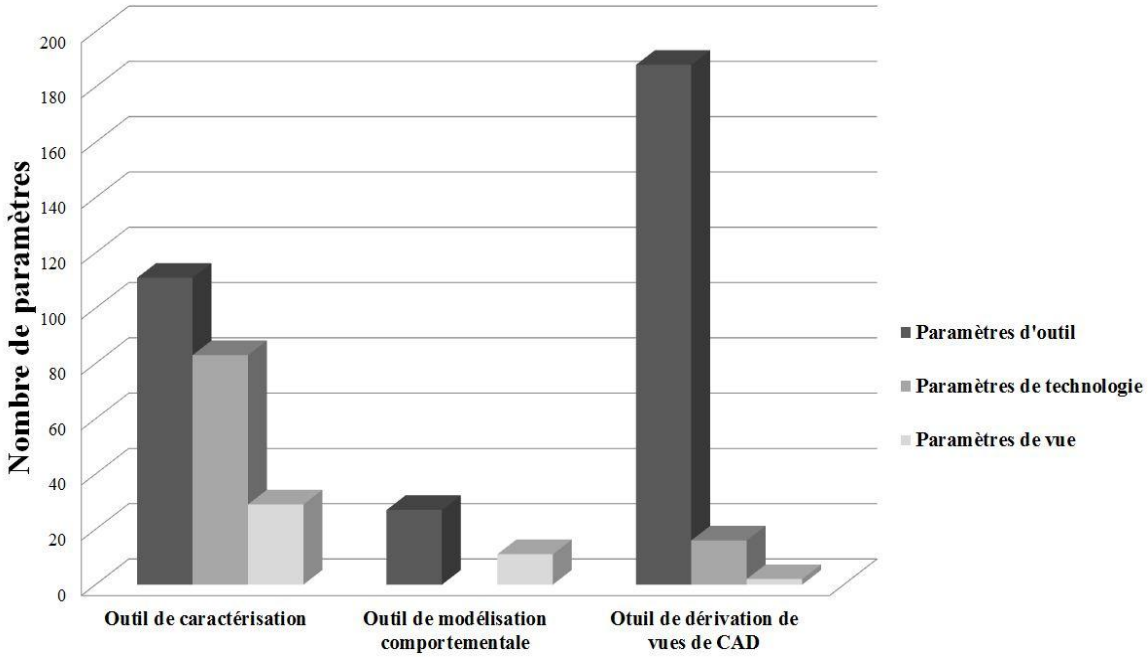

Figure 4.1 Résultats d'analyse des données d'entrée des outils (source : ST)

En conséquence, nous avons proposé une taxonomie des données de la spécification comme le montre la figure 4.2. Cette taxonomie est une fondation du modèle de données.

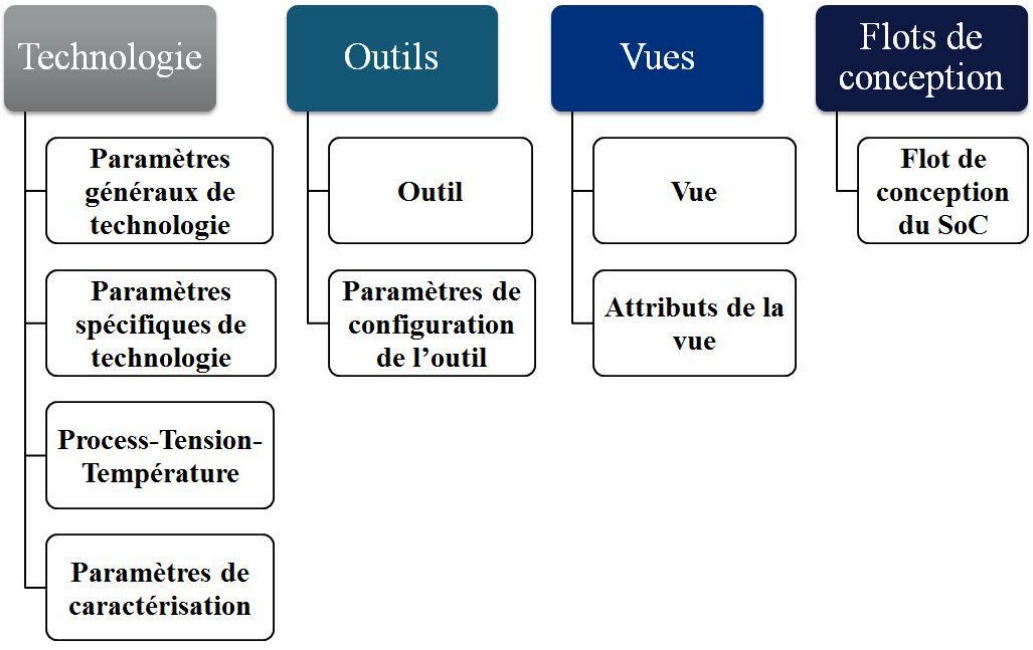

Figure 4.2 Taxonomie des données de la spécification

A l'aide de cette taxonomie, nous avons établi un modèle de donnée approprié. Ce modèle nous permet de représenter toutes les informations nécessaires pour le développement de bibliothèques. La figure 4.6 donne tous les objets modélisant les données et leurs relations.

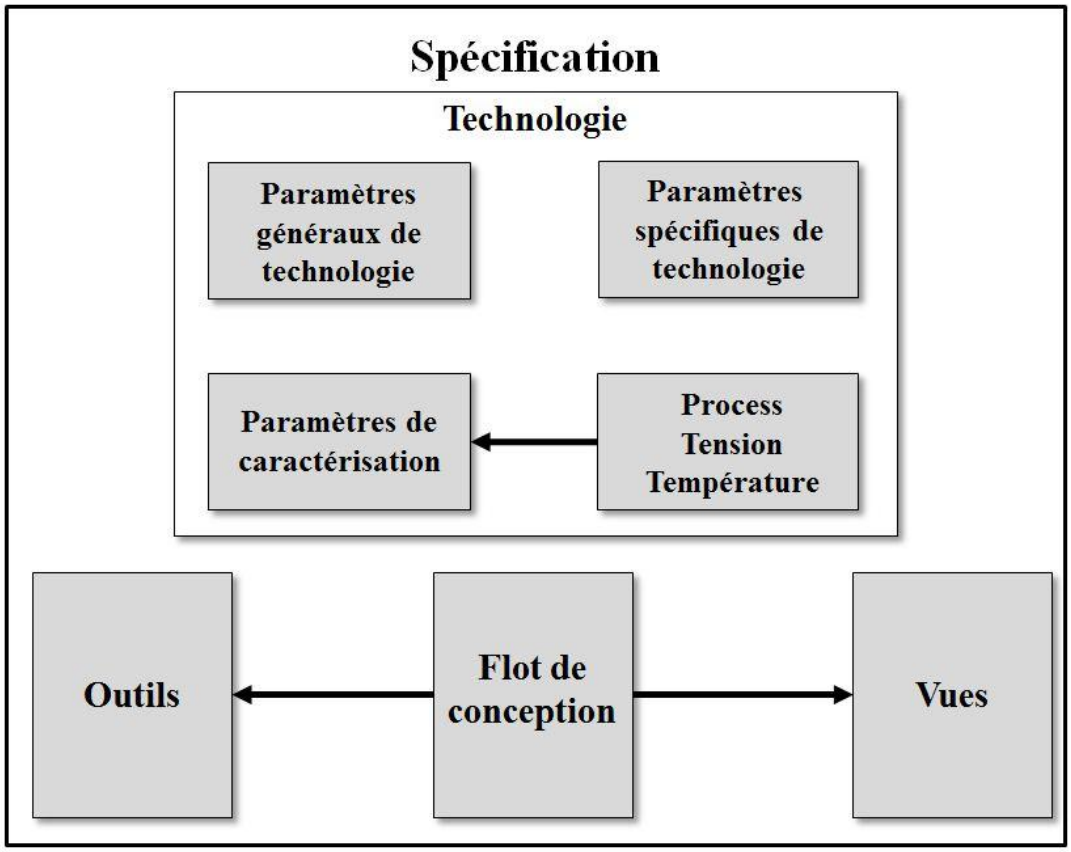

Figure 4.3 Données de la spécification et leurs relations

L'objet modélisant la donnée représente un ensemble de paramètres explicité dans la figure 4.4 en utilisant un diagramme de classe UML. L'information sur un paramètre est donnée par un nom et au moins une valeur. Ce modèle de donnée basique est utilisé pour construire le modèle de donnée pour la spécification.

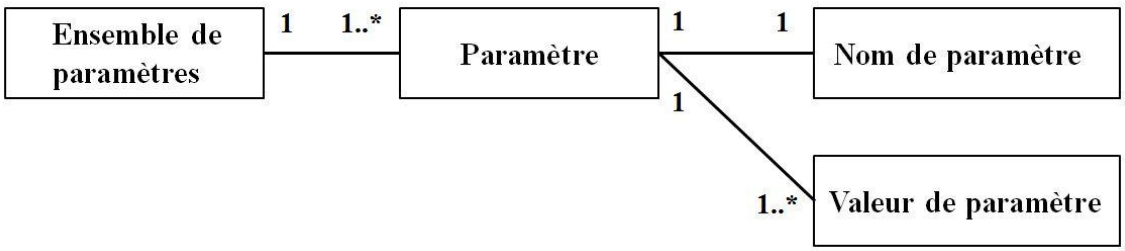

Figure 4.4 Objet de donnée

Il y a deux sortes d'ensembles de paramètres selon qu'ils soient sous-catégorisés ou non. La figure 4.5 donne un exemple de ces deux sortes d'ensembles de paramètres. Le

premier est un ensemble de paramètres généraux de technologie. Le deuxième représente un sous-ensemble de paramètres spécifiques de technologie en fonction de la catégorie de bibliothèque.

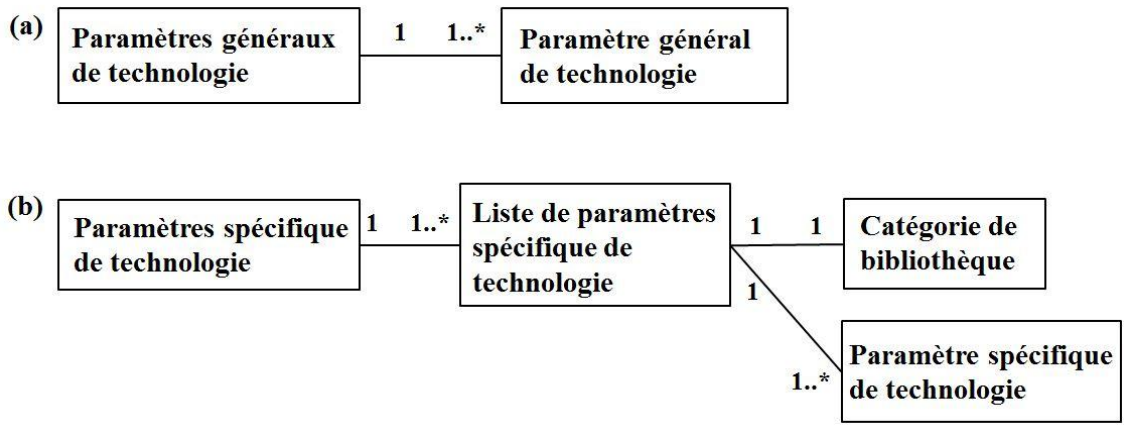

Figure 4.5 Exemples de (a) ensemble de paramètres non sous-catégorisé
(b) ensemble de paramètres sous-catégorisé

D'autre part, pour définir un paramètre, nous proposons une méthode simple et unifiée pour donner un nom et des valeurs. Il y a aussi deux sortes de paramètre selon qu'il a un attribut clé ou non. Cet attribut est utilisé chez STMicroelectronics pour identifier un paramètre au lieu du nom. La figure 4.6 donne des exemples de ces deux sortes de paramètres. Le premier permet de donner un nom et au moins une valeur pour définir un paramètre général de technologie. Le deuxième permet de donner un nom, une version comme valeur et un attribut clé pour définir l'information d'un outil.

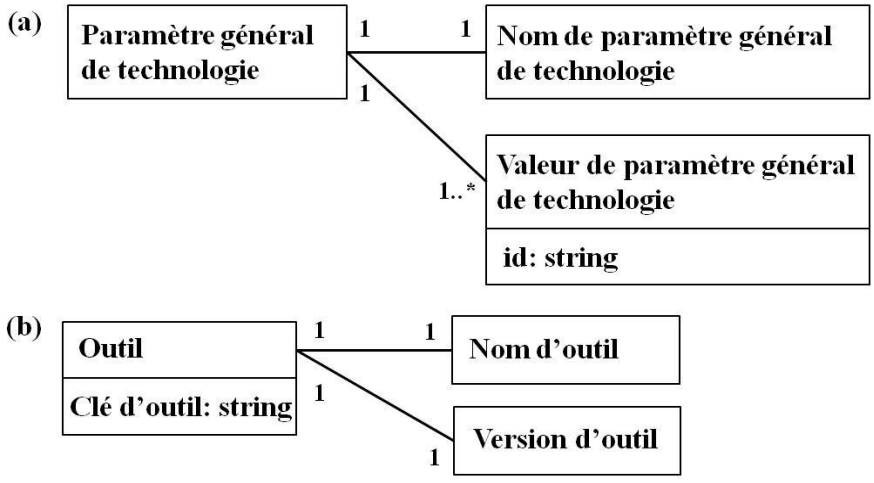

Figure 4.6 Exemples de (a) paramètre sans attribut clé (b) paramètre avec attribut clé

- **Méthode basée sur les références pour la création de la spécification**

L'objectif de cette méthode est de créer une spécification consistante et complète en fournissant autant d'informations prédéfinies que possible à partir d'une base de données de références centralisée. Nous proposons deux modèles pour ces références : le dictionnaire et la liste.

Premièrement, le dictionnaire contient une collection de termes contenant un nom et des valeurs comme le montre dans la figure 4.7. Cette référence couvre le premier type de paramètre en offrant la liste de paramètres et des valeurs possibles du paramètre.

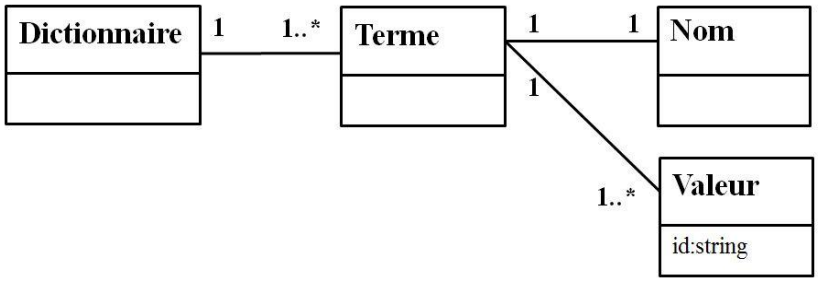

Figure 4.7 Dictionnaire

Deuxièmement, la liste vise à fournir une liste complète des paramètres avec leurs attributs clés pour aider à définir le deuxième type de paramètres comme le montre dans la figure 4.8.

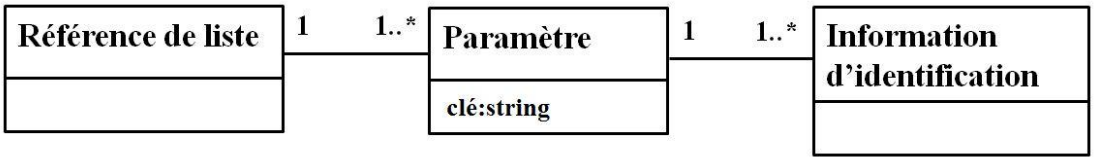

Figure 4.8 Référence de liste

La figure 4.9 illustre le flot de création des spécifications basées sur ces deux types de références. Le premier flot à droite consiste à obtenir une liste de paramètres et leurs valeurs possibles par recherche à partir du dictionnaire. Le deuxième flot à gauche consiste à obtenir une liste complète de paramètres avec leur information d'identification à partir de la liste et d'entrer la valeur du paramètre par le développeur de spécification. A la fin, toutes les données obtenues sont définies dans une spécification.

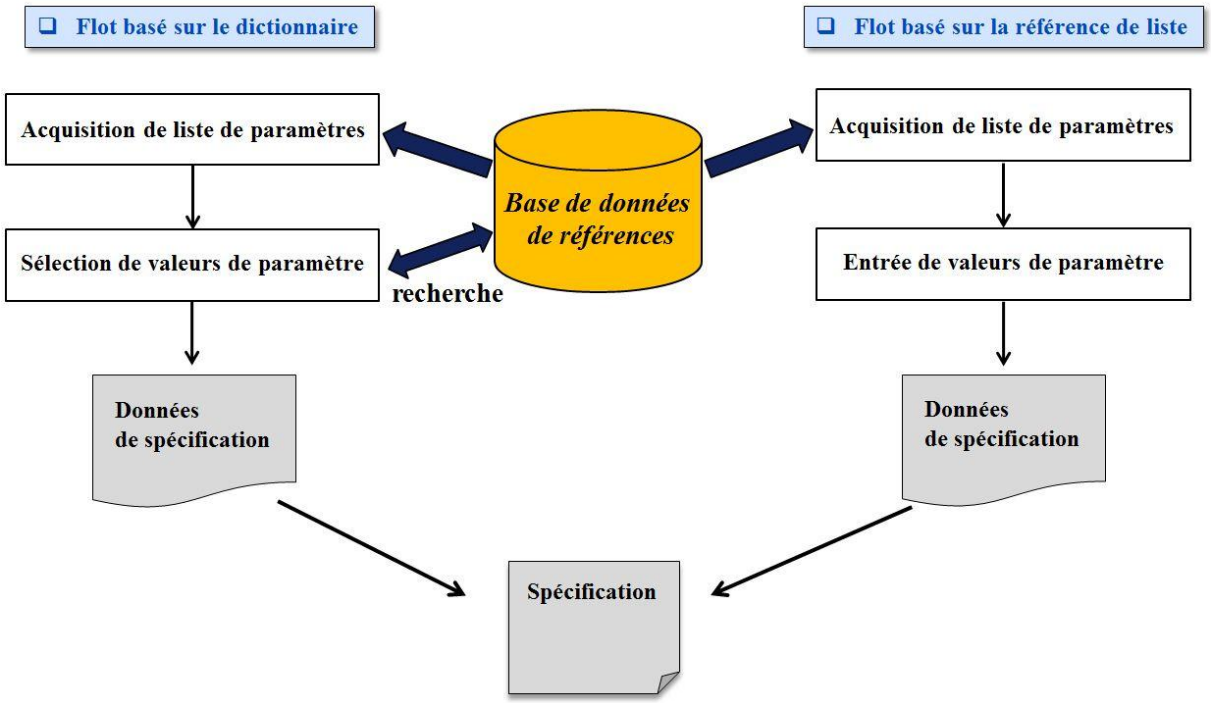

Figure 4.9 Création de la spécification basée sur les références

- **Mots-clés basés sur les tâches pour l'extraction de données**

Le développeur de bibliothèque doit exécuter des tâches pour développer une bibliothèque. Une des difficultés les plus importantes est d'obtenir des données de la spécification pour effectuer chaque tâche. Pour l'extraction de données à partir de la spécification proposée, nous proposons d'abord d'utiliser des mots-clés. Ils peuvent se représenter par un ensemble de quatre éléments : paramètre, catégorie, nom ou clé, et identificateur (id).

$$Mot - Clé = (Paramètre, Catégorie, Nom\ ou\ Clé, Id)$$

Cet ensemble d'éléments permet d'identifier un paramètre ou un ensemble de paramètres de la spécification selon les besoins de l'utilisateur. De plus, pour extraire les données nécessaires pour une tâche à accomplir, nous proposons de définir la relation entre les tâches et les données de la spécification en utilisant ces mots-clés. L'ensemble des mots-clés pour une tâche peut être exprimé par :

$$Mots - Clés_{Tâche} = \bigcup_{i=1}^{m} Mot - Clé_i$$
$$= \bigcup_{i=1}^{m} (Paramètre_i, Catégorie_i, Nom_i\ ou\ Clé_i, Id_i)$$

5. Implémentation : Plateforme de spécification

En utilisant le modèle de données présenté précédemment, nous avons développé un langage de spécification basé sur XML nommé LDSpecX [108] ainsi que la plateforme de spécification. Cette plateforme vise non seulement à créer une spécification consistante et complète avec une grande quantité de données mais également à rapidement et précisément extraire ces données selon la tâche à accomplir.

- **Spécification basée sur XML pour le développement de bibliothèque (LDSpecX)**

Le langage de spécification LDSpecX est développé en proposant un schéma XML qui représente sa syntaxe. Ce schéma est montré dans la figure 5.1. En utilisant ce langage, nous pouvons définir toutes les données nécessaires pour le développement de bibliothèques.

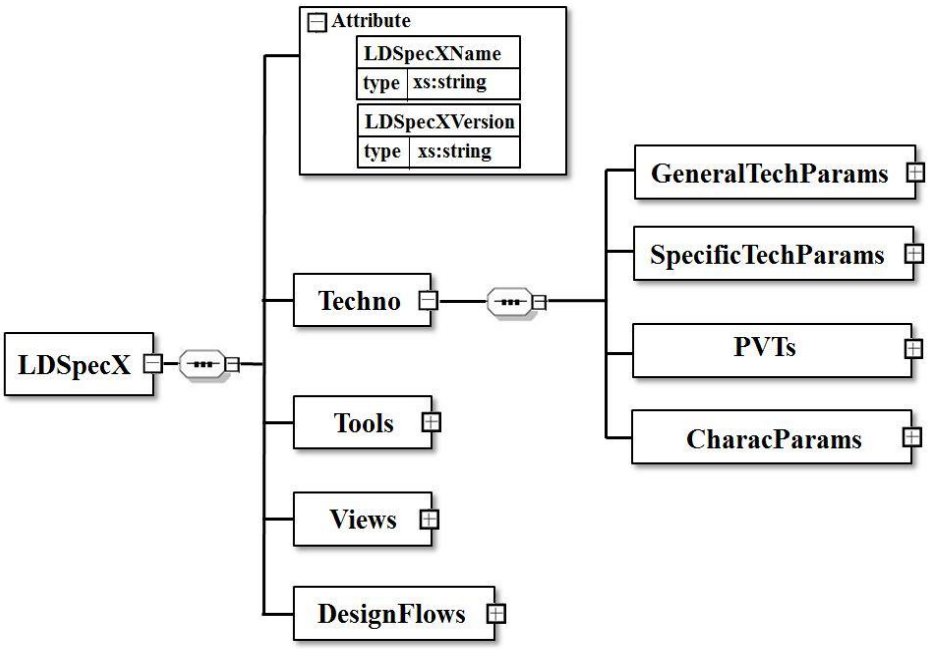

Figure 5.1 Schéma de LDSpecX

- **Création de la spécification**

Nous avons proposé une base de données de références auparavant. Cette base de données est construite en créant un dictionnaire et deux listes de référence pour les outils et les vues. A l'aide de cette base de données, nous avons développé un outil pour la création de la spécification [115]. En utilisant cet outil, une spécification en LDSpecX peut être créée en suivant le flot donné dans la figure 5.2. D'abord, la spécification est divisée en plusieurs

morceaux nommés fragments pour que les données d'une même catégorie soient collectées indépendamment. Le flot est divisé en deux étapes. La première étape est de créer un fragment de la spécification et de vérifier qu'il contient tous les paramètres. Cette étape doit être répétée jusqu'à ce que tous les fragments soient créés. Après la création des fragments, nous pouvons passer à la deuxième étape consistent à les combiner afin de produire une spécification finale. A la fin de cette étape, nous vérifions que le résultat est complet, consistant et correct. De plus, la syntaxe de la spécification est également vérifiée.

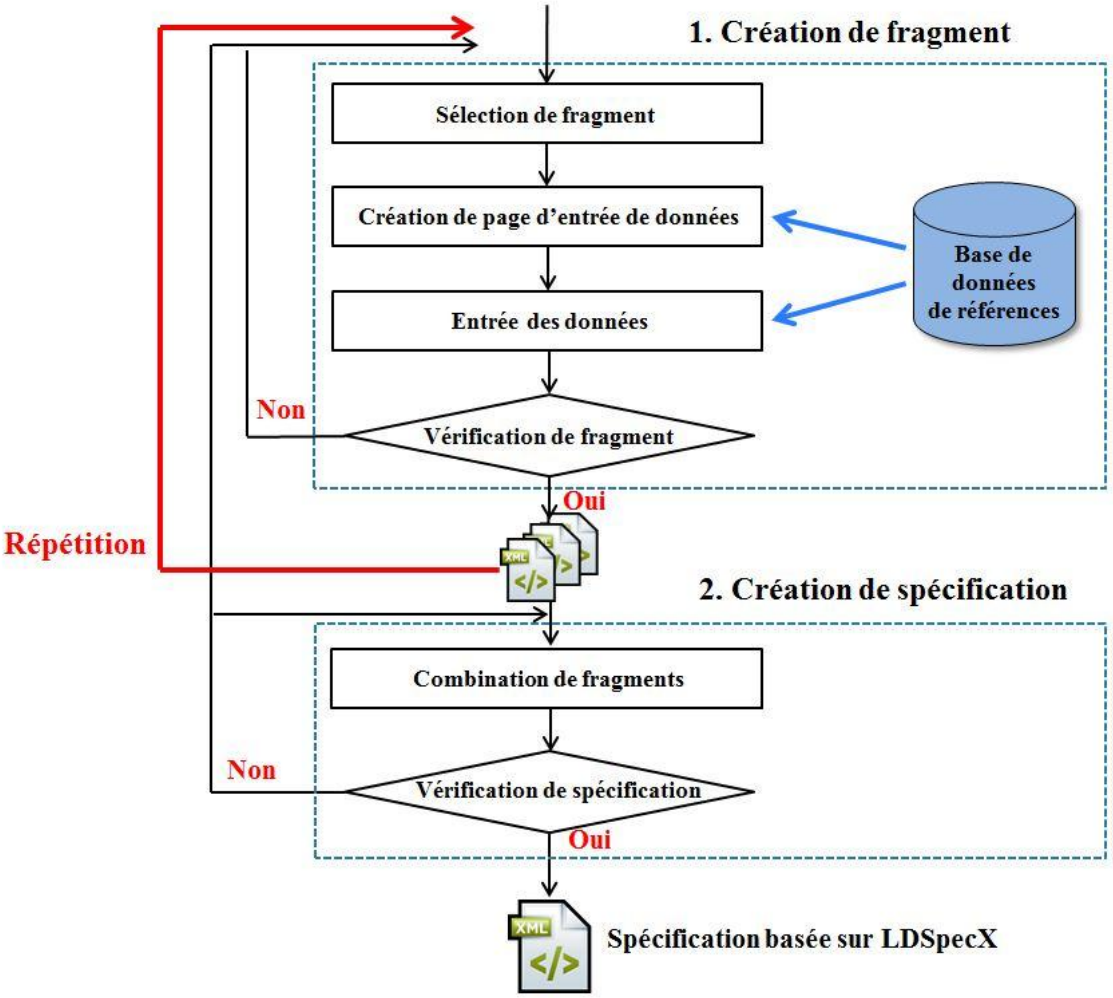

Figure 5.2 Flot de création de spécification basée sur les références

Nous avons développé une interface graphique (*Graphic User Interface* GUI) en Python qui permet d'accéder à la base de données de références pour obtenir les informations prédéfinies. La figure 5.3 montre la page de saisie des données pour créer un fragment sur des paramètres généraux de technologie à l'aide du dictionnaire. Premièrement, une liste de paramètres est obtenue à partir du dictionnaire pour la faire apparaître sur la page de saisie des

données. Deuxièmement, les valeurs de paramètres doivent être définies soit en les obtenant directement à partir du dictionnaire, soit en les entrant manuellement.

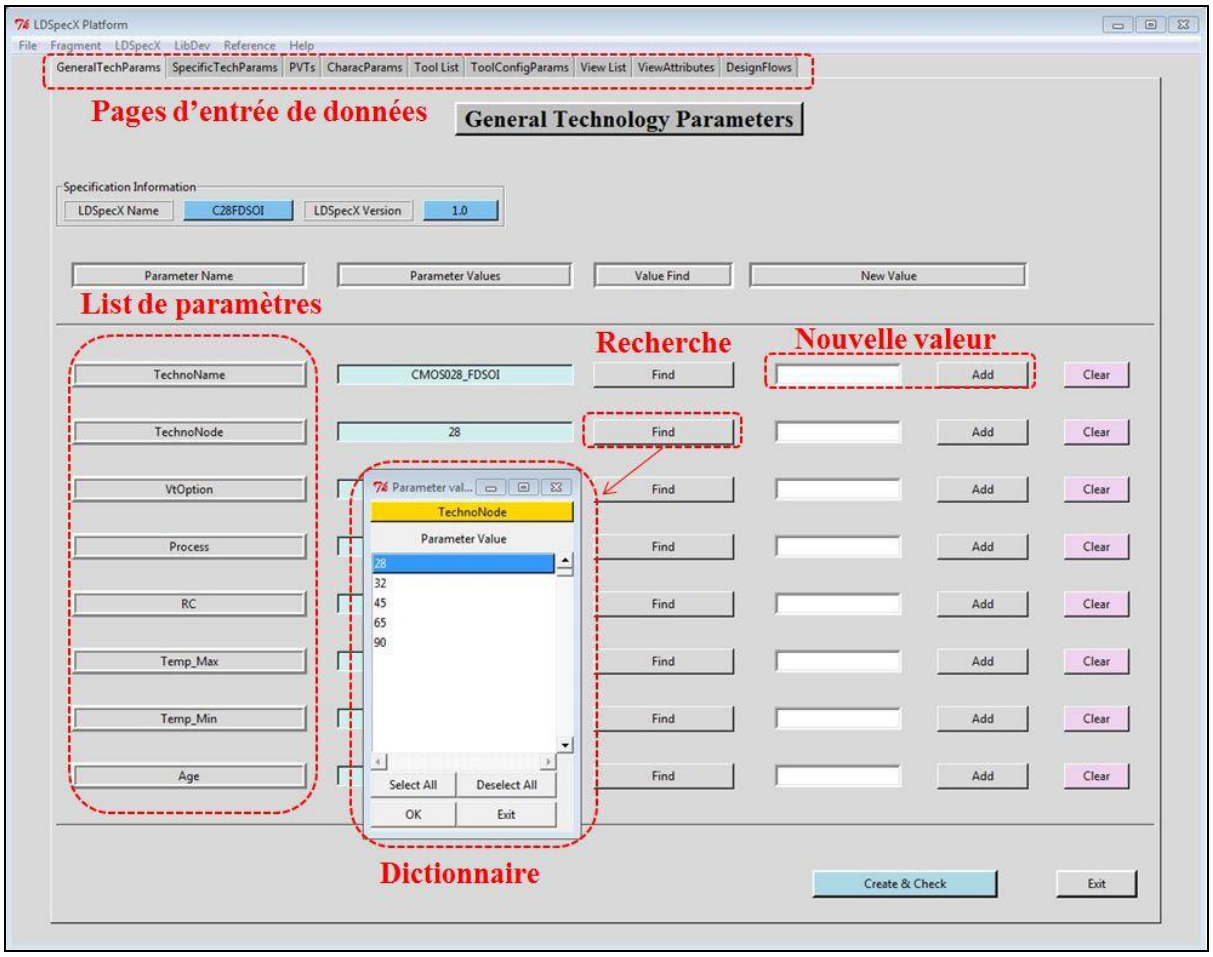

Figure 5.3 Page de saisie des données pour des paramètres généraux de technologie

La figure 5.4 montre la page de saisie des données pour définir l'information liée aux outils à l'aide d'une liste de référence pour les outils. Ceci fournit une liste complète des outils nécessaires avec leur clé et nom. C'est pour cela que le développeur de spécification peut facilement définir toute l'information liée aux outils.

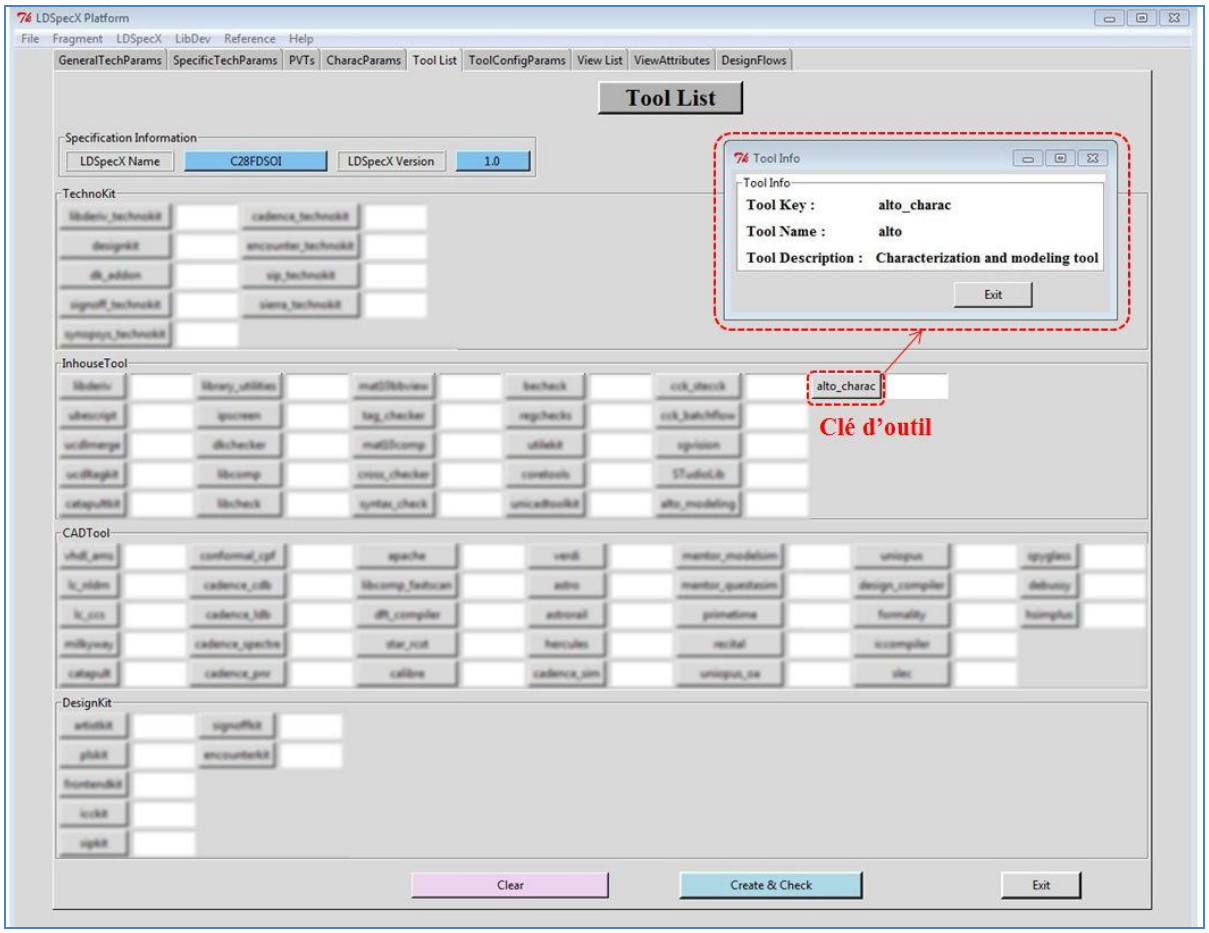

Figure 5.4 Page de saisie de données pour l'information des outils

La figure 5.5 montre la page de création de la spécification en LDSpecX. Tous les fragments obtenus à partir des étapes précédentes doivent être combinés pour générer une spécification en LDSpecX.

Comme décrit précédemment, cette plateforme fournit aux développeurs de spécification une base de données unique de références ainsi qu'un flot unifié. En conséquence, elle peut aider à créer une spécification complète et consistante.

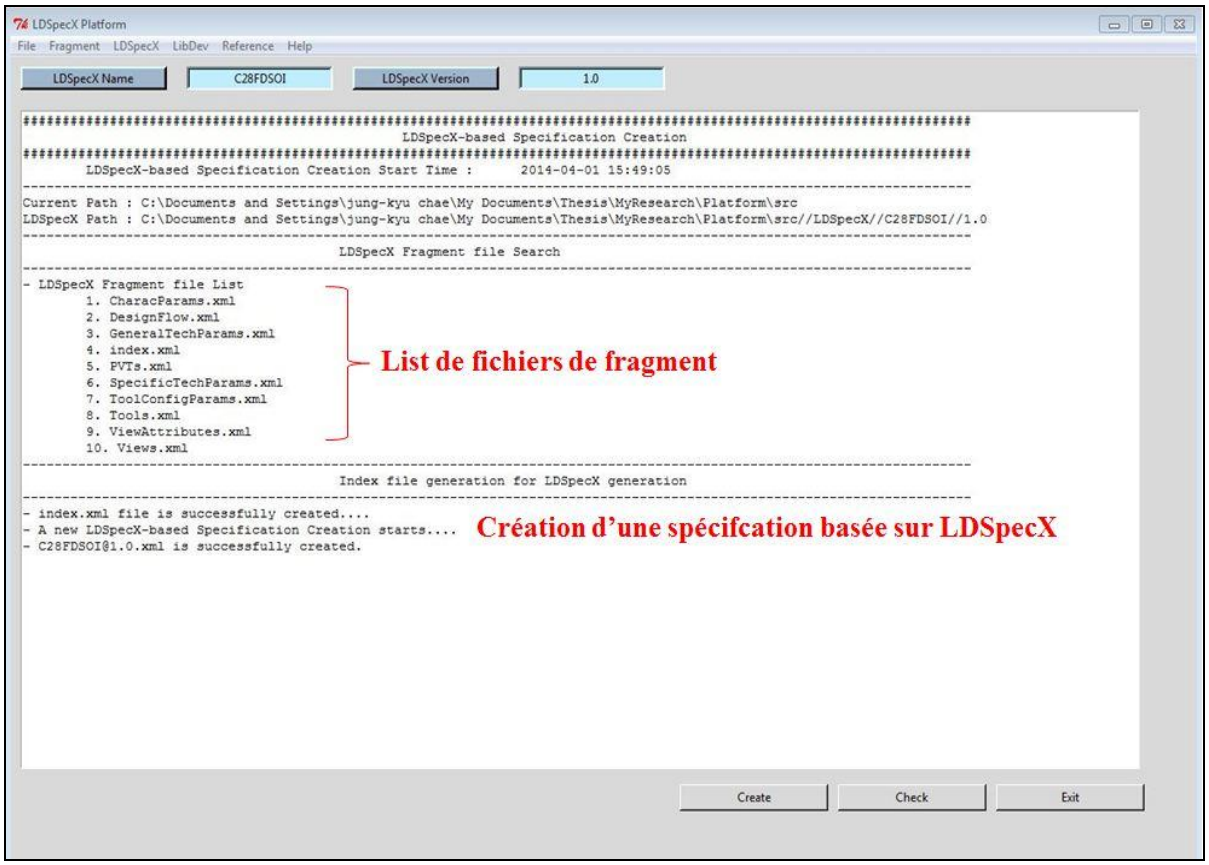

Figure 5.5 Page de création de spécification en LDSpecX

- **Extraction des données de la spécification**

Nous avons développé une interface de programmation d'applications (*Application Programming Interface* API) en Perl basée sur la méthode d'extraction de données vue précédemment. D'abord, nous devons définir la relation entre les tâches et les données de spécification en utilisant des mots-clés. Cependant, les données de spécification nécessaires dépendent de la catégorie de bibliothèque. Pour cette raison, il faut définir les tâches pour le développement de bibliothèques pour chaque catégorie de bibliothèque séparément comme le montre dans la figure 5.6 (a). Chaque tâche peut avoir plus d'un sélecteur qui contient un mot-clé pour faire le lien avec les paramètres de spécification comme l'illustre la figure 5.6 (b). En utilisant ce schéma, des mots-clés pour toutes les tâches du flot de développement de bibliothèques ont été prédéfinis et stockés dans un fichier XML.

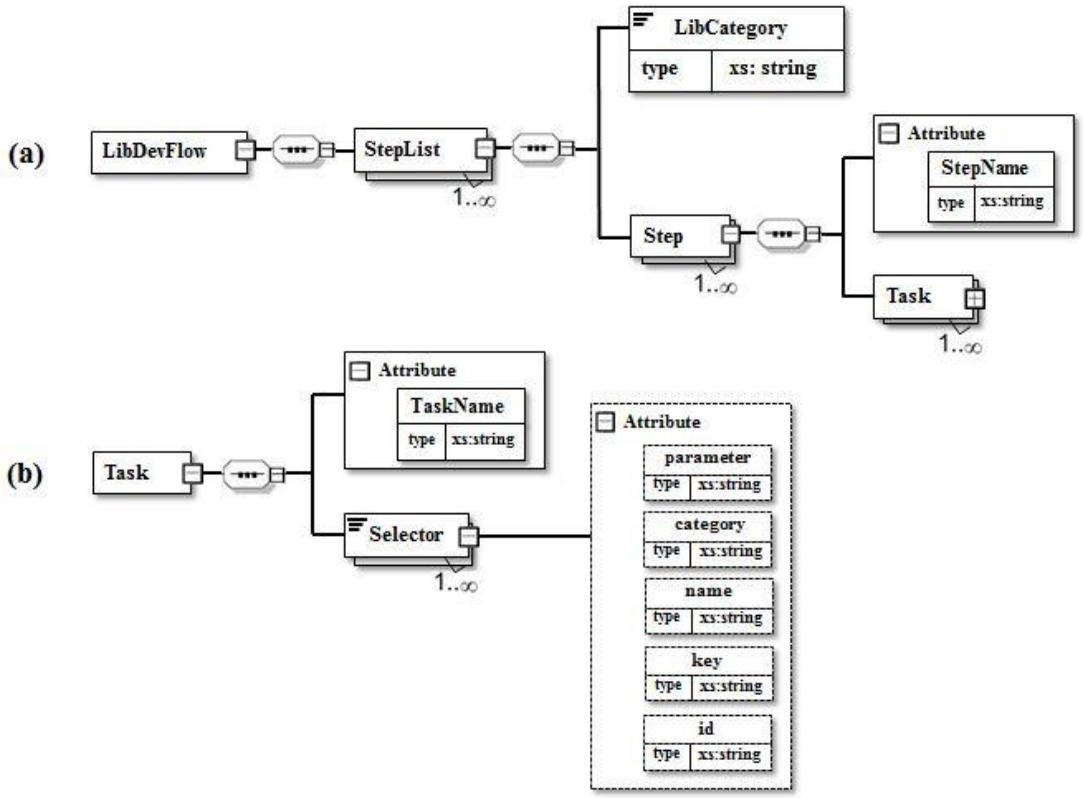

Figure 5.6 (a) Flot de développement de bibliothèques (b) Tâche

Après la création du flot, nous avons développé une API qui permet d'accéder à la base de données de LDSpecX comme l'illustre la figure 5.7.

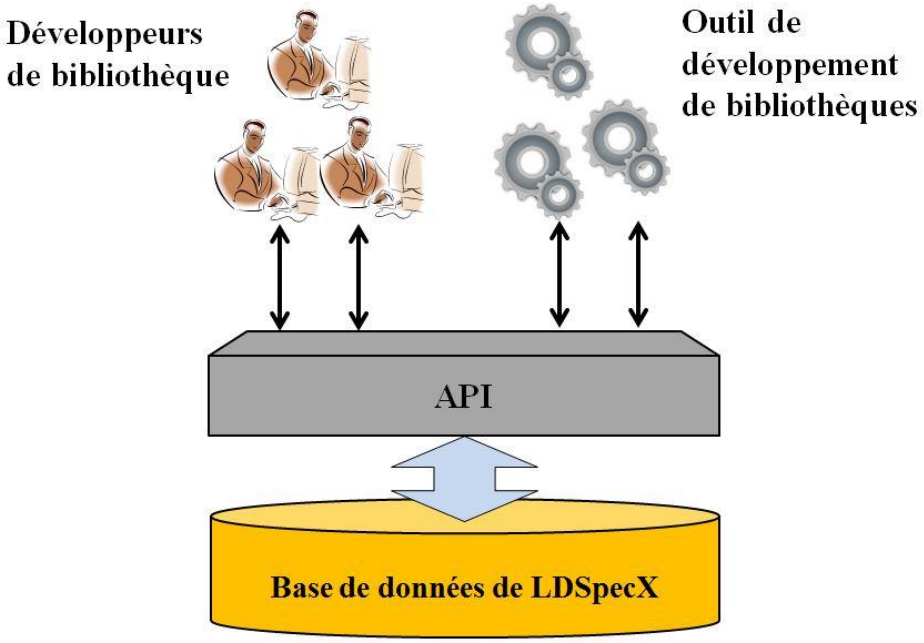

Figure 5.7 API

Cette API fournit aux utilisateurs des fonctions pour extraire des données. Les deux fonctions fondamentales sont :

1) **getSpecData** (spécification, mot-clé): renvoie les données obtenues à partir de la spécification basée sur LDSpecX (fichier XML) utilisant un mot-clé donné (chaîne de caractères).
2) **getFlowInfo** (catégorie de bibliothèque, tâche): renvoie les mots-clés à partir du flot de développement de bibliothèques (fichier XML) selon la catégorie de bibliothèque et la tâche (chaînes de caractères).

De plus, en utilisant ces fonctions, une fonction permettant d'extraire des données selon la tâche désirée est également fournie :

3) **getSpecDataViaFlow** (spécification, catégorie de bibliothèque, tâche): renvoie toutes les données nécessaires à partir de la spécification basée sur LDSpecX (fichier XML) pour une tâche désirée (chaîne de caractères).

La figure 5.8 montre un flot pour l'extraction de données en utilisant des mots-clés basés sur les tâches. Premièrement, les mots-clés sont collectés à partir du fichier de flot. Deuxièmement, en utilisant ces mots-clés, les données peuvent être obtenues à partir de la base de données de spécifications.

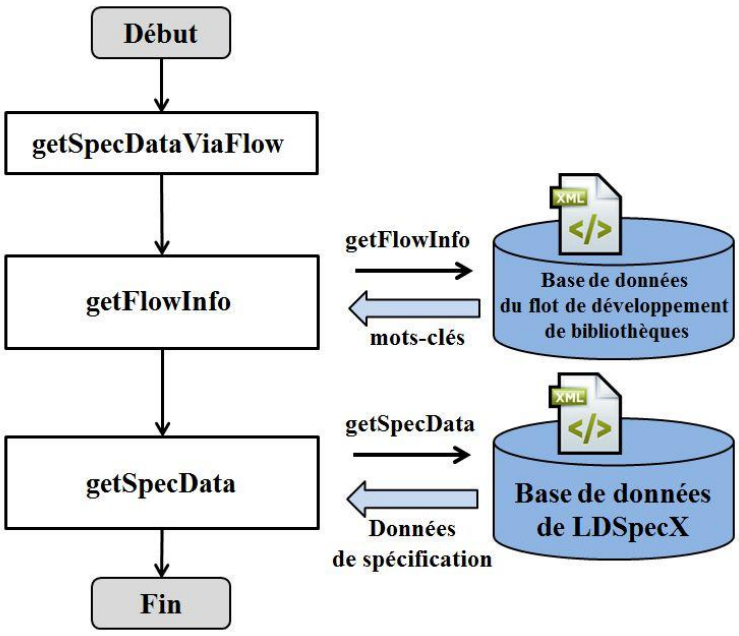

Figure 5.8 Extraction de données en utilisant des mots-clés basés sur les tâches

En utilisant les fonctions de l'API, nous avons développé des outils pour extraire des données automatiquement et un outil de vérification de la bibliothèque par rapport à la spécification.

6. Expérimentation

Expérimentalement, nous avons effectué le développement d'une bibliothèque de cellules standard (portes logiques) à partir de la création d'une spécification jusqu'à la validation de la bibliothèque en utilisant la plateforme de spécification présenté précédemment.

Premièrement, nous avons créé, à l'aide du GUI, une spécification en technologie 28nm CMOS FDSOI pour la bibliothèque de cellules standard avec une réduction considérable du temps de développement. Le tableau 6.1 donne un comparatif de la de spécification basée sur LDSpecX avec les spécifications traditionnelles.

Table 6.1: Comparatif de la spécification basée sur LDSpecX avec les spécifications traditionnelles

Catégorie d'information	Paramètre		Nombre de paramètres	
			Spécifications traditionnelles	LDSpecX
Technologie	Paramètre général de technologie		6	9
	Paramètre spécifique de technologie		6	6
	PVT		20	20
	Paramètres de caractérisation	Référence de caractérisation	-	13
		Pente	240	240
		Charge capacitive	1136	1136
Outils	Outil		73	76
	Paramètre de configuration de l'outil		-	449
Vues	Vue		154	154
	Attribut de la vue		10	10
Flots de conception	Flot de conception du SoC		3	3

Nombre total de paramètres	1648	2116
Nombre de fichiers	2 excel fichiers (5 feuilles), 3 word fichiers	1 xml fichier
Temps de création	2 heures	42 minutes

Deuxièmement, à partir de cette spécification, nous avons développé une bibliothèque contenant 11 cellules combinatoires et séquentielles. Les tâches nécessaires ont été exécutées en utilisant les outils internes et l'API. Cette API permet d'extraire les données nécessaires beaucoup plus rapidement qu'avec la méthode traditionnelle (semi-manuelle). En conséquence, 86 vues ont été produites. Finalement, la bibliothèque obtenue a été vérifiée par l'outil de vérification développé à l'aide de l'API.

Enfin, cette plateforme aide à réduire le gap entre la spécification et le système automatique actuel pour le développement rapide de bibliothèques.

7. Conclusion et perspectives

Dans cette thèse, nous avons proposé des méthodologies, des flots et des outils pour formaliser les spécifications pour le développement de bibliothèques et la traiter efficacement. Cette spécification vise à être utilisée comme une référence pour générer et valider des bibliothèques (e.g. cellules standard, cellules d'entrée/sortie, mémoires, etc.) ainsi que des IPs complexes (PLL, etc.)

Nous avons proposé une spécification unifié basée sur XML nommée LDSpecX. Ce langage de spécification permet d'encapsuler toutes les informations nécessaires pour le développement de bibliothèques de cellules et d'IPs. En utilisant LDSpecX, nous avons également développé une plateforme de spécification. Cette plateforme nous permet de créer une spécification consistante et complète en LDSpecX par un flot unifié et de précisément et rapidement extraire les données nécessaires à partir de la spécification selon une tâche désirée à l'aide des GUI et API respectivement. La figure 7.1 illustre la vue d'ensemble de la plateforme de spécification.

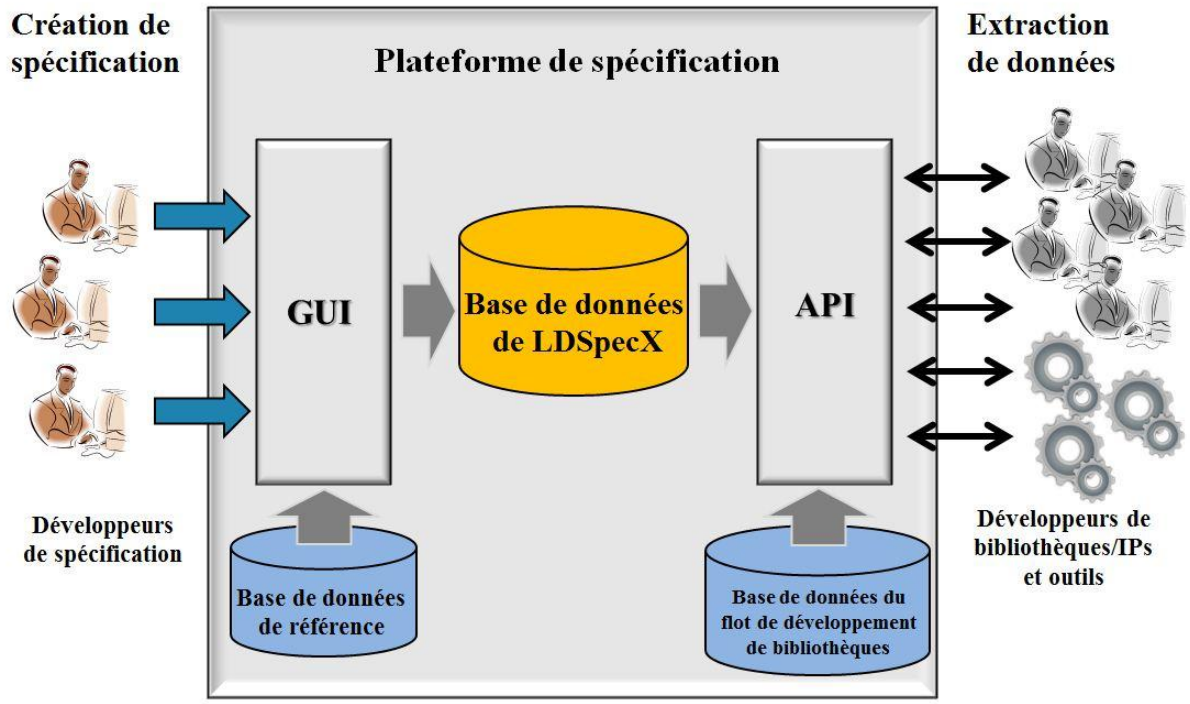

Figure 7.1 Plateforme de spécification

Une principale perspective de recherche qui apparait à l'issue de cette thèse concerne l'extension à la configuration automatique des outils pour couvrir le développement de toutes les catégories de bibliothèques en plus des bibliothèques de cellules standard. En outre, la traçabilité de la spécification serait efficace pour donner l'information détaillée sur le changement des données. En utilisant les mots-clés basés sur les tâches qui définissent la relation entre les tâches et les données de la spécification, nous pouvons facilement reconnaître des tâches liées au changement de données. Cela permettrait d'effectuer uniquement les tâches concernées pour produire une bibliothèque avec une nouvelle version de la spécification. Par conséquence, ces perspectives de recherches pourraient considérablement aider à améliorer la productivité du développement de bibliothèques. De plus, la transformation des spécifications traditionnelles existantes en spécification basée sur LDSpecX serait nécessaire du point de vue du développeur de spécification pour les réutiliser. La spécification proposée et sa plateforme peuvent aussi être étendues pour la conception de systèmes en couvrant les informations additionnelles comme, par exemple, la liste de bibliothèques et d'IPs. Ce travail futur peut faciliter les développeurs de bibliothèque/IP ainsi que les concepteurs de systèmes pour obtenir les données nécessaires à partir de la spécification.