Dipl.-Ing. Christoph Trummer

Automated Simulation-Based Verification of Power Requirements for System-on-Chip Designs

---

Dissertation

vorgelegt an der
Technischen Universität Graz

zur Erlangung des akademischen Grades
Doktor der Technischen Wissenschaften
(Dr.techn.)

durchgeführt am Institut für Technische Informatik
Technische Universität Graz
Vorstand: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Reinhold Weiß

Graz, im Jänner 2010

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz, am ....
(Unterschrift)

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

....
date (signature)

Kurzfassung

Moderne Elektrogeräte werden oft als "energiesparend" angepriesen oder mit begrenzten Energiequellen betrieben. Neben Anforderungen, welche die Funktionalität betreffen, werden auch nicht-funktionale Anforderungen immer wichtiger. Aufgrund dessen ist die Leistungsaufnahme die wichtigste nicht-funktionale Anforderung. In einem zeitgemässen Elektrogerät ist die gesamte Funktionalität oft auf einem einzigen Chip konzentriert. Entwirft man nun ein solches "System-on-Chip" (SoC), so werden, wegen des hohen Grads an Komplexität oft vorgefertigte Komponenten und Teile wiederverwendet, um sich auf die wesentliche Funktionalität konzentrieren zu können. Da die kritischste Einschränkung des Designs deren Leistungsaufnahme ist wird Power Aware Design benötigt. Das Power Aware Design erweitert das logische SoC Design um die Architektur des Versorgungsnetzwerks. Das logische Design interagiert mit dem Power Aware Design um die Leistungsaufnahme während des Betriebs zu reduzieren. Allerdings wird der Aspekt des Power Aware Design oft übersehen und dem SoC Design erst sehr spät im Designprozess hinzugefügt. Dies führt zu weiterer Komplexität in der Verifikation, welche überprüft, ob das Design den Anforderungen entspricht. Ebenso können kritische Fehler im Power Aware Design erst sehr spät entdeckt werden, was eine zeit- und kostenintensive Überarbeitung erfordert.

Das Augenmerk dieser Arbeit liegt auf der Spezifikation und Verifikation von nicht-funktionalen Anforderungen betreffend der Leistungsaufnahme, den sogenannten "power requirements". Diese umfassen Batterielebensdauer, Schranken für die Leistungsaufnahme und Interaktion mit dem Power Aware Design. Sie werden zusammen mit den funktionalen Anforderungen in semi-formalen "use cases" spezifiziert. Die "use cases" werden anschließend automatisch analysiert, um daraus Testfälle abzuleiten. Während der Simulation werden die Testfälle dem System auferlegt, was darin ein entsprechendes Verhalten auslöst. Die korrespondierende Leistungsaufnahme wird abgeschätzt und zur Verifikation der Batterielebensdauer und der Schranken für die Leistungsaufnahme verwendet. Das separat beschriebene Power Aware Design wird in eine ausführbare Form umgewandelt. Es wird gemeinsam mit dem SoC Design simuliert und daraufhin gegen die Anforderungen verifiziert. Die Designkomponenten des Systems, dessen Power Aware Design, Ergebnisses aus Verifikation und Leistungsabschätzung werden in ein Projekt zusammengefasst und in einem standardisierten Format für "Intellectual Property" (IP) gespeichert. Schließlich wird das Projekt in einer IP Bibliothek für zukünfige Verwendung abgelegt.

Die vorgeschlagene Methodik wurde in einer Fallstudie an higher-class Radio Frequency Identification (RFID) tags, die eine häufige Anwendung für Systems-on-Chips darstellen, erfolgreich angewandt.

Abstract

Modern electronic devices are often advertised as "energy saving" or are powered by limited energy sources. Besides their functional requirements the non-functional requirements are also becoming increasingly important. Power dissipation is the most important non-functional requirement. Inside electronic devices the entire functionality is often concentrated on a single chip. When designing such a System-on-Chip (SoC), the high complexity forces designers to reuse pre-designed components and to focus on the main functionality. Since the most important design constraint is power dissipation, power aware design is necessary. Power aware design extends the logic SoC design with the architecture of the power supply network. The logic design interacts with the power aware design to dynamically reduce power consumption. However, the aspect of power aware design is often overlooked. It is often added to the SoC during late stages of the design which causes additional complexity for verifying that the SoC design fulfills its requirements. Serious defects related to the power aware design may be exposed at late stages necessitating time-consuming, costly redesign.

This work focuses on specification and verification of non-functional requirements concerning power dissipation, the so-called power requirements. Power requirements concern battery lifetime, constraints for power dissipation and interaction with the power aware design. They are specified in addition to functional requirements in semi-formal use cases. Use cases are analyzed automatically from which test cases are derived. During simulation the test cases are applied to the system which stimulates a corresponding behavior. The associated power dissipation is estimated to verify the battery lifetime and power constraints. The separate power aware design is translated into an executable supply network. It can be simulated with the SoC design and is verified for conformance to the requirements.

The system's components, the power aware design, results of verification and power estimation are stored in a standardized IP format and added to the IP library for later reuse. The proposed methodology is successfully evaluated in a case study of a higher-class Radio Frequency Identification (RFID) tag, a common application for Systems-on-Chips.

Acknowledgements

First of all, I would like to express gratitude to my supervisor Prof. Dr. Reinhold Weiß for his insights, helpful advice and for providing me with the facilities to conduct my research for this dissertation. I especially want to thank our project leader Dr. Christian Steger for his extraordinary support, skillful guidance and great inspiration during my work at the Institute for Technical Informatics at Graz University of Technology. Moreover, I am grateful to Dr. Damian Dalton for his beneficial feedback, insightful comments and friendly support.

This thesis would not have been possible without the funding from the FFG contract 812424 of the Austrian Federal Ministry for Transport, Innovation, and Technology. I am particularly grateful to Dr. Markus Pistauer, DI Andreas Schuhai and DI Peter Lederer from CISC Semiconductor Design+Consulting GmbH for their knowledge, support and stimulating feedback. Also, I would like to express my appreciation to the people at Neosera Systems Ltd. for contributing the RHEiMS framework.

My project college Dr. Christoph Michael Kirchsteiger I thank for his assistance, interesting discussions and useful hints. I am also indebted to my students for their valuable contributions to this dissertation: Christoph Ruggenthaler, Patrick Pauše, Peter Randeu, Patrick Schrey, Michael Lackner and Matthias Schröfelbauer. Furthermore, my thanks extend to the entire staff at the Institute for Technical Informatics for their support, constructive comments and helpful suggestions during my work.

I thank my family, especially my parents and grandparents for their ongoing support and motivation.

My deepest gratitude goes to my fiancé Anna Helena whose patient love, understanding and continuous encouragements enabled me to complete this work.

Graz, January 2010 Christoph Trummer

Extended Summary

Systems-on-Chips (SoCs) are used in a wide variety of today's electronic appliances. Most commonly they are found in mobile, battery-powered devices. A System-on-Chip (SoC) consists of many components with different functionality, all integrated on a single chip. Advances in chip fabrication enable the SoCs to become very small. This means that more functionality fits onto an SoC. This development is most evident in mobile phones. A few years ago they were bulky devices with monochrome displays. Today's mobile phones are equipped with high-resolution color touch-screens, integrated digital cameras, music players and Internet connection. However, the increase in functionality causes specification, design and especially verification of SoCs to become highly complex.

Traditionally, the SoC development cycle starts with specification of the requirements. Then the SoC is designed in a Hardware Description Language (HDL). This HDL prototype can be simulated to verify its functionality. After the design has undergone several iterations and has been verified thoroughly the chip is manufactured.

In the specification phase the concept of the SoC's desired functionality is described in the requirements document. Additionally, parameters, conditions and constraints are defined, under which the SoC's functionality has to be performed. These are the non-functional requirements. The most important non-functional requirements for SoCs are power requirements. Power requirements describe conditions and limits concerning the SoC's power dissipation. Since most SoCs are battery-powered, the battery's lifetime is such a power requirement. In order to fulfill the battery lifetime requirement, the SoC has to operate with limited power. These power constraints are also part of the power requirements. To reduce power consumption, different power states are defined. During a power state unneeded parts of the SoC are switched off and on again when needed. Therefore, power states are also part of the power requirements.

In common with all non-functional requirements, power requirements are difficult to determine and to specify. When the SoC is initially specified no details about its power dissipation are known. Furthermore, there is no format which assists in their specification. Consequently, despite their importance, power requirements are often neglected.

After a HDL model of the system is produced it is verified to ensure that the requirements are satisfied. The numerous requirements for SoCs increase the verification effort. The more detailed the SoC design becomes, the longer its verification takes. If power requirements are not specified early on implementation of the power aware design materialize in later stages of the SoC design. Verification can only proceed at an even later stage. Extensive redesign following the late discovery of critical power-related design errors is very costly. Estimating the SoC's power demand exposes thermal issues which cause system instability. When they are addressed late in the design process, large and expensive cooling solutions are needed. Late selection of the battery negatively influences size, weight and costs of the device.

This work is part of the SIMBA project which aims to reduce the complexity of verification. Therefore, functional and non-functional requirements are tied to each step of the SoC design process. After specification the informally described requirements are refined into use cases to avoid ambiguity. A use case document is a more formal, unified description of the requirements. At each design iteration, first the use case document and then the system model is refined and detail is gradually added. The use case document is automatically analyzed to generate test cases for simulation-based verification of the system.

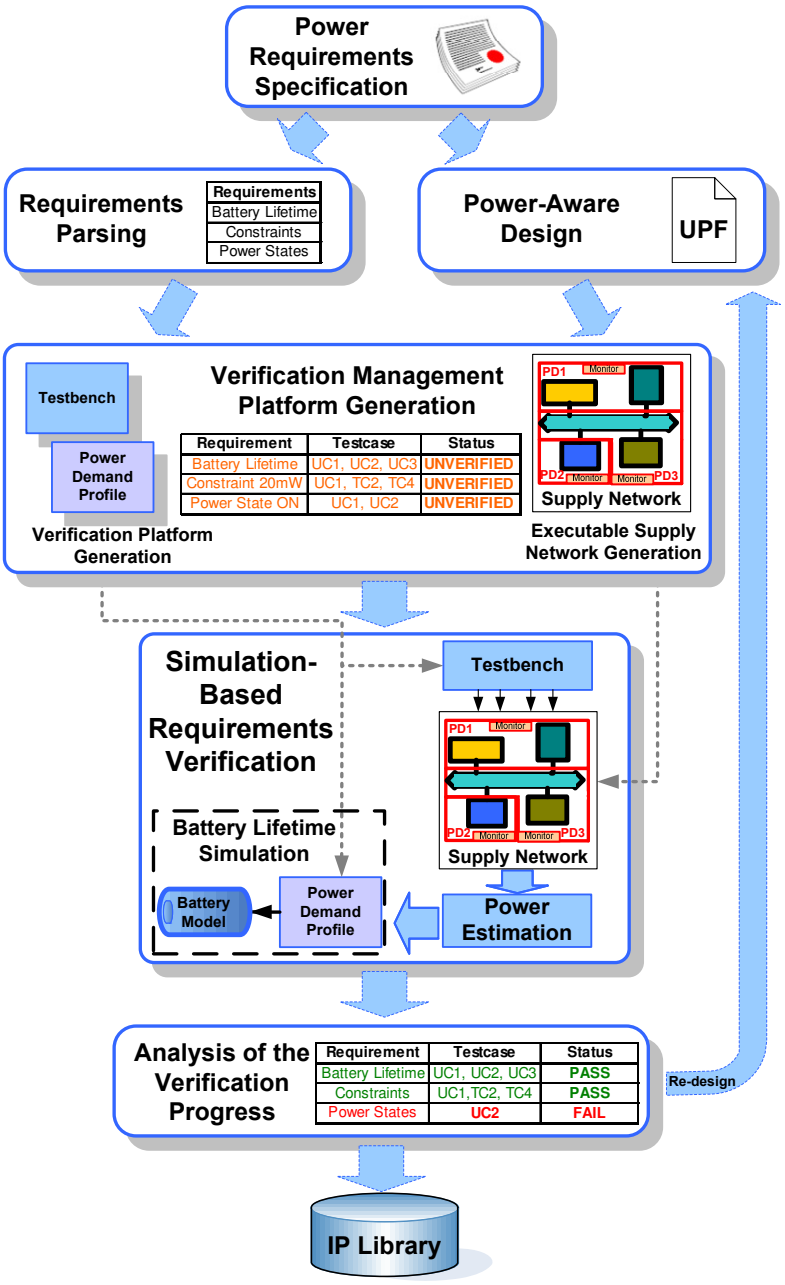

Figure 1: Methodology for Simulation-Based Verification of Power Requirements

This work describes a methodology to verify the SoC's most important non-functional requirements, the power requirements. The methodology is illustrated in Fig. 1. During specification, the highest level power requirements express the battery lifetime for the system. From the battery lifetime other power constraints are derived. These power constraints are imposed on the SoC's functionality to avoid thermal instability and to meet the specified battery lifetime. Fulfilling the power constraints necessitates power states to reduce the system's power dissipation. In addition to the functional description, each use case is extended with power constraints and power state information. Similarly, battery lifetime is specified for a series of use cases describing an application of the SoC. The power requirements are expressed in XML or via IBM DOORS®, a common specification tool [1].

In the design phase the system model is created. Some parts of the system are newly created while others are reused from existing components in a library. In parallel, the power design is created which implements power states and describes the architecture of the system's power supply. The Unified Power Format (UPF), which has recently become the IEEE standard 1801, describes the system's power aware design independent of the HDL. When the SoC design is available, verification begins.

The use case document is automatically analyzed and test cases are generated. During simulation they are used to verify the functional and non-functional requirements. Each use case corresponds to several test cases. When such a test case is launched a stimulus is sent to the SoC model. The model responds by performing the appropriate operation. The system's behavior corresponds to a certain amount of power being dissipated. The Rapid Hierarchical Energy Investigation Modeling System (RHEiMS) is used to estimate the SoC's energy dissipation at system level. After deriving the corresponding power for the SoC's functionality it can be verified against the power constraints[1]. Moreover, the system's power state is determined and compared to the specification[2]. Simulating all test cases verifies that the system properly enters and leaves the power states and that the power constraints are met. After the power dissipation has been determined for all use cases the battery lifetime can be ascertained. A battery model is connected to the SoC's power profile and its lifetime is calculated[3]. When a power constraint is violated, a mistake in the power states is detected, or the required battery lifetime is not met, the SoC design needs to be corrected and re-verified. After all functional and non-functional requirements are fulfilled the design can be further refined. The model or parts of it can be stored in the IP library. This Intellectual Property (IP) can be reused in other designs[4].

The benefit of our methodology is that it provides a unified use case format to specify both power requirements and functional requirements. This allows the creation and verification of the power design during early stages of the SoC design. Our methodology includes fast and accurate power estimation to detect peaks in power dissipation which

[1] Specification and Automated Simulation-based Verification of Power Requirements for System-on-Chips, Joint IEEE Circuits and Systems and TAISA Conference 2009, NEWCAS-TAISA '09, Toulouse, France, June/July 2009

[2] Simulation-based Verification of Power Aware System-on-Chip Designs Using IEEE 1801, IEEE NORCHIP Conference, Trondheim, Norway, November 2009

[3] Verification Methodology for Battery Lifetime Requirements of Higher Class UHF RFID Tags, IEEE International Conference on RFID 2009, Orlando, USA, April 2009

[4] A Component Selection Methodology for IP Reuse in the Design of Power-Aware SoCs Based on Requirements Similarity, 3rd Annual IEEE International Systems Conference, Vancouver, Canada, March 2009

helps to avoid thermal instability. Early determination of the battery lifetime supports the proper battery selection and reduces cost. Automatic generation of test cases greatly decreases the verification effort. In the developed IP library, system components are stored with their use case document, verification environment and verification results. Within the library, IP can be efficiently searched based on its functionality for reuse in other SoC designs. The included verification environment decreases verification effort for future SoC designs which reuse the IP.

In conclusion, this dissertation introduces power requirements into the SIMBA use case format and verification methodology. To reduce complexity, the verification environment is automatically generated to ascertain the system's battery lifetime, power constraints and power states during simulation. The essential contributions are: the extended use case format for power requirements, the automated verification of power requirements within a single methodology and an IP library for functionality-based component search.

Contents

List of Figures

List of Tables

List of Abbreviations

EDA	Electronic Design Automation
FPGA	Field Programmable Gate Array
ESL	Electronic System Level
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronics Engineering
IP	Intellectual Property
MPSoC	Multiple-Processor System-on-Chip
NoC	Network-on-Chip
RFID	Radio Frequency Identification
RHEiMS	Rapid Hierarchical Energy Investigation Modeling System
ROM	Read Only Memory
RTL	Register-Transfer-Level
SoC	System-on-Chip
SyAD	System Architect Designer
TL	Transaction-Level
TLM	Transaction Level Modeling
UML	Unified Modeling Language
UPF	Unified Power Format
VIP	Verification Intellectual Property
XML	Extensible Markup Language

Chapter 1

Introduction to Simulation-Based Verification of Power Requirements for SoC Designs

Advances in semiconductor fabrication technology have enabled high silicon integration densities and increasing chip performance. More functionality now fits onto less chip area. Consequently, an entire system is integrated on a single chip, a so-called System-on-Chip (SoC). Although, SoCs are found in all kinds of consumer electronics they are especially common in portable electronic devices. Typical examples are mobile phones, MP3 players, wireless sensor nodes (WSNs) and active Radio Frequency Identification (RFID) tags.

1.1 Motivation

Before designing an SoC, its functionality is specified in the requirements. However, the increase in SoC functionality leads to an enormous number of requirements. This causes specification of the requirements to be very complex. Subsequently, the requirements are implemented in a logical description of the SoC design. The more functionality the SoC has to perform, the more complex its logical design becomes. To counter complexity of the SoC design different levels of design abstraction (Fig. 1.1) and reuse of existing design components are employed.

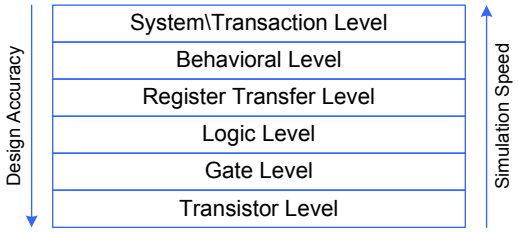

Figure 1.1: Levels of Design Abstraction [1]

The increase in functionality also causes higher power consumption [3]. Power aware design is necessary to reduce both the SoC's power consumption and energy dissipation

since it has to operate with a limited power source (i.e. a battery). However, the power aware design also contributes to the overall complexity of SoC design.

Verification ensures, that the SoC design fulfills its requirements and performs as specified. This also applies to the power aware design. Power estimation is fundamental in verifying that the SoC is able to operate with a limited amount of power. The verification process may use as much as 70 % of the entire design resources [4].

Under these premises the following subsections explain the motivation for this work.

1.1.1 Deficiencies in Specification of Power Requirements

Generally, requirements describe the user's needs of a system. The requirements describe the SoC's purpose, objectives, capabilities and functionality. These are called functional requirements. Additionally, the non-functional requirements specify attributes, conditions, bounds and constraints under which the functionality has to be performed. [5]

Since SoCs are common in mobile devices power is such a constraint as it affects a variety of system parameters. Due to the SoC's limited battery capacity, power consumption directly influences operational lifetime. Power consumption also causes dissipation of heat which affects the system's proper operation, degrades reliability and reduces lifetime [6]. If power consumption is not constrained, costly, bulky and heavy cooling solutions have to be integrated. Lowering power dissipation through power aware design influences the architecture of the supply network on the chip. Consequently, power requirements are the most important non-functional requirements of SoCs [3].

Unfortunately, detailed information about the system's power or energy dissipation is not available in the specification stage, and therefore no unified format or common methodology exists to express power requirements. Often, the power demand is predicted based on experiences from previous or similar projects [7], [8]. For mobile systems a lifetime goal is specified, which affects choosing the energy source. After lifetime requirements and energy source are known, power and energy constraints can be derived and imposed on the system's functionality [9]. In order to meet the power and energy constraints, power states are specified to reduce supply voltage and switch off unneeded parts of the system. Since requirements for SoCs have no unified format, power requirements are rarely specified.

1.1.2 Late Power Aware Design

The decrease in chip area and increase in functionality also affects power dissipation, which has become the most critical constraint for SoCs [10]. The SoC's power dissipation consists of a static and a dynamic part. Dynamic power is caused by changing gate states and signal values while the SoC is active [11], [12]. In contrast, static power is consumed due to leakage when the SoC is powered but the transistors are not switching [11], [12]. With decreasing chip size dynamic power dissipation becomes lower but static power dissipation rises (Fig. 1.2). To manage the SoC's power dissipation power aware design is applied. Power aware design dynamically adapts the system's behavior to the available power [13], [11]. This is achieved with a variety of low-power techniques such as clock gating, power gating, dynamic voltage and frequency scaling which are activated during power states [11]. Meeting power and energy constraints due to limited available power plays a key role in power aware design [13].

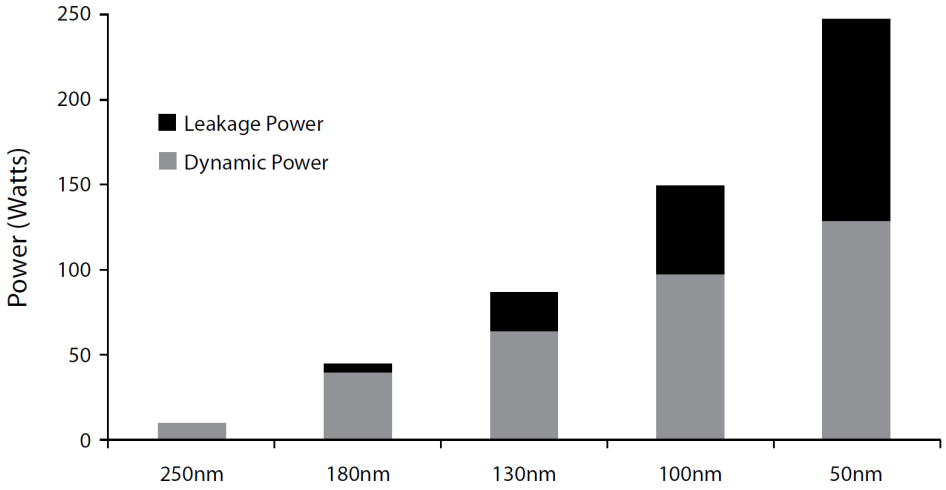

Figure 1.2: Influence of Chip Size on Static and Dynamic Power [12]

Functionality to reduce power is part of the so-called pervasive functionality. Pervasive functionality encompasses parts of the system not directly related to the SoC's main functionality and is often overlooked during early stages of the SoC design [14]. Another reason for the elusiveness of power aware design is that Hardware Description Languages (HDLs) express the system's functionality rather than its power architecture [15], [16]. Consequently, power aware design is often introduced to the system at late design stages [17], [15].

1.1.3 High Complexity in Verification of Power Requirements

Generally, verification examines the system to confirm that it fulfills its requirements. Verification of the power requirements comprises two parts. First, the structure of the power aware design is analyzed and verified against the specification. Second, power dissipation and battery lifetime are estimated and compared to the requirements. [18], [19]

There are two distinct verification methods for power aware design, simulation-based and formal verification. In simulation-based verification a model of the system and its power aware design is executed in behavioral simulation. In contrast, formal verification mathematically proves correctness of system and power aware design. Both methods are highly complex due to the numerous different components, states and interactions. [16], [20]

Presently, functional verification of the power aware design happens at Register Transfer Level (RTL) or Gate Level [16] and uses about one third of the power verification effort [18].

Accurate power estimation to verify the power and energy constraints requires a considerable amount of time and is often only available at late design stages [8]. The battery is typically selected in late phases of the system design which often leads to violations of lifetime or weight constraints and necessitates redesign [21]. Battery models for estimating lifetime often show high computational effort [22], [23]. Consequently, simulating the battery lifetime using a load profile of the system requires a significant amount of time. Early lifetime estimation speeds up the process and helps to avoid costly redesign [24].

The later the power aware design is verified, the longer the system model is "power unaware", which causes an even larger number of potential bugs [18]. Moreover, the later

it is done the longer it takes to simulate the system and to estimate its power and lifetime. Consequently, late verification may also lead to late, costly redesigns.

1.1.4 Missing Verification Information in IP Exchange Formats

To counter the high complexity in SoC design, components are reused from previous designs or third party vendors. These pre-designed components are called Intellectual Property (IP). Within the Electronic Design & Automation (EDA) industry companies use different proprietary storage and exchange formats for their IP [25]. Only in recent years the importance of a common IP exchange format has been recognized. The IP-XACT format developed by the SPIRIT consortium aims for a standardized representation of the IP [26]. However, in most cases the IP neither contains verification information nor a common verification methodology that allows reproduction of verification results [20]. The so-called Verification Intellectual Property (VIP) is a customizable component for verifying a dedicated IP component [19]. It contains the entire verification environment (e.g. test cases, testbenches, etc.) for a specific IP. Without existing VIP, the verification environment has to be manually created, which is challenging and contributes to the verification effort [20].

1.1.5 Low Search Efficiency in IP Libraries

To store and manage IP components for later reuse digital repositories, so-called IP libraries are employed. IP vendors often advertise their IP via online platforms [2]. Even though functionality is a key parameter for IP it is only represented by a few keywords which is inadequate to identify suitable IP accurately. Search mechanisms to find IP components within IP libraries are commonly relying on keywords and/or constraints [2], [27]. Precise functional descriptions of the IP components are scarce because they are difficult to obtain. This impedes efficient IP search and often results in additional effort caused by the accidental selection of unsuitable components.

1.2 Automated Simulation-Based Verification of Power Requirements for SoC Designs

1.2.1 The SIMBA Project

This work is part of the Simulation-Based Requirements Testing of Power Aware SoCs (SIMBA) project[1]. The main goal of the project is a tight integration of requirements into the specification, design and verification phases of power aware SoCs. Therefore, functional and non-functional requirements are specified and linked to test specifications. The linkage process automatically translates the requirements into a verification environment. Through simulation and power estimation the SoC's requirements are verified. [28]

The basis for the SIMBA project is the System Architect Designer (SyAD) framework from CISC Semiconductor Design+Consulting GmbH [29], [30]. The SyAD framework is extended with the novel SIMBA methodology. Another significant contribution to SIMBA

[1]The SIMBA project was funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under contract FFG 812424

is the energy estimation framework Rapid Hierarchical Energy Investigation Modeling System (RHEiMS) from Neosera Systems Ltd. [31], [32]. The SIMBA flow is illustrated in Fig. 1.3. Initially, the requirements are specified and analyzed. Then two interdependent flows originate. The left side of Fig. 1.3 provides verification of functional requirements and is described in [1], while the right side of Fig. 1.3 verifies non-functional, power-related requirements of SoCs and is presented in this thesis.

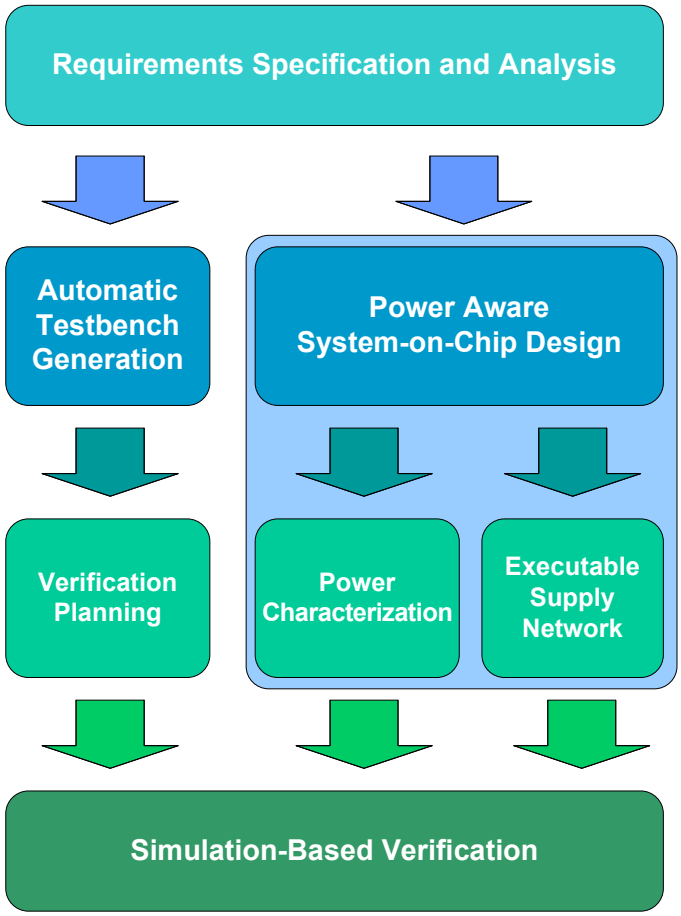

Figure 1.3: Overview of the SIMBA Methodology [28]

1.2.2 Problem Description

This thesis addresses the essential deficiencies and issues of design and verification of power aware SoCs highlighted in section 1.1.

- Specification inconsistencies and elusiveness of different power requirements
- Creating power aware design at late stages of the SoC development process
- High complexity in verification of power requirements
- Missing verification information in IP exchange formats
- Low search efficiency in IP libraries

This dissertation attempts to counter the above issues by tightly integrating different power requirements into specification, design and verification of SoCs. Furthermore, the verification environment and results are linked to the IP for reproducibility in later reuse.

This work's challenges are clearly driven by the high complexity in specification, design and verification of power aware SoCs. Promoting easy specification of power requirements in a unified format may prove to be difficult. Possibilities to establish power aware design early in the design process have to be determined. In addition, the increasing verification complexity caused by power aware design needs to be addressed. Extending the functional verification methodology proposed in [1] and integrating the RHEiMS framework [31] are additional challenges. Means to extend a common IP format, gather verification information and establish a precise but simple functional description have to be implemented. The remaining challenge is to provide a library to store, manage and efficiently search IP.

1.2.3 Contribution and Significance

This dissertation delivers two major contributions:

1. Important power requirements and means to specify them are investigated. The semi-formal format for functional specifications (see [1]) is extended with different power requirements. Then, capabilities for early power aware design are analyzed and established. Simulation-based verification is augmented with power requirements. This includes functional simulation of the power aware design, power estimation and battery lifetime prediction. The high verification complexity is countered by automatic generation of the verification environment and fast system level simulation.

2. Common IP representation formats are evaluated. A suitable format is extended with information from functional and non-functional verification. The verification environment itself is included as VIP. Thereafter, suitable means to accurately describe the IP's functionality are researched. This is closely tied to a study of search mechanisms which are capable of finding IP based on their functional description. IP storage, management and search are implemented by an IP library.

The methodology for specification and verification of power requirements is evaluated in a case study of a higher class RFID tag. The IP library provides components for reuse and assists the design process with verification information.

1.2.4 Organization of Dissertation

The remainder of this dissertation is organized as follows: In chapter 2 state-of-the-art approaches for specification of power requirements, power aware design, simulation-based verification of power requirements, IP representation and IP libraries are evaluated. Chapter 3 explains the methodology for simulation-based verification of power requirements. Additionally, the IP format extended with verification information and our IP library are elaborated. The case study on two SoC implementations of active RFID tags to evaluate the proposed methodology is described in chapter 4. Finally, chapter 5 summarizes this dissertation and provides an outlook on future work.

Chapter 2

Related Work

Since Systems-on-Chips (SoCs) are increasingly complex to specify, design and verify, tools and methodologies have been developed to solve some of the related issues. This chapter evaluates related methodologies and state-of-the-art tools for simulation-based verification of power requirements in the context of SoC design. Additionally, the present state in IP exchange, IP libraries and Verification IP is examined.

2.1 Specification of Non-Functional Requirements for SoCs

This section analyzes which and how non-functional requirements are commonly specified in SoC designs. Special attention and investigation is given to specification formats that allow description of non-functional requirements. Tools for specification of requirements were investigated by Kirchsteiger in [1] and revealed IBM DOORS® to be suitable.

Non-functional requirements describe the system's properties, characteristics, attributes, qualities, constraints, and performance [33]. The distinction between functional and non-functional requirements are often arbitrary [33]. The following constraints are often associated with non-functional requirements of SoCs [10], [24], [34]:

- Timing
- Area
- Power
- Battery lifetime
- Performance
- Reliability

With process technologies below 100 nm, area as a constraint is of less concern [10]. The most controversial parameter is timing [33]. Timing may be regarded as a constraint or as part of the behavior [33]. In the SIMBA methodology we treat timing as part of functionality. Therefore, it is verified together with the functional requirements (see [1]). Power, timing and performance are closely interrelated. If clock frequency is reduced, signal values change less often and power dissipation is lowered. As a result the operation needs

a longer time to complete because speed is reduced and performance degrades. Battery lifetime is strongly dependent on energy (i.e. power * time) [11]. However, peaks in current consumption which are evident in momentary power also have a negative effect on the battery lifetime [23]. There is a consensus that power is the most important constraint for SoC designs [3], [10].

So far, there is no attempt to capture power requirements in a common methodology. In some works however, experiences from SoC designers with specification of power requirements are reported. The following subsection summarizes some representative experience but does not claim to be complete.

2.1.1 Expression of Power Requirements

Since power and energy constraints and battery lifetime requirements are interdependent they can be summarized as power requirements. These parameters influence the structure of the power aware design, which in turn affects fulfillment of the constraints and battery lifetime goal. The power aware design is abstracted into power states which are additional conditions under which functionality has to be performed. Therefore, we regard power states as non-functional requirements. Power and energy constraints, battery lifetime and power states constitute the major areas of concern in specification of power requirements.

Battery or Operational Lifetime Requirements

With mobile devices the battery's lifetime is often equal to the device's operational lifetime. Therefore, battery lifetime is a key requirement in system specification which directly influences customer satisfaction [24]. Battery lifetime is influenced by two factors. First, the capacity which depends on the battery's size, weight and technology [21]. Second, the application which is driven by the desired product lifetime and the required quality of service [35]. Battery lifetime is the most recognized power requirement and is often specified in conjunction with a typical application of the system [24], [36].

> Representative example of a battery lifetime requirement:
> *For 600 read operations per day, the battery shall last for 6 years.* (compare [36])

Power and Energy Constraints

The power constraints are derived from the available power budget. Constraining power avoids excessive dissipation of heat and therefore system instability. Moreover, it limits peaks in current demand which have a negative influence on the lifetime of most batteries [23]. To ensure that the lifetime requirement is met, energy constraints are defined [11]. Even though energy is the more important constraint when using batteries it is rarely expressed. Instead, power constraints are used [7], [8]. Constraints can be applied on the entire system [9] or they can be specified for individual components of the system [37]. Both, power and energy constraints can be imposed on functionality.

> Representative example of a power constraint:
> *The power dissipation of all units active during the read operation shall not exceed 5 mW.* (compare [37])

Power State Requirements

Power state requirements specify how the system is able to fulfill the power constraints. Active and inactive parts of the system and their voltage levels are summarized as power states. Power states are tied to functionality and serve as basis for the power aware design.

Decker et al. from Cadence Design Systems Inc. suggest to specify a strategy for the verification of power aware design in a verification plan. It contains details about requirements for the system's power states, how to trigger a power state, and how power states should be verified. Planning the power aware verification is done when the structure of the power aware design is created for the system. [38]

> Representative example of power state requirements:
> *1.1 Power Mode A*
> *Power mode A is defined by MAC1 being off and MAC2 being at low voltage.*
> *1.1.1 Transition requirements*
> *This power mode can only be entered if the MAC1 link is disconnected, and there are no messages in the*
> *queue assertions : xxx and yyy are required*
> *1.1.2 Functional requirements*
> *The following features must be tested during this power mode*
> *1.1.2.1 Send a transaction on MAC2*
> *Coverage: mac_2_send cross powermode A*
> *1.1.2.2 Attempt to read from MAC1 buffer*
> *1.1.3 Testing Requirements*
> *The following system tests should be run in this mode.*
> *1.2 Power Mode B*
> (from [38], pg. 7, Fig. 4)

Summarizing the examples above, non-functional requirements are often specified informally in natural text. Presently, power states are are not specified in the requirements document. Although, Decker et al. recognize power states as part of the power requirements they fail to express them prior to the design stage. Instead of describing power states in the requirements document during the specification stage, they are often specified implicitly through the power aware design [10]. Also, they are specified parallel to the power aware design in the verification plan [38]. This may be due to the fact that they are often overlooked [14] and that awareness for the necessity of the power architecture arises during later design stages.

2.1.2 Specification Formats for Non-Functional Requirements of SoCs

Sometimes use cases are derived from the initial requirements document. Use cases represent a more technical, interaction-based view of the system. Non-functional requirements are attached either to individual use cases or to the entire use case document [33], [39].

One of these use case formats, capable of accommodating non-functional specifications, is described by Bahill et al. [39]. This format is based on the Unified Modeling Language (UML), which is commonly utilized in software engineering. However, as demonstrated

by Bahill et al. [39] UML can be used to specify hardware systems. The requirements are translated from the customer's point of view to the designer's point of view in a use case analysis model [39]. Initially, the requirements are described in a more detailed, technical manner as use cases [39]. Table 2.1 shows a structured, textual use case format.

Name: Take a sample.
Brief description:
The controller module receives an interrupt from the timer module.
Then it requests a sample from the sensor module and stores it in the memory.
Scope: A system which gathers data from its environment.
Primary actor: The user.
Supporting actor: Timer module.
Frequency: The system operates continuously.
Precondition: The system has been activated by the user.
Main Success Scenario:
1a. A timer interrupt occurs.
2. The controller leaves idle mode.
3. A sample is requested from the sensor module.
4. After the sample has been received it is stored in memory.
5. The controller goes back to idle mode.
Alternate Flow:
1b. The user requests a sample.
Rules:
Rule1: The controller's default mode is idle.
Rule2: When the memory is full, the controller starts to overwrite from the first sample.
Nonfunctional performance requirement:
Taking a sample should last less then 100 ms.

Table 2.1: An Analysis Model Use Case (compare [39], pg. 34)

After the functional and non-functional requirements have been expressed in the use cases, classes are identified. A class is derived from a set of similar objects and abstracts common properties. Thereafter, instances are derived from the classes. [39]

Since the model is based on UML, it utilizes the same nomenclature. A class is related to software development and object-oriented programming. For hardware, classes correspond to modules of the system. Then a collaboration diagram is developed which describes interactions between the class instances and the exchanged messages (Fig. 2.1). [39]

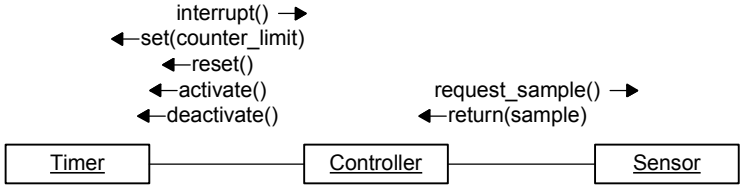

Figure 2.1: A Collaboration Diagram (according to [39])

After functionality and attributes of the classes and their interactions are known, the class diagram is created (Fig. 2.2). It represents classes as boxes which contain the class name, a list of attributes and operations. [39]

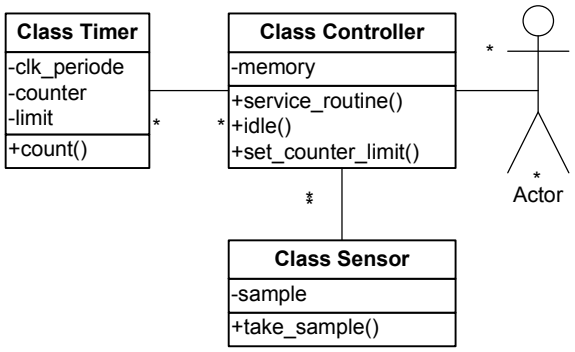

Figure 2.2: A Class Diagram (compare [39])

However, non-functional requirements are only represented in the initial analysis model use case of the format. The collaboration and class diagram are used to add details to architecture and functionality.
In another approach, Glintz [33] regards non-functional requirements as attributes and constraints. He expresses the concern that specifying them into a textual documentation template is helpful but may not be sufficient. Especially when attributes and constraints cannot be clearly identified as entirely local or global, their specification becomes an issue. Instead, it is suggested that an aspect-oriented representation of requirements is employed. A hierarchical multi-dimensional modeling language is used to decompose the system model and describe the requirements. The hierarchical elements represent the functional requirements. Attributes and constraints are described as separate entities which can be attached to the functional elements representing local non-functional requirements. Consequently, global attributes and constraints are attached to the root element in the hierarchy. Attributes and constraints which only apply to some parts of the system are attached to the corresponding parts with a "join relationship". [33]

As a conclusion, the textual format of the use case analysis model presented by Bahill et al. [39] is adapted for the SIMBA project (see [1]). Although the use case format introduced by Bahill et al. allows specification of non-functional requirements they are neglected in later stages of the use case analysis. Also, the non-functional requirements within the use cases do not have a clearly defined structure which leads to inconsistencies. Glintz [33] describes an abstract, tree-like representation of the non-functional requirements instead of a textual template. Detaching non-functional requirements from the functional use cases clearly contributes to an inconsistent specification. Consequently, functional and power requirements are unified in the SIMBA use case format.

2.2 Simulation-Based Verification of Non-Functional Requirements

This section investigates related work in the areas of simulation-based verification of non-functional requirements. However, the focus is on power constraints and battery lifetime. Verification of power state requirements is described in Subsection 2.3.2.

Usually, SoC designers think of constraints as a "backend issue" and are not aware of the complexity of constraints. However, the backend teams are not familiar enough with the design to capture all constraint scenarios. Consequently, verification of constraints is performed late in the design process and some issues may be completely overlooked, which causes redesign after synthesis. [34]

Churiwala et al. describe Atrenta SpyGlass®-Constraints, a tool for verification of timing constraints. The tool is depicted in Fig. 2.3. It is mainly used to uncover missing clock definitions, wrongly inserted clocks, missing signal delays and clock dividers. Besides verifying constraints, SpyGlass® can also be used to automatically insert constraints for synthesis. The constraints are specified in a spreadsheet within SpyGlass® and translated for synthesis tools. The SpyGlass® tool can be used at and below RTL. [34]

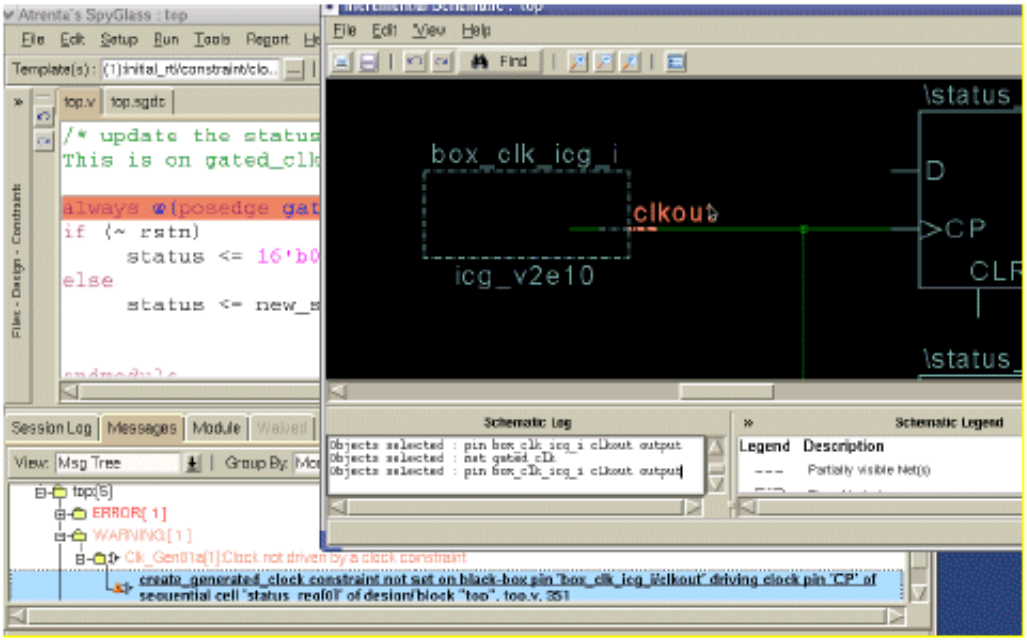

Figure 2.3: Atrenta SpyGlass® [34]

At first glance, the tool described by Churiwala et al. seems to be a general purpose constraints tool (compare Subsection 2.2.2). However, it mainly focusses on low level constraints related to the RTL or gate level layout of the clock tree and power architecture.

2.2.1 Simulation-Based Verification of Battery Lifetime

Battery lifetime is usually verified by using a load profile of the system and a battery model. However, verification of battery lifetime is very difficult due to the complexity of most battery models and their large number of parameters [24]. Therefore, accurate battery models are computationally intensive [23]. Also, missing details about the energy demand and operating conditions at early stages contribute to the complexity of lifetime verification [24]. Since there are numerous battery models available for simulation-based lifetime estimation, only three representative approaches will be discussed. So far, there is no methodology or flow that directly integrates battery lifetime requirements into the verification process or battery lifetime estimation.

Chen et al. propose a battery model which can be simulated with a system model in simulators, compatible to Cadence[1]. An electrical battery model, which is also capable of reflecting a battery's current-voltage characteristics, is developed for lifetime prediction. The model takes non-linear battery effects into account. The authors compare their model for lifetime prediction to actual NiMH and Lithium-Polymer batteries at various discharge conditions and achieve accurate results. Unfortunately, Chen et al. do not provide details about simulation runtime or computational complexity of their model. [40]

In their approach, Rahmé et al. develop a battery model to estimate the lifetime for wireless sensor networks. The battery model is based on an analytical battery model. It is simplified in order to be used on the node itself to calculate its remaining lifetime. Therefore, an average of all currents during a sensor node's state is assumed as the load profile. After a given time the battery model updates its state taking the charge recovery effect into account. It is stated to be accurate compared to other analytical models. [41]

The third approach by Simjee et al. uses a load emulator to estimate the battery lifetime. First, a load profile is recorded with the emulator. This can be either a simulation of a system model or voltage and current measurement of an actual physical system. Second, the emulator applies the stored power profile to a connected physical battery. It is also possible to charge a rechargeable battery with the emulator. The load emulator also allows scripting a sequence of charge and discharge cycles for automating lifetime estimation. The advantage of the proposed load emulator is that it provides reproducible results at acceptable speed and allows to automate battery lifetime estimation on a physical battery. [22]

In [23] Rao et al. investigate battery models with battery and energy optimization in mind. Initially, effects of energy reduction and battery aware optimization are studied. They discovered that under certain conditions energy optimization techniques may achieve longer battery lifetime than battery aware techniques. Moreover, the battery's charge recovery effect plays a key role in maximizing the battery's lifetime. Rao et al. show that the resting period in which the battery recovers charge can be analytically determined. It serves as a criterion when choosing between energy-optimization and battery aware policies. Their conclusion is that for tasks with short and long load duration energy optimization should be preferred over a battery aware policy. [23]

[1] Cadence Design Systems Inc., http://www.cadence.com/

Although, the work of Rao et al. is not closely related to verification of lifetime requirements, it shows the necessity of taking battery lifetime into account when designing a system. Especially when developing policies for task scheduling and energy optimization, which can be done at early stages of system design. Other approaches use recoded power profiles and either emulators or simulators to estimate battery lifetime. None of the related works above provides details how the load profile is acquired (eg. power estimation). The focus is mainly on the battery model or developed policy for task scheduling rather than verification of battery lifetime.

2.2.2 Simulation-Based Verification of Power and Energy Constraints

Power has been recognized as the most important design constraint for SoCs. Therefore, estimating power consumption and verifying the power constraints is now widely performed in the SoC design process. However, most of the effort is focused on low-level power estimation. Although, low-level power estimation provides accurate results, it requires a detailed model of the system, which only becomes available in late design stages. This means that there is little tolerance to change architectural features. Additionally, low-level power estimation is computationally intensive. Consequently, information on the SoC's power dissipation would be required much earlier in the design to have an impact. Since approaches for early, rapid and effective power estimation are not widely available SoC designers commonly rely on spreadsheets, which only provide inaccurate estimates. [8]

Atrenta also enables verification of the power constraints with the SpyGlass®-Power tool. According to Atrenta Inc., SpyGlass®-Power can be used to estimate power consumption at RTL and Gate-level. However, power constraints need to be added manually to design elements (e.g. voltage domains) in order to be verified. Moreover, it can even be used to verify power aware design in both UPF and CPF. It detects areas of activity suitable for clock gating and measures clock gating effectiveness. Furthermore, SpyGlass®-Power is able to expose errors related to power domains, isolation cells and level shifters. [42]

Cadence® Incisive® Palladium® Dynamic Power Analysis uses an in-circuit emulator for power estimation at system level and RTL. Through cell libraries (e.g. for memory) switching activity is estimated and average and peak power is derived. Once a SoC has been emulated its power values and activity data can be stored in a database. Then the values can be reused off-line without the need to re-run power emulation. Via a CPF file, Incisive® Palladium® Dynamic Power Analysis takes power aware design into account. [43], [38]

Sunwoo et al. suggest to use FPGA[2]-Accelerated Simulation Technologies (FAST) to estimate power consumption at early design stages. According to their strategy, the design is divided in a functional model and a timing model. The functional model reflects the functionality of the system and its peripherals. The timing model describes the system's timing which depends on its hardware components. Both models run in parallel, the functional model in a software simulator and the timing model on a hardware emulator (i.e. an FPGA). The functional model executes and sends traces of data and instructions to the

[2]Field Programmable Gate Array

timing model, which reacts accordingly. Since the functional model has no notion of time, it continues sending traces without waiting for the previous task to finish. The authors use activity information from the emulator to estimate power consumption during automated post-processing analysis. [8]

Currently, no design flow is able to handle power constraints, battery lifetime and power state requirements from specification to verification in a single flow. Available tools such as Atrenta's SpyGlass® and Cadence® Incisive® Palladium® indicate that non-functional requirements are of increasing importance in the industry. However, non-functional verification is often done at RTL or below, where simulation suffers from low speed.

2.3 Power Aware System-on-Chip Design

With the increasing importance of power dissipation, techniques to reduce power were developed but had little influence on the design format. As static power became predominant, more techniques to reduce power consumption emerged but could not be accurately reflected in the SoC design languages. Clock gating and power gating effectively reduce static and dynamic power dissipation. Dynamic voltage and frequency scaling reduce the switching activity by lowering the system's performance. In a power state different techniques are combined. However, most techniques are added to the design at low level of abstraction because HDLs do not facilitate their design. Recently, new formats and standards for the description of the power architecture independent of the HDL became available. [11], [44], [15]

2.3.1 Power Aware Design Formats

Both, the Unified Power Format (UPF) and the Common Power Format (CPF) describe the architecture of the power aware design independent of the HDL. CPF was developed by the Power Forward Initiative, a group of EDA companies. In 2007, it was donated to the Silicon Integration Initiative (Si2). Similarly, UPF was introduced by a rivaling group of EDA consortium and later contributed to Accellera[3]. In March 2009, the IEEE approved UPF as IEEE 1801-2009 Standard for Design and Verification of Low Power Integrated Circuits. [44], [15]

Usually, power aware design - the description of elements in the power supply network - is carried out at RTL (see Fig. 2.4). This means that the logic SoC design has been refined and iterated from system level. Then, the architecture of the power domains and their voltage levels are described on top of the logic design in a separate format (ie. CPF or UPF). The power formats specify the architectural elements necessary to apply design techniques for power reduction. CPF and UPF are very similar in their syntax and capabilities to express the power aware design at RTL. In both formats, power or voltage domains, power switches, isolation cells, level shifters and retention cells can be described. Currently, the industry works towards a convergence of the two formats.

[3] http://www.accellera.org

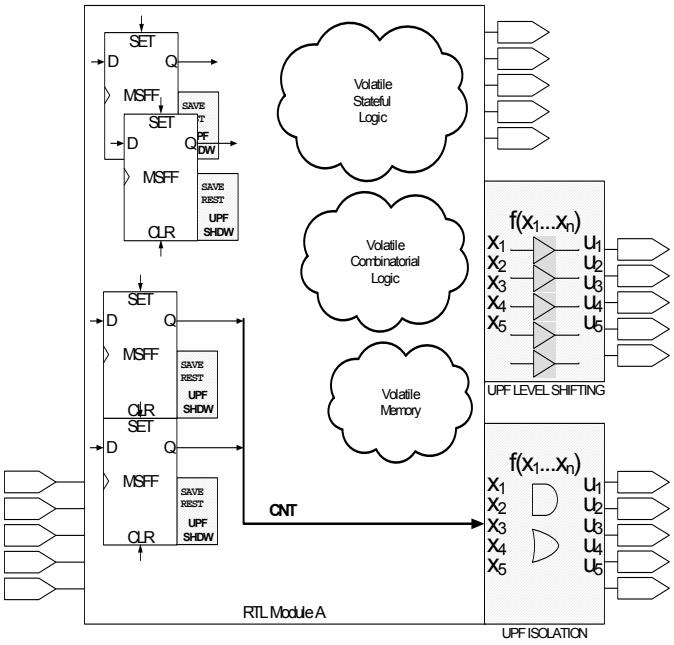

Figure 2.4: Power Aware Design in UPF on Top of the Logic Design [15], pg. 6

However, not all approaches for power aware design use the novel power formats. Jian et al. demonstrate a design flow to reduce power reduction on an RTL SoC design. The authors analyze the RTL SoC design for possible clock gating. Then the Synopsys[4] Power Compiler™ is used to automatically insert clock gating cells during synthesis. Additionally, power management is implemented to dynamically reduce clock frequency and supply voltage. Moreover, Jian et al. optimize usage of on-chip memory to avoid excessive and power intensive external memory access. Also, power gating is applied by turning off inactive parts of the chip and the memories. [3]

UPF and CPF are both specification languages to describe the power aware design on top of the logic design and independent of the HDL. They provide elements to implement power reduction techniques. Several tools of different EDA vendors support power aware design in either CPF or UPF. Since they also provide functionality for verification of the power aware design, they will be evaluated in the following subsection.

2.3.2 Simulation-Based Verification of Power Aware Design

Tools for formal and simulation-based verification of the power aware design are available for UPF and CPF. However, in this section the focus lies on simulation-based approaches. Commonly, power aware design is verified at RTL or below.

Bembaron et al. describe power aware design at RTL in UPF. To verify the power aware design, a three step methodology is applied. First, the authors employ a tool for static

[4]Synopsys, Inc. http://www.synopsys.com/

analysis to detect design errors prior to simulation. Assertions are inserted into the design to identify certain conditions (e.g. activity in a powered-down domain) during simulation. Second, the RTL model of the system, the power aware design in UPF and special power aware behavioral models are compiled. The power aware behavioral models are expressed in the Verilog HDL and contain functionality which triggers special events for the simulator. Third, the compiled design is run on a simulator. The simulator recognizes the events from the behavioral models and handles corruption of memories and signals when the power is switched off. By applying the verification methodology at RTL the authors were able to detect bugs in isolation, retention and power domain connectivity. [45]

A power aware simulator for UPF, as described by Bembaron et al. [45], is included in the Questa® framework from Mentor Graphics® (Fig. 2.5). Questa® PASim interprets the power aware design in UPF and its connection to the HDL models. During simulation it applies corruption behavior for deactivated power domains and infers isolation and retention functionality. Additionally, it supports power aware behavioral models. [46]

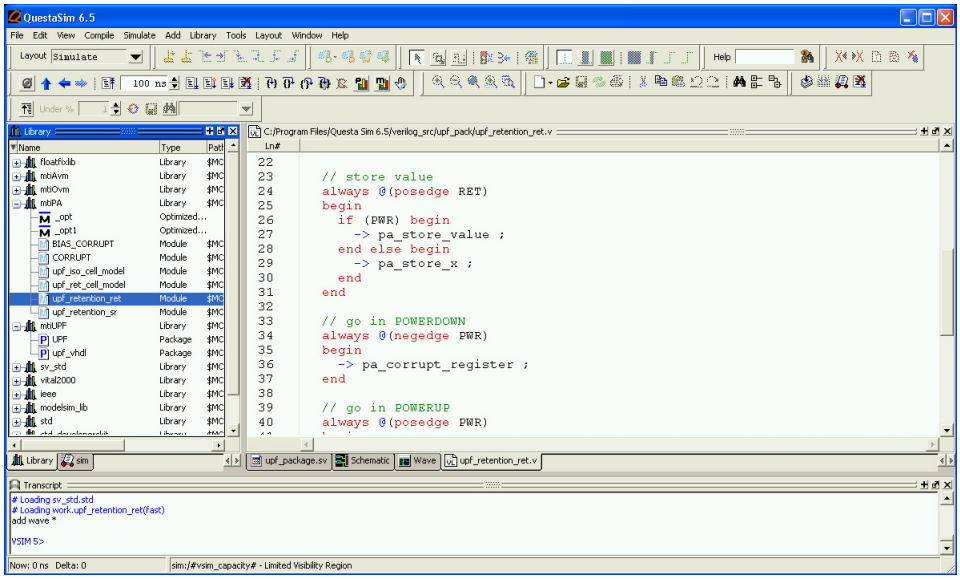

Figure 2.5: Mentor Graphics® Questa®

Cadence Design Systems, Inc. also offers tools for verification of power aware design. They use CPF to describe the power aware design. Cadence® Encounter® Conformal® Low Power performs formal verification without simulating the design. Additionally, the Incisive Unified Simulator is able to interpret CPF commands to simulate power shutdown and startup sequences. Therefore, power domains, level shifters, proper isolation and save-restore sequences for retention can be verified. Above all, the Cadence Design Systems, Inc. simulator operates at system level. This enables early verification of the power aware design with abstract, high-level models. More detail can be added gradually until the system can be verified at RTL. The simulation engine creates a model of the power aware design defined in CPF for RTL and system level designs. [38]

Available design tools use custom simulators which interpret UPF or CPF to verify the power aware design. Assertions and behavioral models of the power aware design can be specified to augment the simulation. Most of the tools perform the verification at RTL. The simulator introduced by Cadence uses CPF for system level verification.

2.4 IP Exchange

Many tools, formats and libraries for IP exist. Basically, each company uses a proprietary format for IP representation [47]. Not only does this complicate IP exchange and reuse, it also impedes automated search [47]. Therefore, the SPIRIT consortium[5], which comprises many major EDA companies, introduced the IP-XACT format [26]. IP-XACT is based on the XML format and allows to encapsulate IP of different HDLs and abstraction levels [26]. In 2009, the IEEE working group P1685 was founded to standardize IP-XACT.

2.4.1 IP Libraries and Tools

Due to the numerous IP formats and libraries, a comprehensive overview would be beyond the scope of this work. Since IP-XACT is the most recent and widely accepted IP exchange format, an overview of representative IP-XACT-based libraries and tools will be provided.

ChipEstimate.com is an online IP provider offering IP from many third party IP developers and semiconductor companies. IP cores are available in many formats, also in IP-XACT. ChipEstimate.com utilizes a keyword and parameter search to find IP within the library. The search dialog is shown in Fig. 2.6. Besides regular IP cores, also Verification IP can be acquired independently from ChipEstimate.com. Generally, the IP does not have VIP attached and is delivered without any VIP. Instead, as quality measure for the IP ChipEstimate.com provides a GSA risk profile and QIP rating. [2]

The GSA is a non-profit organization performing risk assessment based on IP design, verification, documentation, etc. which is available as a spreadsheet [48]. Similarly, the Virtual Socket Interface Alliance (VSIA) QIP Metric analyzes manually specified IP design, integration and support practices [49]. However, the VSIA has been dissolved recently.

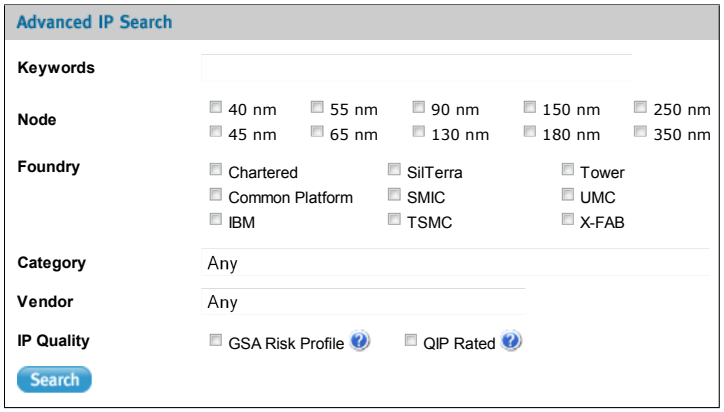

Figure 2.6: ChipEstimate.com IP Search Dialog [2]

[5] http://www.spiritconsortium.org

A similar online IP library is Design and Reuse[6]. Besides Silicon IP, Design and Reuse also offers separate Verification IP and even Software IP. However, it does not provide similar IP quality descriptions as ChipEstimate.com. Only a datasheet summarizes the IP's features. Design and Reuse allows searching for IP based on keywords, vendors and semiconductor manufacturing processes. [50]

The OpenFPGA CoreLib project, introduced by Wirthlin et al., plans to create a standardized design environment for FPGA-based systems. Therefore, the authors intend to link design tools, programming tools and an IP library for FPGA cores with a common standard. Especially interesting is the concept of the library standard to allow tool-independent access for widespread use of the IP component. According to Wirthlin et al. the IP within the library will be represented in the IP-XACT format. However, some extensions to IP-XACT will be necessary to support the suggested higher level of design abstraction. The low-level FPGA IP cores in a HDL are mapped to a high-level programming language such as C and C++. Also, the importance of additional meta-data and information for IP description is pointed out. Even though some concepts are outlined by the authors, the entire project seems to be in an early stage and no details about the library are available. [51]

The Synopsys[7] coreTools are a bundle of tools for IP packaging and management. The coreTools are compliant to the SPIRIT IP-XACT format. They provide a framework for packaging and configuring IP. [52]

- *coreBuilder*^{TM} allows the designer to pack all files of a specific SoC module into an IP-XACT component regardless of its HDL. Different views of the IP can also be created and included in the package. Additionally, coreBuilder^{TM} can be used to create graphical and command based menus for configuring the IP. [52]
- *coreAssembler*^{TM} automates integrating new IP into an SoC design. The coreAssembler^{TM} automatically configures the IP component and connects it to the design. Moreover, it documents details about connection and parameters which can be employed to generate a testbench for the IP. Besides packaging IP with the coreBuilder^{TM}, the coreAssembler^{TM} is also able to handle any new and IP-XACT conforming IP. [52]
- *coreConsultant*^{TM} is the backbone of the coreTools. It guides the designer through configuration, verification and implementation phases and generates the XML files according to the IP-XACT standard. [52]

The Magillem[8] IP-XACT Packager is capable of generating an IP-XACT-conform IP out of HDL source files. It utilizes the Magillem Compliance Checkers Suite to ensure IP-XACT conformity. However, the most remarkable feature is the IP-XACT extension for different Analog-and-Mixed Signal (AMS) HDLs. [53]

Magillem Platform Assembly's goal is to speed up SoC and FPGA-based design. It supports and guides the SoC designer through import, configuration and verification of IP bundles. A graphical design environment is utilized to design and modify the system.

[6] http://www.design-reuse.com
[7] Synopsys, Inc. http://www.synopsys.com/
[8] MAGILLEM The Ontology Company S.A., http://www.magillem.com/

It also allows to connect IP automatically to the system's hierarchy. An environment to control and monitor verification of the IP can also be generated automatically. Platform Assembly checks the IP for IP-XACT compliance. [54]

Arpinen et al. describe how a system can be created on a component basis using UML and IP-XACT. The authors create a system level model containing lower-level IP cores described in IP-XACT. Instead of an HDL, UML is used to describe the IP interconnection in a testbench. Since UML does not have the proper capabilities to compose SoC design, Arpinen et al. extend its diagrams and elements. A framework is implemented which interprets the UML description. With a set of transformation rules it generates an SoC design from the referenced IP-XACT components. On an FPGA, the authors demonstrate generating and synthesizing an SoC from UML and IP-XACT components. [55]

Conclusively, IP-XACT has stimulated research and development in the EDA industry and academia. Libraries of online IP vendors already support the new IP-XACT standard. Many tool vendors provide frameworks to package IP in an HDL into the IP-XACT format. Approaches such as the OpenFPGA CoreLib aim for a library of components synthesizable on FPGAs. Automatically composing and synthesizing a system from a customized UML description is demonstrated. Also, extending IP-XACT for analog components and higher levels of design abstraction is suggested.

2.4.2 Verification IP

Verification IP (VIP) is used to verify other IP components. VIP contains an entire reusable verification environment to reproduce verification results without having to manually create test cases. Despite its obvious importance VIP does not have a common, standardized format and is usually provided separately from the IP. This means that it has to be bought separately (see [50], [2]). Without available VIP, additional effort is required to analyze the IP and to re-create the verification environment.

In September 2009, Accellera published the Verification Intellectual Property (VIP) Best Practices Interoperability Guide [56]. The document contains a set of guidelines which specify how to create and interchange testbenches written in the SystemVerilog[9] HDL. It particularly aims to lower verification costs and effort by providing a reference on how to translate OVM to VMM VIP and vice versa. The Verification Methodology Manual (VMM) and Open Verification Methodology (OVM) are two common SystemVerilog verification libraries. The VMM Standard Library was developed by Accellera, whereas the Open Verification Methodology (OVM) library represents a joint effort from Cadence Design Systems Inc. and Mentor Graphics®. A working group continues to develop the approach into a Common Base Class Library (CBCL) targeting IEEE standardization.

Recently, Accellera and the SPIRIT consortium announced to join their efforts. With their knowledge and expertise in VIP and IP exchange formats, a new standardized VIP-extension for IP-XACT may be possible in the future.

[9] IEEE 1800 Standard for SystemVerilog - Unified Hardware Design, Specification, and Verification Language

2.4.3 Efficient IP Search

When searching IP for reuse the goal is to find components which are best suited for the system-under-design. The high complexity of the system, the large number of requirements and many available components aggravate the situation. The difficulty is to find components which contribute to the system's functionality and match its constraints. Consequently, functionality is the most important parameter for IP search. Additional parameters are constraints such as area, timing and power dissipation.

However, commercial IP libraries only offer simple IP search mechanisms. Keywords, categories, vendors and gate size are commonly used parameters. [50], [2]

Matthaikutty et al. introduce a framework for automatic IP selection based on structural information. The authors utilize metadata gathered from SystemC models to reflect IP. The SystemC model is analyzed to determine whether it is a component, channel or transactor which identifies the component's level of abstraction. Communication and interaction between sub-components are determined by signals and channels. The structural information and interaction metadata are used to gather data types and signatures. Additionally, the IP's name and version can be annotated into the model for cataloging and advanced selection. The IP's gathered information is represented in XML, linked to the corresponding SystemC source and stored in an IP library. A visual architectural template allows the designer to compose the system. It is even possible to mix different levels of design abstraction. IP with the structure and metadata which match the specified architectural template best is automatically selected from the library. Additionally. the authors plan to import metadata from IP-XACT components into their framework. [57]

Hamza-Lup et al. describe a methodology to select components for a system based on its non-functional system requirements. The authors assume that each component is characterized by a series of functional and non-functional requirements which comprise implementation, interface and performance. The functional requirements need to be mapped to functional attributes of the component. Due to a potentially large IP library, thoroughly determining all IP combinations is infeasible because of the high computational complexity. Therefore, the authors reduce the search space by removing components which do not match functional requirements and target interfaces. Then an optimization algorithm is employed to find a combination within the remaining subset of components. The algorithm composes the system from a combination of components matching the non-functional requirements. The resulting components for the system are selected from the IP library. However, the authors do not utilize an actual requirements document to perform their component search. Their work focuses on optimizing of a combination of components based on performance attributes. Functionality can be expressed but only with keywords. [27]

Altough academic approaches suggest more efficient search methodologies for IP components, commercial IP libraries still rely on keyword search. The approach from Matthaikutty et al. gathers structural information from a SystemC component which is utilized for selecting IP from an architectural rather than a funtional description of the system. In contrast, Hamza-Lup et al. select components from a library by employing non-functional requirements to determine an optimal combination of suitable components.

2.5 Summary

From related works and state-of-the-art tools it is apparent that non-functional requirements and constraints are of increasing importance. Especially power constraints, battery lifetime requirements, and power state requirements are gaining the attention of SoC designers. However, there is no common format which would ease specification of those power requirements. Despite their interdependence available methodologies only allow to verify power constraints, battery lifetime requirements and power state requirements separately. Since power requirements are not treated with the same priority as functionality they are often neglected and verified late in the design process. This also complicates power aware design which is usually specified in UPF or CPF. Although the EDA industry has recognized the importance of power aware design, tools commonly support its verification at RTL or gate level.

To keep up with today's fast-paced development cycles designers have to rely on predesigned IP components. The new IP-XACT standard simplifies exchange of IP and is supported by the EDA industry with automated packaging tools. However, IP for verification of the component (i.e. VIP) is commonly sold separately. Metadata and additional information describing the IP and its verification status are scarce. Also, libraries which allow to search IP based on their functionality are not available in the industry. A few promising academic approaches for sophisticated component search exist. Non-functional requirements and parameters are used to optimize combinations of components selected from a library. Another approach relies on structural information from a high-level specification to search for suitable IP. So far behavioral or functional descriptions from requirements are not considered to search for IP in the library.

The main objective of this dissertation is to develop a new methodology for verification of different power requirements. Additionally, efficiency for IP search is improved and verification information is included within the IP. The individual objectives are as follows:

- Extending the SIMBA use case format to specify battery lifetime requirements, power and energy constraints and power state requirements.
- Automatically generating an environment to verify the specified power requirements.
- Verification of power requirements has to be performed at early stages of the design.
- Extending the IP-XACT format to accommodate Verification IP and additional information.
- Development of a functionality-based search for components in an IP library.

Chapter 3

Novel Methodology for Simulation-Based Verification of Power Requirements

Overview

To solve the previously explained issues in SoC design a novel methodology is proposed. The SIMBA use cases in the XML format are extended with power requirements. Power aware design at system level is described in UPF and translated into an executable supply network in SystemC. The automatically generated SystemC verification environment is extended to verify power requirements. Design and Simulation is facilitated in the extended SyAD framework. Energy estimation is performed during simulation with the RHEiMS tool. The system design and simulation environment are translated into the IP-XACT format and stored in the IP library.

Figure 3.1 depicts an overview of the individual stages of the methodology. These stages are discussed in greater detail within the publications in Chapter 6. The publications in Section 6.1–6.4 elaborate the entire methodology for each power requirement. Therefore, the entire flow is applied to the individual power requirements and evaluated separately. Section 6.5, comprehensively explains the simulation-based verification methodology for all power requirements.

The next work in Section 6.6 analyzes important features for Intellectual Property besides the core. Furthermore, it elaborates details on the design of the IP library and explains the concept of the search mechanism. Section 6.7 discusses the extension of the SPIRIT IP-XACT format and the IP library. A summary of the entire IP library and its features is given in final publication in Section 6.8.

Specification of Power Requirements

Power requirements, specified in addition to the functional requirements, comprise battery lifetime requirements, power constraints and power state requirements. Initially, the battery lifetime requirements are specified for a group of several use cases in the semi-formal format. From these high-level requirements, power constraints for functionality of the

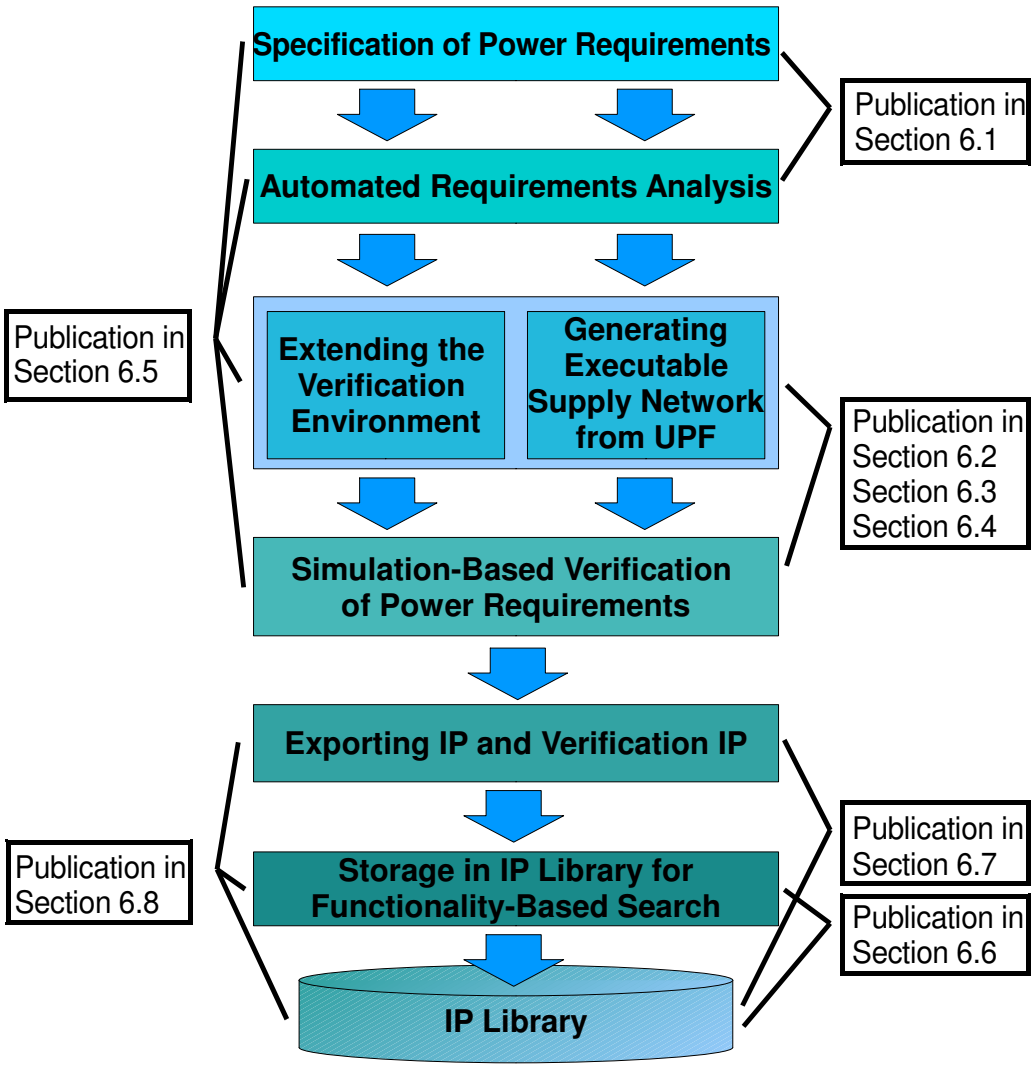

Figure 3.1: Overview of Simulation-Based Verification of Power Requirements

system-under-design are derived. Power and/or energy constraints are applied to entire use cases or individual parts of a use case. The power state requirements specify the power state the system is in while performing the functionality described in a use case. Each power state comprises a description of the supply state for the power domains, isolation and retention information.

Specification of the functional requirements in the use cases is comprehensively discussed in *Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs* (Section 6.1). Additionally, Section 6.2, Section 6.3 and Section 6.4 explain how to specify power requirements.

Automated Analysis of the Requirements

The semi-formal nature of the use case format complements the automated analysis of the requirements. Their sentence structure is evaluated and important information is extracted. The minimum battery lifetime for a sequence of use cases (i.e. scenarios) and their duty cycle is determined. The identified power constraints are tied to entire use cases or use case steps. Moreover, information about supply states for the power domains, isolation for inactive domains and memory retention is gathered.

Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs, in Section 6.1, substantially describes the automated analysis of the semi-formal use cases. The same approach is applied to the power requirements.

Generating the Executable Supply Network From the UPF

The power aware design is developed in UPF on top of the logic design. System modules can be grouped to power domains, which can be switched on and off via power switches. When a power domain is switched off its output signals are floating. Isolation can be defined to avoid unwanted behavior in connected modules. Switching off a power domain causes loss of contents in memory. Retention cells, which save the memory's contents, are required to allow the system to continue its operation from the last known state. The supply network is connected to the power sources and power states are defined.

To avoid design errors, the power aware design represented in UPF is automatically compared to the specification. UPF can not be simulated by itself. Therefore, it is translated into an executable supply network in SystemC[1]. It contains the modules of the logic design and elements from the UPF design as SystemC modules. Monitors inside the supply network keep track of voltage supply and power states. This data is relayed to the verification environment.

In *Simulation-based Verification of Power Aware System-on-Chip Designs Using IEEE 1801* the implementation of the power aware design in UPF and the automatic generation of the executable supply network are outlined. Refer to Section 6.2 for details.

Extending the Verification Environment

Since battery lifetime, power/energy constraints and power state requirements are closely tied to the SoC's functionality, the functional verification environment is extended. The verification environment keeps track of simulation time and to which use case and steps the executed test cases belong. Moreover, it gathers results from energy estimation and the monitors inside the executable supply network. As soon as these results become available they can be utilized for analysis of power states and power dissipation for each use case.

For the battery lifetime requirements, a separate verification environment is generated for each scenario in the application. A scenario comprises sequences of use cases and steps. For each sequence test cases are generated.

[1] SystemC is an open-source HDL based on the C++ programming language. It provides several levels of design-abstraction, a fast simulator and is approved as IEEE Standard 1666™-2005 [58].

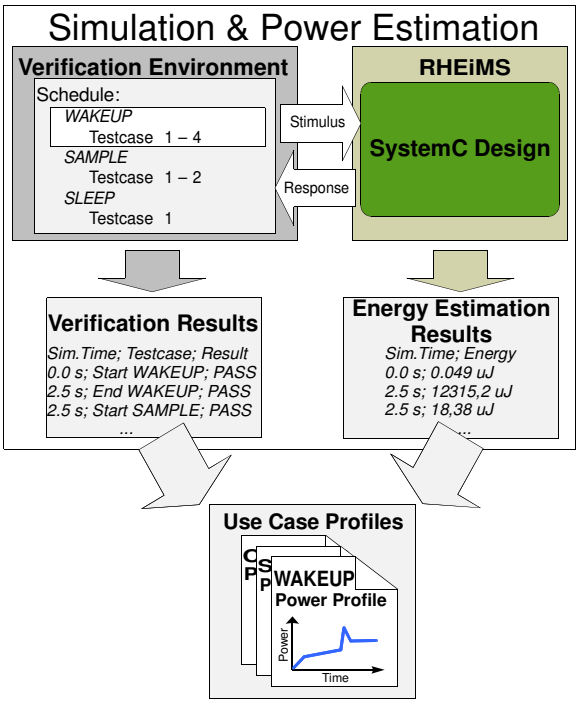

Figure 3.2: Simulation and Power Estimation

Section 6.3 and Section 6.4 extensively discuss the extensions of the verification environment. *Verification Methodology for Battery Lifetime Requirements of Higher Class UHF RFID* refers to the verification environment for verification of battery lifetime requirements (see Section 6.3). Similarly, *Specification and Automated Simulation-based Verification of Power Requirements for Systems-on-Chips* (Section 6.3) presents details on generating the verification environment for power constraints.

Simulation-Based Verification of Power Requirements

During simulation, the verification environment executes test cases and gathers information. The system-under-verification is stimulated by the test cases and responds by executing the according functionality. This functionality corresponds to power being dissipated in the system. The RHEiMS framework is used to estimate the energy dissipation at system level. In conjunction with timing information from the verification environment, the average power consumption is determined. The simulation and power estimation stage is shown in Fig. 3.2.

Simulation of battery lifetime comprises two steps. First, the verification environment needs to be simulated for all scenarios of the application. After power profiles for each scenario are available, they are arranged to a demand pattern according to the application. This power demand pattern is integrated into a SystemC module. Second, this SystemC demand module is connected to a user-defined battery model. In another simulation the power demand pattern is continuously applied to a battery model. At the same time, the verification environment monitors the battery model. When it runs out of charge, the verification environment aborts the simulation and the total battery lifetime is determined.

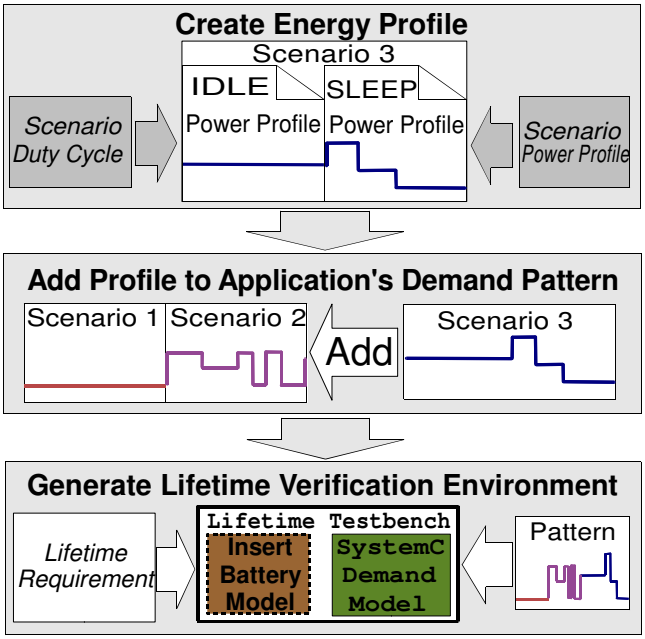

Figure 3.3: Battery Lifetime Verification Environment

After simulation, the verification environment reports the results for power estimation, battery lifetime and the executable supply network. The results are visualized next to the functional requirements. Details about power dissipation, violations of the power constraints or mistakes in the power states are elaborated. After defects in the design have been corrected, verification is re-run until functional coverage goals are met.

The publications in Sections 6.2–6.4 explain simulation-based verification for power state requirements, battery lifetime requirements and power constraints. Moreover, the entire methodology for verification of power requirements is collectively applied in *Automated Simulation-based Verification of Power Requirements for Systems-on-Chips* (Sections 6.5).

Exporting IP and Verification IP

The IP-XACT format, an industry standard and upcoming IEEE standard, has been extended to add requirements and verification information to the IP. The entire project containing the SoC design and its resources is exported (Fig. 3.4). Therefore, the SyAD project is translated into the IP-XACT representation. The extended IP-XACT format contains the logic design, the power aware design, the use cases, documentation, the verification environment and simulation results.

Exporting the IP to the standardized IP-XACT format is elaborated in Publication 7. Furthermore, it explains the IP-XACT extensions for the additional resources and VIP.

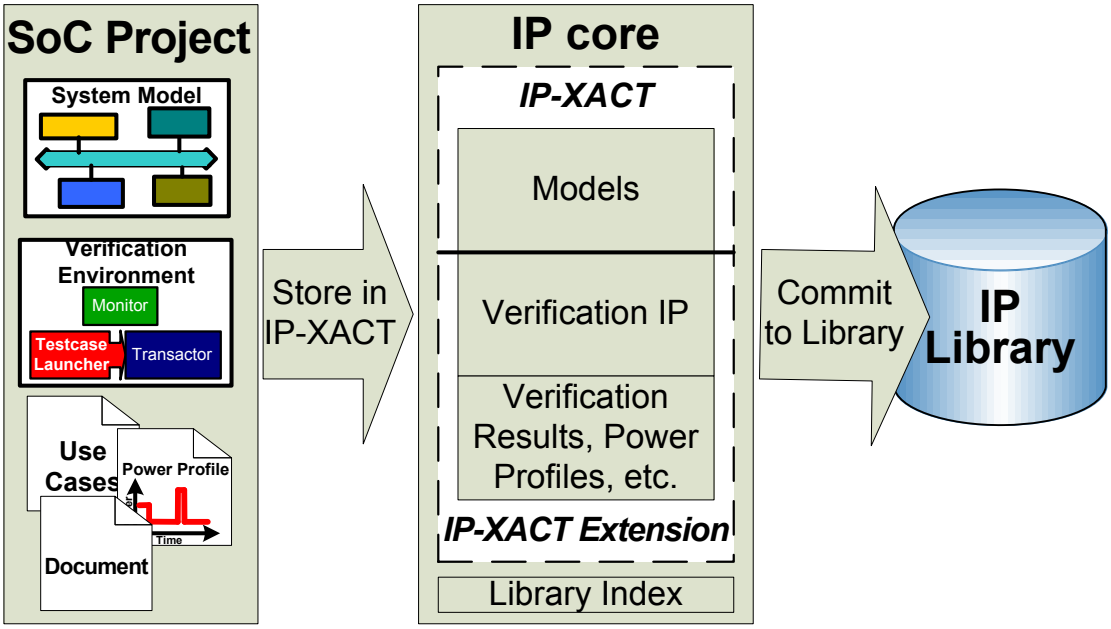

Figure 3.4: Storing IP in the Extended IP-XACT Format and Committing it to the Library

Storage in IP Library for Functionality-Based Search

After exporting the designed SoC project into the extended IP-XACT format it can be stored in a library for later re-use . Instead of relying on simple keywords our novel search mechanism allows to find components based on their functionality. The functionality is represented by the SoC's use cases. When committing the IP to the library, a search index is automatically generated (see Fig. 3.4). During IP search in the library a filter for keywords and constraints is applied initially. Then similarity analysis compares the use cases of the remaining set of IP to the use cases specified as a search parameter. The best matches are returned. The SoC designer is now able to choose from functionally suitable components and is given additional information concerning the IP's verification status. Therefore, the IP library supports IP selection for SoC designs and reduces complexity. The tightly integrated requirements along with added verification status and verification environment inside the IP are crucial for successful future designs.

The library's concept is essentially provided in *A Component Selection Methodology for IP Reuse in the Design of Power-Aware SoCs Based on Requirements Similarity* (Section 6.6).

IP Library

To store and manage the IP for later re-use a server-based IP library has been developed. It acts as a repository for all project resources such as requirements, use cases, power estimation, design and verification information. The integrated IP management tool provides web-based access to the repository for IP management. When IP is submitted to the library, it is marked as "to review". It does not appear in the search result. After the library manager accesses the IP-under-review and inspects the project and its verification information it can be approved. Then the IP is finally committed to the library and appears in search queries. This helps to avoid incomplete and unverified IP within the library.

The publication, in Section 6.7, *An IP-XACT Library extended with Verification Information for Functionality-based Component Selection* explains the server-based IP library in greater detail. Finally, *Search for Extended IP-XACT Components in a Library for Power Aware SoC Design based on Requirements Similarity* summarizes the approach with the extended IP, IP library and functionality-based search mechanism (Section 6.8).

Chapter 4

Methodology Evaluation and Case Studies

To prove the feasibility of the novel methodology it is demonstrated on a case study. A power aware SoC is designed to be used as a battery-powered higher class RFID tag [59]. To show that our methodology is applicable during early stages of the design we develop our SoC at system level. At this stage individual modules and components of the design are available. Timing can be expressed accurately for the components. However, their functional behavior and communication remain abstract. This chapter elaborates how the individual stages of the proposed methodology are applied on the example SoC design.

4.1 Requirements Analysis

Before the requirements for the systems can be specified the intended application has to be defined. Higher class RFID tags are commonly utilized for tracking goods, containers and monitoring environmental parameters. They are equipped with sensors, memories, and processing capabilities. Usually, an RFID reader communicates with the tags. However, active tags are even able to communicate with other tags and the RFID reader.

The chosen application for the developed RFID tag is refrigeration monitoring. RFID tags for refrigeration monitoring are equipped with temperature sensors. They are used to determine if crates of cooled or frozen food may have been spoiled due to exposure to higher temperature. To avoid replacing the RFID tag frequently they are designed to be integrated into a pallet. The plastic pallet with crates of food on it is either stored in a refrigerated depot or transported in a refrigerated truck or container. Every tag has a unique identifier, therefore the pallets are distinguishable. Fig. 4.1 illustrates the concept of our refrigeration monitoring tag.

4.1.1 Functional Requirements

For communication between reader and tag we choose the ISO/IEC 18000-7 protocol [60]. In addition to commands for communication it supports higher class tags with commands for entering and leaving low power states. Based on the ISO/IEC 18000-7 protocol we describe the functional requirements with the help of the SIMBA use case format, instead of a purely textual description. A brief description of all use cases follows below.

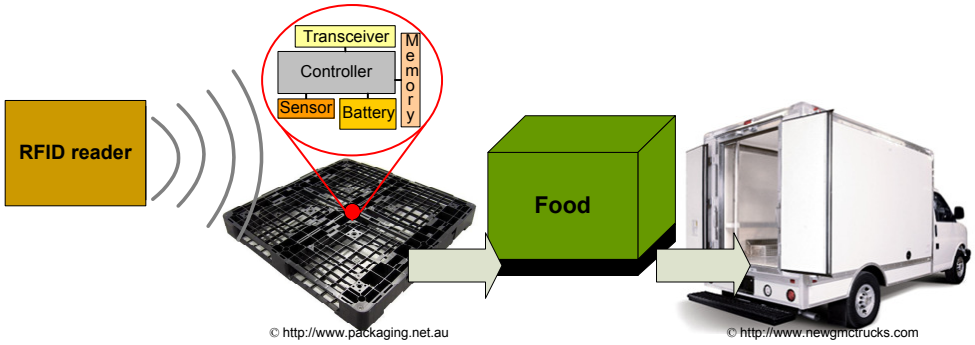

Figure 4.1: Higher Class RFID Tag for Refrigeration Monitoring

Standby - The tag is in a low power mode. Most of its components are inactive.

Wakeup - After a timer triggers an interrupt, the tag awakes from its standby state. Then it determines whether a wakeup command has been sent by the reader. If such a command has been detected it enters idle mode. Otherwise it returns to standby mode.

Idle - During idle all of the tag's components are active. It awaits further commands from the reader.

Collection - This mode is used to determine the tag's unique identifier. The tag reads the identifier stored in its ROM and transmits it to the reader.

Read - If the tag receives a read command it fetches the highest temperature sample and the average temperature from its memory. Thereafter, it transmits the read response containing these values.

Write - With the write command the user can write data into the tags memory. It is also used to program the tag's sampling interval and to erase the memory contents. The tag confirms that the write command was successfully executed.

Sleep - The sleep command causes the tag to shut down and enter the standby mode.

Sampling - When the second timer reaches the time set by the user, another interrupt is generated. If the tag is in a standby state parts of the system wake up to take a sample. If the tag is already active it finishes its current task, then takes a sample. The sample is used to calculate the moving average and to determine the highest temperature since the last read operation. Both, the highest temperature and the average are stored in memory.

Detect State - After the sample has been stored, the tag determines whether its previous state was standby or active. Then it returns to that state.

4.1.2 Power Requirements

After specifying the functional requirements as use cases the power requirements are elaborated. IBM DOORS®, a common tool for requirements analysis [1], is employed to express the requirements. The use cases in IBM DOORS® can be converted to XML. IBM DOORS® is illustrated in Fig. 4.2.

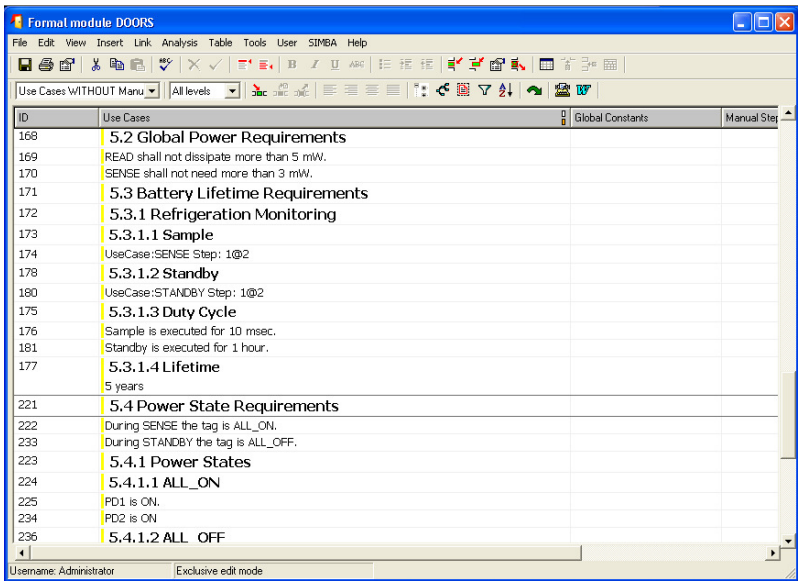

Figure 4.2: SIMBA use cases in IBM DOORS®

The first power requirement is the tag's battery lifetime. The tag is assumed to be sealed to protect its electronics from moisture and damage. Therefore, the battery cannot be replaced easily. Consequently, after the battery runs out of charge the entire tag has to be replaced. If the tag needs to be replaced all too frequently, customers will be dissatisfied.

First, a potential application scenario for the RFID tag is developed. Since the food is most likely to be frozen or kept at very low temperature short exposure to room or ambient temperature are negligible (e.g during loading and unloading). Hence, we assume the tag takes a sample every minute. Every two hours a reader requests the tag's identifier, the average and highest temperature sample. Then the tag is sent back to standby mode. The application is illustrated in Fig. 4.3.

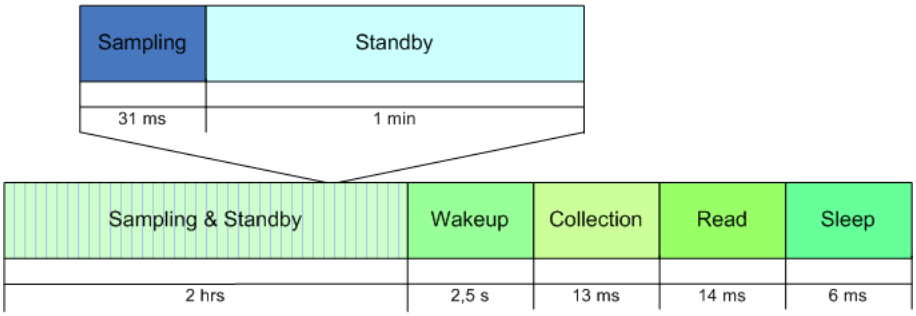

Figure 4.3: Application Overview

As typical battery lifetime for the refrigeration monitoring we specify a time period of 5 years. Next the application and the corresponding battery lifetime are specified in the use case document. The application and lifetime requirement are added to the non-functional requirements section below the functional use cases. We summarize sequences of use cases which occur during the application in scenarios. The three scenarios *Query_ Temperature*, *Sleep_ Mode* and *Take_ Sample* are specified in the use cases as shown in table 4.1.2.

Global Power Requirements:
Application: Refrigeration_Monitoring
Scenario: Sleep_Mode
Use Case: STANDBY *Step:* 1@6
Scenario: Query_Temperature
Use Case: WAKEUP *Step:* 1@4
Use Case: IDLE *Step:* 1@2
Use Case: COLLECTION *Step:* 1@12
Use Case: READ *Step:* 1@12
Use Case: SLEEP *Step:* 1@5
Scenario: Take_Sample
Use Case: SAMPLING *Step:* 1@2
Use Case: DETECT_STATE *Step:* 1a1@1a3
Duty Cycle:
1. Take_Sample is executed 120 times.
2. Sleep_mode is repeated for 2 hours.
3. Query_Temperature is executed once.

Lifetime:
5 years

Table 4.1: Specification of the Refrigeration Monitoring Application

Now a battery has to be found which is able to withstand the harsh conditions of the application's environment. Tadiran[1] manufactures special Lithium batteries which are able to withstand extreme temperatures. After examination of various available batteries the TL-5920 [61] battery was chosen. It features a nominal capacity of 8.5 Ah, a rated voltage of 3.6 V and a temperature range from -55°C to +85°C. However, for the refrigeration monitoring we assume a temperature range of -30°C to 0°C.

Before we can calculate the battery lifetime the power budget is determined. From timing of each RFID command specified in the ISO/IEC 18000-7 protocol [60] and the application the duty cycles are derived. For sampling we took typical values for the conversion time from a datasheet of a digital temperature sensor [62]. Similarly, for memory access time we determined the value from an EEPROM's datasheet [63]. With the timing and the assumed power constraints we are able calculate the battery lifetime. This serves as a quick estimation before we even start the simulation. Table 4.1.2 shows the power constraints for the RFID command. Since the Collection and Read command require transmission of a response we assume a higher power dissipation and, thus, a higher constraint.

Since low temperature affects the battery's capacity we consult the TL-5920 battery's

[1] http://www.tadiranbat.com/

Time	Command	Power Constraint
2.5 s	Wakeup	50 mW
14 ms	Read	60 mW
13.36 ms	Collection	60 mW
50 ms	Idle	50 mW
5.88 ms	Sleep	50 mW
2 hr	Standby	100 μW
3.72 s	Sampling	10 mW

Table 4.2: Determining the Power Constraints

datasheet [61]. Assuming the highest discharge current to be 20 mA, the battery's capacity is 2.4 Ah for -30°C and 4.5 Ah for 0°C. Using Peukert's formula $T = C/I^n$ we quickly estimate the battery's lifetime to see whether our assumptions are feasible and the lifetime requirement of 5 years can be reached. In the equation, T specifies time, C denotes the battery's capacity and I is the discharge current. The parameter n is the so-called Peukert's exponent which is different for each battery time. Since we only need a quick estimate of the battery lifetime we assume n=1 for our Lithium battery. Calculating the discharge current is done straight-forward from the power constraints and timing values.

Temperature	Capacity	Calculated Lifetime
-30°C	2.4 Ah	5 years 212 days
0°C	4.5 Ah	10 years 169 days

Table 4.3: Quick Estimation of the Battery Lifetime

The calculation result from table 4.1.2 shows that the assumed lifetime of 5 years is feasible for the refrigeration monitoring application. The power constraints can now be added to the use cases to ensure the specified battery lifetime is achieved. The corresponding section of the use case document is given in table 4.1.2.

Global Power Requirements:

...

Power Constraints:

STANDBY shall not consume more than 50 uW.
READ shall not use more than 60 mW.
IDLE shall not need more than 50 mW.
SLEEP shall not use more than 50 mW.
COLLECTION shall not need more than 60 mW.
WAKEUP shall not dissipate more than 50 mW.

Table 4.4: Specification of the Power Constraints

After specification of the power constraints we can determine the system's power states. There are three main modes of operation. First, the idle mode during which all modules of the system are active. The RFID tag performs its operation, receives commands and transmits responses to the commands. The second mode, is the standby mode. This is

the low-power state where all modules of the tag are inactive. However, at least a timer and some power management module have to remain active. They are needed to awake the system from the standby mode and to transform it into active mode. The third mode, is the sampling mode in which the tag takes a temperature sample. Only some modules of the tag have to be active to actually take the sample. Since the RFID tag does not need to receive or transmit commands during that time the transceiver module can be deactivated.

We define the state where all modules of the tag are active as "ALL_ON" mode. The standby mode, during which most of the modules are deactivated, is "ALL_OFF". The state during sampling is called the "SAMPLE_ON" mode. Table 4.1.2 contains the specification of the power states linked to the system's use cases. Note the transition from ALL_ON to ALL_OFF during the SLEEP use case.

Global Power Requirements:
...
PowerStateRequirements:
During STANDBY the Tag is in powerstate ALL_OFF.
At WAKEUP the Tag goes to ALL_ON.
For READ the Tag is in ALL_ON.
At COLLECTION the Tag is ALL_ON.
While IDLE the Tag is in state ALL_ON.
During SLEEP the Tag goes to ALL_OFF.
For SAMPLING the Tag is in SAMPLE_ON.

Table 4.5: Specification of the Power State Requirements

For the power aware design it is necessary to specify details about the individual power states. Therefore, individual modules of the system have to be identified.

- Transceiver: It receives and decodes commands sent by the RFID reader. If a response is necessary it encodes the data from controller and transmits it.
- Controller: The controller is the heart-piece of the system. It interprets the commands from the transceiver and reacts to interrupts from the timers. It also holds the tag's identifier in its ROM.
- Timers: The timers send interrupts to awake the tag from standby mode. They also initiate temperature sampling.
- Memory: A memory is required to store the temperature samples.
- Sensor: The temperature sensor is necessary to take and digitize the samples.
- Power Management Unit: This module is responsible for activating and deactivating the power supply of the other modules. It triggers the standby and active states.

Now the specification of the power aware design can be finished. Each power state is a set of power domain states, isolation and retention states. The three previously defined power states necessitate three power domains. Each power domain is defined by its elements and possible supply states (e.g. ON/OFF). The first power domain, PD1, contains the

Transceiver module. Therefore, it also requires a higher voltage. The second domain, PD2, encompasses the Controller module and the Sensor module. These modules require a lower supply voltage. The third power domain contains the Memory module and has the same supply voltage as PD2. Table 4.1.2 shows an excerpt from the specification of a power state and its domains. The first line of the SAMPLE_ON state describes isolation for the power domain PD1. The remaining specifications define the supply states of the domains during the power state.

PowerStateRequirements:
...
PowerState: SAMPLE_ON
All inputs of PD1 are set to low.
The powerdomain PD1 is OFF.
PD2 is at LowVoltage.
Domain PD3 is at LowVoltage.
...
PowerDomain: PD1
DomainElement: Transceiver
DomainStates:
DomainState: HighVoltage
DomainVoltage: 3.0 V
DomainState: OFF

Table 4.6: Specification of Power Domains for the Power States

Now we begin to create the model of higher class RFID tag in SystemC. The use case document is carefully studied and the individual components are created at system level. The system's architecture and its power aware design are illustrated in Fig. 4.4.

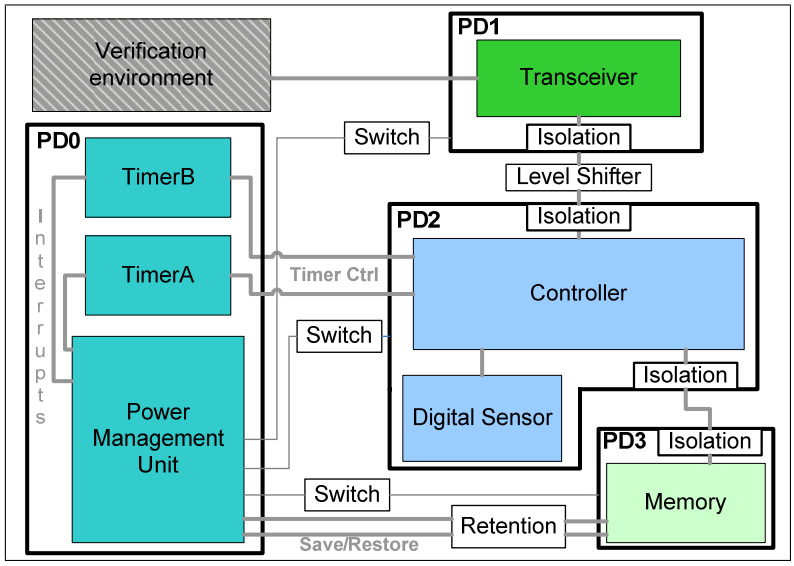

Figure 4.4: Power Aware Design of the Higher Class RFID Tag

4.2 Verification of the Power Requirements

After the system level model of the higher class RFID tag is finished, the novel verification methodology for the power requirements is applied. Initially, the requirements are automatically analyzed and interpreted. For each scenario the sequence of use cases is determined and an XML file is created. From this file a verification environment is created. For each scenario the power dissipation needs to be estimated with the RHEiMS framework. Therefore, the system model is annotated with cases from the RHEiMS database which correspond to its activity. Now the system is repeatedly simulated for each scenario to create a power profile. Finally, all power profiles are automatically joined according to the application description. A demand model is generated and can be connected to a battery model. A simple battery model has been previously developed and is configured according to the Tadiran TL-5920 battery's datasheet. After simulating the demand model and the battery model for -30°C and 0°C we verify the battery lifetime requirement. In table 4.2 the verification results are summarized.

Temperature	Capacity	Simulated Lifetime	Verification Result
-30°C	2.4 Ah	5 years 306 days	**PASS**
0°C	4.5 Ah	10 years 355 days	**PASS**

Table 4.7: Verification Results Battery Lifetime

The verification results for the battery lifetime correspond to the previously calculated results (compare table 4.1.2). Fortunately, the achieved battery lifetime is even longer than expected. The difference comes from the assumed power constraints which were utilized for the initial calculation of the battery's lifetime.

To verify the power constraints the verification environment is created directly from the functional use cases. Subsequently, the verification environment is simulated with the model of the RFID tag and the power for each use case is estimated. The results are verified against the specified power constraints. All use cases satisfy their constraints (table 4.2).

Use case	Power Constraint	Power Estimated	Verification Result
IDLE	50 mW	42.99 mW	**PASS**
COLLECTION	60 mW	46.32 mW	**PASS**
SLEEP	50 mW	49.3 mW	**PASS**
WAKEUP	50 mW	49.27 mW	**PASS**
STANDBY	100 μW	47.52 μW	**PASS**
READ	60 mW	51.91 mW	**PASS**
SAMPLING	10 mW	6.93 mW	**PASS**

Table 4.8: Verification Results Power Constraints

Finally, the power aware design is verified. The RHEiMS power estimation framework estimates the power consumption based on activity in the system and does not take the power aware design information into account. Even though power estimation has confirmed that the power constraints are met, there may be still some mistakes in the power aware design. The power aware design in UPF is translated into the executable supply network

in SystemC. The static checker evaluates the UPF statements and warns about mistakes in the power aware design before simulation. If no mistakes are reported, it can be simulated together with the system model. The monitors inside the supply network report events. After simulation the reported events are automatically analyzed and compared to the power state requirements. In the verification plan the results are visualized and can be inspected by the designer (see Fig. 4.5).

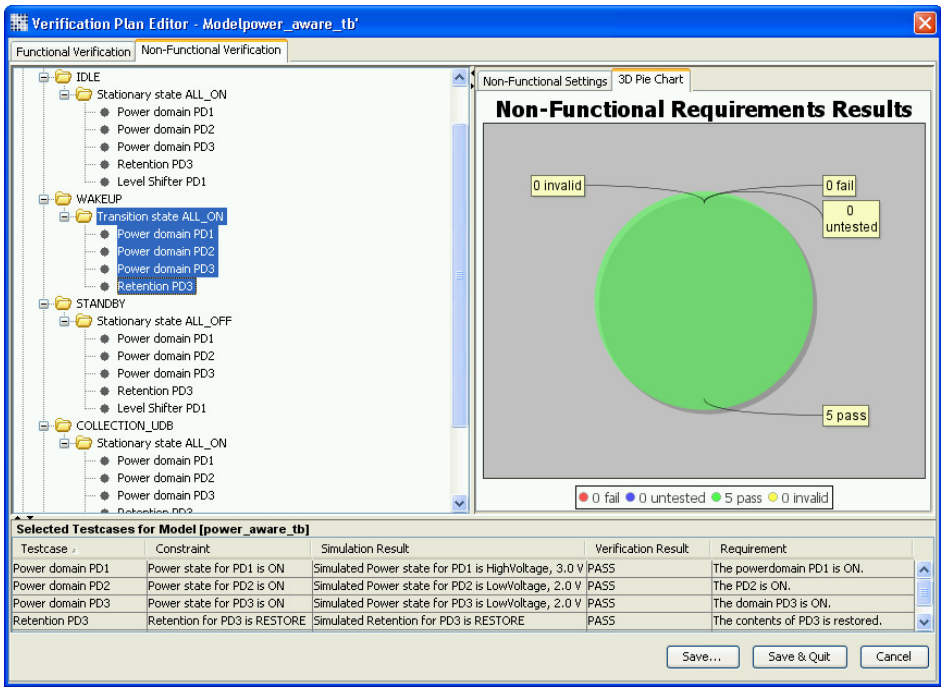

Figure 4.5: Verification Results for Power State Requirements in the Verification Plan

Initially, many mistakes in the power aware design were reported. Table 4.2 summarizes the simulation results which identify the design errors.

The most obvious mistake was the missing level shifter between the power domains PD1 and PD2. A level shifter translates the voltage levels between input and output signals of domains with different supply voltage. The level shifter was not defined in UPF.

Another error was discovered during the wakeup sequence of the tag model. Although the power domains were properly switched on, the retention restore signal was not transmitted by the Power Management Unit. Since system model and power aware design are independent of each other, this problem does not have any consequence during simulation. However, in reality the system would be unable to restore its previous state. This could lead to non-deterministic behavior, loss of data and malfunctions in communication.

A serious mistake was found in the SLEEP and STANDBY use case. The Controller module was properly set into its logical standby mode and ceased all activity. However, the Power Management Unit did not correctly activate the power switches. Therefore, the power domains remained on and the transition to power state ALL_OFF did not take place. Also, the retention save signal was not correctly triggered.

During the SAMPLING use cases a bug concerning the isolation of power domain

Use case	Specified State	Simulated State	Verification Results
IDLE	stationary in ALL_ON	stationary in ALL_ON	**FAIL**
- PD1 and PD2 have different voltages but no level shifter			FAIL
COLLECTION	stationary in ALL_ON	stationary in ALL_ON	**FAIL**
- PD1 and PD2 have different voltages but no level shifter			FAIL
SLEEP	transition to ALL_OFF	stationary in ALL_ON	**FAIL**
- PD1 is ON, supply is HighVoltage			FAIL
- PD2 is ON, supply is LowVoltage			FAIL
- PD3 is ON, supply is LowVoltage			FAIL
- Retention for PD3 is NOT at SAVE			FAIL
WAKEUP	transition to ALL_ON	stationary in ALL_ON	**FAIL**
- Retention for PD3 is NOT at RESTORE			FAIL
STANDBY	stationary in ALL_OFF	stationary in ALL_ON	**FAIL**
- PD1 is ON, supply is HighVoltage			FAIL
- PD2 is ON, supply is LowVoltage			FAIL
- PD3 is ON, supply is LowVoltage			FAIL
READ	stationary in ALL_ON	stationary in ALL_ON	**FAIL**
- PD1 and PD2 have different voltages but no level shifter			FAIL
SAMPLING	stationary in SAMPLE_ON	stationary in SAMPLE_ON	**FAIL**
- Outputs of PD1 are not isolated to ZERO			FAIL
- PD1 and PD2 have different voltages but no level shifter			FAIL

Table 4.9: Verification Results Power Constraints

PD1 was discovered. Without proper isolation the signals of a powered-down domain are floating. This could cause unwanted activity in modules which are sensitive to these signals. In our case, in UPF isolation was wrongly specified “ONE” instead of “ZERO”. After the errors in the power aware design and Power Management Unit were corrected verification is re-run and succeeds.

4.3 Design Space Exploration for the RFID Controller

Since the next step in the design process would be to refine the design to a lower level of design abstraction, a specific implementation for the Controller has to be chosen. Therefore, a design space exploration is performed and the IP library is searched for a suitable component. A search for a functionally compatible implementation is performed with the use cases of the current RFID tag implementation. The IP library suggests suitable IP for an RFID tag which are illustrated in Fig. 4.6.

Amongst the top ranked IP are several useful components for an RFID tag. On the first and on the third position are RFID controllers. Closer inspection reveals that the IP “RFID Controller” on rank three is a very high level implementation for an RFID controller. Since a similar model of such a controller is already used in our project it is discarded. The “RFID_Controller_8051” on the top-most rank is an 8051 microcontroller with an implemented ISO/IEC 18000-7 state machine. Although also at system level it is a more

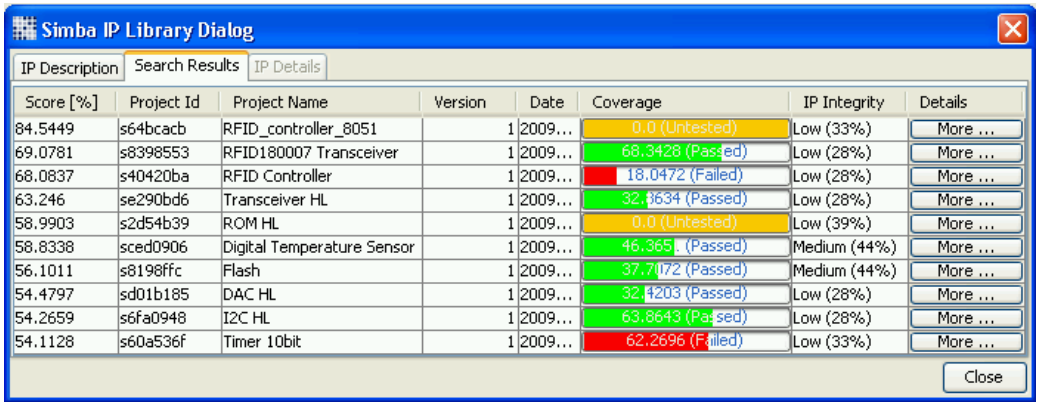

Figure 4.6: Search Results for Functionally Suitable IP

specific model than the current controller in our project. However, its verification status indicates that it has not been verified yet. Since, the power requirements need to be re-verified for the project with the "RFID_Controller_8051" component, we import the IP into our project despite its verification status.

Repeating the verification process reveals that the newly imported 8051 RFID controller dissipates less power than the originally developed controller. The verification results of the two different implementations can be compared. Verification of the power constraints clearly shows that the 8051 implementation of the RFID controller dissipates less power than the previous implementation (Table 4.3).

Use case	**Power Constraint**	**Power Estimated** HL_RFID_Controller	**Power Estimated** RFID_Controller_8051
IDLE	50 mW	42.99 mW	868 μW
COLLECTION	60 mW	46.32 mW	4.21 mW
SLEEP	50 mW	49.3 mW	4.49 mW
WAKEUP	50 mW	49.27 mW	869 μW
STANDBY	100 μW	47.52 μW	24.36 μW
READ	60 mW	51.91 mW	5.64 mW
SAMPLING	10 mW	6.93 mW	4.18 mW

Table 4.10: Comparing Power Constraints Results for Two Different RFID Controllers

As a result from the lower power dissipation we also expect a longer battery lifetime. Simulation with the battery model confirms our expectations. Table 4.3 summarizes the battery lifetime results. Although the Taridan TL Lithium-Ion batteries are stated to last up to 20 years (according to [64]), a lifetime of 49 years is very unlikely. The simple battery model does not account for self discharge and deterioration of the battery's materials. The result from battery lifetime estimation shows that a battery with less capacity would also be sufficient. The Tadiran TL-5903 is a suitable candidate from the same series [65]. It is cheaper and also smaller than the TL-5920 but is still able to withstand the extreme temperatures of refrigeration monitoring.

Use case	Battery Lifetime Requirement	Simulated Lifetime HL_RFID_Controller	Simulated Lifetime RFID_Controller_8051
Battery TL-5920 -30°C, 2.4 Ah	5 years	5 years 306 days	26 years 189 days
Battery TL-5920 0°C, 4.5 Ah	5 years	10 years 355 days	49 years 264 days
Battery TL-5903 -30°C, 0.6 Ah	5 years	1 year 144 days	6 years 229 days

Table 4.11: Comparing Battery Lifetime Results for Two Different RFID Controllers

4.4 Summary

The novel methodology was applied to an SoC implementation of a higher class RFID tag for refrigeration monitoring. The functional requirements were specified in the SIMBA use case format in IBM DOORS®. The refrigeration monitoring application was specified and a lifetime requirement was expressed. After choosing a battery the power constraints were imposed on functionality, so the lifetime goal can be met. To fulfill the power constraints, power states were specified for the individual use cases. For the power aware design a UPF file was created which describes several power domains and the supply network. The verification environment was automatically generated from the use cases and simulated. During simulation, verification and power estimation were performed. The verification results revealed a series of errors related to the power aware design. After they were corrected, the desired lifetime was ascertained and the power constraints were fulfilled. Design space exploration was performed by searching functionally suitable components in the IP library. An 8051 microcontroller with an implemented RFID state machine was found. Re-verifying the SoC design after replacing the old controller with the 8051 microcontroller revealed a far lower power dissipation. The lifetime goal could be reached with a smaller, cheaper battery.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This work presents a new methodology for verification of power requirements through simulation. The existing SIMBA use case format for functional requirements is extended with power requirements. These power requirements are power/energy constraints, battery lifetime and power state requirements. After specifying the functional use cases and describing a potential application, the battery lifetime requirements are expressed. Then power constraints are imposed on the functional use cases to limit power dissipation. Power state requirements elaborate the power state the system is in during a use case.

According to the specifications, the power aware design is implemented in the Unified Power Format (IEEE Std. 1801) independent of the system design. In our IP library, SoC designers can search components for reuse. Suitable components can be found through automatic similarity analysis which identifies components compatible to the system's functionality. After the system and its power aware design are created verification begins.

Initially, the use case document is analyzed and a verification environment is generated for functional verification (see [1]). This verification environment was extended with power requirements. To verify the battery lifetime the application's power demand is estimated with the RHEiMS framework (see [31]) while simulating the system with the verification environment. With the estimated power profile and a battery model the lifetime of the system's battery is verified for the given application. Similarly, the power and energy constraints are verified by estimating the power/energy dissipation for each use case. To verify the power state requirements an executable supply network is automatically generated from the UPF design. After it is simulated together with the system, reports from the network's elements are compared to the power state requirements.

The verification results for the power requirements are visualized in the verification plan for inspection by the designer. After all requirements and constraints are fulfilled the system can be submitted to the IP library. The IP-XACT format was extended to accommodate the system's design, power aware design, use case, documentation, verification environment and verification results. Since IP-XACT is widely supported, the newly created IP can be exchanged easily. The verification results increase confidence during IP selection. Moreover, the included verification environment even allows to recreate the verification results.

The benefits of this work are as follows. A functional use case format is extended for specification of different power requirements in a single document. Through automated analysis of this document the verification environment is extended for verification of the power requirements. Therefore, the verification effort is reduced. Power aware design can be created and verified already at system level utilizing the IEEE Std. 1801 UPF. Additionally, the introduced methodology supports verification of the specified power requirements in one complete simulation-based flow within the SyAD framework.

An IP library was developed to store components comprising specification, design, documentation and verification information. Therefore, the IP-XACT standard was extended. An efficient search mechanism was developed to retrieve components based on their functional compatibility to the current system-under-design. By linking the verification information and environment to the IP the verification effort is reduced and the IP can be reused with confidence.

The developed methodology was evaluated on a case study of a higher class RFID tag which was implemented as a System-on-Chip. The battery lifetime requirement was specified and verified for an example application where the tag was used for refrigeration monitoring. To achieve the desired battery lifetime, power constraints were specified. Through simulation and power estimation using the RHEiMS framework the power constraints were successfully verified. According to the specified power state requirements a power aware design was created in UPF. Verification revealed deviations between design and specification and errors in the power management. After the errors were corrected, a different implementation for the RFID controller was searched in the library. Design space exploration with functionally suitable components from the IP library resulted in a new design. The selected 8051 microcontroller turned out to be a better suited component. Verification was re-run for the system and succeeded. With the 8051 microcontroller the higher class RFID tag dissipated far less power. Consequently, it achieved the required battery lifetime for the refrigeration monitoring application with a smaller, cheaper battery. The project was submitted to the IP library for future reuse.

5.2 Future Work

The next step in the evolution of SoCs are Networks-on-Chips (NoCs) or Multiprocessor Systems-on-Chips (MPSoCs). NoCs are usually a layered stack of SoCs which are interconnected by a bus. MPSoCs are SoCs with additional processing units for parallel processing. It is obvious that design and verification of those systems are even more complex than for SoCs. New design paradigms have to be developed to manage the rising complexity.

Specifications will become more abstract and will already describe the structure and functionality of the design. Additional constraints and parameters can be imposed on functionality, components and the entire system. From the specifications, IP will be automatically gathered from different, distributed IP libraries (compare [57]). During simulation different matching components will be swapped ad-hoc when their constraints are violated. Future IP will have linked configurable verification IP which is automatically inserted during verification.

As pointed out by Rahmé et al. energy management of Wireless Sensor Nodes (WSNs) becomes increasingly important [41]. Even though sensor networks comprise either WSNs or active RFID tags, design and development of their power aware design is done individually. The idea is to take the power states of neighboring nodes into account and to develop strategies for power aware behavior already in the hardware design stage. The approach of Rahmé et al., which calculates the battery's state-of-charge on-the-fly and adapts the WSN's behavior based on neighboring nodes (see [41]) could be seen as a first step into that direction. The entire network of nodes will cooperatively develop a power management strategy constantly adapting their power-states. Therefore, future work will include tools and methodologies to design and verify a new generation of power aware SoCs.

Verification of these systems with "interdependent power aware design" requires a new paradigm of power aware design to handle this bird's eye view of the entire system of wireless sensor nodes. Novel languages and formats for specification, system design and verification will be necessary. Pervasive functionality has to be integrated into the specification process even more tightly. Additionally, communication and interaction between several of these systems-of-systems have to be taken into account already during specification. However, this would cause complexity to increase even more. Instead of a single, already complex system, a multitude of interdependent, complex systems has to be specified, designed and verified. Design for such systems would necessitate a new level of abstraction to compensate the extreme complexity. Instead of the system level which treats one system as a whole a new system-of-systems level (SSL) will be necessary to design the interdependent power aware system.

Understandably, designing such a system will heavily rely on the design-reuse paradigm to take advantage of standardized IP format. Structural and functional information are combined to search for IP in several remote libraries in parallel. This necessitates new and fast algorithms for evaluating and combining results from different IP libraries. Feedback and ranking systems which take opinions and experience from other designers into account will be required in addition to information on verification status. This improves IP quality and confidence in unknown IP. Since structural information is used for search, the IP can be automatically connected to the remainder of the system. Similarly, included power aware design is connected to the system's power supply network and power management scheme. This allows establishing inter-system power awareness.

Future verification would require additional automation and abstraction to handle the huge complexity. Standardized VIP which is linked to each IP component could be automatically integrated into the global verification environment for the system. Through standardized interfaces the component's VIP communicates with the system's verification environment. Distributed simulation, with each IP being simulated on the IP vendors' distributed servers, accelerate verification. This means that a low level IP (e.g. gate level) is either simulated or synthesized on a dedicated FPGA at the IP vendor's facilities. Through several levels of abstraction the results are communicated through the System-of-Systems level. Therefore, the designers and verification engineers receive accurate results at fast simulation speed. However, this would require new verification tools to handle the computationally intensive (low level) simulation and overhead from distributed communication. Additionally, the simulation results from several IP components at various locations and in different HDLs have to be translated, combined and evaluated for verification.

Chapter 6

Publications

This chapter contains publications which explain the approach presented in Chapter 3 in greater detail.

Publication 1: *Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs*, 2nd Annual IEEE International Systems Conference, Montreal, Canada, 7–10 April 2008

Publication 2: *Simulation-based Verification of Power Aware System-on-Chip Designs Using IEEE 1801*, IEEE NORCHIP Conference, Trondheim, Norway, 16 –17 November 2009

Publication 3: *Verification Methodology for Battery Lifetime Requirements of Higher Class UHF RFID Tags*, IEEE International Conference on RFID 2009, Orlando, USA, 27–28 April 2009

Publication 4: *Specification and Automated Simulation-based Verification of Power Requirements for System-on-Chips*, Joint IEEE Circuits and Systems and TAISA Conference 2009, NEWCAS-TAISA '09, Toulouse, France, 28 June – 1 July 2009

Publication 5: *Automated Simulation-based Verification of Power Requirements for Systems-on-Chips*, to be published

Publication 6: *A Component Selection Methodology for IP Reuse in the Design of Power-Aware SoCs Based on Requirements Similarity*, 3rd Annual IEEE International Systems Conference, Vancouver, Canada, 23–26 March 2009

Publication 7: *An IP-XACT Library extended with Verification Information for Functionality-based Component Selection*, Austrochip 2009, Graz, Austria, 7 October 2009

Publication 8: *Search for Extended IP-XACT Components in a for Power Aware SoC Design based on Requirements Similarity*, IEEE Systems Journal 2010, to be published

SysCon 2008 – IEEE International Systems Conference
Montreal, Canada, April 7–10, 2008

Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs

Christoph M. Kirchsteiger[1], Johannes Grinschgl[1], Christoph Trummer[1], Christian Steger[1], Reinhold Weiß[1] and Markus Pistauer[2]
[1]Institute for Technical Informatics
Graz, University of Technology
Inffeldgasse 16/1, 8010 Graz, Austria
E-mail: *(c.kirchsteiger, steger, rweiss)@tugraz.at*
[2]CISC Semiconductor Design+Consulting GmbH, Austria
Lakeside B07, 9020 Klagenfurt, Austria

Abstract *– In common design flows of System-on-Chip (SoC) designs functional verification requires 70% of the entire design effort. Most of the effort for functional verification is spent on finding and creating adequate testcases to verify that the modeled design corresponds to its specification. This is done manually, since automatic testcase generation from the specification is often not possible due to the informal, non-machine readable structure of the specification document. Formal specification languages would ease the parsing process, however, these formats are difficult to use by system engineers from different domains. A promising trade-off are semi-formal specification formats, which are both easy-to-parse and easy-to-use.*
The SIMBA[1] project focuses on semi-formal use case-based specification formats, which are used to automatically generate a transaction-based SystemC verification platform. Finally, these SystemC testcases are simulated together with the System-under-Verification (SuV) to verify that it fulfills the given specification.
This results in a novel design methodology regarding requirements elicitation and automatic testcase generation. A demonstration is given by applying this methodology to a SystemC RFID controller model. It is shown that the demonstrated approach automates and improves the functional verification of SoCs.

Keywords *– Automatic Testcase Generation, Specification-based Verification, Dynamic Functional SoC Verification, Semi-formal Specification.*

I. INTRODUCTION

Due to the increasing complexity of System-on-Chip (SoC) designs, a specification-based functional verification of these systems is a time and resource consuming process. The objective in functional verification [1] is to verify that the modeled system behaves according to its informal specification. Clearly, deriving testcases manually by reading the large SoC specification document is very time and resource intensive and error-prone. On the other hand, it is infeasible to perform this task automatically due to the informal, non-machine readable structure of the specification document.

A number of research projects, most of them from the software domain, try to resolve these issues. Depending on the chosen specification format they can be divided into:

- Non-formal specifications, which use natural language for requirements specification. Stakeholders, who specify requirements, can use them easily but it is difficult to parse them automatically.

- Formal specifications based on formal description formats. Languages like the Unified Modeling Language (UML), the Specification Description Language (SDL) or temporal-logic properties are used to define an entirely formal specification document. These languages can be parsed easily, however they are in large parts unknown and difficult to learn.

- Semi-formal specifications are a desirable trade-off between the techniques above. They are accepted by stakeholders and the almost automatic processing does not impose too much interactive effort onto the designer.

Field Name	Description
Name	Name of the use case
Brief description	Optional description of the use case
Scope	Name of the System-under-Verification (SuV) or Design-under-verification (DUV)
Primary actor	External components interacting with the SuV (=environment)
Precondition	Valid conditions before the use case is executed
Main Success Scenario	Step-based interaction of the System-under-Test with its environment
Alternative Flow	Alternative step-based interaction with the environment
Postcondition	Valid conditions after the use case is executed
LocNonfunctional Requirements	Specified non-functional requirements (timing constraints, power constraints,...) for each interaction step or flow of steps in the current use case
GlobNonfunctional Requirements	Specified non-functional requirements (timing constraints, power constraints,...) for all specified use cases
Constants	Message types, time constants, ...

Fig. 1. Enhanced Textual Use Case (Semi-Formal Specification).

1 This project has been funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the FFG contract FFG 812424

The approach presented here focuses on semi-formal description formats. A very promising and well-known semi-formal specification style are textual use cases [2]. They are much more detailed than the graphical UML use cases. They are both widely accepted by stakeholders and suitable for automatic post-processing. They define the interaction and behavior of a system under certain conditions (pre-/post-conditions, trigger) as a sequence of interaction steps with the environment (represented as so-called "*actors*"). Their structure is formal, table-based and composed of several fields for the name, the pre-/postconditions and the interaction scenarios. However, within each field the description is entirely informal. Thus, textual use cases are similar to natural language but used in a structured way, which makes them easy-to-learn for stakeholders from various domains.

A common textual use case description contains the following fields:

- Actor (communicates with the specified system)
- Pre-/postcondition and trigger
- Main success scenario (i.e. main interaction sequence)
- Extensions (i.e. alternative flows to the main scenario)

We have extended the common textual use case description with additional fields to cover non-functional requirements, like power and performance requirements and constants, like message types and time constants. Our textual use case format is shown in Fig. 1.

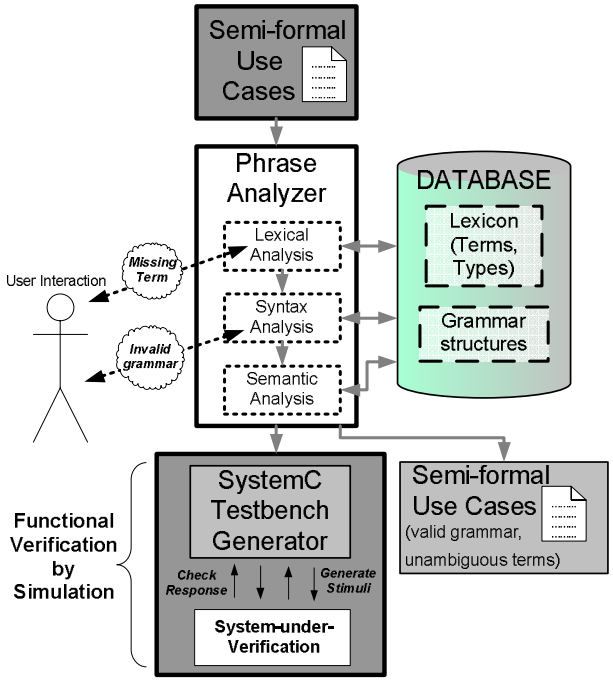

Fig. 2. Textual Use Case (Semi-Formal Specification).

In this paper, we propose a novel design methodology (see Fig. 2) for the specification-based functional verification of SoCs by simulation. We use simulation for verification without being concerned with the state-space explosion problem as in static verification techniques. The main idea behind our approach is based on automatic testcase generation from the specification document. Fig. 2 shows that our approach encompasses both the correction of ambiguities and misinterpretations by performing a lexical, syntax and semantic analysis of the specification document. In addition it enables functional verification of the system model by the generated SystemC testcases (see bottom block in fig. 2). The testcases are based upon the SystemC Verification Standard (SCV) [3] and can be used to verify both transaction-level models and model implementations at RTL [4].

The remainder of this paper is organized as follows: We start with an overview of related work in section 2. In section 3 we present our methodology and describe its implementation in section 4. Section 5 provides a case study of a SoC-based RFID controller to present the application of our methodology. Results are demonstrated in section 6. Finally we give a conclusion and list further work in section 7.

II. RELATED WORK

Testcase generation from the specification has been widely studied in the research community. The focus of our approach is to support a specification format, which is both easy-to-use by various stakeholders and suitable for automatic processing. In contrast, some research approaches like [5] and [6] that favor UML or SDL as specification languages lack a highly accepted and easy-to-use language for specification of SoC components. This constitutes a large burden for stakeholders from various domains, who are not familiar with these specification languages.

In [7] natural language requirements are used as specification format. However, due to the limitations of natural language regarding automatic processing, additional SDL descriptions have to be implemented for each requirement to enable testcase generation.

Other approaches, like [8], [9], Torx from Tretmans [10], and the B-language [11], are based on temporal-logic properties to automate specification-based testing. Although this specification formats are unambiguous, precise and consistent, it is very difficult for stakeholders from various domains to get familiar with these formats. In contrast, our approach is based on semi-formal textual use case-based descriptions as defined in [2]. They are both widely accepted and easy-to-use by stakeholders and suitable for automatic post-processing.

There are some significant approaches in literature dealing with textual use case-based descriptions. The closest to our work are [12] and [13]. In [12] a software use case specification is parsed and converted to another specification format, known as UML activity diagrams. This is done by lexical, linguistic and semantic analysis. These steps are quite similar to our approach; however, the ability to automatically generate testcases from the UML diagrams is completely missing. In [13] linguistic techniques are applied to extract

useful information from the use cases. Nevertheless, no testcases are automatically created either. Other relevant approaches [14], [15] use natural language patterns to eliminate the inherent natural language drawbacks, like ambiguity, incompleteness and inaccuracy. These patterns are applied during the elicitation phase. The stakeholder gets a number of recommendations and guidelines of how to apply them. Although, these approaches result in a more unambiguous and complete specification, no effort was done to transform these formats to formats suitable for automatic testcase generation.

Approaches dealing with an unstructured natural language specification as [16], [17] require a lot of effort and interaction in the field of requirements engineering. In [17] explicit properties of each requirement have to be defined manually, together with a model to check these properties. In [16] each requirement has to be analyzed manually to indicate what simulation data and waveform it corresponds to and how to test it. Clearly, these approaches require a lot of interaction effort when the number of requirements is large. In contrast, our approach requires minimum time for requirements specification and automates most of the steps for testcase generation. This greatly reduces time and effort to fulfill the short time-to-market required for SoC designs.

III. NOVEL APPROACH

We propose a novel specification-based functional verification by simulation methodology that aims for:

1. Resolve ambiguities and incorrect grammar in the specification document (=Specification checking).

2. Focus on automated functional testbench generation from a textual use case-based specification (=Minimal user interaction).

3. Verify that the developed system model corresponds to its specification (=Functional Verification).

As shown in Fig. 2 our approach starts with a semi-formal use case-based specification of the System-under-Verification (SuV). Then, the phrase analyzer performs a lexical, syntax and semantic analysis to extract information from the use case-based specification required by the subsequent testcase generator. The phrase analyzer requires some user interaction and is tightly linked with the database. The database contains a lexicon of known terms and their types and a set of supported grammar structures. The extracted information is used to generate a syntax-corrected and unambiguous specification document. It is used as input for our testcase generator. During simulation these testcases are applied to the SuV to check if it corresponds to the specification. Output messages convey information on the test progress, the test coverage as well as the test results to inform the verification engineer on-line about the current status of the simulation-based verification.

As illustrated in Fig. 2 some interaction with the user is still required by our approach. However, the effort for these interactions is decreasing with the number of processed requirements and depends on how much the user sticks to our provided specification guidelines. In case of a missing term, the user has to specify the type of this unknown term or it has to map the term to an existing one in the database. This decision is remembered the next time this term is analyzed. Thus, no user interaction is required for this term any more. During the syntax analysis step an invalid grammar prompts the user to rewrite or delete the specified phrase. Clearly, if the use cases are specified correctly using the provided guidelines no user interaction will be necessary at all.

A. Define the Use Cases

Mandatory fields in the use case description are "*Actor*", "*Trigger*" and "*Precondition*". As shown in Fig. 1. the "*Main Success Scenario*" and "*Extension Scenarios*" fields are also necessary, since. They contain a step-by-step description of the interaction between the SuV and its environment, which is the main information for the test case generator.

As shown in Fig. 1. we have extended the common textual use case descriptions [2] by additional fields to cover non-functional requirements and constants. This is not explained here any further since it goes beyond the scope of this paper.

The value of each field in the textual use case description is described in a non-formal way. Thus, our methodology has to deal with typical natural language issues [18]. Therefore, we define a grammar and a lexical subset of the natural language to be solely used for specifying the use case. This is done in collaboration with our industry partner, who has strong experience with common grammar structures and terms used for specification. A list of guidelines is provided to keep the stakeholder to the given grammar structure and focus on terms from the lexical subset. It is not mandatory for the stakeholder to stick to these guidelines. Nevertheless, this decreases the required user interaction during the lexical and syntax analysis steps. Since not all terms can be covered by our lexical subset, the lexicon can be extended with new terms interactively by the stakeholder.

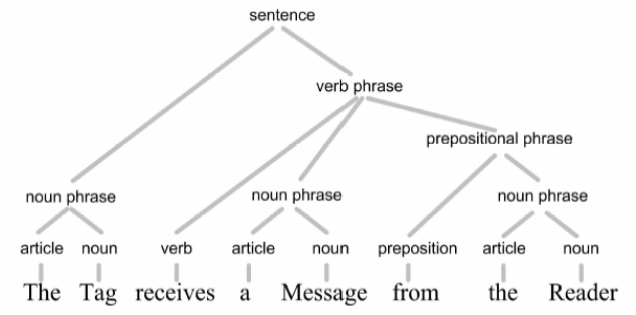

Fig. 3. Syntax Tree with type information for each term generated by the Syntax Analyzer.

B. Lexical Analysis

After the use cases have been written, a lexical analysis on each interaction step listed in the use case scenarios is performed. It checks if the specified terms exist in the lexical database. If a term is missing, the user can add it to the database together with additional information on its type (e.g. noun, verb ...). The lexical analysis identifies the type of each term [19] e.g. reader -> noun, receives -> verb ...etc.

C. Syntax Analysis

The syntax analysis checks the grammar of each interaction step. If it has an invalid grammar (i.e. it is not in the database), it cannot be parsed. In this case the user has to rewrite the entire interaction step. In case the grammar is correct, a syntax tree [19] as in Fig. 3 is generated.

D. Semantic Analysis

The semantic analysis applies linguistic techniques [13] to the generated tree structure. It identifies the meaning of each term and identifies actions, actors, subcomponents and message data. After the semantic analysis step the now syntax-corrected and unambiguous specification document is stored in XML. This provides a portable generic input for various testbench generators of different languages and domains.

E. Generate a SystemC Testbench

In this step, SystemC testcases based on the extracted information from the previous steps are generated. Each testcase is either generated from a single *interaction step* or several *interaction steps* in the use case scenarios. The set of testcases derived from an entire scenario is combined into a verification state machine. This state machine generates stimuli and checks the response of the SuV. The execution of such a state machine performs the entire interaction sequence specified in the corresponding scenario of the use case. It verifies whether the modeled system design fulfills the use case scenario.

We use the SystemC Verification Standard (SCV) [3] for our verification state machine, which provides practical support for constrained and weighted randomized stimuli and transaction recording. In addition, it allows reusing our verification components on lower levels for the functional verification of the SuV's RTL model implementation. (see Transaction-based Verification (TBV) [4]).

IV. IMPLEMENTATION

Fig. 4 shows the implementation of our approach. JAXB [20] is used to generate Java classes similar to the structure of the input XML specification document. It fills the instances of these classes with information from the XML requirements specification. An instance is created for each use case. These instances are analyzed by the phrase analyzer module. It consists of a Java CUP parser [21], which invokes JFlex [22] for each term stored in the instances. JFlex performs a lexical analysis to identify the type of each term. For syntax analysis the CUP parser uses an LR-Parser [23] to check the syntax and generates a syntax tree from each phrase stored in the use case instances (see Fig. 3). This is used by the semantic analyzer to determine the meaning of each term and to generate the error-corrected XML specification.

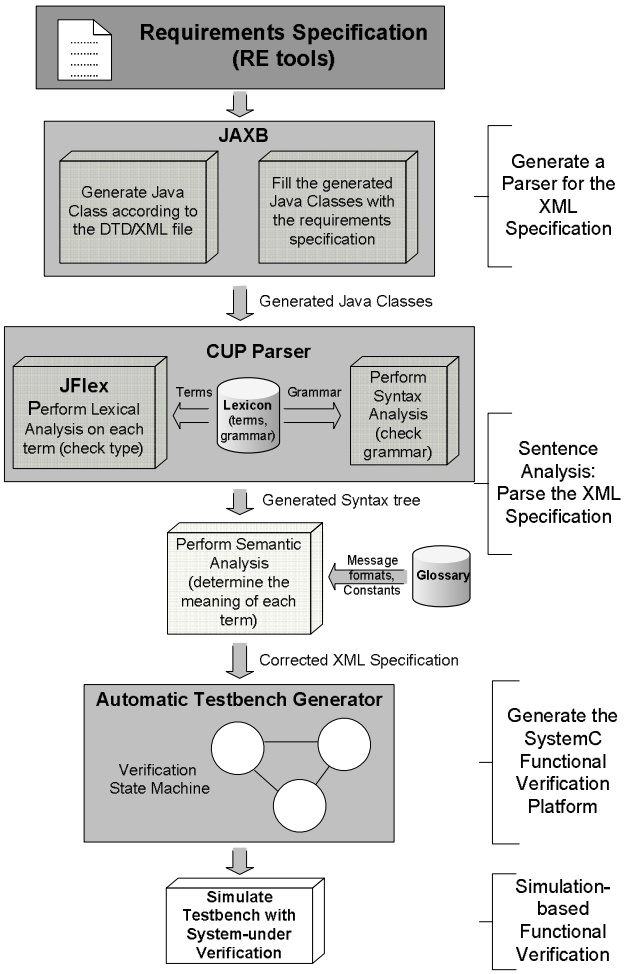

Fig. 4. The implementation is based on the Java CUP Parser [21] and JAXB [20] to extract data. This data is the input for our implemented testcase generator.

Finally, the automatic testbench generator module uses this XML specification to generate use case and testcase objects, which are used by the testcase generation algorithm illustrated in Fig. 5.

Each use case object contains a list of testcase objects, whereas each testcase object also contains a list of links to subsequent testcases as specified in the use case scenarios. This list covers all ramifications to extension scenarios (=alternative flows) at the current step (=testcase object) in the use case scenario. The testcase generator is a SystemC model. It consists of three separate threads: random test

generation, test execution and the checking of non-functional requirements.

The algorithm of the random test generation thread is shown in Fig. 5. The entire algorithm is reiterated by a user-specified number of times. During each iteration the algorithm uses SCV constructs to randomly select a use case object from the use case list. This list is generated each time the testcase generator reads the XML specification file provided by the parse analyzer. A use case may contain a list of predecessor use cases. These are defined in the pre-condition statement of the use case specification. They are executed by the algorithm before the current use case is processed. For each use case our algorithm goes through the stored list of its testcases starting at the first element in the list. If this testcase has no alternative testcases (i.e. no use case extension scenarios), it is executed. This is done by the test execution thread as explained in subsection A. It generates stimuli, estimates and stores the tag's internal state and checks the SuV response. After the execution, our algorithm turns to the next testcase from the list. If there is a list of alternative testcases available, the SCV random function is used to randomly select a testcase from the list and executes it.

This algorithm facilitates the random processing of arbitrary sequences of main and extension scenarios. It also supports dependencies between the various use cases by considering the precondition use cases.

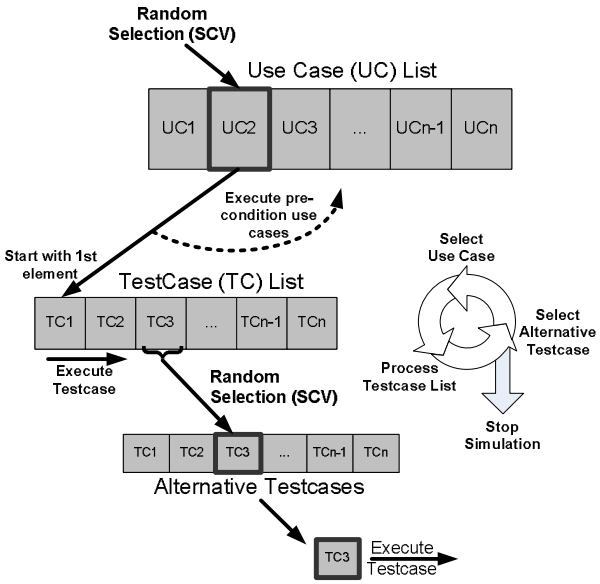

Fig. 5. Random SystemC testcase generation from use cases using the SystemC Verification Standard (SCV).

A test reporting feature during simulation prints the number of passed and failed use- and testcases. It yields the rate between the number of already processed tests and the entire set of testcases. This provides the verification engineer with an important metric to stop the verification process ahead of time if the test execution rate exceeds a certain value.

A. Test execution thread

Fig. 6 illustrates the test execution thread, which is based on the verification statemachine generated from the input XML specification.

```
int testbench::test_execution(){
 switch(state){
  case SET_UP_TAG_1_RECEIVES_ACTION :
   transmit_Message->set_value(QUERY_MESSAGE, SL_FLAG);
   send_to_DUT(transmit_Message);
   break;

  case SET_UP_TAG_4A_SETS_ACTION :
   transmit_message.storeInternalState(SL_FLAG};
   transmit_message.storeInternalState(INVENTORIED_FLAG};
   break;

  case SET_UP_TAG_PREC_0_COMES_ACTION :
   if(strcmp(last_use_case.name,"arbitrate tag")!=0){
    next_use_case_stack.pop("Set up Tag");
    goto_use_case("arbitrate tag");
   }
   break;

  case REPLAY_TAG_11B_TRANSMITS_ACTION :
   message *received_msg;
   receive_from_DUT(received_msg, PC_EPC_CRC_MSG);
   if(check_message(received_msg))
    return TB_PASSED;
   else
    return TB_FAILED;
   break;
 }
```

Fig. 6. Generated statemachine of the test execution thread.

As shown in Fig. 6 the case blocks to handle the *receive* and *transmit* actions do not invoke the corresponding SystemC write and read functions of the output and input ports connected with the SuV (=DUT: Design-under-Test). Instead, the *send* and *receive* functions (=transactions) of a SystemC transactor component, which is also generated automatically by our approach, are invoked. These functions are marked as grey-tone in Fig. 6. We use the transactor component to separate the testcase generation and execution module from the SuV's interface as shown in Fig. 8. The actual mapping of the generated testcases to the input and output ports of the SuV is "outsourced" to the transactor (also known as bus-functional model or adapter) and has to be implemented together with the transactor's RTL interface by the designer of the SuV. As shown in Fig. 7 the generated transactor component source code contains comments and placeholders to help the designer to find the right place for adding the mapping statements. Thus, the designer decides to which SuV ports to map the generated testcase messages. The generated testcase messages are derived from the "constants" field in the use case-based specification document and are defined as C++ structs, having a struct member variable for each protocol field. This allows accessing each message protocol field separately and mapping it to the corresponding SuV port by the designer.

In addition to the *receive* and *transmit* actions, the verification statemachine also handles *set* actions defined in the specification. A *set* is specified when the SuV changes its internal state. In case of an RFID specification, a *set* action can be specified as: "*The RFID tag sets its SL Flags to the Message setting"*. In this case the verification statemachine maintains an array to store the SuV internal state, since it knows the settings of the message it has sent to the SuV. The stored information can be used for testcase generation, which require the knowledge of the SuV's internal state. For instance, the *SET_UP_TAG_1_RECEIVES_ACTION* case block uses the stored RFID tag's *SL* Flag setting to determine and send a *matching SL* Flag that corresponds to the RFID tag's internal *SL* flag.

The verification statemachine also maintains the correct execution sequence of the use cases. This is shown in the case block *SET_UP_TAG_PREC_0_COMES_ACTION,* which selects the use cases listed in the precondition to be executed before the testcases of the current use case are executed.

```
#include "transactor.h"
 void send_to_DUT(message msg)
 {
  // TO BE DONE HERE: MAP MESSAGE TO PORTS

  // e.g. DUT_address_port.write(msg.address);
 }

 void receive_from_DUT(message* msg, msg_type)
 {
  // TO BE DONE HERE: MAP PORT SIGNALS TO MESSAGE
  switch(msg_type)
  {
     case PC_EPC_CRC_MSG:
       // e.g. msg->epc_pc_crc = DUT_port.read();
       break;
     //....
  }
 }
```

Fig. 7. Source code of the transactor, which is enhanced by the designer to map the testcases to the SuV interface

B. HW/SW Co-Design Tool SyAD® (System Architect Designer)

We have implemented our methodology in the HW/SW co-design tool SyAD® (System Architect Designer) [25] as shown in Fig. 8, which demonstrates the proposed linking of the testcase module with the SuV via the generated transactor component. This linking allows using the same testcase module for different interfaces of the SuV and it also enables to reuse the same testcase module for various SuV's, which implement the same interface.

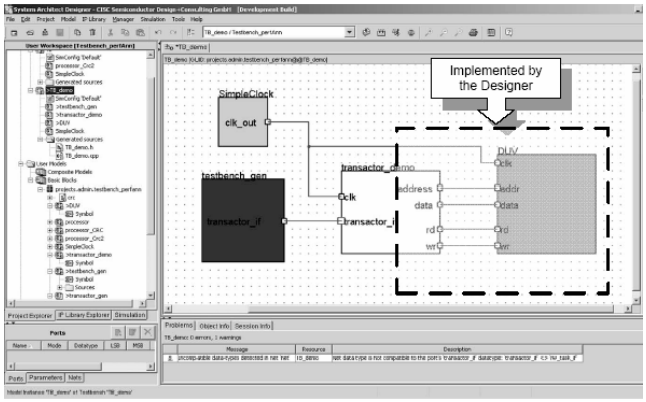

Fig. 8. Proposed testbench architecture consisting of the testcase generator, the transactor and the SuV (=DuV: Design-under-Verification) modeled in the co-design tool SyAD [25].

SyAD® enables the development of system-level HW/SW co-designs and supports a multi-language and multi-level co-simulation framework supporting SystemC, VHDL, VHDL-AMS and MATLAB Simulink.

V. CASE STUDY: A SOC-BASED RFID CONTROLLER

As a case study we have considered a use case-based specification of an RFID controller provided by our industrial partner. The use case specification is derived from the RFID controller state diagram specified in the EPCGlobal Class-1 Generation-2 UHF RFID protocol for communications [24]. It covers the entire controller state diagram (see Fig. 6.19. in [24]) and encompasses 53 use case scenarios (see Table 1). Fig. 9 shows a small excerpt from the generated use case-based specification document.

Name: Set up Tag
Description: Use case accessed when tag enters the Reader field.
Scope: UHF RFID Tag (=Tag).
Primary actor: Interrogator (=Reader)
Precondition/Trigger: Tag (re)-enters the Reader Field
Main Success Scenario:
1. Tag receives Query command from Reader with Receiver unit
...
10. Tag transmits 16bit Random number with Backscatter unit.
11. Tag exits use case and goes to "Reply Tag" Use Case
Alternate Flows:
1a. Tag receives Select command from Reader with Receiver unit
...
LocNonfunctional Requirements:
Timing Constraints: Step 1 until step 11 shall be done within t1.
Power Constraints: The power consumption in step 1 shall not exceed 1mW
Others: Tag shall work in a range between 800Mhz – 900Mhz
GlobNonFunctionalRequirements:
Timing Constraints: The tag shall respond within t1
Power Constraints: The power consumption shall not exceed 1mW
Others: …

Fig. 9. Specified use case derived from the protocol specification of an RFID controller state machine.

The illustrated use case is accessed when the RFID controller enters the reader field (see Precondition/Trigger).

Fig. 10 demonstrates the application of our methodology to the first interaction step in the "*Main Success Scenario*" of Fig. 9. It is defined as *"Tag receives Message from Reader with Receiver unit"*. All the tasks in Fig. 10, apart from mapping the testbench ports to the SuV, are processed automatically, without any user interaction. This is due to the correct syntax of the phrase and the fact that no unknown terms are used.

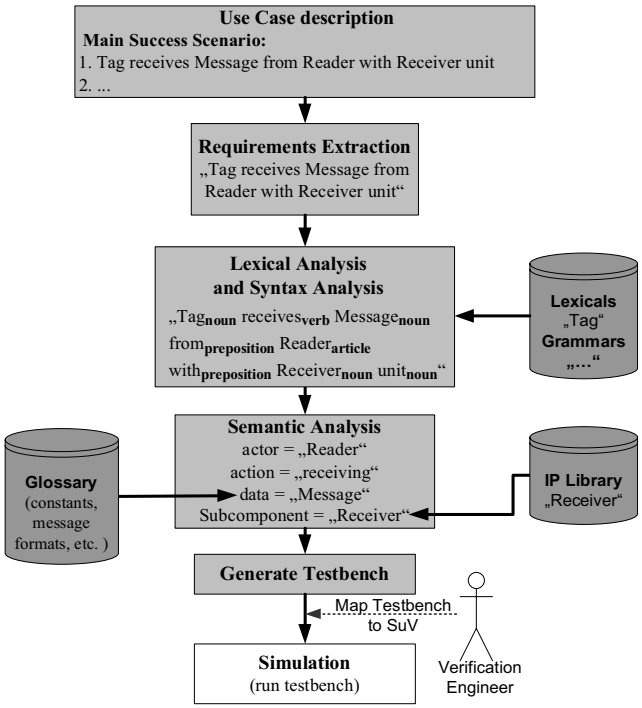

Fig. 10. The application of our methodology to the use case specification of an RFID Tag.

VI. RESULTS

We applied our approach to a SystemC transaction-level model of an RFID controller. First, we used our methodology to check the correctness of the use case specification document of the RFID controller. Table 1 lists the results. We discovered 6 syntax errors (due to missing verbs and articles) and added 8 unknown terms to the RFID specific lexicon.

Table 1. Recorded results when applying our approach to the entire RFID use case specification.

Characteristics	Tag Specification
Syntax Errors (missing verb, article,…)	6
Unknown terms	8
#Use Cases/ #Use Case Scenarios	5 / 53
# generated Testcases	131

In a second step we generated the verification statemachine from the now corrected specification document. Our testcase generator generated 131 testcases from the specification. We then started the simulation to demonstrate our random testcase execution algorithm (see Fig. 5) for 5, 10, 15 and 20 iterations.

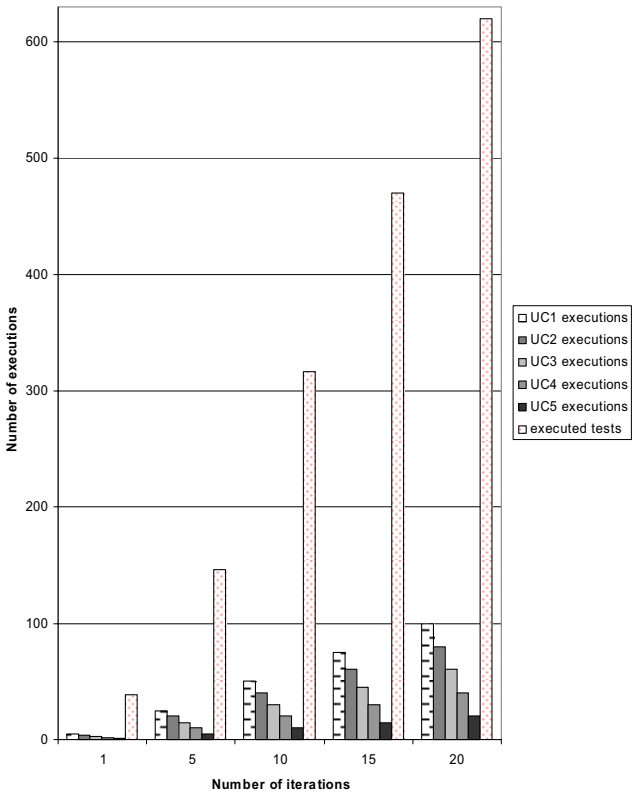

Fig. 11. Number of executions of each use case and number of executed testcases with the random testcase execution algorithm for 5, 10, 15 and 20 iterations

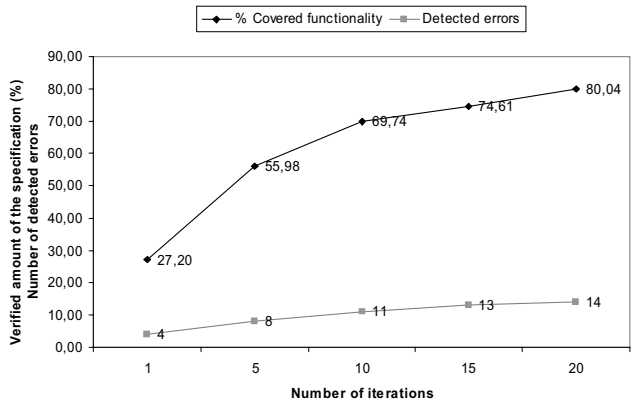

Fig. 12. Covered portion of the specification (=executed use case scenarios) and number of detected errors by the random testcase execution algorithm for 5, 10, 15 and 20 iterations

We used the generated report statements during simulation to determine the executions of each use case, each passed use case scenario and the number of executed tests as shown in Fig. 11. Use case 1 (UC 1) is executed most of all, since it is in the precondition list of all other use cases. In contrast, UC 5 is less often executed since it does not occur in the precondition list of any other use case.

The ratio of executed use case scenarios to the total number of use case scenarios specifies the covered amount of the specification by the simulated testcases. Fig. 12 shows a comparison of the number of identified errors and the verified portion of the specification at 5, 10, 15 and 20 iterations. A 80% functional coverage detects 14 errors and requires 20 iterations of the testcase execution algorithm, which results in more than 600 executed tests as illustrated in Fig. 11.

VII. CONCLUSION AND FURTHER WORK

In this paper we presented a novel design and verification methodology for System-on-Chip (SoC) designs. Common flows lack a sophisticated support for specification checking and automated specification-based functional verification of the modeled system. This is due to the missing link between the specification and the implemented design. Our methodology closes this gap by tightly integrating the specification into the design flow. Testcases are automatically derived from widely accepted use case-based specifications and are used for functional verification by simulation.

Our approach focuses on use case-based specifications, which are suitable for describing protocols. We therefore use a case study based on the semi-formal specification of a higher class RFID controller to demonstrate and prove our methodology. We showed that our methodology can be used to automatically generate SystemC testcases to verify whether the design fulfills its specification throughout the development process. This reduces the time for testcase generation and the number of required redesigns later in the design flow. The applicability of our approach is limited to the external interface of the System-under-Verification (SuV). If there is almost no communication with the external world, black-box tests are not sufficient to verify the system. Therefore, as a further step, we plan to enhance the methodology with white-box verification features by integrating automatically generated interface monitors.

Much more designs have to be verified with our approach to prove it. This is reflected in our current work.

Ongoing research also focuses on the integration and verification of power constraints with the demonstrated approach.

REFERENCES

[1] Andrew Piziali. Functional Verification Coverage Measurement and Analysis. Kluwer Academic Publishers, 2004.

[2] A. Cockburn. Writing Effective Use Cases. Addison-Wesley Professional, 2001.

[3] S.Swan, C.Norris. A tutorial introduction on the new systemc verification standard. Technical report, 2003.

[4] The Transaction-Based Verification Methodology. Technical report, Cadence Labs, 2000.

[5] Q. Zhu, R. Oishi, T. Hasenawa, and T. Nakata. System-On-Chip Validation using UML and CWL. In Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on, pages 92–97, 2004.

[6] JinShan Yu, Tun Li, and QingPing Tan. The Use of UML Sequence Diagram for System-on-Chip System Level Transaction-based Functional Verification. In Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on, volume 2, pages 6173–6177, 21-23 June 2006.

[7] L.H. Tahat, B. Vaysburg, B. Korel, and A.J. Bader. Requirement-Based Automated Black-Box Test Generation. In Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International, pages 489–495, 8-12 Oct. 2001.

[8] C. Nebut, F. Fleurey, Y. Le Traon, and J.-M. Jezequel. Requirements by Contracts allow Automated System Testing. In Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on, pages 85–96, 17-20 Nov. 2003.

[9] Ajitha Rajan. Automated requirements-based test case generation. SIGSOFT Softw. Eng. Notes, 31(6):1–2, 2006.

[10] Jan Tretmans and Ed Brinksma. Torx: Automated model based testing - cˆote de resyste.

[11] J.R. Abrial. The B-book : assigning programs to meanings. Cambridge University Press, August 1996.

[12] M. Riebisch and M. Hubner. Traceability-driven Model Refinement for Test Case Generation. In Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the, pages 113–120, 4-7 April 2005.

[13] A. Fantechi, S. Gnesi, G. Lami, and A. Maccari. Application of Linguistic Techniques for Use Case Analysis. In Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on, pages 157–164, 9-13 Sept. 2002.

[14] C. Denger, D.M. Berry, and E. Kamsties. Higher Quality Requirements Specifications through Natural Language Patterns. In Software: Science, Technology and Engineering, 2003. SwSTE '03. Proceedings. IEEE International Conference on, pages 80–90, 4-5 Nov. 2003.

[15] S.F. Tjong, N. Hallam, and M. Hartley. Improving the quality of natural language requirements specifications through natural language requirements patterns. In Computer and Information Technology, 2006. CIT '06. The Sixth IEEE International Conference on, pages 199–199, Sept. 2006.

[16] D. Myers, N. Vincent, K. O'Loughlin, and Marks. Simulation-Based Requirements Testing. In Systems and Information Engineering Design Symposium, 2003 IEEE, pages 189–194, 24-25 April 2003.

[17] V. Gervasi and B. Nuseibeh. Lightweight Validation of Natural Language Requirements: a case study. In Requirements Engineering, 2000. Proceedings. 4th International Conference on, pages 140–148, 19-23 June 2000.

[18] Centre for Language Technology. Controlled Natural Languages, 2007.

[19] E. Loper, S. Bird, E. Klein. Introduction to Natural Language Processing. 2001.

[20] Sun. Java architecture for xml binding (jaxb), 2003.

[21] Cup - lalr parser generator for java, 2007.

[22] JFlex - the fast scanner generator for java, 2007.

[23] Lang B.G Naumann. Parsing. 1994.

[24] EPCGlobal. EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz ?U 960 MHz, 1.0.9.

[25] A. Schuhai, M. Pistauer, S. Kajtazovic, C. Steger. Automatic generation of a verification platform for heterogeneous system designs. In Advances in Design and Specification Languages for SoCs - Selected Contributions from FDL'05, 2005.

Simulation-based Verification of Power Aware System-on-Chip Designs Using UPF IEEE 1801

Christoph Trummer, Christoph M. Kirchsteiger, Christian Steger, Reinhold Weiß*,
Damian Dalton† and Markus Pistauer‡
*Institute for Technical Informatics, Graz University of Technology, Austria
email: (trummer, c.kirchsteiger, steger, rweiss)@tugraz.at
†School of Computer Science and Informatics, University College Dublin, Ireland
email: damian.dalton@ucd.ie
‡CISC Semiconductor Design+Consulting GmbH, Austria
email: m.pistauer@cisc.at

***Abstract*—For System-on-Chips (SoCs) the most critical design constraint is power dissipation. Therefore, power aware design should be introduced at early stages of SoC design where it has the highest benefits for power reduction. This also lowers the design complexity and verification effort. Until recently, capabilities to describe and verify the power design early were inadequate which often led to late re-design. Lately, the IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits, an extension of the Unified Power Format (UPF) was approved. This work uses the new IEEE 1801 standard to describe power aware design. The power design is automatically translated into an executable hierarchy parallel to the system design. Simulation results from system and power design are used to automatically verify the SoC's power aware design against its specifications.**

I. Introduction

Due to rising silicon integration density today's System-on-Chips (SoCs) are able to perform more functionality. However, the increase in functionality causes higher power dissipation [1]. Consequently, power consumption is the most critical constraint for SoCs [2]. Moreover, the increasing functionality also rises complexity for specification, design and verification of SoCs. Verification uses up to 70% of the design effort [3]. To counter the rising complexity implementation details are abstracted away in early design stages. This allows to focus on behavior and enables early verification. To reduce the system's power dissipation power aware design is used. Despite its importance power design is often added at late stages to the system [4]. This has two reasons. First, functionality to reduce power dissipation is part of the pervasive functions. Pervasive functions are operations beyond normal system operation. Therefore, they do not directly contribute to the SoC's main functionality and are often overlooked [5]. Second, is the lack of means to adequately express and verify power design at high levels of abstraction [6]. With the Unified Power Format (UPF) and Common Power Format (CPF) two formats to describe the power design were introduced [7], [8]. They enable power aware design and verification at Register Transfer Level (RTL). However, the earlier power design is added to the system the higher the benefit for power reduction [1] and the earlier it can be verified. Early verification benefits from fast simulation and helps to lower the risk of late and costly re-designs [5].

We introduce a novel methodology enabling automated simulation-based verification of the power-aware design at system-level. Our methodology comprises three stages:

In the first stage the requirements for power aware design are analyzed and refined into semi-formal use cases. Each use case describes the system's functionality and power state. By expressing the requirements as use cases faults from ambiguity and misinterpretation are reduced. The semi-formal nature of our use cases enables automated parsing and analysis.

The second stage is the design stage. Based on the specifications the system engineers create the SoC design in a hardware description language (HDL). We describe the SoC design in SystemC beginning at system-level. In parallel, the power design is developed using the IEEE 1801 standard (UPF).

In the third stage the UPF power design is analyzed and automatically translated into SystemC modules. The use cases are parsed and a verification environment is automatically created. The SoC design and the power design are simulated. The verification environment launches test cases derived from the use cases invoking an equivalent behavior in the system. Finally, results from the verification environment and the supply network are automatically evaluated verifying the system.

Our contribution comprises automatic generation of the verification environment and the executable supply network from the use cases and the UPF design. Additionally, by simulating the executable supply network and the verification environment the system is verified automatically. The main benefits are the high degree of automation and the capability to simulate and verify power aware design at system-level. The automatic generation of verification environment and executable supply network reduces the verification effort. Since, power design can be added at early design stages the high potential for power reduction is retained. Early verification of the power design lowers the risk of late and costly re-design.

This paper is organized as follows. In section II background and related works about specification and verification of power aware design are analyzed. Section III provides details about the three stages of our methodology. The case study in section IV applies our methodology to a SoC design. Finally, section V summarizes our work and outlines future work.

978-1-4244-4311-6/09/$25.00 ©2009 IEEE

II. Background and Related Work

A. Use case specification for hardware design

The Unified Modeling Language (UML) can be used to specify use cases for hardware design. With an analysis model the requirements are refined into use case-, collaboration- and class diagrams [9]. The benefit of the formal UML use cases is that they can be automatically interpreted. However, formal descriptions are not easy to read without prior knowledge of the format [10]. A semi-formal use case format is described in [11]. The format is a mixture of a formal structure and informal text. Therefore, the textual use cases have a predefined structure which consists of a series of steps and interactions expressed in natural language [11]. In contrast to formal use cases, semi-formal use cases can be understood more easily and automatic processing is also possible [10].

B. Automated test case generation from use cases

Automatic deriving test case from use cases and generating a verification environment is presented in [10]. The use cases are similar to the semi-formal use case format described in [11]. The structured use cases are parsed and interpreted with semantic analysis. A verification environment is created automatically for verifying a model of the system in a Hardware Description Language (HDL). During simulation the verification environment applies test cases to the system model. This invokes a behavior corresponding to the functionality described in the use case. By monitoring the system's behavior and response the functionality is verified [10].

C. Power design formats

To counter the rising complexity in system design Hardware Description Languages (HDLs) with higher levels of abstraction were developed. To reduce the predominant dynamic power dissipation adding power design information was not necessary at high level [6]. Therefore, it was introduced to the system at a late stage (below RTL) [6], [12]. Due to the increase in static power at process technologies below 100 nm power suddenly became a highly critical design constraint for SoCs. Power management and power aware design became important and the capabilities of HDLs to express power design were insufficient. Because of the benefits of early power aware design the Electronic Design and Automation (EDA) industry realized the need for a standardized, HDL-independent power specification format. Two similar formats to specify the power design were developed by industrial consortiums, the Common Power Format (CPF) and the Unified Power Format (UPF) [7], [8]. With the emergence of these specification languages power design can be described at RTL without modifying the system design [2], [12], [13]. Recently, the IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits was approved [6]. The new IEEE 1801 standard extends UPF.

D. Verification of the power design

In general, verification of the power design falls into three steps. First, it is verified that the design correctly enters the power state. Second, the system has to provide the correct result at the each power state. Third, it is ensured that the logic behaves correctly when a power state is entered e.g. the unit has no activity when switched off [5]. Two approaches use UPF to describe the power design and perform simulation-based functional verification at RTL [2], [13]. In [13], behavioral models for the power aware functionality are simulated together with a testbench and the system. However, the power aware models and the testbench need to be created manually.

Many functional defects can be exposed by simulating the power design. The most important are incorrect switching of supply voltages for power domains, falsely timed state transitions, wrong power states, improper save/restore sequences for retention. Moreover, inadequate isolation, problems to reset blocks to a good state when power is restored and incorrect level shifting between domains can also be detected [2], [13].

III. Methodology for Simulation-based Verification of Power Aware SoC Designs

A. Specification Stage

In the specification stage the SoC's requirements are refined into a semi-formal use case document with three sections:

The first section, *Use Cases*, contains all use cases as subsections. Each use case describes functionality as stepwise interactions between the system and the actor(s). The most important parts of an use case are as follows (compare [10]).

- **Use case:** specifies the use case name
- **Description:** contains a short abstract of the use case
- **Scope:** states the system to which the use case applies
- **Actors:** refers to entities interacting with the system
- **Primary Scenario:** contains a series of interaction steps in natural text which may contain constants
- **Alternative Scenarios:** lists deviations from the primary scenario which branch into a separate series of steps

The second section is the *Power State Scenario* section. It consists of three subsections specifying the power design.

- **State Description:** the system's state for each use case
- **Power State:** lists power states as set of domain supply states, specifies when isolation and retention are active
- **Power Domains:** specifies all of the system's modules part of the power domain and specifies valid supply states

The third section of the use case document is the *Constants* section. It defines variables used in the interaction steps of the use cases. Each constant is assigned a value and data type. This specifies which data the actor exchanges with the system.

The interactions between actor(s) and system are described as natural text in the use case's primary and alternative scenarios. After the use cases are finished the Power State Scenario is specified. The Power Domain section states a set of domains. Each power domain comprises a list of modules which are part of the domain and a list of domain supply states. In the Power State part the power state is defined as natural text by assigning a supply state to each power domains. Moreover, it specifies when isolation and retention are active. As a result, each State Description contains a textual description of the power state the system is in during an use case.

B. Design Stage

The designers create the system according to the use case document. During design the system is divided into functional units and modules. Each module is described in the SystemC HDL. At system-level the modules communicate via interfaces and transactions. When the system is refined communication is performed via signals. Our methodology supports interfaces and signals. However, each interface has to implement the SystemC "default event". This is necessary for the monitor in the power domain to detect activity in inactive domains. Controller modules need to use signals to interact with the power design to be compatible with the IEEE 1801 standard.

While the system is created, the power design is elaborated in UPF. Power domains are described and connected to the supply network. Power switches to change domain supply states are created. For communication between domains with different supply voltages level-shifters are defined. Isolation is used to avoid undefined signal values when a domain is switched off. When the supply of a domain is turned off the modules in the domain lose their internal state. To avoid the loss of information retention is applied to a few crucial modules to save their internal states. The power state table summarizes supply ports and their states into power states. The isolation, retention and the power switch elements are controlled by signals from the system design.

C. Simulation and Verification Stage

Before the system is simulated, the executable supply network and verification environment are generated automatically. Therefore, the use case document is parsed and its sections are interpreted. Initially, actor, system and references to constants are determined for each use case. Then the sequence of steps is examined. The semantic analysis identifies the actions, the direction and constants in each sentence. In this way test cases are derived the interaction steps of each use case [10]. Then the verification environment is created. It contains a schedule for each test case. During simulation it launches test cases and triggers a corresponding behavior in the system [10]. The verification environment logs simulation time, executed use case and test case, the stimuli and system responses.

To simulate the power design an executable supply network is generated. Information about modules, ports and connections are extracted from the design. Thereafter, the power design is parsed and automatically translated into a parallel hierarchy. For each element in the power design a corresponding module is created in SystemC. Inside the modules a monitor logs the simulation time, internal state and supply information.

For the UPF supply ports SystemC modules are created accordingly. Each module propagates its voltage level to the connected supply network as floating point number. If a supply port is off or at ground negative infinite is assigned.

The power domains are translated into SystemC modules containing the specified system modules as sub-modules. The domain's supply and ground are connected to the supply network. The monitor logs the domain's supply state. It also listens for signal and interface activity when the domain is off.

The generated SystemC isolation module monitors its supply state, isolation control and correct isolation values.

Modules are created containing elements for which retention applies. The supply, save and restore signals are monitored.

The level shifters translate a domain's communication ports to the appropriate voltage levels. From UPF SystemC modules are created to monitor correct shifting between voltage levels.

To switch between supply voltages and to deactivate domains power switches are created. The switch is sensitive to the signal of its controller module. Depending on the control signal the switch applies a supply or "off" state to its output.

Finally, the specified supply network is used to properly connect the newly generated power design modules.

When the verification environment launches test cases during simulation different power states are entered. The modules in the executable supply network propagate their state throughout the network. When the control signals for power switches, isolation and retention are applied the monitors inside each module log the sequence of events. After simulation the logged events are automatically evaluated. For each use case iteration the specified and simulated power states are compared. The state for each domain is determined, compared to the specification and activity inside a deactivated domain is reported. Sequences for control signals and supply states for isolation and retention are verified. Finally, the system designer receives a verification report describing all discovered faults.

IV. Case study

To demonstrate our methodology we implement a higher-class RFID tag as SoC. The higher-class tag (HCT) is equipped with a temperature sensor and extended memory. Initially, we analyzed the ISO/IEC 18000-7 protocol [14] for higher-class tags. Use cases are specified for the wakeup, collection, read, write and sleep commands. More use cases for the sampling, standby and idle are created. When the RFID reader sends a command the tag acts accordingly. The tag enters the standby use case if it receives a sleep command. After an interval, set by the write command, a timer interrupt occurs. A sample is taken from the sensor and stored in memory. Every 2.5 sec. the tag awakes from standby to determine if the RFID reader has sent a wakeup command. If wakeup is detected the HCT enters idle mode waiting for further commands, else it returns to standby. The functionality to enter a power state is described in the use cases. The use cases specify the actions of the power management unit while their three power states are specified in the Power State Scenario section (see excerpt in table I).

The Power State section provides details for each power state. After a wakeup command, in idle mode or when processing a command all power domains are on (ALL_ON). When the tag receives a sleep command it enters standby and all domains except PD0 are off (ALL_OFF). In this way the power management unit remains able to control the switches. To take a sample a timer interrupt occurs and the SAMPLE_ON state is entered. In SAMPLE_ON the domains for controller, sensor and memory are active, while the transceiver remains off. Four power domains are specified. The domain PD0 with power

TABLE I
EXCERPT FROM THE POWER STATE SCENARIO SECTION

State Description
While in STANDBY the Tag is ALL_OFF.
For WAKEUP the Tag goes to ALL_ON.
At READ, WRITE, IDLE and COLLECTION the Tag is ALL_ON.
During SLEEP the Tag goes to ALL_OFF.
For SAMPLING the Tag is in SAMPLE_ON.
Power State: SAMPLE_ON
At entry: Retention RESTORE is HIGH for PD3.
Description:
PD0 is ON.
The domain PD1 is in state OFF.
PD2 is in the LowVoltage state.
Power domain PD3 is ON.
At exit: Retention SAVE is ON for PD3.
Power Domain: PD2
Elements: Controller; Digital Sensor;
Domain States: OFF; LowVoltage, 2 V;

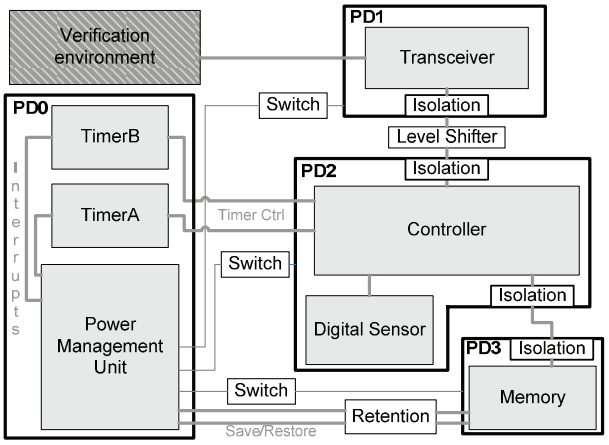

Fig. 1. Case study: power aware design of a higher class RFID tag

management unit and the timers is always active. PD1 contains the transceiver unit. The controller and sensor unit are in PD2. For the memory in PD3 retention is defined. All domains are supplied with 2 Volts except PD1 which needs 3 Volts. Table I provides an example of the power domain specification.

Based on the use cases the system and power design are created. The modules for the HCT are written in SystemC. The power domains, switches, isolation, retention and supply network are defined in UPF. Fig. 1 depicts a schematic of the system and power design. From the use cases and the power design the verification environment and executable supply network are generated. After initial functional verification additional mistakes are deliberately inserted into system and power design. During simulation the monitors create a report which shows the detected faults. Table II only contains some inserted faults and verification results due to space limitations.

V. CONCLUSION AND FUTURE WORK

This paper demonstrates a methodology to verify power aware SoC designs. In semi-formal use cases the system and its power states is specified. Then the SoC is developed in SystemC and the power design is created using the IEEE 1801 standard (UPF). The use cases are automatically analyzed and a verification environment is generated. From UPF the power design is translated into SystemC models with internal monitors. During simulation the verification environment launches test cases to exercise the system. This triggers different power states. Results from monitors and verification environment are automatically verified against the specifications. In a case study a higher class RFID tag is specified and designed. Faults were deliberately inserted into the SoC design to show the capabilities of our approach. After automatically evaluating the simulation results all faults were detected.

Currently, we are researching how the SystemC Verification Standard could enhance our methodology. Moreover, we plan to integrate custom supply modules such as battery models.

TABLE II
EXCERPT FROM THE VERIFICATION RESULTS

Inserted Fault	Specification	Simulation
Activity when domain is off	PD2 is OFF	PD2 has activity
Incorrect domain state	PD2 is ON	PD2 is OFF
Wrong power state in IDLE	ALL_ON	SAMPLE_ON
No isolation	PD1 inputs to latch	No isolation in PD1
Isolation not active	PD2 inputs to latch	PD2 isolation OFF
No level shifting	PD1 3V; PD2 2V	No level shifter at PD1 or PD2
Wrong level shifting	PD1 3V; PD2 2V	Inputs PD1 high-to-low
Wrong retention save-restore	Entry ALL_OFF save	Entry ALL_OFF restore

REFERENCES

[1] H. Jian and S. Xubang, "The Design Methodology and Practice of Low Power SoC," in *Embedded Software and Systems Symposia, 2008. ICESS Symposia'08. International Conference on*, 2008, pp. 185–190.
[2] A. Crone and G. Chidolue, "Functional Verification of Low Power Designs at RTL," *Lecture Notes in Computer Science*, vol. 4644, pp. 288–299, 2007.
[3] R. Lissel and J. Gerlach, "Introducing new verification methods into a company's design flow: an industrial user's point of view," in *Design, Automation & Test in Europe, Conference & Exhibition DATE'07*. IEEE, April 2007, pp. 689–694.
[4] W. Nebel, "System-Level Power Optimization," *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pp. 27–34, 2004.
[5] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification the Complete Industry Cycle*. Elsevier/Morgan Kaufmann, 2005.
[6] "IEEE Standard for Design and Verification of Low Power Integrated Circuits," *IEEE Std 1801-2009*, pp. C1–218, 2009.
[7] Accellera, "Accellera: Unified Power Format (UPF) 1.0 Standard," pp. I–96, February 2007.
[8] The Power Forward Initiative (PFI), "A Practical Guide to Low-Power Design - User Experience with CPF," pp. 1–281, 2008.
[9] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.
[10] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiß, and M. Pistauer, "Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs," in *Systems Conference, 2008 2nd Annual IEEE*. IEEE, April 2008, pp. 1–8.
[11] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.
[12] S. Bailey, "Kick-Starting RTL Power-Aware Verification," *Chip Design Magazine*, vol. October/November 2007, 2007.
[13] F. Bembaron, S. Kakkar, R. Mukherjee, and A. Srivastava, "Low Power Verification Methodology Using UPF," in *Conference on Electronic Systems Design and Verification Solutions, DVCON*, 2009, pp. 228–233.
[14] International Standardization Organization, "ISO/IEC 18000-7:2008 - Information technology – Radio frequency identification for item management – Part 7: Parameters for active air interface communications at 433 MHz," 2008.

2009 IEEE International Conference on RFID 3B.4

Verification Methodology for Battery Lifetime Requirements of Higher Class UHF RFID Tags

Christoph Trummer, Christoph M. Kirchsteiger, Alex Janek, Christian Steger, Reinhold Weiß*,
Markus Pistauer† and Damian Dalton‡
*Institute for Technical Informatics, Graz University of Technology, Austria
email: (trummer, c.kirchsteiger, alex.janek, steger, rweiss)@tugraz.at
†CISC Semiconductor Design+Consulting GmbH, Austria
email: m.pistauer@cisc.at
‡School of Computer Science and Informatics, University College Dublin, Ireland
email: damian.dalton@ucd.ie

***Abstract*—Today's higher class tags usually are powered by batteries. The battery's capacity and the application's power demand influence the operational lifetime of the tag. Therefore, the designated application and lifetime requirement have to be kept in mind when designing a higher class tag. Moreover, the lifetime requirement needs to be verified in order to ensure the application will be successful. However, verification of the lifetime requirement is usually a very complex task. A verification environment for the application and its lifetime requirement needs to be created manually. After simulation with a battery model the results can be compared to the requirements document. Due to the complex and time-consuming nature of verification this often results in later time-to-market and increasing costs. In this work we present a novel, highly automated methodology to verify battery lifetime requirements. From the requirements document of the higher class UHF RFID tag a verification environment is created automatically. After power estimation is performed a battery model can be connected to the automatically generated lifetime verification environment. Finally, simulation is performed to verify whether the higher class UHF RFID tag fulfills the lifetime requirement of the application. The main benefit of our methodology is a decrease in the verification effort due to the high degree of automation in the creation of the verification environment. Moreover, simulation time is decreased which enables faster exploration of various batteries. This results in faster time-to-market and a reduction of costs.**

I. Introduction

Nowadays, ultra high frequency (UHF) RFID is used in a wide variety of applications. Consequently, UHF RFID systems have to fulfill a rising number of requirements. Some of these requirements are high operating ranges, sensing and monitoring, large memory and processing capabilities for the measured data [1]. These systems are called higher class UHF RFID tags [2]. When designing a higher class RFID tag to meet the requirements, developers are facing an increasing complexity. Moreover, the requirements concerning the functionality of the system and non-functional requirements also have to be verified. Non-functional requirements such as costs and operational lifetime are especially important for higher class UHF RFID tags [1]. As for all battery-powered devices the lifetime of higher class UHF RFID tags mainly depends on its power dissipation and the battery's capacity. In turn battery capacity affects size and overall cost of the system.

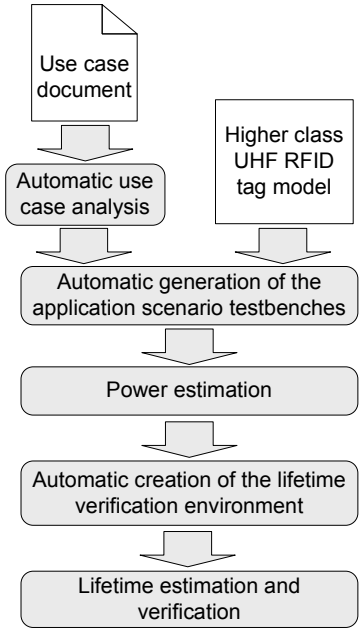

Fig. 1. Principle of the methodology for verifying lifetime requirements

Due to the high system complexity, 70% of the entire design effort is spent to verify the requirements are fulfilled [3]. Battery lifetime requirements are verified using a battery model which is simulated with a model [4] or a pre-recorded power profile of the system [5]. Either approach requires a testbench for the system which is usually created manually. Testbench and battery model are simulated and the results are often manually compared against the requirements.

In this paper we propose a novel approach that allows to specify and verify lifetime requirements for designated applications in an highly automated flow (Fig. 1). Therefore, the applications are more tightly integrated into the requirements analysis process. This helps the developers to better understand conditions, environment and context the system operates in.

978-1-4244-3338-4/09/$25.00 ©2009 IEEE

Additionally, the specified application contains the lifetime requirement. Through semantic analysis the applications and their lifetime requirements are automatically evaluated. Then a verification environment is generated for power estimation of the device's application. Finally, the application's lifetime requirement is verified by simulating the estimated power demand with a battery model.

The main benefit of our approach is its high degree of automation. Thus, the effort spent in the verification of the system is greatly reduced resulting in a faster time-to-market. The major contribution of our methodology is the automated analysis of the requirements and the automatic generation of the lifetime verification environment. Importantly, also simulation speed is greatly increased performing faster battery lifetime estimation. In addition, the actual verification of lifetime requirements for target applications can already be carried out at early stages of the design. This permits to rapidly explore the impact of different battery capacities on the lifetime and helps in selecting the appropriate battery for the application of the higher class tag.

The remainder of this paper is organized as follows. In section II related work is analyzed. Our proposed methodology is explained in section III. We validate our approach in section IV by estimating the lifetime of an higher class UHF RFID tag. In a case study (section V) of a higher class tag in aircraft landing gear monitoring the verification of lifetime requirements is demonstrated. Finally, we summarize our approach and outline future work in section VI.

II. Background and related work

A. Specification of lifetime requirements

A key factor in the operational lifetime of a higher class RFID tag is the lifetime of its battery. Battery lifetime is mainly influenced by the tag's power dissipation. Depending on the task it performs the higher class tag consumes more or less power. Thus, to estimate power consumption and battery lifetime the application has to be taken into account. An application consists of a series of tasks which are performed by the device. For a higher class UHF RFID tag this could be to communicate with a reader and/or other active tags, read sensor data and store the samples in memory. In [4] the authors mention that operational lifetime requirements are usually specified in terms of minimum battery lifetime. Sometimes, manufacturers specify the lifetime of their higher class RFID tag for a certain amount of readouts per day (see [6]). This is often an abstract generalization of a typical application.

Examples of UHF RFID applications with lifetime requirements are tire pressure monitoring (TPM) and aircraft landing gear monitoring. In active TPM systems a higher class RFID tag is integrated into a car tire. The tag continuously measures inflation pressure and tire temperature [7]. Typically TPM systems require lifetimes of more than 10 years [7], [8]. In airplanes a higher class tag can be integrated into landing gear to monitor brake temperature or pressure on the shock absorber. Parts of the aircraft landing gear are durable and typically have long lifetimes before being replaced [9]. Consequently, a higher class UHF RFID tag monitoring a landing gear needs to have a lifetime of 7-12 years [9].

B. Requirements specification for hardware design

In [10] the authors present a methodology which uses the unified modeling language (UML) to specify requirements in hardware design. Through an analysis model the use cases are refined into use case-, collaboration- and class diagrams [10]. With this methodology test cases have to be derived manually from the descriptions and diagrams. Non-functional requirements can be included into the use case descriptions and diagrams [10]. The format to express hardware requirements in UML is similar to the textual use cases described in [11]. These use cases have a predefined structure which consists of a series of steps and interactions expressed in natural language. A structured use case format with text in natural language is adequate to capture requirements of higher class RFID tags.

C. Automated verification platform generation from use cases

A verification platform can be automatically generated from textual use cases [12]. A use case format similar to the one described in [11] is employed to capture functional requirements. Through semantic analysis of the structured use cases a verification platform is created automatically. It is used to verify a model of the system in a hardware description language. During simulation the verification environment sends stimuli derived from the use case descriptions to the system model. This provokes a behavior corresponding to the functionality described in the model's use case. By monitoring the model's response the functionality is verified [13]. However, this approach only considers functional requirements for test case generation, non-functional requirements are not verified.

D. Battery lifetime estimation

To estimate the lifetime of batteries various battery models are used to simulate or calculate battery lifetime for different load conditions. Many different battery models exist with a wide variety of accuracy and calculation effort [14]. In most of the cases the battery models are simulated with a model of the system or are connected to pre-recorded power profiles to estimate the battery lifetime [4], [5].

E. Functional verification of higher class UHF RFID tags

In [1] requirements of higher class UHF RFID tags are identified. It is emphasized that the most important non-functional requirements are costs and lifetime. The authors developed a high-level simulation model of a higher class UHF RFID tag to verify its functionality. For components contributing the most to power consumption more detailed models are employed. These models are used to accurately simulate the higher class tag's power dissipation [1]. Additionally, models of a battery and energy harvesting devices are used to simulate the tag's power supply. In a proceeding work the authors use a system-on-chip development platform to implement their higher class UHF RFID tag in hardware [15].

III. Battery lifetime verification methodology

For successfully verifying the lifetime requirements of higher class UHF RFID tags its requirements need to be specified. To automate analysis of the requirements they are expressed as textual use cases. The use case document contains the description of the higher class tag's functionality as a series of interaction steps. Moreover, it specifies potential applications for the tag and states the lifetime requirement of the tag's energy source. Next, the use cases are automatically parsed and analyzed. Thereafter, a testbench corresponding to the application is automatically generated for the model of the tag. It is used to exercise the higher class UHF RFID tag's functionality for power estimation. This results in a power profile for the application specified in the use case document. Due to the complexity of the system battery lifetime simulation is usually computationally intensive and time-consuming. Therefore, only the tag's power demand not its functionality is simulated to speed up lifetime estimation. To achieve this, the power profiles are analyzed and automatically embedded within a SystemC demand model. This SystemC model can be connected to an arbitrary battery model. During simulation the demand model continuously repeats the profile's duty cycle and draws a current from the battery model. When the battery model runs out of charge simulation stops and simulation time is compared to the lifetime requirement. Through this automated sequence it is verified whether the application's lifetime requirement is fulfilled.

The remainder of this chapter provides a detailed description of the main phases of our approach.

A. Requirements analysis

In the beginning of a higher class RFID tag's design phase its functional and non-functional requirements are specified. Then they can be refined into use cases. We extend the textual use case format described in [12] since it is able to capture both functional- and non-functional requirements. Moreover, it can be analyzed automatically as described in [12]. The use case document is divided into several sections. This structure supports automated analysis by assisting in classification of the information in the individual sections. Our extended use case document comprises three main sections which contain a series of use cases, the global constants and applications.

- **Use Cases:** contains an arbitrary amount of use cases
- **Global Constants:** defines constants used in the interaction steps of the use cases; constants may contain references to other constants or values
- **Applications:** specify a series of applications for which lifetime requirements must hold

Table I contains an excerpt of an use case document. The use case example shows the general structure and the primary scenario's interaction steps. Additionally, parts of the Global Constants section are included to demonstrate the concept. An example of an application is described in table II.

Within the Use Cases section all use cases are specified as subsections. The most important sections of each use case are as follows (also compare [12]).

TABLE I
EXAMPLE OF AN USE CASE DOCUMENT (EXCERPT)

Use Cases
Use case: COLLECTION_WITH_UDB
Description:
This use case describes the tag receiving a
Collection with Universal Data Block command.
Scope: Tag
Primary Actor: Interrogator
Preconditions:
1a. The Tag comes from the WAKEUP use case.
1b. The Tag comes from the IDLE use case.
Primary Scenario:
1. The tag receives a COLLECTION_UDB_COMMAND.
2. The tag identifies the UDB_TYPE.
3. The tag reads the MAX_PACKET_LENGTH.
4. The tag determines the WINDOW_SIZE.
5. The tag selects a RANDOM_SLOT.
6. The tag goes to the IDLE use case.
Alternative Scenarios:
-
Global Constants:
Constant: COLLECTION_UDB_COMMAND
Parameter: COMMAND_CODE
Parameter: MAX_PACKET_LENGTH
Parameter: WINDOW_SIZE
Parameter: UDB_TYPE
Constant: WINDOW_SIZE
Parameter: @bit@15@0@

- **Use case:** specifies the use case name
- **Description:** contains a short abstract of the use case
- **Scope:** refers to the system to which the use case applies
- **Primary Actor:** an entity interacting with the system
- **Preconditions:** preceding use cases which lead to the current use case
- **Primary Scenario:** contains a series of interaction steps in natural text which may contain constants
- **Alternative Scenarios:** specifies deviations from the primary scenario; starting at the step in which the deviation occurs it branches into a separate series of steps

Constants used in the steps of the use cases need to be defined in the Global Constants section. The constants contain references to other constants or specify a value, data type and bit range. If the value is left unspecified it is randomized during simulation. The constants are used to determine which data the actor sends to and receives from the system.

B. Application specification

After specifying the use cases, the applications can be defined. This is done within the case document after the Global Constants section. An application describes the order in which use cases are executed and their duty cycle. Further, the lifetime requirement for the application is specified.

The application section comprises three subsections. In the first subsection one or more use cases and their steps can be summarized to scenarios. This simplifies specification as several use case sequences may occur repeatedly. Also, it allows to deliberately branch into the alternative scenarios of a use case if the application demands it. Table II demonstrates the specification of several small scenarios and a larger scenario containing all small scenarios.

TABLE II
EXAMPLE OF AN APPLICATION SPECIFICATION

Application: Collection_with_UDB_sequence
Scenario: Wakeup
Use Case: WAKEUP *Step:* 1@4
Use Case: IDLE *Step:* 1@2
Scenario: Receive_collection_command
Use Case: COLLECTION_WITH_UDB *Step:* 1@5
Scenario: Idle_waiting
Use Case: IDLE *Step:* 1@2
Use Case: IDLE *Step:* 3b1@3b5
Scenario: Transmit_UDB_response
Use Case: FETCH_ID *Step:* 1@3
Use Case: UDB_RESPONSE *Step:* 1@6
Scenario: Receive_sleep_command
Use Case: SLEEP *Step:* 1@3
Scenario: Sleep_mode
Use Case: POWER_DOWN *Step:* 1@3
Use Case: CARRIER_SENSE *Step:* 1@4
Scenario: Collection_with_UDB_round
Use Case: WAKEUP *Step:* 1@4
Use Case: IDLE *Step:* 1@2
Use Case: COLLECTION_WITH_UDB *Step:* 1@5
Use Case: IDLE *Step:* 1@2
Use Case: IDLE *Step:* 3b1@3b5
Use Case: FETCH_ID *Step:* 1@3
Use Case: UDB_RESPONSE *Step:* 1@6
Use Case: SLEEP *Step:* 1@3
Duty Cycle:
1. Wakeup is executed for 2.5 sec.
2. Receive_collection_command executes for 5 ms.
3. Idle_waiting runs for 57.3 ms.
4. Transmit_UDB_response shall be executed for 8.1 ms.
5. Receive_sleep_command takes 5.8 ms.
6. Sleep_mode is repeated for 9597.424 sec.
7. Collection_with_UDB_round shall be executed 0-2 times.

Lifetime:
5 years 150 days

The second part of the application contains the order of individual scenarios and specifies their duty cycle. This subsection is expressed in natural text and contains a series of steps. Each step specifies the scenario and its duty cycle. Instead of a time the expected number of executions can be used to specify the duty cycle. In this case the duration of the scenario's power profile and the number of executions are used to calculate the duty cycle. It is possible to specify random executions with a range from minimum to maximum expected executions. Step 7 in table II is executed randomly between zero and two times. The random number within the range is re-calculated during simulation after each duty cycle.

The third subsection contains the lifetime requirement for this application. The lifetime requirement is expressed by simply specifying a sequence of numbers and time. This expresses the minimum required lifetime of the energy source for the application to be successful.

As an example we specify the "Collection with UDB" sequence according to the ISO/IEC 18000-7:2008 standard [16] for UHF RFID tags. (table II). The duty cycle of the application is chosen so the "Collection with UDB" sequence is executed every 2 hours and 40 minutes. Additionally, up to two random "Collection with UDB" sequences are specified for each duty cycle. If repeatedly executed this corresponds to 10 fixed and up to 20 random collection sequences per day.

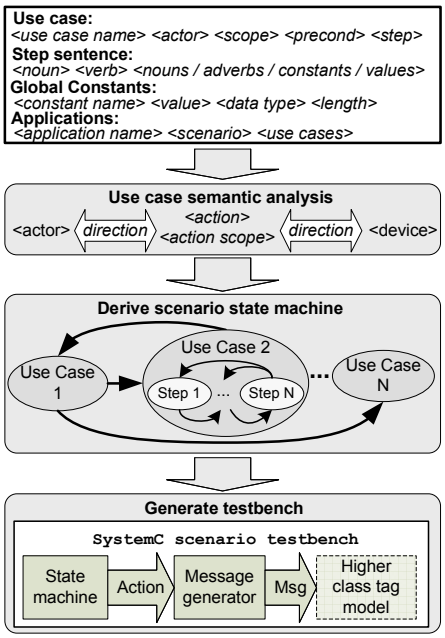

Fig. 2. Automatic generation of an application scenario testbench

C. Automatic use case and application analysis

After specification of the use cases and the applications they have to be analyzed. The lifetime of a device is dependent on its power consumption. Hence, the power dissipation of the system-under-design has to be determined. However, for estimating the power consumption we require a behavior corresponding to the application to be invoked in the system-under-design. To avoid time-consuming manual analysis of the use cases and laborious creation of test cases, a testbench is automatically derived from the specified application.

To accomplish this, the use case document is parsed and the individual sections and sentences are interpreted. Then the applications are parsed and the individual scenarios with the references to use cases and their steps are resolved. Semantic analysis is used to correctly interpret the use cases. Initially, the actor, system-under-design and references to constants are determined from the use case. The specified sequence of steps is examined and the individual parts of each sentence are classified. Then, semantic analysis identifies the actions and their direction for each sentence. The constants are resolved by interpreting the Global Constants section. Finally, semantic analysis is applied to determine each scenario's duty cycle and the application's lifetime requirement.

D. Automatic creation of the application scenario testbenches

After successfully analyzing the use case document the testbenches for power estimation are generated automatically. Therefore, we extend the approach for automatic testbench generation described in [12]. Instead of creating a SystemC testbench for functional verification for the entire set of use cases (as in [12]), only the use cases described in each application scenario are used.

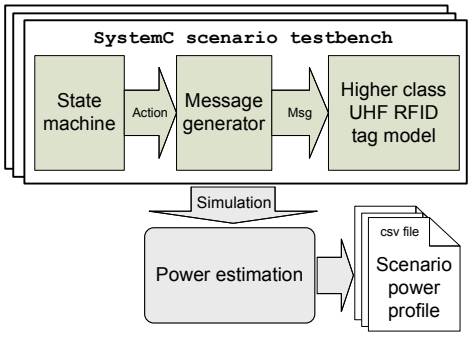

Fig. 3. Power estimation for each scenario

From the previous semantic analysis of the use case document a SystemC testbench is automatically generated. The testbench contains a state-machine for each specified application scenario (Fig. 2). From the use cases and their steps described in the scenario actions have been identified through semantic analysis. Each action provokes either an internal state transition or a transition to another use case. During a transition the actor or the system may send and receive messages. A message is determined from the action and a constant stated in the action's scope. The direction in which the message is sent is determined from the action, the system and the actor. The state machine with its transitions and messages is translated into a SystemC testbench. After the testbench is generated it can be applied to the system model to exercise the behavior specified by the scenario. Power estimation can be performed to determine the power consumption corresponding to the exercised behavior.

E. Estimation of the scenario's power dissipation

For power estimation the automatically generated state machine within the SystemC testbench is used. Messages corresponding to the interaction steps in the use cases are exchanged between testbench and system model. In this way the functionality described in the use case is invoked in the system-under-design. Thus, each message causes the dissipation of a corresponding amount of power by the system model.

To estimate the power consumption, either a power estimation tool or a very accurate simulation model of the system can be used. Our approach permits either possibility. For system-level power estimation the RHEiMS tool from Neosera Systems Ltd. [17] is employed. To accurately estimate a higher class UHF RFID tag's power consumption the model described in [1] is used. In both cases the model of the system is simulated with the testbench for each scenario. As a consequence, for the given scenario a power profile is created of the system-under-design (Fig. 3). Power estimation is repeated for each scenario's testbench. Eventually, a power profile exists for every scenario in all applications. We use the simple comma-separated-value (csv) format to store the time-stamp and the corresponding power value.

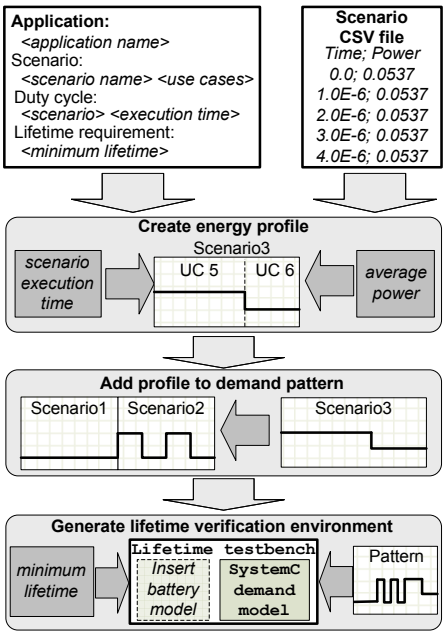

Fig. 4. Automatic creation of the verification environment

F. Automatic creation of the lifetime verification environment

Having performed power estimation a verification environment is automatically created to carry out the lifetime estimation of the energy source (see Fig. 4).

First, the power profiles for all scenarios are arranged according to the order and duty cycle specified by the application. The power profiles are analyzed, the power values are averaged and their duty cycle is stored.

Second, the average power from the log file is converted into energy (Power $\times$ Time). This permits to repeat each scenario profile for the amount of time specified in the scenario duty cycle. However, if the number of executions is specified instead of a time the log file's duty cycle is used to calculate the scenario's duty cycle. Consequently, we receive an energy profile from the duty cycle and the averaged power values.

Third, energy values from all scenarios are used to mimic the demand of the specified application. This means the energy profile for each scenario is appended in the order it appears in the steps of the application. The result is an energy demand pattern over the specified period of time. For random executions a corresponding code statement within the specified range is generated. The number of executions is randomized and re-calculated after every simulated duty-cycle. Then duty cycle and energy are updated accordingly.

Finally, the code for a SystemC demand model is generated automatically. The model repeatedly executes the energy pattern over and over again. Thereby, the random executions are re-calculated and the duty cycle information is updated. The demand model is integrated into a verification environment where it can be connected to the desired battery model. Thus, the input of the battery model is the current drawn at each time-step of the given application scenario.

TABLE III
VERIFICATION OF THE PROPOSED LIFETIME ESTIMATION METHODOLOGY

Profile	Energy
single read 128-bit	9.32E-03 J
single sleep mode	7.14E-05 J
total 128-bit (600 times per day)	2.21 J
total sleep (remainder of the day)	6.16 J
Lifetime i-Q8 data sheet:	> 6 years
Lifetime calculated Peukert:	6 years, 175 days
Lifetime estimated:	6 years, 145 days[1,2]

[1] Avg. Error 7.63% according to battery data sheets ($\approx \pm 178$ days)
[2] Error 1.27% compared to Peukert's formula ($\approx \pm 30$ days)

G. Lifetime estimation & verification of lifetime requirements

A battery model is connected to the demand model within the automatically generated verification environment. During simulation the demand model repeats the energy pattern until the battery model cannot meet the demand any more. When this happens the lifetime simulation is aborted.

The verification environment reports the simulation time and it is compared to the specified minimum lifetime for the application. If the simulated lifetime is longer than the specified lifetime the system fulfills the lifetime requirement for the given application. Otherwise, the system is not able to fulfill the lifetime requirement. Either different system components, another low-power strategy or a larger energy source has to be chosen.

IV. VERIFICATION OF THE PROPOSED LIFETIME ESTIMATION METHODOLOGY

To verify our approach we choose to apply our methodology to estimate the lifetime of a sample higher class UHF RFID tag. To avoid inaccuracies from high-level power estimation or a simulation model we choose to measure the power profiles. Therefore, power profiles from the i-Q8 tag from IDENTEC SOLUTIONS [6] have been recorded. As stated in the i-Q8 tag's product sheet the battery lifetime is longer than 6 years for 600 readings of 128 bits from the tag's memory per day. Therefore, profiles for reading 128 bits and sleep mode have been digitally measured and logged. Then the power profiles were converted into the csv format with entries for time stamp and power. Consequently, we received a csv file for the 128-bit memory readout and one for the sleep mode.

Then the application is specified in the use case document. It assumes a 128-bit memory readout repeated for 600 times. Afterwards, the tag returns to sleep mode for the remainder of the day. The minimum lifetime for the application was specified to be 6 years as stated in the i-Q8 tag's product sheet. After automatic analysis of the specified application and the csv files we received the generated SystemC demand model. The i-Q8 tag's power source is a Tadiran SL-760 Lithium battery with a capacity of 2.2 Ah [18]. A simple battery model was developed and configured with the data of the 2.2 Ah battery. Finally, we connected both SystemC models and started the simulation. After the battery model ran out of charge, the simulation stopped.

For the i-Q8 tag's designated application of 600 daily memory readouts the battery lifetime was estimated to be 6 years and 145 days (see table III). This corresponds to the "longer than 6 years" lifetime statement from the i-Q8 tag's product sheet [6]. Consequently, the lifetime requirement for the application specified in the product sheet is confirmed.

However, the battery model used for lifetime estimation is very simple and does not account for any battery effects. The inaccuracy of the battery model has been determined by using continuous discharge on models with data from the Varta CRAA [19] and Tadiran SL-760 [18] batteries. Lifetime simulation results have been compared to the battery data sheets. For continuous drain the average error of the battery model is 7.63%. Calculating the battery lifetime for the iQ-8 tag with Peukert's formula ($C_p = I^k \times t$) yields an error of 1.27%. However, the Peukert formula is inaccurate [14]. In general, the accuracy for lifetime estimation can be improved by using a more accurate battery model.

This example showed that our specification format is capable of accurately reflecting a higher class UHF RFID tag's application. Moreover, the automatic analysis of the application and the generation of the demand model from the specified duty cycle and the power profiles were demonstrated. Typically the automatic requirements analysis and generation of the verification environment does not exceed 30 seconds. Simulation time mostly depends on the profile length and the computational effort of the used battery model. For the i-Q8 UHF RFID tag the generation of the verification environment took less than 15 seconds. Battery lifetime simulation with the simple linear battery model takes approximately 5 seconds.

V. CASE STUDY OF LIFETIME REQUIREMENTS IN LANDING GEAR MONITORING

As case study for lifetime estimation aircraft landing gear monitoring has been chosen. In this application higher class UHF RFID tags are used to monitor the brake temperature of aircraft landing gear. The lifetime requirement is based on the information from [9]. This means battery lifetime has to be between 7 years and 12 years. We assume monitoring each of the aircraft's landing gears is performed independent from other gears. In addition, for monitoring brake temperature we further assume one higher class UHF RFID tag and one interrogator per landing gear.

A. Use case and application specification

The ISO/IEC 18000-7:2008 standard [16] provides a detailed description of the higher class UHF RFID tag's command sequences. The use cases for the higher class tag are specified in accordance to the ISO/IEC 18000-7:2008 standard [16]. Additionally, a set of varying applications using a higher class UHF RFID tag to monitor aircraft landing gear according to [9] are specified. For all of the following landing gear monitoring applications the aircraft is assumed to be on the ground for two hours and in flight for two hours.

The readout sequence is illustrated at the top of Fig. 5. A collection round is followed by a memory read command.

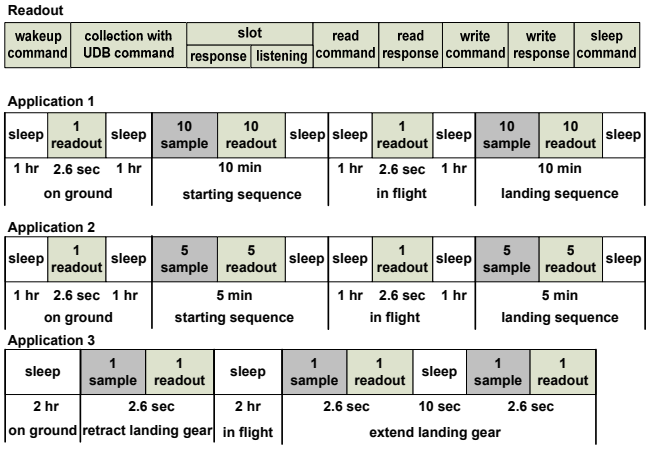

Fig. 5. Different applications for aircraft landing gear monitoring

After the tag's response to the memory read the sampling interval is set. Then the sleep command sends the tag to a low-power sleeping mode. After the given interval the tag awakes, takes a temperature sample from the sensor, stores it in memory and returns to sleep mode.

Three periodic event sequences for the application (see Fig. 5) are specified. Their main difference is the number of readouts per application. In addition, each of the three periodic applications is extended with a random readout sequence. For these random applications it is assumed that the ground maintenance personnel performs zero to three readouts while the aircraft is on the ground. This allows a comparison of the six applications to determine which fulfills the minimum lifetime requirement of more than seven years.

- *Application one:* The first application for brake temperature monitoring features a readout sequence every hour and a 10 minute readout sequence before start and landing. During the 10 minute readout sequence a collection round is started, the sensor data is read from the memory and the new sample interval is set. The tag is sent to sleep mode for one minute. After the previously set interval the sample is taken and stored in memory.
- *Application two:* Instead of ten readouts during the start- and landing sequences application two only performs five readouts. Each readout is followed by a two minute sleep period. Otherwise it is identical to the first application.
- *Application three:* In application three the number of readouts has been reduced to only three readouts. One sample is taken and read when the landing gear is retracted after take-off. During flight the UHF RFID tag is in sleep mode. When the landing gear is extended a sample is taken and read. Then the tag is idle for 10 seconds and another sample is taken and read.

After specification of the use cases and different application scenarios they are automatically analyzed. For each application scenario a testbench for power estimation is generated.

B. Power estimation for the application scenarios

To estimate the power dissipation of our SystemC model of an ISO/IEC 18000-7:2008 standard-conform higher class UHF RFID tag we used the RHEiMS framework [17]. RHEiMS uses statistical methods to estimate the power consumption for a model by comparing it to a similar model stored in its database. Therefore, input, output vectors and activity are compared to the stored data and power consumption is calculated. RHEiMS provides fast simulation at system level and shows results with 1.5% accuracy from gate level [17].

To store values in the RHEiMS database, we use results from the simulation model of an UHF RFID tag conforming to the ISO/IEC 18000-7:2008 standard described in [1]. The model has also been implemented on a system-on-chip (SoC) development platform [15]. Comparison of the power consumption of the tag model and the SoC implementation shows high accuracy with an average error of 2.57% [20].

Our SystemC model of the higher class RFID tag has been simulated with RHEiMS. As a result, power profiles have been determined for all scenarios of the applications. From the specified duty cycles of all applications and the power profiles the demand models are generated. The demand model of each application is embedded into a separate verification environment. Each demand model repeatedly executes the application's energy profile.

C. Lifetime simulation for the applications

The demand models are connected to the simple SystemC battery model. The battery model is configured with data from three different Lithium batteries with a shelf-life of 10 years.

- Varta CRAA with 2.0 Ampere-hours [19]
- Tadiran SL-760 with 2.2 Ampere-hours [18]
- Tadiran SL-2770 with 8.5 Ampere-hours [18]

Each application is simulated with these three batteries to explore different battery sizes.

D. Results of the lifetime requirements verification

In table IV the results of the battery lifetime estimation for aircraft landing gear monitoring are displayed. Column one contains the application, column two the battery and column three shows the tag's lifetime with that battery.

The results clearly shows a trend in battery lifetime. The lifetime requirement for each application could only be fulfilled with the Tadiran SL-2770 (8.5 Ah) Lithium battery.

Taking into account errors from the battery model and the power profiles application one has a lifetime of 7 years and 315 days. Thus application one with the Tadiran 8.5 Ah battery fulfills the minimum lifetime requirement of 7 years. The Tadiran SL-2770 battery has a shelf-life of more than 10 years. This means with the Tadiran SL-2770 battery applications two and three would also fulfill the minimum lifetime requirement.

Automatic generation of the verification environment, simulation and the verification of the battery lifetime requirement was performed in under 20 seconds for each application and battery. Thus, the capabilities of our methodology for rapidly exploring different batteries are demonstrated.

TABLE IV
LIFETIME ESTIMATION RESULTS FOR AIRCRAFT LANDING GEAR MONITORING

Application	Battery	Lifetime
Application 1	Varta 2.0 Ah	2 years 21 days
Application 1	Tadiran 2.2 Ah	2 years 96 days
Application 1	Tadiran 8.5 Ah	8 years 277 days
Application 1 random	Varta 2.0 Ah	1 year 345 days
Application 1 random	Tadiran 2.2 Ah	2 years 51 days
Application 1 random	Tadiran 8.5 Ah	8 years 96 days
Application 2	Varta 2.0 Ah	2 years 313 days
Application 2	Tadiran 2.2 Ah	3 years 51 days
Application 2	Tadiran 8.5 Ah	12 years 54 days
Application 2 random	Varta 2.0 Ah	2 years 232 days
Application 2 random	Tadiran 2.2 Ah	2 years 329 days
Application 2 random	Tadiran 8.5 Ah	11 years 83 days
Application 3	Varta 2.0 Ah	4 years 294 days
Application 3	Tadiran 2.2 Ah	5 years 104 days
Application 3	Tadiran 8.5 Ah	20 years 155 days
Application 3 random	Varta 2.0 Ah	4 years 65 days
Application 3 random	Tadiran 2.2 Ah	4 years 217 days
Application 3 random	Tadiran 8.5 Ah	17 years 280 days

VI. CONCLUSION AND FUTURE WORK

In this paper we presented a methodology to verify battery lifetime requirements of higher class UHF RFID tags. It was determined that the battery lifetime is highly dependent of the application the RFID tag is used in. Therefore, we augmented a use case format to specify hardware requirements. The use case format is now capable to capture the target application and its lifetime requirement. Our highly automated approach generates a testbench for power estimation of the application specified in the use case document. From the application's estimated power dissipation a verification environment is created. With the verification environment and a battery model, lifetime estimation and verification of the application's lifetime requirement is performed.

The presented methodology has been verified using digitally measured power profiles of an existing higher class UHF RFID tag. The lifetime verification results correspond to the tag's product sheet and calculated lifetime.

In a case study, lifetime requirements for aircraft landing gear monitoring with a higher class UHF RFID tag have been verified. Different periodic and random sampling and tag readout sequences were specified as different applications in our extended use case format. Power estimation and battery lifetime estimation were performed for the specified applications. Due to the high degree of automation and fast simulation speed different batteries could be explored rapidly. As a result, a battery could be determined for which the higher class UHF RFID tag fulfills the lifetime requirements of the landing gear monitoring application.

Currently, we are investigating different battery models and their impact on simulation speed and accuracy. In addition, we plan to explore the effects of energy harvesting devices and other energy sources with our methodology.

ACKNOWLEDGMENT

This project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the contract FFG 812424.

REFERENCES

[1] A. Janek, C. Steger, R. Weiß, J. Preishuber-Pfluegl, and M. Pistauer, "Functional Verification of Future Higher Class UHF RFID Tag Architectures based on Cosimulation," in *Proceedings of the IEEE International Conference on RFID*. IEEE, April 2008, pp. 336–343.

[2] EPC Global Inc., "EPC Radio-Frequency Idendity Protocols Class-1 Generation-2 UHF RFID Protocol for Communication at 860 MHz 960 MHz Version 1.1.0," 2007.

[3] R. Lissel and J. Gerlach, "Introducing new verification methods into a company's design flow: an industrial user's point of view," in *Design, Automation & Test in Europe, Conference & Exhibition, 2007. DATE '07*. IEEE, April 2007, pp. 689–694.

[4] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Discrete-Time Battery Models for System-Level Low-Power Design," in *IEEE Transactions on very large scae integration (VLSI) systems*, vol. Vol. 9. IEEE, October 2001, pp. 630–640.

[5] F. Simjee and P. Chou, "Accurate battery lifetime estimation using high-frequency power profile emulation," in *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*. IEEE, 2005, pp. 307–310.

[6] IDENTEC SOLUTIONS, http://www.identecsolutions.com/fileadmin/user_upload/PDFs/product_sheets/ILR/EN/ID.0601.EN_i-Q8.pdf, February 2009, last accessed - 12/03/2009.

[7] H. Bochmann, R. Kessler, and G. Schulze, "Current and Future Developments in Tire Pressure Monitoring Systems," *Automobiltechnische Zeitschrift (ATZ)*, 2005, reprint available at http://www.beru.com/download/produkte/fachaufsatz_tss_3gen_e.pdf.

[8] C. Kolle, W. Scherr, D. Hammerschmidt, G. Pichler, M. Motz, B. Schaffer, B. Forster, and U. Ausserlechner, "Ultra low-power monolithically integrated, capacitive pressure sensor for tire pressure monitoring," in *Sensors, 2004. Proceedings of IEEE*. IEEE, Oct. 2004, pp. 244–247.

[9] B. Pátkai, L. Theodorou, D. McFarlane, and K. Schmidt, "Requirements for RFID-based Sensor Integration in Landing Gear Monitoring - A Case Study," July 2007.

[10] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.

[11] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.

[12] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiß, and M. Pistauer, "Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs," in *Systems Conference, 2008 2nd Annual IEEE*. IEEE, April 2008, pp. 1–8.

[13] C. M. Kirchsteiger, C. Trummer, C. Steger, R. Weiss, and M. Pistauer, "Automatic verification plan generation to speed up soc verification," *NORCHIP, 2008.*, pp. 33–36, Nov. 2008.

[14] Ravishankar Rao and Sarma Vrudhula and Daler N. Rakhmatov, "Battery Modeling for Energy-Aware System Design," *IEEE Computer*, vol. VOL. 36, no. NO. 12, pp. 77–87, December 2003.

[15] A. Janek, M. Schöfelbauer, C. Steger, R. Weiß, J. Preishuber-Pfluegl, and M. Pistauer, "Design and Implementation of a Power Aware Communication Protocol for Active UHF RFID Tags based on Transceiver SoC CC1110," in *European DSP Education and Research Symposium 2008, Proceedings*. Texas Instruments, 2008, pp. 155–162.

[16] International Standardization Organization, "ISO/IEC 18000-7:2008 - Information technology – Radio frequency identification for item management – Part 7: Parameters for active air interface communications at 433 MHz," 2008.

[17] Neosera Systems Ltd., "http://www.neosera.com," 2009, last accessed - 12/03/2009.

[18] Tadiran Batteries GmbH, http://www.tadiranbatteries.de/eng/downloads/LITHIUM/pdc06engSL-760.pdf, last accessed - 28/11/2008.

[19] VARTA Microbattery GmbH, "http://www.varta-microbattery.com," 2009, last accessed - 11/03/2009.

[20] A. Janek, "Architecture Design and Simulation of Energy Harvesting Sensors," Ph.D. dissertation, Graz University of Technology, 2008.

Specification and Automated Simulation-based Verification of Power Requirements for System-on-Chips

Christoph Trummer, Christoph M. Kirchsteiger, Christian Steger, Reinhold Weiß*,
Markus Pistauer† and Damian Dalton‡
*Institute for Technical Informatics, Graz University of Technology, Austria
email: (trummer, c.kirchsteiger, steger, rweiss)@tugraz.at
†CISC Semiconductor Design+Consulting GmbH, Austria
email: m.pistauer@cisc.at
‡School of Computer Science and Informatics, University College Dublin, Ireland
email: damian.dalton@ucd.ie

***Abstract*—Today's advances in silicon integration density allow more and more functionality to fit into a system-on-chip (SoC). However, this has made power consumption the most critical design constraint for system-on-chips (SoCs). Power consumption especially affects portable devices as it influences battery lifetime. Moreover, specification, design and verification of System-on-Chips have become increasingly complex. In this work we present a novel methodology which supports specification and automates verification of power requirements through simulation and power estimation. We demonstrate our approach on an example SoC. Therefore, we specify power requirements for a higher class radio frequency identification (RFID) tag. Then we automatically generate test cases which allow to estimate the RFID tag's power consumption. To verify the power requirements we apply the test cases to the system model in simulation.**

I. Introduction

For today's System-on-Chips (SoCs) power consumption is the most important design constraint. It directly affects system stability and battery lifetime of portable devices [1]. Consequently, power requirements are the most important non-functional requirements. Due to the rising silicon integration densities a SoC performs more functionality using the same or even less chip area. In turn complexity increases for specification, design and verification of SoCs. Particularly, verifying that a system fulfills its requirements often takes 70% of the entire design effort [2].

Verification of power requirements is usually performed through power estimation [1]. Test cases have to be derived manually from the requirements which is laborious. Then the test cases are applied onto the system to estimate its power consumption. Finally, verification is performed by ensuring the estimated power does not violate the constraints. To reduce the effort of verification an automated flow is necessary.

In this paper we present a methodology for specification and automated verification of power requirements. We introduce a use case format for specifying power requirements. The semi-formal nature of the format permits the automatic derivation of test cases through semantic analysis. During simulation the generated test cases are applied to the system and invoke a corresponding behavior. The power consumption of the system's behavior is estimated using RHEiMS, a fast system-level power estimation tool [3]. The power estimation results are automatically verified against the power requirements. In fig. 1 the principle of our methodology is illustrated.

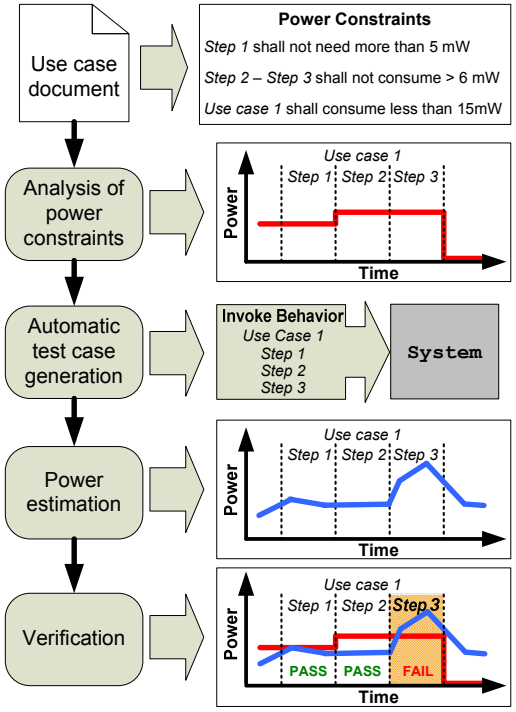

Fig. 1. Principle of the Methodology for Verifying Power Requirements

Our main contribution is a novel, self-contained methodology from specification to verification of power requirements. Further, we reduce complexity in requirements specification by using a simple, structured use case format. By automatically generating test cases to verify the power requirements the verification effort is decreased as well.

This paper is organized as follows. Section II analyzes related work in the area of specification and verification of power requirements. In section III details about the use case format and our verification methodology are explained. Our approach is evaluated on a case study of a higher class RFID tag (section IV). Finally, section V summarizes our work and provides an outlook to future work.

II. Related work

A. Power requirements

Power requirements do not describe a behavior or functionality. Consequently, power requirements are non-functional requirements or constraints. At the time the requirements are specified exact power consumption and constraints may not be known. They need to be approximated or derived from the power demand of similar systems, devices or protocols [4], [5]. Power consumption affects battery lifetime, therefore high-level power requirements often emerge from battery lifetime requirements [6]. Sometimes power requirements are expressed after knowledge of the system's energy source and power consumption is available [7]. Thus, power requirements are constraints describing the maximum amount of power the system is allowed to consume while performing a certain operation.

B. Specification formats for power requirements

Power requirements are most commonly stated informally in natural text: "The system shall consume less than 5mW" [7]. Often, this statement applies to the entire system and poses a global constraint. High-level power constraints such as battery lifetime are often expressed similarly: "The system's battery shall last 5 years for 600 read operations per day" [8]. This statement can later be refined into specific power requirements.

Formal specifications are used to capture power requirements. In [9] power requirements are expressed informally and later transformed into formal temporal logic expressions. A well known formal specification format is the unified modeling language (UML). UML is also used to specify requirements in hardware design [10]. Non-functional requirements can be included into the use case descriptions and diagrams [10].

The semi-formal use case format combines the structure of formal use cases with informal textual expressions [11]. Functionality is described through a series of interaction steps between the system and the actor(s) [11]. Similar to UML it also allows to specify non-functional requirements.

C. Automatic test case generation from requirements

Due to the high complexity of SoCs it often has large number of requirements. Manually writing test cases for verification needs considerable amount of time and effort. Therefore, it is desirable to automatically derive test cases from the requirements. However, automated analysis requires a specification format that is easy to parse and interpret.

In the software domain test cases can be automatically generated from formal descriptions such as UML state machine diagrams [12]. For hardware designs test cases can be automatically generated through semantic analysis of semi-formal textual use cases [13]. The test cases are applied to a model of the system in a hardware description language during simulation. By monitoring the systems behavior and its responses the requirements are verified [13].

D. Simulation-based verification of power requirements

Power requirements are usually verified by with power estimation tools and simulation [1]. Traditionally, these tools estimate power consumption by analyzing low-level models of the system. This is accurate but also very time-consuming [1]. Recently, tools have become available that allow high-level power estimation [1]. These tools are usually fast but not very accurate. However, RHEiMS [3] a high level power estimation tool uses special macro models to retain the accuracy from lower abstraction levels while providing fast simulation results.

III. Methodology

A. Specification of power requirements

Power requirements are constraints imposed on functionality which means the system has to perform under certain conditions. In our case the system has to operate with limited power. Therefore, a format is needed that captures both functional and non-functional requirements. Due to their structured nature semi-formal use cases strike a balance between readability and automatic processing capabilities [13]. For verification of power requirements the use case format has to be extended. To specify power requirements two additional sections are necessary. The local power requirements are a subsection of the use cases which allow to assigning power constraints to a single step or a series of steps. The global power requirements section is used for constraints affecting one or more use cases. The extended use case document comprises three main sections: the use cases, the global constants and the global power requirements.

The "Use Cases" section contains a series of all use cases. The most important sections for each use case are (see [13]):

- **Use case:** the name of the use case
- **Description:** an informal abstract of the use case
- **Scope:** names the system the use case applies to
- **Primary Actor:** the external entity interacting with the system
- **Preconditions:** preceding use cases leading to the current use case
- **Primary Scenario:** the interaction steps which describe functionality and may contain constants
- **Alternative Scenarios:** deviations from the primary scenario for conditional branching into a separate series of steps
- **Local Power Requirements:** power constraints for steps of the use case

In the corresponding sections of the use case document power requirements are specified in natural text. However, for automated semantic analysis the text has to follow a certain syntax.

TABLE I
EXCERPT OF A SAMPLE USE CASE DOCUMENT

Document: Specifications for an ISO18007 conform RFID tag
Use Cases:
Use case: COLLECTION_WITH_UDB
Description:
This use case describes the RFID tag receiving a Collection with Universal Data Block command.
Scope: Tag
Primary Actor: RFID reader
Preconditions:
1a. The tag comes from the WAKEUP use case.
1b. The tag comes from the LISTENING use case.
Primary Scenario:
1. The tag receives a COLLECTION_UDB_COMMAND.
2. The tag identifies the UDB_TYPE.
3. The tag reads the MAX_PACKET_LENGTH.
4. The tag determines the WINDOW_SIZE.
5. The tag selects a RANDOM_SLOT.
6. The tag goes to the LISTENING use case.
Alternative Scenarios:
-
Local Power Requirements:
Step 2 - Step 4 shall not consume more than 30mW.
Step 5 shall dissipate less than 20mW.
Global Power Requirements:
COLLECTION_WITH_UDB should not consume more than 110 mW.
Global Constants:
Constant: COLLECTION_UDB_COMMAND
Parameter: COMMAND_CODE
Parameter: MAX_PACKET_LENGTH
Parameter: WINDOW_SIZE
Parameter: UDB_TYPE
Constant: WINDOW_SIZE
Parameter: @bit@15@0@

1. <*Step*> <*Text*> <*Less/More*> <*Value*> <*Unit*>
   EXAMPLE:
   Step 1 shall not dissipate more than 5,5mW.
2. <*StepRange*> <*Text*> <*Less/More*> <*Value*> <*Unit*>
   EXAMPLE:
   Step 1 - Step 3 shall consume less than 10mW.

A <*Step*> consists of the word "Step" and its number. The <*StepRange*> is expressed by using "-" or "to" between two <*Step*> statements. For global power requirements <*UseCase*> and <*UseCaseRange*> are used instead. <*Text*> consists of a verb and an optional negation. In <*Less/More*> written expressions or the smaller/greater characters ("<" or ">") can be used. For <*Value*> any integer or float is allowed but instead of the decimal point a comma is used. The <*Unit*> may contain a prefix for the magnitude and "W" for Watt.

B. Automatic generation of test cases for power estimation

To verify that a functionality does not violate its power constraint the associated power consumption has to be examined. Consequently, we need to analyze the use cases, determine constraints and generate a test case for estimating power.

The approach from [13] is extended to include power requirements in automatic test case generation. The use cases specify a series of interaction steps and state transitions. From these specifications the test cases are generated. Then the verification environment creates a test case schedule considering their preconditions. During simulation it mimics the actor and executes the schedule by sending messages

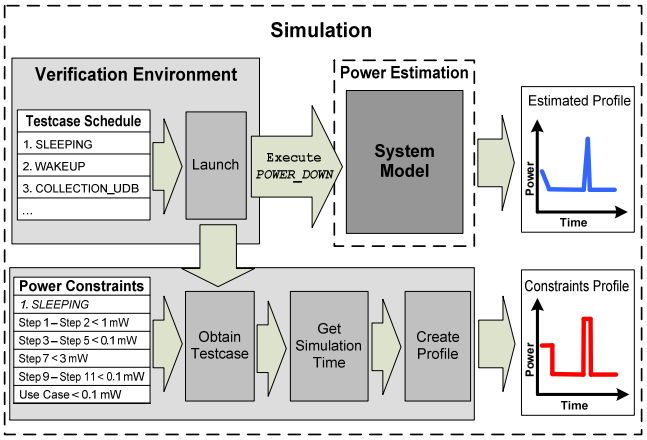

Fig. 2. Creating Power Profiles from Constrains and Power Estimation

to the system. This causes state transitions, changes internal values and triggers responses from the system. The verification environment monitors the system to verify that it operates correctly (see [13]). When the test case schedule is created local and global power requirements are written into a power constraints profile. During execution of the test cases their power consumption is estimated. To verify the system's power requirements the simulation time of the currently executed test case is recorded. This allows the constraint to be linked to the executed test case and its time frame. Fig. 2 shows the verification environment and creation of the constraints profile.

C. Power estimation

The RHEiMS framework [3] is used to estimate power consumption. It compares input, output and activity of the current model to a similar model with known power values. Through case-based reasoning and statistical methods RHEiMS calculates the power values for the current system. It combines fast simulation with high accuracy (ca. 1.5% at system level [14]). RHEiMS needs statements inside the system model for accessing the input/output vectors. Consequently, the system model is annotated with these statements before simulation. When the verification environment and the system model are simulated RHEiMS monitors activity and estimates power consumption. Finally, a profile of the estimated power consumption for each time step is created (see Fig. 2).

D. Verification of the power requirements

During verification each time frame of the estimated profile from RHEiMS is analyzed. The power value is compared to corresponding entry in the constraints profile from the verification environment. When a violation of a local or global constraint is detected use case, step and time are reported.

IV. CASE STUDY

We evaluate our methodology on a System-on-Chip implementation of a higher class ultra-high frequency (UHF) radio frequency identification (RFID) tag. This RFID tag is based on the ISO/IEC 18000-7:2008 [15] specifications and shall be used for long-term temperature measurement.

TABLE II
POWER CONSTRAINTS FOR A HIGHER CLASS UHF RFID TAG

Use case	Power constraint (mW)	Duty cycle (s)
WAKEUP	50	2.5
COLLECTION_UDB	110	0.013358
LISTENING	50	0.05
READ_MEMORY	110	0.013948
GOTO_SLEEP	50	0.00588
SLEEPING	0.1	3597.33
SAMPLING	10	0.09

TABLE III
VERIFICATION RESULTS OF THE POWER CONSTRAINTS

Use case	Estimated power (mW)	Test case
WAKEUP	49.277	PASS
COLLECTION_UDB	46.361	PASS
LISTENING	42.998	PASS
READ_MEMORY	104.369	PASS
GOTO_SLEEP	49.304	PASS
SLEEPING	0.0952	PASS
SAMPLING	6.126	PASS

A. Use case specification

After analyzing the ISO/IEC 18000-7:2008 document the semi-formal use cases are created (excerpt in table I). For our application, the tag awakes every two minutes from sleep mode to store a temperature sample in its memory. Once per hour a RFID reader retrieves the logged samples from the tag.

To specify the power requirements we need to select a battery and analyze the system's lifetime. Therefore, a battery with 2 Ampere-hours was chosen to achieve a minimum lifetime of four years. Power constraints are derived from the above duty cycle and battery lifetime. Since a high-level model is used the power requirements are specified on a per use case basis. Transmitting data usually consumes the most power. Therefore, we set the power constraint in the corresponding use cases to 110 mW. For the sleeping state less than 100 nW shall be used. When the tag is active and receiving data the constraint is 50 mW. Table II contains a summary of the power requirements in the use case document.

B. Verification and results

The automatically created verification environment launches test cases as specified in the steps of the use cases. For power estimation we annotated our higher class UHF RFID tag model with the RHEiMS statements to monitor its activity. Therefore, we selected a similar model of a higher class ISO/IEC 18000-7 UHF RFID tag [16] which was pre-characterized and stored in RHEiMS. During simulation one duty cycle for each test case is simulated. After simulation the results from power estimation and constraints were compared automatically. Table III contains the verification results of the power requirements. All test cases were executed and fulfill the power constraints.

V. CONCLUSION AND FUTURE WORK

In this paper we presented a methodology to specify and verify power requirements. A semi-formal use case format was extended to allow specification of power requirements. From the use cases and power requirements, test cases are automatically created. Through simulation and power estimation the power requirements are verified.

On a case study of a system-level model of a higher class UHF RFID tag our approach was demonstrated. The power requirements were derived from battery lifetime and specified in the use case document. After simulation and power estimation the power requirements were successfully verified.

Currently, we elaborate our use cases into a lower level of abstraction. Then, we will refine our model of a higher class RFID tag into a more detailed RTL model. Finally, we plan to verify the RTL model's power requirements with our approach.

ACKNOWLEDGMENT

This project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under contract FFG 812424.

REFERENCES

[1] D. Sunwoo, H. Al-Sukhni, J. Holt, and D. Chiou, "Early models for system-level power estimation," *Microprocessor Test and Verification, 2007. MTV '07. Eighth International Workshop on*, pp. 8–14, Dec. 2007.

[2] R. Lissel and J. Gerlach, "Introducing new verification methods into a company's design flow: an industrial user's point of view," in *Design, Automation & Test in Europe, Conference & Exhibition, 2007. DATE '07.* IEEE, April 2007, pp. 689–694.

[3] Dalton, McCarthy, Quigley, and Leeney, "A system-level power evaluation method," Irish Patent PCT 31 498IESP, 2008.

[4] J. Taneja, J. Jeong, and D. Culler, "Design, modeling, and capacity planning for micro-solar power sensor networks," *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pp. 407–418, April 2008.

[5] F. Fereydouni Forouzandeh, O. Mohamed, and M. Sawan, "Ultra low energy communication protocol for implantable body sensor networks," *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, pp. 57–60, June 2008.

[6] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Discrete-Time Battery Models for System-Level Low-Power Design," in *IEEE Transactions on very large scae integration (VLSI) systems*, vol. Vol. 9. IEEE, October 2001, pp. 630–640.

[7] S. Mikami, T. Matsuno, M. Miyama, M. Yoshimoto, and H. Ono, "A Wireless-Interface SoC Powered by Energy Harvesting for Short-range Data Communication," in *Asian Solid-State Circuits Conference, 2005*, 2005, pp. 241–244.

[8] IDENTEC SOLUTIONS, http://www.identecsolutions.com/, October 2008, last accessed - 31/1/2009.

[9] S. Ray, P. Dasgupta, and P. Chakrabarti, "Formal verification of power scheduling policies for battery powered mobile systems," *India Conference, 2006 Annual IEEE*, pp. 1–6, Sept. 2006.

[10] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.

[11] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.

[12] P. Samuel, R. Mall, and A. Bothra, "Automatic test case generation using unified modeling language (uml) state diagrams," *Software, IET*, vol. 2, no. 2, pp. 79–93, April 2008.

[13] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiß, and M. Pistauer, "Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs," in *Systems Conference, 2008 2nd Annual IEEE.* IEEE, April 2008, pp. 1–8.

[14] Neosera Systems Ltd., "Rheims: Rapid hierarchical energy investigation modelling system," http://www.neosera.com, 2009, last accessed - 31/1/2009.

[15] International Standardization Organisation, "ISO/IEC 18000-7:2008 - Information technology – Radio frequency identification for item management – Part 7: Parameters for active air interface communications at 433 MHz," 2008.

[16] A. Janek, C. Steger, R. Weiß, J. Preishuber-Pfluegl, and M. Pistauer, "Functional Verification of Future Higher Class UHF RFID Tag Architectures based on Cosimulation," in *Proceedings of the IEEE International Conference on RFID.* IEEE, April 2008, pp. 336–343.

Automated Simulation-based Verification of Power Requirements for Systems-on-Chips

Christoph Trummer, Christoph M. Kirchsteiger, Christian Steger, Reinhold Weiß*,
Markus Pistauer† and Damian Dalton‡
*Institute for Technical Informatics, Graz University of Technology, Austria
email: (trummer, c.kirchsteiger, steger, rweiss)@tugraz.at
†CISC Semiconductor Design+Consulting GmbH, Austria
email: m.pistauer@cisc.at
‡School of Computer Science and Informatics, University College Dublin, Ireland
email: damian.dalton@ucd.ie

***Abstract*—Today power dissipation is the most important constraint for Systems-on-Chips (SoCs). Consequently, it is necessary to consider power requirements for mobile, battery-powered devices in which SoCs are often used. Power requirements describe battery lifetime, power constraints and functionality to enter different low-power states. Also, the power requirements need to be verified beside functionality. However, to verify that the complex SoC design fulfills its requirements needs considerable effort. We introduce a methodology to reduce the verification effort through a high degree of automation. Our novel approach to verify battery lifetime, power constraints and the power aware design comprises three parts. First, a semi-formal use case format unifies specification of power and system requirements. Second, these specifications are used to automatically derive test cases and to generate a verification environment. Third, fast simulation and power estimation are employed to verify battery lifetime, power constraints and the power aware design against the requirements.**

I. Introduction

Systems-on-Chips (SoCs) are typically used in mobile, battery-powered devices. Process technologies below 100 nm enable more functionality but also cause static power consumption to rise [1]. Since power directly affects system stability and battery lifetime it is the most important design constraint for SoCs [2]. Consequently, battery lifetime and power dissipation are considered non-functional requirements imposing constraints on the system's functionality. To reduce power dissipation, design techniques are employed to turn off inactive areas of the SoC, reduce supply voltages and scale operating frequency. Functionality for power reduction is part of the pervasive functions which are operations beyond normal system operation [3]. Consequently, functionality to reduce power does not directly contribute to the SoC's main functionality [3]. It is part of the power requirements.

The increase in the SoC's functionality affects power dissipation and contributes to complexity in specification, design and verification. Particularly, verifying the numerous system requirements often uses 70% of the entire design effort [4].

Traditionally, battery lifetime, power requirements and the power design are verified at late design stages and incoherently. This has two reasons. First, to reduce the predominant dynamic power dissipation in process technologies above 100 nm it was not necessary to add power design information at early stages [1]. With the increase in static power consumption the need arose to specify and verify power design earlier [1]. Second, power estimation techniques are inaccurate at the high levels of abstraction of early design stages. With inaccurate power estimation results verification of battery lifetime and power constraints remains vague.

Verifying of functional and non-functional power requirements at late stages has several disadvantages. Adding the power aware design at stages (i.e. below Register Transfer Level (RTL)) forfeits some of the potential to optimize power reduction [5]. Late verification of the power aware design may expose critical bugs too late which necessitates costly redesign. Disregarding power consumption at early design stages may result in unwanted power peaks which severely impact battery lifetime and cause thermal issues. To compensate system instability from heat, costly and bulky cooling solutions have to be deployed. Verifying battery lifetime early allows optimization of battery size and helps in reducing costs. Additionally, early estimation of power and battery lifetime benefits from fast simulation speeds. Consequently, by applying power aware design and verifying power requirements in early phases better architectures are achieved and cost is reduced [5].

Our approach starts at the requirements phase and simplifies specification through semi-formal use cases. The format unifies specification of power requirements and the system's functionality. From these use cases semantic analysis automatically generates test cases and a verification environment. During simulation the verification environment launches the test cases to trigger a corresponding behavior in the system design. With a power estimation tool the power dissipation for the executed behavior is calculated. From the power estimation results the power constraints imposed on functionality are automatically verified. Additionally, it is used to determine the lifetime for a given battery which verifies the battery lifetime requirements. By comparing the system's power state during simulation to the specification the power design is verified.

The main benefit of our methodology is the high degree of automation during the verification process which reduces the verification effort. The easily comprehensible, unified use

case format reduces complexity in specification of power requirements. Importantly, our methodology also facilitates early verification of power constraints, battery lifetime and power design to lower costs and the risk of late redesign.

This paper is organized as follows. In section II background and related works about specification and verification of power requirements are analyzed. Section III explains our methodology in detail. Our approach is demonstrated on a case study in section IV. Finally, section V summarizes our paper.

II. Background and related work

A. Requirements specification for hardware design

Requirements can be expressed in different ways and are often ambiguous, imprecise and misleading. Typically, requirements are specified informally in natural text which is easy to read but ambiguous and difficult to process. In contrast, formal requirements can be processed easily because of their clearly defined structure. However, reading them without knowledge of the syntax is difficult [6]. Semi-formal requirements are a trade-off between readability and automatic processing [6].

Similar to the software domain the unified modeling language (UML) can be used to specify requirements for hardware design [7]. With an analysis model requirements are refined into formal use case-, collaboration- and class-diagrams [7]. Non-functional requirements can be included into the use case descriptions and diagrams but lack a unified structure [7].

A semi-formal format to express use cases is introduced in [8]. The use cases have a predefined structure similar to UML but within the individual sections a series of steps and interactions is expressed in natural language.

B. Power and battery lifetime requirements

Power requirements can be classified into functional and non-functional requirements. Functional power requirements specify functionality of the power design. From the specification the power design is created to reduce the system's power dissipation [1]. It can be described with two similar design formats. The Unified Power Format (UPF) was recently approved as IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits [1]. Alternatively, the Common Power Format (CPF) is well-established in the industry [9]. Both formats allow description and verification of the power aware design independent of the Hardware Description Language (HDL). However, as the power aware design does not directly contribute to the SoC's main functionality it is often overlooked in specification [3].

Since power consumption affects battery lifetime, high-level power requirements are often expressed as battery lifetime requirements [10]. They are usually specified in terms of minimum battery lifetime [10]. Some manufacturers specify the lifetime for their device in a generalized application [11].

Power constraints limit the amount of power the system is allowed to consume to perform certain functionality. When the system's functional requirements are specified details about power consumption and constraints may be unknown. Commonly, they are approximated from the power demand of similar systems, devices or protocols [12], [13]. Sometimes, power constraints are expressed after the system's energy source and power consumption become available [14].

C. Verification of non-functional power requirements

Power requirements are usually verified with power estimation tools and simulation [15]. These tools estimate the system's power consumption by analyzing low-level models which is accurate but also very time-consuming [15]. Recently, tools for high-level power estimation have become available [15], [16]. Due to rapid simulation speed at high levels of abstraction these tools are often fast but not very accurate.

To verify lifetime requirements, battery lifetime needs to be simulated or calculated with battery models. Many battery models exist and vary in accuracy and calculation speed [17]. Commonly, battery models are simulated with a model of the system or are connected to pre-recorded power profiles to estimate their lifetime [10], [18].

D. Verification of functional power requirements

In general, verification of functional power requirements comprises three steps. First, correct transition to each power state is ensured. Second, it is verified that the system performs correctly at the each power state. Third, the system design has to behave properly when entering a power state [3]. Two approaches use UPF to describe the power design and perform simulation-based verification [2], [19]. The first approach [2], focuses on specification of the power design and identifies power related bugs simulation is able to expose. While in [19], behavioral models for the power aware functionality are simulated with a testbench and the system. However, the power aware models and the testbench need to be created manually.

E. Automated verification platform generation from use cases

A verification platform can be automatically generated from textual use cases [6]. A semi-formal use case format similar to [8] is employed to capture functional requirements. Through semantic analysis of the use cases a verification platform is created automatically. It is used to verify a system model in a HDL. During simulation the verification platform sends stimuli derived from the use cases to the system model. This provokes a behavior corresponding to the functionality described in the model's use case. By monitoring the model's response and state transitions the functionality is verified [6]. We extend this approach to verify the different power requirements.

III. Methodology

Our methodology is oriented on the level of detail available at different design stages. At early design stages the power requirements are very abstract and expressed as battery lifetime requirements. When the system becomes more detailed lifetime requirements are refined into power constraints and imposed on the system's functionality. Finally, the functional design is detailed enough for the power design to be specified. The functional power requirements are expressed as power states which the system enters during a use case.

TABLE I
EXAMPLE OF AN *Use Case*

Use case: COLLECTION_UDB
Description:
This use case describes the tag receiving a
Collection with Universal Data Block command.
Scope: Tag
Actor: Interrogator
Preconditions:
1a. The Tag comes from the WAKEUP use case.
1b. The Tag comes from the IDLE use case.
Primary Scenario:
1. The Tag receives a COLLECTION_UDB_COMMAND.
2. The Tag identifies the UDB_TYPE.
3. The Tag reads the MAX_PACKET_LENGTH.
4. The Tag determines the WINDOW_SIZE.
5. The Tag selects a RANDOM_SLOT.
6. The Tag goes to the IDLE use case.
Alternative Scenarios:
-
Constants:
Constant: COLLECTION_UDB_COMMAND
Parameter: COMMAND_CODE
Parameter: MAX_PACKET_LENGTH
Parameter: WINDOW_SIZE
Parameter: UDB_TYPE
Constant: WINDOW_SIZE
Parameter: @bit@15@0@

A. Use case specification

The system's requirements are refined into use cases as explained in [6]. In the semi-formal use cases functional requirements are expressed in natural text as step-wise interactions between system and actor(s). To accommodate power requirements the format from [6] is extended. The XML-based use case document contains use cases, summarizes sequences of use cases as applications and specifies the power requirements. The use case document is structured as follows:

- Use Cases
- Applications
  - Scenarios
  - Duty Cycles
- Power Requirements
  - Battery Lifetime Requirements
  - Power Constraints
  - Power State Requirements

The *Use Cases* section contains all use cases describing the functionality of the system. The most important subsections of each use case include the use case name, a description, the scope, the actor(s), the primary scenario and alternative scenario(s) (see table I). Scope refers to the system and the actor specifies an entity interacting with it. The primary scenario describes the step-wise interaction between system and actor in natural text. The data exchanged in the interactions is specified in the constants section. Each constant may contain other constants or consists of a value, datatype and bit range. If the value is unspecified it is randomized during simulation.

Table II shows the *Application* section where potential applications for the system are described. Therefore, *Scenarios* are created which contain sequences of use cases and steps.

TABLE II
EXAMPLE OF AN *Application*

Application: Sampling
Scenario: Sleep_mode
Use Case: STANDBY *Step:* 1@6
Use Case: IDLE *Step:* 1b1@1b4
Scenario: Collection_with_UDB_round
Use Case: WAKEUP *Step:* 1@4
Use Case: IDLE *Step:* 1@2
Use Case: COLLECTION_UDB *Step:* 1@12
Scenario: Receive_Sleep_Command
Use Case: SLEEP *Step:* 1@5
Scenario: Take_Sample
Use Case: WAKEUP *Step:* 1a1@1a4
Use Case: SAMPLE *Step:* 1@2
Use Case: SLEEP *Step:* 1a1@1a3
Scenario: Read_Memory
Use Case: READ *Step:* 1@13
Duty Cycle:
1. Take_Sample is executed 3600 times.
2. Sleep_mode is repeated for 1 hour.
3. Collection_with_UDB_round is executed once.
4. Read_Memory shall be executed once.
5. Receive_Sleep_Command is executed once.

TABLE III
EXAMPLE OF THE *Power Requirements*

Battery Lifetime Requirements:
For Sampling the Tag's battery shall last for more than 4 years.
Power Constraints:
COLLECTION_UDB should consume less than 70 mW.
READ shall not consume more than 70 mW.
WAKEUP should not need more than 50 mW.
SLEEP shall not use more than 50 mW.
IDLE should dissipate less than 50 mW.
SAMPLE should not consume more than 7 mW.
STANDBY shall not need more than 100 nW.
Power State Requirements
While in STANDBY the Tag is ALL_OFF.
For WAKEUP the Tag goes to ALL_ON.
At READ, IDLE and COLLECTION_UDB the Tag is ALL_ON.
During SLEEP the Tag goes to ALL_OFF.
For SAMPLE the Tag is in SAMPLE_ON.

This simplifies specification as several use case sequences may occur repeatedly. Also, it allows deliberately branching into the alternative scenarios of a use case if the application demands it. The *Duty Cycles* and *Scenarios* define an *Application*.

The *Power Requirements* section shown in table III specifies *Battery Lifetime Requirements*, *Power Constraints* and *Power State Requirements*. *Battery Lifetime Requirements* relate to an *Application* for which a minimum battery lifetime is specified. The *Power Constraints* are imposed on entire use cases or on individual steps. Entries in *Power State Requirements* describe the system's power state during a use case. The functionality to trigger a power state is described in the *Use Cases*. Each power state is defined as a set of power domain supply states which are specified similar to the constants.

B. Automatic generation of the verification environment

The sections of the XML-based use case document are automatically parsed. Initially, actor, system and constants are determined for each use case. Semantic analysis interprets and classifies each sentence from the step-sequence to identify test

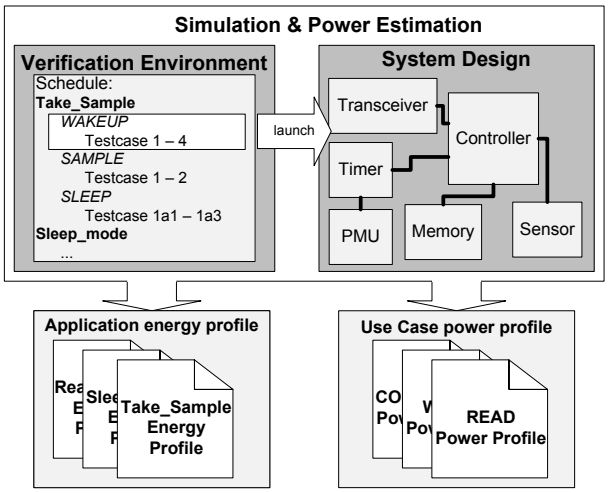

Fig. 1. Simulation and power estimation

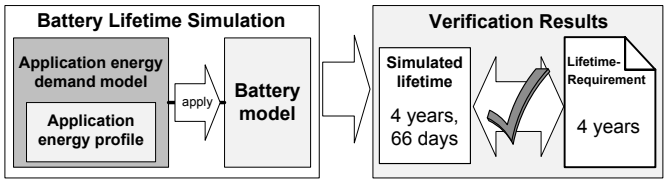

Fig. 2. Battery lifetime verification

cases from the specified actions, exchanged data and state transitions [6]. An action either provokes a transition to a step or to another use case. During such a transition actor and system may send and receive data which is determined by the action and the constant. The direction in which the data is sent is derived from the action, the system and the actor. After the test cases are created they are scheduled according to their appearance in the use case. A verification environment containing the schedule is generated in SystemC. During simulation it launches the scheduled test cases and triggers a corresponding behavior in the system [6]. The verification environment keeps track of simulation time, use case and associated test cases, stimuli and system's responses.

C. Power estimation

To estimate power and energy dissipation the RHEiMS framework from Neosera Systems Ltd. [16] is used. It achieves high estimation accuracy by utilizing pre-characterized values from low-level power models stored in its database. Through cased-based reasoning and statistical analysis of system activity and input vectors a power and energy profile is created during simulation. Consequently, RHEiMS is used at system level to benefit from fast simulation speeds while retaining the high precision of low-level power estimation [16].

D. Verification of battery lifetime requirements

To predict battery lifetime the system's power dissipation for the application needs to be determined. Therefore, the automatically generated SystemC verification environment is connected to the system. The schedule of test cases is executed according to the *Scenarios* of the *Application*. During simulation the system reacts to the test cases by performing functionality. A corresponding power consumption is estimated.

Resulting from power estimation with RHEiMS, a power and energy profile is created for the executed sequence of test cases. Power estimation is repeated until profiles exist for each *Scenario* of the *Application*. The profiles are comma-separated-value (csv) files containing time-stamps and the associated power or energy value respectively.

From the energy values of all scenarios and the specified duty cycle a SystemC demand model of the specified application is automatically generated. At each simulation time-step the model repeatedly applies the application's energy pattern to the connected battery model. When the battery model runs out of charge the simulation stops and simulation time is verified against the specified battery lifetime for the application. Since the SystemC demand model simply repeats the demand pattern, simulation speed and estimation accuracy only depend on the complexity of the battery model.

E. Verification of power constraints

After the battery lifetime is verified power constraints are specified for the design. To achieve the battery lifetime and to avoid power peaks and thermal issues the system's functionality has to fulfill power constraints. The power constraints can be derived from the duty cycle and battery lifetime. They are imposed on an entire use case or on individual steps.

To verify the specified power constraints the system is simulated with the verification environment and power estimation is performed. All test cases for each use case are executed to trigger a behavior in the system. Again, RHEiMS estimates the corresponding power dissipation for the behavior.

After simulation the RHEiMS power profile for each use case is analyzed automatically. The power value is compared to the corresponding constraint in the *Power Constraints* section of the use case document. If a violation is detected, use case, step and the power value are reported.

F. Verification of power state requirements

The power aware design is implemented according to the IEEE 1801 standard (UPF) parallel to the HDL system model. It contains power domains, the power supply network, power states and supply voltages. However, it can not be simulated by itself. Therefore, an executable supply network is generated to verify the power design. Information about modules, ports and connections are extracted from the system. Then, the power design is parsed and automatically translated into a parallel hierarchy (Fig. 3). For each element in the power aware design a corresponding module is created in SystemC. Inside each module a monitor logs simulation time, power states and supply information.

For the UPF supply ports SystemC modules are created accordingly. Each module propagates its voltage level to the

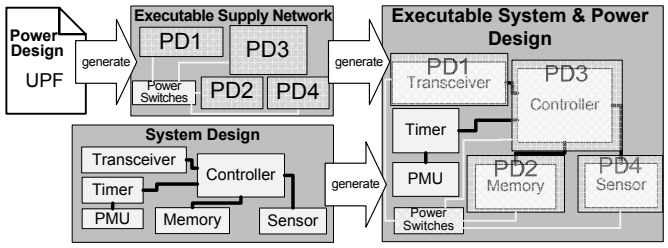

Fig. 3. Automatic generation of the executable supply network

TABLE IV
VERIFICATION RESULTS FOR THE POWER CONSTRAINTS

Use case	Power constraint (mW)	Power estimated (mW)
COLLECTION_UDB	70	46.36
READ	70	51.7
WAKEUP	50	49.28
IDLE	50	42.99
SLEEP	50	49.3
STANDBY	0.1	0.095
SAMPLING	7	6.12

connected supply network as floating point value. If a supply port is "off" or at ground negative infinite is assigned.

The power domains are translated into SystemC modules containing the specified system modules as sub-modules. The domain's supply and ground are connected to the supply network. A monitor constantly logs the domain's supply state.

To switch between supply voltages and to deactivate domains power switches are created. The switch is sensitive to the signal of its controller module. Depending on the control signal the switch applies a supply or "off" state to its output.

Finally, the specified supply network is used to connect the newly generated power design modules.

When the verification environment launches test cases during simulation different power states are entered. The modules in the executable supply network propagate their state throughout the network. When the control signals for power switches are applied the monitors inside each module log the change in the supply network. After simulation the logged events are automatically evaluated. For each use case iteration the simulated power states are verified against the power state requirements. Also, each power domain state, sequences for control signals and supply states are compared to the specification. A verification report informs the system designer about discrepancies between simulation and requirements.

IV. CASE STUDY

To demonstrate our methodology we implement a SoC in a case study. We design an active, higher-class RFID tag (HCT) intended for a refrigeration monitoring application.

Initially, the ISO/IEC 18000-7 protocol [20] for higher-class tags was analyzed. According to the protocol, use cases are specified for the wakeup, collection with UDB, read and sleep commands. Further, use cases for sampling, standby and idle mode are created. As representative example, table I contains the collection with universal data block (UDB) use case. The use cases describe the tag's functionality, state transitions and interactions between RFID reader and tag.

After the use cases are specified the application to monitor temperature is designed. Every second the tag takes a temperature sample from the sensor and stores it in memory. Once per hour an employee (or a machine) reads the temperature samples in order to detect interruptions in the cooling chain. Therefore, the wakeup command is sent and the HCT enters idle mode waiting for further commands. With the collection command the tag's ID is determined. Retrieving all samples from the tag's memory would need excessive read operations which would soon exhaust the tag's battery. Instead only two 16 bit numbers for maximum and average temperature during the last hour are read. Upon receiving a sleep command the tag enters standby mode. Table II summarizes the application.

According to the use cases a system-level model of the higher class tag is implemented in SystemC. It comprises a transceiver, controller, timer, sensor, memory and power management unit. A 2.2 Ah Lithium battery is chosen as power source for our RFID tag. Four years is specified as minimum battery lifetime requirement for the temperature sampling application (table III). After generating the verification environment we started simulation and the test case sequence was executed according to its schedule. RHEiMS produced a corresponding energy profile for the application. From this profile a demand model was generated and connected to a simple, linear battery model. The model predicted a lifetime of 4 years and 66 days and the verification report indicated the battery lifetime requirement as successfully verified.

After knowing the power source the power constraints are specified so the HCT meets its lifetime requirement. Therefore, a power budget was calculated from the lifetime and the known duty cycle. It designates the available amount of power per hour. Transmitting data usually consumes the most power. Consequently, the collection and read use cases were assigned constraints of 70 mW. When the tag is active and receiving data the constraint is 50 mW. For the standby state power is restricted to less than 100 nW. Since sampling requires the tag to be partially active for short periods it has a power constraint of 7 mW. Table III contains the specified power constraints. Again the automatically generated verification environment is simulated with the tag's model. This time the schedule is executed on a per use case basis and the corresponding power is estimated. The power profile is verified against the specifications and the results are shown in table IV.

In the power state requirements three power states are specified (see table III). During wakeup, collection, read or in idle mode all power domains are on (ALL_ON). When the tag receives a sleep command it enters the standby state and its domains are off (ALL_OFF). Only the power management unit remains active to control the power switches. To take a sample the SAMPLE_ON state is entered activating the domains for controller, sensor and memory while the transceiver remains off. In total four power domains are specified. The power aware design is implemented in UPF describing the supply network, switches, power domains and power states. The

TABLE V
VERIFICATION RESULTS FOR THE POWER STATES

Use case	Specified power state	Simulated power state
COLLECTION_UDB	ALL_ON	ALL_ON
READ	ALL_ON	ALL_ON
WAKEUP	ALL_ON	ALL_ON
IDLE	ALL_ON	SAMPLE_ON
SLEEP	ALL_ON	ALL_ON
STANDBY	ALL_OFF	ALL_OFF
SAMPLING	SAMPLE_ON	SAMPLE_ON

executable supply network was generated from the power aware design and simulation started. While the verification environment applies test cases the monitors inside the supply network track the system's power states. An incorrect power state could be detected (table V). After correcting the bug the system was re-verified and succeeded.

To validate our results we compared the power model of the ISO/IEC 18000-7 RFID tag in our RHEiMS database to an equivalent implementation on a SoC development platform. Analysis of the RHEiMS power model and measurements from the SoC implementation shows high accuracy with an average error of 2.57% which corresponds to the usual precision of RHEiMS [16]. Calculating the battery lifetime for the tag with the Peukert formula ($C_p = I^k \times t$) yields an error of 1.17% compared to the simulation result.

V. CONCLUSION

In this paper we present a methodology to verify power requirements. Therefore, functional and non-functional power requirements are specified in a semi-formal use case format. Through semantic analysis test cases are derived automatically from the use case document. All test cases are scheduled inside a verification environment. During simulation the verification environment executes the test cases which triggers a corresponding behavior in the system. While performing this functionality the system's power dissipation is estimated. With the power estimation results and a battery model the battery's lifetime is predicted for an application and automatically verified against the requirements. Similarly, power constraints imposed on functionality are verified. To verify functional power requirements it is ensured that the system enters the specified power states. Therefore, an executable supply network is generated from the power aware design in UPF. When the verification environment applies the test cases in simulation the system enters the power state. The simulated power state is verified against the specified power state. Finally, the designer receives a detailed report about fulfilled or violated power requirements.

In a case study we apply our methodology to an SoC design of an active RFID tag for temperature monitoring in a cooling chain. Use cases are specified according to the ISO/IEC 180007 protocol for active tags. The lifetime requirement of four years for the temperature monitoring application was successfully verified. The power constraints for the individual use cases were specified and verified against the requirements. Finally, the executable supply network was generated from the power aware design. By simulating the system and supply network the specified power states were verified.

ACKNOWLEDGMENT

This project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the contract FFG 812424.

REFERENCES

[1] "IEEE Standard for Design and Verification of Low Power Integrated Circuits," *IEEE Std 1801-2009*, pp. C1–218, 2009.

[2] A. Crone and G. Chidolue, "Functional Verification of Low Power Designs at RTL," *Lecture Notes in Computer Science*, vol. 4644, pp. 288–299, 2007.

[3] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification the Complete Industry Cycle*. Elsevier/Morgan Kaufmann, 2005.

[4] R. Lissel and J. Gerlach, "Introducing new verification methods into a company's design flow: an industrial user's point of view," in *Design, Automation & Test in Europe, Conference & Exhibition, 2007. DATE '07*. IEEE, April 2007, pp. 689–694.

[5] W. Nebel, "System-Level Power Optimization," *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pp. 27–34, 2004.

[6] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiß, and M. Pistauer, "Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs," in *Systems Conference, 2008 2nd Annual IEEE*. IEEE, April 2008, pp. 1–8.

[7] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.

[8] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.

[9] The Power Forward Initiative (PFI), "A Practical Guide to Low-Power Design - User Experience with CPF," pp. 1–281, 2008.

[10] L. Benini, G. Castelli, A. Macii, E. Macii, M. Poncino, and R. Scarsi, "Discrete-Time Battery Models for System-Level Low-Power Design," in *IEEE Transactions on very large scae integration (VLSI) systems*, vol. Vol. 9. IEEE, October 2001, pp. 630–640.

[11] IDENTEC SOLUTIONS, http://www.identecsolutions.com/fileadmin/user_upload/PDFs/product_sheets/ILR/EN/ID.0601.EN_i-Q8.pdf, February 2009, 5/9/2009.

[12] J. Taneja, J. Jeong, and D. Culler, "Design, modeling, and capacity planning for micro-solar power sensor networks," *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pp. 407–418, April 2008.

[13] F. Fereydouni Forouzandeh, O. Mohamed, and M. Sawan, "Ultra low energy communication protocol for implantable body sensor networks," *Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008. 2008 Joint 6th International IEEE Northeast Workshop on*, pp. 57–60, June 2008.

[14] S. Mikami, T. Matsuno, M. Miyama, M. Yoshimoto, and H. Ono, "A Wireless-Interface SoC Powered by Energy Harvesting for Short-range Data Communication," in *Asian Solid-State Circuits Conference, 2005*, 2005, pp. 241–244.

[15] D. Sunwoo, H. Al-Sukhni, J. Holt, and D. Chiou, "Early models for system-level power estimation," *Microprocessor Test and Verification, 2007. MTV '07. Eighth International Workshop on*, pp. 8–14, Dec. 2007.

[16] Neosera Systems Ltd., "RHEiMS: Rapid Hierarchical Energy Investigation Modelling System," http://www.neosera.com, 2009, 5/6/2009.

[17] Rao, R. and Vrudhula, S. and Rakhmatov, D. N., "Battery Modeling for Energy-Aware System Design," *IEEE Computer*, vol. VOL. 36, no. NO. 12, pp. 77–87, December 2003.

[18] F. Simjee and P. Chou, "Accurate battery lifetime estimation using high-frequency power profile emulation," in *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*. IEEE, 2005, pp. 307–310.

[19] F. Bembaron, S. Kakkar, R. Mukherjee, and A. Srivastava, "Low Power Verification Methodology Using UPF," in *Conference on Electronic Systems Design and Verification Solutions, DVCON*, 2009, pp. 228–233.

[20] International Standardization Organization, "ISO/IEC 18000-7:2008 - Information technology – Radio frequency identification for item management – Part 7: Parameters for active air interface communications at 433 MHz," 2008.

IEEE SysCon 2009 —3rd Annual IEEE International Systems Conference, 2009
Vancouver, Canada, March 23–26, 2009

A Component Selection Methodology for IP Reuse in the Design of Power-Aware SoCs Based on Requirements Similarity

Christoph Trummer, Christoph M. Kirchsteiger, Christian Steger, Reinhold Weiß*,
Andreas Schuhai, Markus Pistauer† and Damian Dalton‡
*Institute for Technical Informatics, Graz University of Technology, Austria
email: (trummer, c.kirchsteiger, steger, rweiss)@tugraz.at
†CISC Semiconductor Design+Consulting GmbH, Austria
email: (a.schuhai, m.pistauer)@cisc.at
‡Neosera Systems Ltd., Ireland
email: damian.dalton@neosera.com

***Abstract*—To counter today's rising complexity in System-on-Chip (SoC) design intellectual property (IP) cores are reused. In libraries often many different pre-designed components are available and selecting suitable IP is difficult and laborious. Power consumption is a key constraint in mobile devices, where SoCs are often used. Therefore, it is important to consider power when selecting components for reuse in the current SoC design. This paper introduces our approach to support component search and selection. Initially, a repository and container for IP is presented. Onto our repository the novel component selection methodology is applied. It considers constraints and properties of IP. Then it performs a similarity analysis between system and component requirements. The result is a ranking of the best-suited components for reuse in the current system under design. To demonstrate our approach a case study is performed on a SoC. With our methodology ranking of components matching the system requirements and constraints is generated. Our work is part of the SIMBA[1] project which focuses on simulation-based requirements testing of power-aware SoCs.**

I. Introduction

A System-on-Chip (SoC) is a system consisting of a multitude of different components integrated on a single chip [1]. During the design of today's System-on-Chips (SoCs) engineers are faced with continuously increasing complexity. As a result of the high silicon integration densities more and more functionality uses the same or even less chip area. Consequently, the industry is facing two important issues. First, the productivity of system designers cannot keep up with the increasing SoC complexity (design-productivity gap) [2]. Second, power dissipation rises as dynamic and especially leakage power increase with the reduction of SoC area [3], [4].

These issues are solved by reusing IP components and by applying power-aware methodologies in SoC design. However, information about the IP core's power dissipation is needed early in the design flow. Early, system-level power estimation permits the identification of critical parts of the design.

[1]The Simulation-based Requirements Testing of Power Aware SoCs (SIMBA) project is funded by the Austrian Federal Ministry for Transport, Innovation, and Technology under the contract FFG 812424.

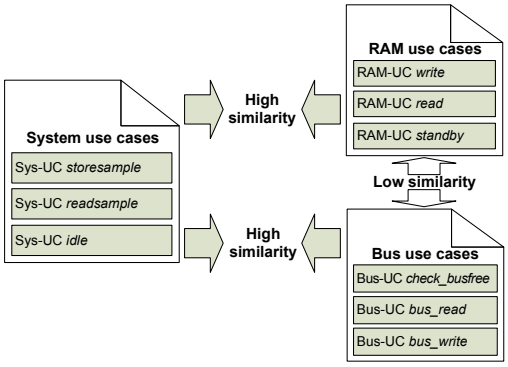

Fig. 1. Similarities between system and component use cases

The power demand can be decreased by choosing different components and by applying power-aware design techniques.

When reusing IP the main difficulty is to find and select components which are best suited for the current system under design. This is the so-called component selection problem [2]. To find the proper IP system engineers need to select components which contribute to fulfill the system's requirements and satisfy its constraints. Due to the many functional requirements and the complexity of the SoC under design this is not a trivial task. Available third-party or legacy IP may consist of different implementations, formats and vary in available information. Manually finding suitable components in this incoherent set of IP needs considerable time and effort. The set of available IP may also be large which complicates manual search even more. Consequently, an automated approach to search and select the appropriate component is highly desirable.

Our contribution comprises two parts. First, parameters for IP selection are investigated to serve as basis for our IP container and repository. Second, we propose automated matching and selection of IP components from this repository.

9781-4244-3463-3/09/$25.00 ©2009 IEEE

Our approach is based on similarities in the IP's requirements and the requirements of the system under design. Fig. 1 illustrates the main idea behind our approach. The result is a ranked list containing components which correspond to the system's functionality and match the given constraints. Extensive research in related work about component selection allows us to believe that our approach is the first to use automated similarity analysis of requirements.

This paper is organized as follows. In section II related works on component selection, requirement formats, similarity analysis and IP representation are investigated. Section III explains our IP repository and components. The next section (IV), describes the component selection methodology using search space reduction and similarity analysis is described. Our methodology is demonstrated on a case study in section V. Finally, section VI summarizes our approach and outlines future work.

II. Related work

A. Hardware requirement specification formats

Requirements can be expressed in many different ways. They are often ambiguous, imprecise and misleading. Most commonly requirements are specified informally, in natural text. Natural text is easy to read but is usually ambiguous and difficult to process automatically. Whereas formal requirements have a clearly defined structure and can be processed easily. However, understanding them is difficult without prior knowledge of the syntax [5]. Semi-formal requirements such as textual use cases are a trade-off between readability and automatic processing capabilities [5].

In the software domain the unified modeling language (UML) has been widely used to provide a common and concise way to express requirements. In [6] the authors present a methodology which uses the UML to specify requirements in hardware design. Non-functional requirements can also be included into the UML descriptions and diagrams [6].

Semi-formal use cases contain a predefined structure which consists of a series of steps and interactions expressed in natural language [7]. This structured, textual use case format format is similar to the UML use cases described in [6]. Moreover, the semi-formal use case format can be used to express hardware requirements [5].

B. Component selection

The target of component selection is to find a subset of components matching the system's functionality and its constraints [2], [8]. Methodologies have been researched in both the hardware and software domain.

Some of these methodologies for component selection use similarity analysis of the component's implementation and/or its interface definitions [9], [10]. Others rely solely on querying for keywords which describe functionality and constraints [8], [11]. A more sophisticated approach uses fuzzy logic for keyword and constraints matching to select components [2].

Few approaches exist which include requirements for selecting components. However, they either rely on a manually created matrix that links similar requirements [12] or on complex additional specifications [10].

From the related works it is apparent that the most important criteria for selecting suitable components is functionality [2]. Additionally, system constraints especially power and delay as well as implementation details (e.g. architecture, technology) need to be considered when selecting IP for SoC design [11]. The IP component itself needs to be linked with the additional information and has to be archived and managed.

C. Textual similarity analysis

Various textual similarity analysis approaches exist. A few exemplary methods will be mentioned as evaluating all approaches is beyond the scope of this work.

A common approach is to use word occurrence to determine text similarity [13]. Thereby, keyword occurrence in documents is analyzed and compared based on a vector space model and information retrieval methods [13]. However, if the sentences are expressed differently by using synonyms very low similarities are reported.

A similar approach analyzes the documents on a per sentence basis [14]. Each word is represented by an integer hashcode which makes comparing sentences very fast. Two equal sentences have the same integer sum. Similarity of sentences is detected by comparing the intersections of their sets of unique words [14]. By representing the words as integers the approach is able to perform a fast similarity analysis.

The approach described in [15] uses natural language processing techniques to detect similarities in documents. Therefore, syntactic and semantic evaluation is performed. Additionally, the frequency of word occurrences is also taken into account for similarity analysis [15].

D. IP representation formats

Many different formats exist for representation of IP [16]. Most electronic design and automation (EDA) companies have their own proprietary standard or format. This means over time a collection of legacy, third-party and in-house IP is formed. Searching in such an inhomogeneous set of different IP formats with varying details and representations is very difficult and laborious.

Recently, a common format for IP component representation and exchange was introduced by the SPIRIT Consortium. The SPIRIT IP-XACT format uses the extensible markup language (XML) and describes intellectual property for development, implementation and verification [17]. IP-XACT is tool-independent and flexible as it can be augmented with additional information [17], [18].

III. IP repository and component design

In the first part of this section the structure of a IP container to store and search for an IP component is explained. The following subsections describe the component and the semi-formal use case format used in similarity analysis.

A. IP repository

To be able to find the IP core for reuse in the current design we first need to define a repository where it can be archived. Apart from being able to store and manage the many different components it also needs to support categorization, classification and description of IP. Within this repository searching is necessary to retrieve and select the IP.

Therefore, the IP repository L is defined as a hierarchical set of categories K.

$$L\{K_1, K_2, K_3, ..., K_m\} \tag{1}$$

Each set of categories K may contain a hierarchy of further categories or components C.

$$K\{K_i, K_{i+1}, ..., K_n, C_1, C_2, ..., C_j\} \tag{2}$$

To efficiently find and select suitable IP components each set L, K and C is defined as unique entity.

B. Components

Components need a representation that includes functional and non-functional information (e.g. constraints, documentation). Functionality is described in the IP component's source-code and its requirements document. Extracting functionality out of source code in different hardware description languages (HDLs) needs complex semantic analysis. Additionally, the analyzed functionality needs to be classified to enable searching for it. Consequently, a language-independent, additional description of the component's behavior is necessary. To avoid additional complexity and effort we suggest to use the already existing requirements document. In our case requirements are expressed in structured semi-formal use cases.

With non-functional information the situation is more complex. The information may not be available or incomplete since the IP component can be legacy IP, from a third party or may be still under development.

In short, a component is a collection of information which has to be archived in the repository for management and later use. We define such a collection of an IP core and its related data as component. However, it is not assumed that each component C is complete since all information may not be available. Consequently, the component C can be defined as a set of use cases, HDL source-code, verification-related data (VIP), documentation, constraints and properties.

$$C\{UC, HDL_{code}, VIP, Doc, Const, Prop\} \tag{3}$$

Within UC the component's requirements are stored as a set of semi-formal use cases (refer to subsection III-C).

The HDL_{code} is a set of files containing the component's source-code in a hardware description language. Verification IP (VIP) similarly comprises testbenches, results and additional verification-related documents.

The documentation (Doc) consists of a mandatory brief description of the component in natural text. Optionally, additional documents such as application notes and code-documentation can be included in Doc.

TABLE I
HIGH-LEVEL USE CASE - WRITING TO A SRAM

Name: WRITE
Description:
This use case describes a write operation to the static RAM.
Primary actor: Memory controller
Supporting actor: -
Trigger:
1. The SRAM receives the WRITE_ENABLE_COMMAND.
Primary Scenario:
1. The SRAM receives the WRITE_COMMAND.
2. The SRAM receives the ADDRESS.
3. The SRAM receives the DATA.
4. The SRAM writes the DATA to the ADDRESS.
5. The SRAM sends the ACKNOWLEDGE.
6. The SRAM goes to the OUTPUT_DISABLE usecase.
Alternative Scenarios:
Alternative Scenario 1
1a_1. The SRAM receives the WRITE_DISABLE_COMMAND.
1a_2. The SRAM sends the ACKNOWLEDGE.
1a_3. The SRAM goes to the OUTPUT_DISABLE usecase.
Constants:
ADDRESS=@bool[14:0]
DATA=@bool[7:0]

The constraints $Const$ comprise maximum and average values for power and delay. Since some or all entries for the constraints may not be available $Const$ is entirely optional.

The properties ($Prop$) contain keywords and informations about architecture and technology. The architecture field contains information about structural specifics (e.g. serial/parallel, synchronous/asynchronous). Moreover, it describes the implementation's level of abstraction (e.g. system-level, register transfer level). Technology specifies the library used for synthesis and power estimation. These are optional entries as they may not be available yet.

We found that the IP-XACT format would be well-suited to represent our IP container. Due to its flexibility it can be extended to accommodate all information of C. Its XML format allows easy parsing and processing of the stored data.

To determine a component's power dissipation the RHEiMS power estimation framework from Neosera Systems Ltd. [19] is used. RHEiMS employs statistical methods and case-based reasoning to estimate power. It compares similarity of input and output of the current component to a similar component with known power consumption [20]. Thereby, it is capable of rapidly estimating the component's power dissipation.

The results are stored in the power entries of the $Const$ field in C and taken into account when selecting the component.

C. Use case format

We refine requirements into semi-formal use cases as described in [7], [5]. The use cases describe interactions between a system (or component) and its environment. Thus, the functionality of a component is entirely described by the set of all its use cases. The semi-formal use cases can contain both functional and non-functional requirements. They are structured into several sections which contain natural text.

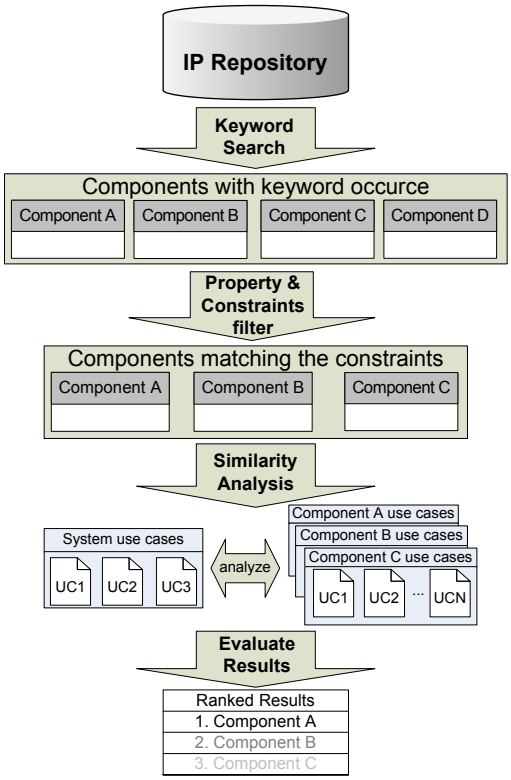

Fig. 2. Component selection methodology

The most important sections of each use case include the use case name, a brief description, the actor(s), the primary scenario and alternative scenario(s) (see table I). With the use cases high level requirements can be expressed and further refined into low-level requirements. Due to its structure the semi-formal use cases can be processed easily which is ideal for fast search (see also [5]). Our use cases are stored in XML which complements its structure.

IV. Methodology for component selection

Our component selection methodology comprises two phases. First, it filters all available components based on keywords and non-functional information. Second, similarity analysis of the individual use cases is applied to the remaining components to determine the most suitable. Fig. 2 illustrates the concept of our component selection methodology.

A. Keyword search and filtering

Since an exhaustive similarity analysis over all available components is computationally intensive, it is desirable to reduce the search space. The difficulty is to remove unsuitable components from the search without inadvertently removing useful ones. Consequently, we evaluate components based on simple keyword occurrence and generate a ranking. The better the component matches the search criteria the higher its rank. Keywords are searched in the properties, the use case description and the component description.

$$f(keywords) : L \rightarrow R \tag{4}$$

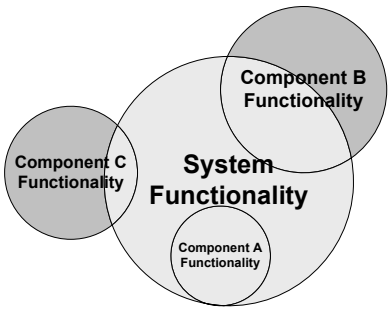

Fig. 3. Functional similarities between components and system

$$R\,\{keywords \in C_i, ..., keywords \in C_j\} \tag{5}$$

We reduce the search space even further by removing unsuitable architectural implementations or components not matching power and timing constraints.

$$g(Const, Prop) : R \rightarrow R' \tag{6}$$

$$R'\,\{(Const, Prop) \in C_k, ..., (Const, Prop) \in C_l\} \tag{7}$$

After keyword search and filtering by constraints $Const$ and properties $Prop$ (e.g. architecture) we receive a set of components. Onto this set R' the similarity analysis is applied.

To retrieve a list of all potentially suitable components for the system keyword search may be omitted. Since constraints or implementation details for a component may not be known in early design stages filtering is optional as well.

B. Similarity analysis

A suitable component for the system needs to be able perform some of its functionality. Therefore, functionality of the system and the component is overlapping. Consequently, the overlapping functionality can be performed by the respective component. Fig. 3 schematically displays three functionally different components and the target system. The larger the overlapping area the more functionality of the system can be performed by the component. The more the functionality of the component corresponds to the system's functionality the better it is suited for the system. In our case, functional requirements are independently expressed in the system's use cases and in the component's use cases. For this reason parts of the system's use cases and the component's use cases are very similar.

We exploit this commonality by applying similarity analysis h on the use cases of each component in R'. The function h compares all use cases of the system under design to each use case of the components in R'. This produces a list of components with the highest similarity to the system's use cases. The algorithm used in similarity analysis compares sentences for their similarity and is described in [14].

After determining the use cases with the highest similarity a ranking for similarity analysis is calculated for the components. Use cases from components showing less than 55% similarity are thereby discarded as they show too little resemblance to the system's functionality.

$$h(UC_{system}) : UC_{component} \in R' \rightarrow R'' \tag{8}$$

The previous ranking of components R' is updated with the new information from similarity analysis. The new set R'' now contains the final ranked list of components. This ranking contains the best suited components for the system under design.

V. Case study

In this section we demonstrate our component search methodology on a case study. An IP repository was created and filled with components. The following subsections describe the system and two cases in which methodology is applied.

A. System-under-design

As example for a SoC we consider a small system for temperature monitoring. The system is capable of measuring ambient temperature and digitizing the temperature samples. For long term monitoring the system stores the samples into its memory. Internal communication is performed by sending data over a bus. A higher class radio frequency identification (RFID) tag [21] could be an application for such a system.

Based on the system's functionality potentially suitable components are a temperature sensor, an analog-digital converter (A/DC), a bus, a memory and a controller. The functional requirements of the system-under-design are expressed by three high-level use cases.

1) *Storesample*, on wake-up the system sends a query over the bus, reads the output of the A/DC and stores it in the memory.
2) *Retrievesample*, on a command the system sends a query over the bus and reads the sample from the memory.
3) *Idle*, the system is idle until a timer interrupt occurs or it receives an external command.

Our sample IP repository comprises 15 different components typically used in SoC design. All components consist of different data, constraints, properties and use cases. Information on some components is left partially incomplete to represent legacy or third party IP. Some components are able to perform the same functionality. Not all components would be suitable for the target system under design.

To select components for the above system we consider two cases. In the first case we want to select suitable components based on their functionality. The search for suitable IP is conducted without constraining the search space. All use cases of the components in the repository are analyzed for their similarity to the system. The second case describes the selection of a specific component. In this case it is a suitable memory component. The search space is reduced by applying the filter to the components in the repository. Then the use cases are analyzed to select the best suited memory component based on its functional similarity.

B. Case 1 - Functionality search

In this example we perform a similarity analysis h over all component use cases. This means the system's *idle*, *storesample* and *retrievesample* use cases are compared to all use cases of the components in the repository.

TABLE II
Functionality search - results similarity analysis

Component	#UC 55-60%	#UC 60-65%	#UC 65-70%	#UC 70-75%	#UC 75-80%
ADC	0	0	0	0	3
ComparatorA	0	0	0	0	0
ComparatorD	2	0	0	0	0
DAC	0	0	0	0	3
EEPROM	0	1	0	0	2
FLASH	1	0	2	0	0
I2C bus	0	2	0	1	0
RAM_A	1	0	2	0	0
RAM_B	0	0	2	0	0
RndNumGen	0	2	0	0	0
EPROM	0	2	0	0	0
SimpleBus	0	0	1	0	0
TempSensorA	0	2	0	0	0
TempSensorD	0	0	2	1	0
Timer	1	0	1	1	0

Table II shows the number of component use cases with the highest similarity (in %) to the system use cases. The higher the similarity the more functionality of the IP corresponds to the functionality of the target system. From this list a ranking of the five most suitable components is calculated.

1. A/DC
1. D/AC
2. EEPROM
3. TempSensorD
4. I2C bus
4. Timer
5. FLASH
5. RAM_A

The above list displays the ranking from the most suitable to the least suitable component based on its functionality. In fact almost all components would be suitable for our system. The A/DC component converts analog samples into digital values. TempSensorD is a digital temperature sensor with built-in analog to digital conversion. The EEPROM, FLASH and RAM_A memories store the temperature data. Bus functionality is covered by the I2C component and the timer's interrupts awake the system from its idle state.

The only exception is the digital-analog converter (D/AC) which does not fit into the system under design. However, at high level the component's functionality is very similar to the A/D converter and therefore, it receives the same ranking.

Although, similarity analysis of use cases is capable of selecting functionally suitable components it is advisable to narrow the search space. Firstly because the exhaustive search is computationally intensive. Similarity analysis over our small set of IP is quite fast (ca. 30 sec) but speed decreases with the size of the repository. Secondly, false positives can be better avoided by providing additional parameters for selection.

C. Case 2 - Specific component search

To show selection of a specific component for the system-under-design our entire methodology is applied. The component we want to select from the set of components is a memory allowing to store and retrieve sensor data.

TABLE III
RANKED RESULTS FOR COMPONENT SEARCH

Ranked set after f	**Ranked set after g**	**Ranked set after h**
1. RAM_B	1. RAM_B	1. EEPROM
2. RAM_A	2. EPROM	2. FLASH
2. EPROM	2. EEPROM	2. RAM_B
2. EEPROM	2. FLASH	3. EPROM
2. FLASH	-	-

The initial search $f('memory')$ returns a ranked subset of five components which contain the "memory" keyword. The intermediate results are displayed in column one of table III. As additional constraints for the filter g we choose not to limit architecture but to keep power consumption low. The filter $g('maxpower < 200e-3')$ excludes components dissipating more than 200mW. The remaining components are displayed in the column two of table III. Finally, similarity analysis h is applied and the ranked results for the search of a memory component are listed (table III).

The resulting components are memory components with less than 200mW of maximum power dissipation. Although the EPROM cannot be written to except for reprogramming it achieves some similarity. This is because part of its functionality (i.e. read) overlaps the system's functionality.

The ranking should serve as a guideline for the system designer to support in selecting components. However, the designer has to make the final decision based on the suitability and additional information (i.e. verification summary).

VI. CONCLUSION AND FUTURE WORK

In this paper we presented an approach to search and select the best suited components for a given System-on-Chip design. Initially, crucial parameters for successful selection of IP were identified. With regards to these aspects an IP container and repository were designed. To select the most suitable IP from the repository a similarity-based approach was introduced. Our methodology exploits the fact that the component's functionality is described in its use cases and the system's use cases. Consequently, the use cases show high similarity and the matching component can be determined through similarity analysis.

To demonstrate our approach our methodology was applied to a case study of a system for temperature measurement. Two different cases for component selection were explored. The first search intended to select components with functionality matching the target system. The resulting list of components clearly demonstrated the feasibility of our approach. Second, a search for a specific component (i.e. low-power memory) was performed. The selection methodology provided a ranked list of the best suited memory components.

Future work includes integration of the methodology into our project's simulation-based verification flow. Therefore, we plan to consider the IP's verification history in our component selection methodology.

REFERENCES

[1] C. Wenwei, Z. Jinyi, L. Jiao, R. Xiaojun, and L. Jiwei, "Study On a Mixed Verification Strategy for IP-Based SoC Design," in *High Density Microsystem Design and Packaging and Component Failure Analysis, 2005 Conference on*, June 2005, pp. 1–4.

[2] T. Zhang, L. Benini, and G. De Micheli, "Component Selection and Matching for IP-Based Design," in *Design, Automation and Test in Europe, 2001. Conference Proceedings*. IEEE, March 2001, pp. 40–46.

[3] K. Usami, "Overview on Low Power SoC Design Technology," *Proceedings of the 2007 conference on Asia South Pacific design automation*, pp. 634–636, 2007.

[4] T. Hattori, "Challenges for Low-power Embedded SOC's," *VLSI Design, Automation and Test, 2007. VLSI-DAT 2007. International Symposium on*, pp. 1–4, 2007.

[5] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiss, and M. Pistauer, "Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs," in *Systems Conference, 2008 2nd Annual IEEE*. IEEE, April 2008, pp. 1–8.

[6] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.

[7] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.

[8] G. Hamza-Lup, A. Agarwal, R. Shankar, and C. Iskander, "Component selection strategies based on system requirements' dependencies on component attributes," in *Systems Conference, 2008 2nd Annual IEEE*. IEEE, 2008, pp. 1–5.

[9] D. Mathaikutty and S. Shukla, "SoC Design Space Exploration through Automated IP Selection from SystemC IP Library," in *International SOC Conference, 2006 IEEE*. IEEE, September 2006, pp. 109–110.

[10] L. Wang and P. Krishnan, "A Framework for Checking Behavioral Compatibility for Component Selection," in *Proceedings of the Australian Software Engineering Conference (ASWEC'06)*. IEEE, 2006, pp. 49–60.

[11] L. Reynari, F. Cucinotta, A. Serra, and L. Lavagno, "A Hardware/Software Co-design Flow and IP Library Based of Simulink™," in *Proceedings of the 38th Design Automation Conference (DAC'01)*. IEEE and ACM, 2001, pp. 593–598.

[12] M. Martinez and A. Toval, "COTSRE: A COmponenTs Selection Method Based on Requirements Engineering," in *Composition-Based Software Systems (ICCBSS) 2008, Seventh International Conference on*. IEEE, 2008, pp. 220–223.

[13] The Apache Software Foundation, "Apache lucene," ttp://lucene.apache.org/java/2_3_0/scoring.html, 2006, last visited - 02/02/2009.

[14] C. Collberg, S. Kobourov, J. Louie, and T. Slattery, "SPlaT: A System for Self-Plagiarism Detection," in *Proceedings of IADIS International Conference WWW/INTERNET 2003*, 2003, pp. 508–514.

[15] K. Indukuri, A. Ambekar, and A. Sureka, "Similarity analysis of patent claims using natural language processing techniques," *Conference on Computational Intelligence and Multimedia Applications, 2007. International Conference on*, vol. 4, pp. 169–175, Dec. 2007.

[16] M. Visarius, J. Lessmann, W. Hardt, F. Kelso, and W. Thronicke, "An xml format based integration infrastructure for ip based design," *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, pp. 119–124, Sept. 2003.

[17] The SPIRIT Consortium, "IP-XACT v1.4: A specification for XML meta-data and tool interfaces," http://www.spiritconsortium.org, 2009, visited - 14/1/2009.

[18] M. Strik, A. Gonier, and P. Williams, "Subsystem Exchange in a Concurrent Design Process Environment," *Design, Automation and Test in Europe, 2008. DATE'08*, pp. 953–958, 2008.

[19] Neosera Systems Ltd., "http://www.neosera.com," 2009, last visited - 01/02/2009.

[20] Dalton, McCarthy, Quigley, and Leeney, "A system-level power evaluation method," Irish Patent PCT 31 498IESP, 2008.

[21] EPC Global Inc., "EPC Radio-Frequency Idendity Protocols Class-1 Generation-2 UHF RFID Protocol for Communication at 860 MHz 960MHz Version 1.1.0," 2007.

An IP-XACT Library extended with Verification Information for Functionality-based Component Selection

Christoph Ruggenthaler[1], Christoph Trummer[1], Christian Steger[1], Reinhold Weiß[1],
Andreas Schuhai[2], Markus Pistauer[2], Damian Dalton[3],
[1]Institute for Technical Informatics, Graz University of Technology, Austria
[2]CISC Semiconductor Design+Consulting GmbH, Austria
[3]School of Computer Science and Informatics, University College Dublin, Ireland
ruggenthaler@student.tugraz.at, {trummer, steger, rweiss}@tugraz.at,
{a.schuhai, m.pistauer}@cisc.at, damian.dalton@ucd.ie

Abstract

In the Electronic Design and Automation (EDA) industry a gap between design and productivity exists and is even expanding. Also, consumers demand fancy products and new features at a faster pace. Consequently, to fulfill these rigid design cycles companies are reusing Intellectual Property (IP) components provided by third-party suppliers instead of creating them anew. However, no common format exists amongst companies which complicates IP exchange and reuse. Moreover, verification information, if existing, is provided separate from the IP.
This paper introduces a novel approach to enable IP exchange with added verification information and metadata. To represent IP the new IP-XACT format is extended. Our developed library is able to store the extended IP with all its resources. The novel component selection strategy uses this information to retrieve matching IP based on their functionality and achieves accurate results.

1 Introduction

In today's fast moving economy hardly any other business is as driven by innovation as the EDA industry. Customers demand a steady stream of improved and new products to make their life easier, more comfortable or simply more entertaining. This forces the companies to release new products frequently.

The increasing number of product requirements makes system design more and more complex. Due to the strict time-to-market constraints System-on-Chip (SoC) designers cannot create all parts of the product from scratch. Therefore, pre-designed IP components are reused to fulfill the numerous requirements and to reduce the complexity of the project. However, system architects have to find the best-suited IP within a large amount of available components. The IP is usually supplied from thrid-party vendors or is available from previous projects [17].

Until recently no common standard for representation and delivery of IP components existed. Consequently, each company uses a different, proprietary storage and exchange format for its IPs [12]. Moreover, either no verification information is included in the IP or it is provided separately, also in a proprietary format [15].

The new IP-XACT standard provides a format for Hardware Description Language (HDL)- and vendor independent IP representation. However, components still lack information describing the IP's architecture, functionality and verification status. The IP-XACT format is based on eXtensible Markup Language (XML) which can be easily extended. This leads to our approach outlined in Fig. 1.

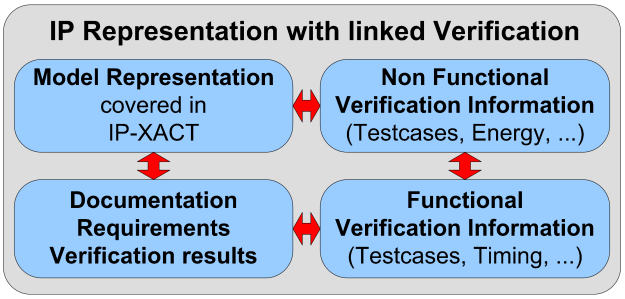

Figure 1. The Extended IP-XACT Format

Our IP is structured as follows. The HDL design is described in IP-XACT to ensure compatibility with any other compliant Design Environment (DE). Verification status and additional resources such as power estimation results or requirements are covered by our extensions. Consequently, we achieve an unique relation between the IP model and data which provides useful information about the verification status. Moreover, the extension allows using the additional information for conducting efficient and precise component search. Unlike traditional lookup searches our approach performs functionality-based component search. With similarity analysis the requirements of each IP in the library are compared to the requirements of the current system-under-design and achieves functionally matching results [15].

This paper is organized as follows. The next section (2) is dedicated to related work in the areas of IP representation, component selection and IP libraries. Section 3 explains our novel methodology and discusses the design of our library. In section 4 a case study evaluating this approach is performed. Finally, section 5 summarizes our work and provides an outlook to future improvements.

2 Related Work

2.1 IP Representation Formats

Trying to enable fast and efficient IP exchange and integration leads to the problem of diversity in SoC representation formats. Commonly proprietary solutions are used within companies. Any newly acquired third-party IP is converted into this proprietary format [16]. Common standards are mostly limited to special purposes or certain levels of design abstraction [4], [13]. IP formats which cover additional metadata are scarcely used. Instead, additional resources (e.g. verification status, documentation) are either missing or delivered separately from the IP. Furthermore, most formats do not encapsulate the used HDL for separation of source and IP description.

2.1.1 The Open Modeling Coalition IP format

Silicon Integration Initiative (Si2) launched the Open Modeling Coalition (OMC) with many participating EDA companies such as ARM Ltd., IBM Corp., Cadence Design Systems Inc., NXP Semiconductors. The Si2 integrates 3rd-party IP into designs through an unified characterization and modeling format. Further, it deals with delay modeling, statistical timing and increased model accuracy for 90nm and 65nm technologies [10] [13].

In general, the format aims for Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA) projects using traditional design flows. For high level and non-traditional design flows extensions have to be adopted and integrated in the DE. These are specific techniques enabling reuse and new formats describing the additional functionality [18]. This means the OMC's format is not intended to cover the entire IP and all its sources. Instead, the format describes the functional criteria for the designated IP [13].

2.1.2 Advanced Library Format

The Advanced Library Format (ALF), IEEE Std. 1603-2003, specifies a modeling language for IP technology, cells and blocks. In contrast to the Si2's format it is designed for low level descriptions from Register-Transfer Level (RTL) to physical level. The specification covers many facets of models such as behavior, timing, power and signal integrity. Since it does not allow to include IP from higher levels as Transaction-Level Modeling (TLM) or Electronic System Level (ESL) the format is rarely used today. In fact, the ALF is obsolete because of the rapidly changing demands of the EDA industries [4].

2.1.3 SPIRIT IP-XACT

The upcoming IEEE standard IP-XACT (IEEE P1685 Working Group) was founded by the SPIRIT Consortium in 2003. Since then it has drawn attention from EDA companies and communities. In comparison to former standards it focuses on vendor-neutral IP descriptions. IP-XACT enables multi-vendor design flows by creating views for specific purposes such as documentation, Verification IP (VIP), etc.. Through the possibility to create different views various abstraction levels can be implemented and combined in the same IP. Each view is distinguished by the type and desired DE for editing [1].

IP-XACT allows to add documentation or mixed abstraction levels from ESL to RTL SoC design. Since all resources and metadata are stored in XML documents it can be accessed with every programming language which supports XML processing. The sources are encapsulated and separated from the IP description. This means the HDL source code is embedded via links to the source file. Thus the HDL code can even be stored on a remote server [14].

2.2 IP Libraries

As a consequence from the lack of a common IP standard, no unified IP library is available. However, companies realized the need for a repository to store and manage their IP. Currently, available IP storage architectures can be categorized into local file-based IP collections, online platforms or client-server based repositories. One representative for each type is discussed below.

2.2.1 GRLIB IP Library

In this client-based library all components are modeled in Very High Speed Integrated Circuit Hardware Description Language (VHDL) and centered around a common on-chip bus (AMBA-2.0 AHB/APB). The files are provided under the GNU GPL license which means they can be utilized freely within the terms of use. Since the IPs are delivered in vendor- and tool independent form they can be efficiently included in current designs. Through Plug & Play functionality and common interfaces the components are easily configured and connected. For the GRLIB IP Library the IP's identification register is split into three parts. First, the vendor ID which is managed by Aeroflex Gaisler assigning unique strings for each company. Second, device IDs are appointed by the IP vendors to their components. Third, different implementations of the IP are distinguished by the version field.

Inside the library each component is uniquely identified by its name and contains subcomponents clustered in packages. As simulation and synthesis need different configurations most of the IPs provide both resources [2]. Currently, no metadata is included in the components. Thus, each IP can be found only by its name and version ID. This greatly impedes efficient IP search and selection.

2.2.2 NXP Yellow Pages

In 2006 NXP Semiconductors introduced a novel lookup service to enhance the reuse functionality for IP components using a client - server based architecture. This proprietary repository, called NXP Yellow Pages, is located in the internal NXP network and hosted on a secure server. Designers may browse through all available IPs and view

a profile page of each part. Therein attributes, documentation, deliverables and development status are shown [17].

For IP exchange the recently published (available on a commercial website) CoReUse standard [5] is used. This representation provides specifications, guidelines and templates for analog and digital IP. The underlying sources are described in SPIRIT IP-XACT [3].

Since the library only acts as a repository no sophisticated search queries including verification or (non) functional data can be performed.

2.2.3 Online Portals

In addition to the proprietary version from NXP other online platforms also host IP components either for free or commercially. ChipEstimate.com provides IPs from over 200 suppliers and foundries. However, its IPs are delivered by partners and have to be purchased separately.

IP from Opencores.org is specifically designed for FPGAs and also contains some related resources (e.g. documentation). The components are provided by a large community and published under free-to-use licenses.

The IP-Extreme.com portal also uses the CoReUse standard for their hosted IPs. Additionally, they provide tools for compliance checks and fast deployment into DEs.

Most online IP exchange platforms do not provide verification information which would support in choosing suitable IP. Furthermore, no sophisticated parametrized search is available to find functionally suited IP.

2.2.4 Academic Approaches

In [8] a Component Composition Language (CCL) is used for designing and describing IPs. CCL abstracts functionality to describe the design. On the other hand untyped or partially typed architectures are handled by the Metamodeling based Visual Component Composition Framework (MCF). Untyped or partially typed means either unknown or incomplete specification or a non-conform description which hinders automatic processing. This framework provides a solution of the so called "Type Inference Problem". Since the original MCF is not able to handle untyped architectures a multi-stage solution was introduced. This solves type assignment and substitution without violating or changing the original types and constraints [7] [9].

Another approach is outlined in [11]. In this paper Matlab Simulink™is used to determine hardware / software co-design aspects. The goal is to find a trade-off between hardware and software implementations. Both implementation possibilities provide benefits and drawbacks with respect to performance, timing, power consumption and flexibility. The described IP library consists of parameterized Simulink™blocks with additional implementation and model information.

2.3 Component Selection Strategy

In general, IP searches use parameters such as name, category or keywords to find desired IP components. Few IP formats support verification information and metadata. Due to this lack of additional information it is difficult to deploy sophisticated IP selection algorithms.

In the works from the previous subsection 2.2.4 some approaches are introduced. Their aim is to provide information by preprocessing fragmentary or non-conform metadata and sources to get an uniform description of each IP. The other approach describes co-design issues. The parameterized blocks are tagged as hard- or software. The algorithm assists in picking the optimal selection from a library. Depending on this selection, different values are estimated in two steps. First a directed graph is built. Second the overall performance is calculated. Thus the best co-design configuration can be chosen.

A new methodology is described in [15]. Each component contains metadata and use cases describing its functionality. The use cases are specified in a semi-formal format [6]. Component selection is performed in two steps. In the first step keyword matching and filtering of properties and constraints is used for pre-selecting potentially relevant IP. Through pre-selection and filtering the search space is reduced for the next step. In the second step, similarity analysis is applied to the use cases of this reduced set of components. This permits to search for components based on their functionality [15].

3 Novel Methodology and Design

In this section the design and methodology of our approach is described. First, we explain the extension of the IP-XACT standard for providing verification data and additional meta information. Second, an overview of the architecture and the component selection strategy is given.

3.1 IP-XACT Extension

IP-XACT allows including verification IP as whitebox elements or additional views to the component. However, more advanced descriptions can only be added via XML Schema extensions [14]. In our case a wrapper is added to the XML Schema to cover the additional data as outlined in Fig. 2. This enables compliance with the standard and allows import of existing IP-XACT conform IPs.

Each folder and file is linked relative to its corresponding XML file. This also depends on the resource type, properties restricting visibility, access or tools for editing. Each of the stored IP is assigned a unique ID instead of a name.

3.2 Project Summary

We link IP and VIP with all related results and estimation files. The System Architect Designer (SyAD®) performs energy and timing estimation and stores the results in separate files. Moreover, it facilitates verification for each use case resulting in information about successful of failed testcases and test coverage.

By extracting information from these result file we generate a comprehensive summary for each IP. In the summary properties, constraints and version information is

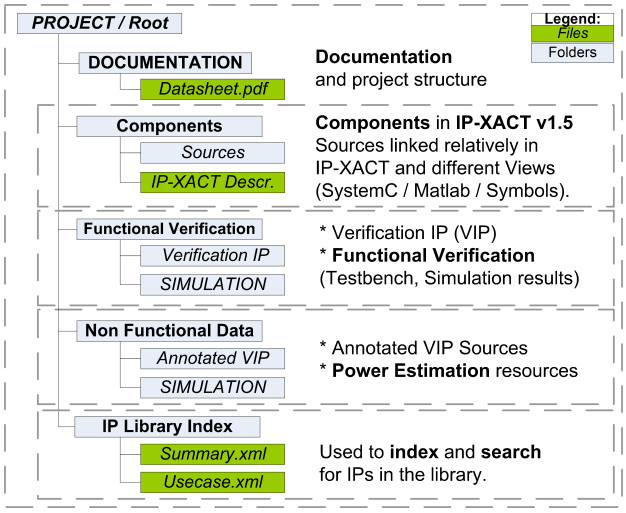

Figure 2. Extended Project Structure

stored. The summary is used for pre-selecting components from the IP library as described in section 3.4. However, not all stored data serves component selection. Some parts also determine the version identifier, design history and contain information for the system architect.

3.3 Library Architecture

As shown in Fig. 3 the architecture is split into two parts.

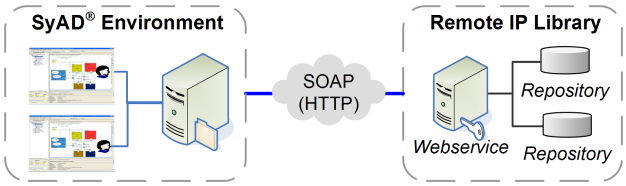

Figure 3. Architecture of the IP Library

The SyAD® DE is used as graphical interface for the SOAP over HTTP communication with our library. The library consists of a webservice which manages data exchange and provides user- and IP management functionality. As back-end a repository implementation is used to store the components and avoid concurrent interactions. Since HTTP is commonly used it enables the global IP infrastructures to be always accessible instead of client-based solutions or servers hidden behind firewalls.

3.4 IP Selection

The concept behind IP selection is described in [15]. For our approach we modify the first step regarding preselection and filtering. Our IP comprises properties (Prop) such as description, keywords, category, architecture and constraints (Const) about energy, timing and technology. Use cases are added for similarity analysis (see Fig. 4).

Since properties and constraints are written in natural language the search engine uses a Vector-Space Model.

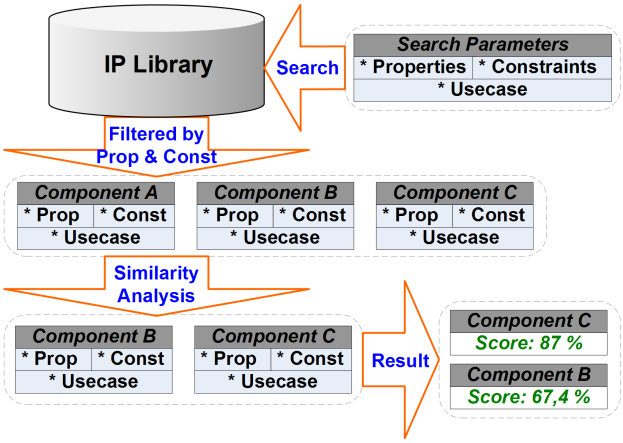

Figure 4. Component Search based on [15]

Each field is compared to the matching components in the library and ranked in a fuzzy or strict way to reduce the search space. A different algorithm is utilized for the semi-formal use cases (see [6]) which contain natural text structured into several sections. Similarity analysis takes advantage the informal sentence structure of the use cases and calculates a similarity score for each IP component. Finally, the components are ranked in percent of their suitability and matching functionality.

In comparison to other projects (see section 2) our approach is able to deal with more sophisticated information about the IPs. Thereby, IP can be searched based on its functionality. Consequently, results are more precise than those from simple keyword searches. However, search speed is slower with the functionality-based approach because of the more complex IP selection algorithm.

4 Case Study

In this section our approach is demonstrated. For the case study we designed a higher class RFID tag which is already equipped with transceiver, microcontroller and digital temperature sensor. It still lacks a memory component to store the temperature samples, which we will retrieve from the library. Our IP library consists of 30 components and covers a wide spectrum of component categories. As the focus for our search lies on different memory components, this category has been filled with various implementations. The next subsections describe two scenarios for selecting memory IP from the library.

4.1 Search with manually specified parameters

In the first scenario our design still misses a memory to store the temperature samples. Since the design is built from scratch no memory component has been chosen so far. Therefore, the designer needs to search by manually specifying criteria. Table 1 shows the search parameters for a RAM component. Since the library contains a tree of categories the designer may choose the appropriate. Parameters for timing and energy are omitted because they

are not available at the current stage of our design. Finally, we select the use cases for our target system the higher class RFID tag.

Field	Inputs
Name	-
Category	/Hardware/Memory/RAM
Keywords	ram, memory
Timing	-
Energy	-
Use cases	RFID_use_cases.xml

Table 1. Manually entered criteria for desired RAM IP

Step one filters and ranks the components regarding their properties and constraints in the IP summary. Table 2 enumerates the five best matches as intermediate results. The first four entries are RAMs which result from an exact matches for category and keywords. The REGISTER was selected due to a partial match in category and some corresponding keywords.

Score	Component	TCs & Coverage		
[%]	Name	passed	failed	[%]
81.9	RAM B	23	0	14
77.3	RAM C	44	1	47
77.1	RAM D	57	0	62
76.9	RAM A	78	0	83
28.9	REGISTER A	27	4	35

Table 2. Intermediate Top 5 Ranking

The next step compares the use cases of the pre-ranked IPs with the specified system use cases through similarity analysis. As this step directly relates to functionally matching components its weights for ranking are higher than the parameter search from the previous step.

Score	Component	TCs & Coverage		
[%]	Name	passed	failed	[%]
61.5	RAM C	44	1	47
61.0	RAM A	78	0	83
60.3	RAM D	57	0	62
53.5	RAM B	23	0	14
37.2	REGISTER A	27	4	35

Table 3. Final Ranking of Top 5 Components

The final ranking is listed in table 3. In comparison to the preliminary results (table 2) the score is lower for all matches. This is because the memory IPs are only able to fulfill parts of the system's functionality. Usually both steps are processed in one search query without intermediate results. Thus table 2 only serves demonstration. Since the summary also filters components only memory IPs are compared in the similarity analysis step.

Before actually selecting a RAM component we access the verification status linked to the IP. From the functional point of view "RAM C" would be best suited for our system. However, the verification status indicates that one testcase (TC) has failed. This means not all requirements have been verified successfully and the IP still needs refinement and re-verification. Consequently, we choose RAM A and add it to our design.

4.2 Design space exploration

In this example we already have a RAM component in our design and are performing a design space exploration. The currently used RAM IP is to be replaced with a different but functionally similar implementation to explore design alternatives. Since we still require a RAM in terms of functionality the parameters are automatically gathered from the current RAM component (see table 4).

Field	Inputs
Name	RAM
Category	/Hardware/Memory/RAM/
Keywords	CMOS static RAM, memory, stram
Timing	Min. $55ns$, Max. $100ns$
Energy	Max. $70nWs$
Use cases	static_RAM_use_cases.xml

Table 4. Parameters of a static RAM

After search we receive different results than in the previous scenario. Especially, the high influence of use case similarity can be clearly seen. RAM A is ranked higher than the other RAM implementations. However, the "read" use cases are also very similar for RAM and ROM. Thus, one ROM model is ranked amongst the five best entries in table 5.

Regarding the verification the best suited RAM A also provides the highest coverage of 83%.

Score	Component	TCs & Coverage		
[%]	Name	passed	failed	[%]
72.4	RAM A	78	0	83
66.7	RAM D	57	0	62
59.9	RAM C	44	1	47
49.0	RAM B	23	0	14
25.4	ROM A	36	0	23

Table 5. Design space exploration top 5 IPs

Both scenarios show that use case similarity analysis provides an innovative approach to select functionally suitable components. Additionally, to gain better confidence in choosing from the recommended IP the models are linked with verification information.

5 Conclusion and Future Work

In this paper a novel approach linking IP models with their verification information is presented. Based on an

IP-XACT extension an IP library was designed. The additional data is used to select components from the library based on functionality and constraints. Although, the search algorithm does not consider the IP's verification status it provides valuable information and supports the designer in choosing the proper component. In a case study two scenarios were considered for demonstrating our approach.

Future work includes improvement of performance and security. More IP is currently developed to facilitate further evaluation of the approach with a larger set of IP. Further, comparison to other similarity analysis algorithms and fine-tuning of the weight factors is to be performed.

6 Acknowledgements

This work is part of the Simulation-Based Requirements Testing of Power Aware System-on-Chips (SIMBA) project which is funded by the Austrian Federal Ministry for Transport, Innovation and Technology under the contract FFG 812424.

References

[1] V. Berman, S. Fazzari, C. Ussery, M. Indovina, M. Strik, J. Wilson, O. Florent, F. Rémond, and Bricaud P. Industrially Proving the SPIRIT Consortium Specifications for DesignChain Integration. In *Proc. Design, Automation and Test in Europe DATE '06*, volume 2, pages 1–6, 6–10 March 2006.

[2] Aeroflex Gaisler. *GRLIB IP Library User Manual*, 2008. `http://gaisler.com/products/grlib/grlib.pdf` - last visited 05.07.2009.

[3] V. Haridas, V. Ramchandra, K. Santhosh, and NXP Semiconductors. Automation in IP based SoC development: Case study of a media processorsubsystem. Technical report, NXP Semiconductors, 2007.

[4] IEEE. Advanced Library Format (ALF) describing integrated circuit (IC) technology, cells, and blocks. *IEC 62265-2005 First edition 2005-07 IEEE Std 1603*, (IEEE Std 1603):1–300, 2005.

[5] IPextreme. Coreuse, 2009. `http://www.ip-extreme.com/coreuse.html` - last visited 04.07.2009.

[6] C. M. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiss, and M. Pistauer. Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs. In *Proc. 2nd Annual IEEE Systems Conference*, pages 1–8, 7–10 April 2008.

[7] D. Mathaikutty and S. Shukla. Mining Metadata for Composability of IPs from SystemC IP Library. In *Proceedings of Forum on specification and Design Languages*, 2006.

[8] D. Mathaikutty and S. Shukla. SoC Design Space Exploration through Automated IP Selection from SystemC IP Library. In *Proc. IEEE International SOC Conference*, pages 109–110, 24–27 Sept. 2006.

[9] D. Mathaikutty and S. Shukla. Type Inference for IP Composition. In *Proc. 5th IEEE/ACM International Conference on Formal Methods andModels for Codesign MEMOCODE 2007*, pages 61–70, May 30–June 2 2007 2007.

[10] Dylan McGrath. Si2 forms Open Modeling Coalition. Technical report, EETimes, 2005. `http://www.eetimes.com/showArticle.jhtml?articleID=170703000` - last visited 04.07.2009.

[11] L. M. Reynari, F. Cucinotta, A. Serra, and L. Lavagno. A Hardware / Software Co-Design Flow and IP library based of Simulink. In *Proc. Design Automation Conference*, pages 593–598, 2001.

[12] L. Sarno, R. Wilson, S.-Kwan Eo, Laurent Lestringand, John Goodenough, Guri Stark, Serge Leef, and Dave Witt. IP Exchange: I'll Show You Mine if You Show Me Yours. In *Proc. 44th ACM/IEEE Design Automation Conference DAC '07*, pages 990–991, 4–8 June 2007.

[13] Si2. Open modelling architecture. Technical report, Si2, 2007. `http://www.si2.org/?page=863` - last visited 04.07.2009.

[14] SPIRIT-Consortium. *IP-XACT Version 1.5 Release*. SPIRIT Consortium, 2009.

[15] C. Trummer, C. M. Kirchsteiger, C. Steger, R. Weiss, A. Schuhai, M. Pistauer, and D. Dalton. A Component Selection Methodology for IP Reuse in the Design of Power-aware SoCs based on Requirements Similarity. In *Proc. 3rd Annual IEEE Systems Conference*, pages 133–138, 23–26 March 2009.

[16] M. Visarius, J. Lessmann, W. Hardt, F. Kelso, and W. Thronicke. An XML format based integration infrastructure for IP based design. In *Proc. 16th Symposium on Integrated Circuits and Systems Design SBCCI 2003*, pages 119–124, 8–11 Sept. 2003.

[17] Ralph Von Vignau. Time-to-market drives SoC Design to higher Levels of Abstraction. *Embedded Systems Europe*, November - December:14–16, 2006. `http://www.embedded.com/columns/technicalinsights/196600752?_requestid=166618` - last visited 05.07.2009.

[18] M. Wirthlin, D. Poznanovic, P. Sundararajan, A. Coppola, D. Pellerin, W. Najjar, R. Bruce, M. Babst, O. Pritchard, P. Palazzari, et al. OpenFPGA CoreLib core library interoperability effort. *Parallel Computing*, Volume 34:231–244, 2008.

Search for Extended IP-XACT Components in a Library for Power Aware SoC Design based on Requirements Similarity

Christoph Trummer, *Member, IEEE,* Christoph Ruggenthaler, Christoph M. Kirchsteiger, *Member, IEEE,* Christian Steger, *Member, IEEE,* Reinhold Weiß, *Member, IEEE,* Markus Pistauer and Damian Dalton

***Abstract*—To counter today's rising complexity in System-on-Chip (SoC) design Intellectual Property (IP) components are reused. In IP libraries many different pre-designed components are available. However, finding and selecting functionally suitable IP is difficult and laborious. Additionally, certain constraints need to be considered when searching for IP. One such constraint is power consumption which is highly important for most SoCs. However, additional information such as power dissipation and verification status are rarely considered in currently available IP representation formats. This paper introduces an extension to the IP-XACT format and describes our innovative approach for searching for suitable components. Our approach extends the IP-XACT format with relevant information, manages the IP in a library, and utilizes a novel selection process which takes into account similarities between system and component requirements. The result is a ranking of the best-suited components for reuse in the current system-under-design. To demonstrate our approach a case study is performed on a SoC. With our methodology a ranking of components matching the system's requirements and constraints is generated.**

***Index Terms*—Intellectual Property (IP), IP reuse, library, component selection, similarity analysis, requirements, power.**

I. Introduction

A System-on-Chip (SoC) is a system consisting of many different components integrated on a single chip [1]. System-on-Chips (SoCs) are typically used in portable and battery-powered consumer electronics. Today's customers demand a steady stream of new products with unique features and innovative functionality. Therefore, new SoCs are faced with a continuously rising number of requirements. Due to high silicon integration densities and the large number of requirements the design and verification of today's SoCs is increasingly complex [2]. Consequently, the industry faces three important issues. First, companies have to deal with strict time-to-market constraints. Second, the productivity of system engineers cannot keep up with the increasing complexity of SoC designs (design-productivity gap) [3] [4]. Third, static power dissipation increases with the reduction of SoC area [5]. These issues can be addressed by reusing IP, applying power aware methodologies and early verification in SoC design.

When reusing IP the main difficulty is finding and selecting components which are best suited for the current system-under-design. This is the so-called component selection problem [4]. System engineers need to select components which fulfill the system's requirements and satisfy its constraints. Available third-party or legacy IP may consist of different implementations and formats, and wide variation in the amount of available information. Manually finding components in this incoherent, and often large set of IP, needs considerable time and effort. Consequently, an automated approach to search and select the appropriate component is desirable.

To reduce the system's power dissipation, power-efficient components are chosen and power aware design techniques are applied. Therefore, information about the individual IP's power dissipation is needed early in the design flow.

In order to avoid costly redesign at a later stage, the system has to be verified as early as possible. Moreover, it has to be ensured that each re-used IP component is verified. A comprehensive report would clarify the IP's verification status and confirms the designer in his or her choice.

Typically, used IP representation formats mainly describe the IP's implementation and do not support additional information. Important details about verification status, power estimation results or power aware design, consequently remain vague. This often leads to time-consuming (re-)verification and new power estimation of the IP.

Our contribution is a methodology which comprises three parts. First, the IP is extended with additional information. Suitable parameters and formats for IP representation are investigated. Second, a library is created to accommodate the extended IP. Third, we propose an automated search for IP components from this library. Our approach is based on similarities between the IP's requirements and system's requirements. The result is a set of IP which is able to perform part of the system's functionality and matches its constraints. Fig. 1 shows the main idea behind our approach.

This paper is organized as follows. In section II background and related works are investigated. Section III explains our extended components and the IP library. The next section (IV), describes the component selection methodology using similarity analysis. Our methodology is demonstrated in a case study in section V. Finally, section VI summarizes our approach and outlines future work.

Manuscript received August 31st, 2009.

This work is funded by the Austrian Federal Ministry for Transport, Innovation and Technology under the contract FFG 812424.

Christoph Trummer, Christoph Ruggenthaler, Christoph M. Kirchsteiger, Christian Steger and Reinhold Weiß are with the Institute for Technical Informatics, Graz University of Technology, Austria e-mail: trummer@tugraz.at

Markus Pistauer is with CISC Semiconductor Design+Consulting GmbH, Austria.

Damian Dalton is with the School of Computer Science and Informatics, University College Dublin, Ireland.

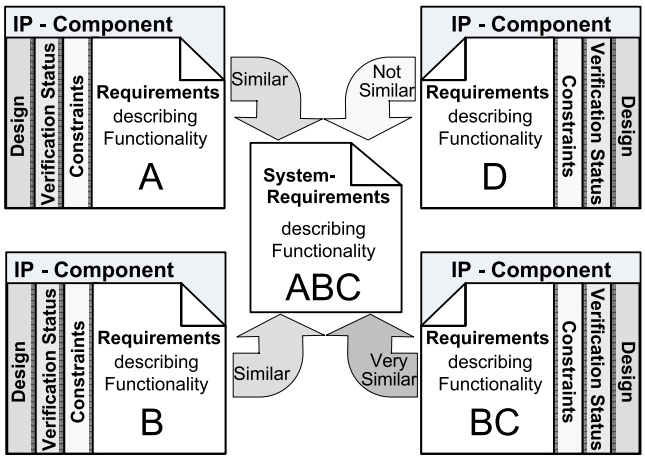

Fig. 1. Similarities between system and component requirements

II. Background and Related Work

A. Hardware Requirement Specification Formats

Requirements can be expressed in many different ways. They are often ambiguous, imprecise and misleading. Most commonly, requirements are specified informally, in natural text. Natural text is easy to read but is usually ambiguous and very difficult to process automatically. In contrast, formal requirements have a clearly defined structure and can be processed easily. However, understanding them is difficult without knowledge of the syntax [6]. Semi-formal requirements such as textual use cases are a trade-off between readability and automatic processing capabilities [6].

In the software domain the Unified Modeling Language (UML) is widely used to provide a common and concise way to express requirements. In [7] the authors present a methodology which uses the UML to specify requirements in hardware design. Non-functional requirements can also be included into the UML descriptions and diagrams [7].

Semi-formal use cases contain a predefined structure describing a series of steps and interactions expressed in natural language [8]. This structured, textual use case format is similar to UML described in [7]. Moreover, the semi-formal use case format can be used to express hardware requirements [6].

B. Component Selection

Since the component selection problem is extensively researched in both the soft- and hardware domain only a few representative approaches will be elaborated. The target of component selection is to find a subset of components matching the system's functionality and its constraints [4], [2].

Some of these methodologies for component selection use similarity analysis of the component's implementation and/or its interface definitions [9], [10]. Others rely solely on querying for keywords which describe functionality and constraints [2], [11]. A more sophisticated approach uses fuzzy logic for keyword and constraints matching to select components [4].

Few approaches exist which include requirements for selecting components. However, they either rely on a manually created matrix that links similar requirements [12] or on complex additional specifications [10].

From the related works it is apparent that the most important criteria for selecting suitable components is functionality [4]. However, additional parameters and information need to be considered for proper component selection.

C. Additional Parameters for IP Selection

Apart from system functionality, constraints are also considered when selecting IP for SoC design. Especially power and delay but also implementation details (e.g. architecture, technology) are the important constraints [11].

Information about the existence and layout of a power aware design is important to reduce the IP's power consumption. This aspect is covered in two different specification formats for power design. The Unified Power Format (UPF) was recently approved by the IEEE as IEEE 1801 Standard for Design and Verification of Low Power Integrated Circuits [5]. Alternatively there is the well-established industry standard, Common Power Format (CPF) [13]. Both formats allow description and verification of the power design to be expressed independently of the Hardware Description Language (HDL).

Knowledge about the IP's verification status or even Verification IP (VIP) are also desirable [3]. Information about the verification status allows the designer to select IP with the confidence that it was verified and tested. Added VIP allows tests to be re-run and eases re-verification if a bug is detected.

To avoid improper or erroneous use of IP a thorough documentation of design and implementation is necessary and also use cases and requirements are helpful.

Despite their importance, current IP formats do not facilitate addition of these parameters and informations to the IP.

D. Textual Similarity Analysis

Various textual similarity analysis techniques exist. A few exemplary methods will be mentioned as evaluation of all approaches is beyond the scope of this work.

A common technique uses word occurrence to determine text similarity [14]. Keyword occurrence in documents is analyzed and compared based on a vector space model and information retrieval methods [14]. However, success of the approach is sensitive to sentence format and wording.

A similar approach analyzes the documents on a per sentence basis [15]. Each word is represented by an integer hash-code for comparing sentences. Two equal sentences have the same integer sum. Similarity of sentences is detected by comparing the intersections of their sets of unique words [15]. By representing the words as integers the approach is able to perform a fast similarity analysis.

The approach described in [16] uses natural language processing techniques to detect similarities in documents using syntactic and semantic evaluation. Also, the frequency of word occurrences is taken into account [16].

E. IP Representation Formats

Today, many different IP representation formats exist [17]. However, available IP formats rarely consider additional resources (e.g. verification status, documentation). Most companies even use their own proprietary format. Any newly

acquired third-party IP is converted into the proprietary format [17]. This means over time a collection of legacy, third-party and in-house IP is formed. Searching in this heterogeneous set of IP with different information, varying details and mixed representations is very difficult and time-consuming.

The Advanced Library Format (ALF), IEEE Std. 1603-2003, specifies a format for IP technology, cells and blocks. ALF is designed for abstraction levels of Register-Transfer Level (RTL) and below. The ALF specification covers many facets of IP such as behavior, timing, power and signal integrity [18]. However, IP at higher levels of abstraction (e.g. System Level) can not be represented. Consequently, the ALF has become obsolete because today's complexity of SoC designs demands higher levels of abstraction.

The upcoming IEEE standard IP-XACT (IEEE P1685 Working Group) was founded by the SPIRIT Consortium in 2003. The SPIRIT IP-XACT format uses the extensible markup language (XML) and describes IP for development, implementation and verification [19]. In comparison to other formats, IP-XACT focuses on vendor-neutral IP descriptions. Therefore, it is tool-independent, flexible and can be extended with additional information [19], [20]. Through, so-called views different levels of abstraction can be represented within the same IP. Each view is distinguished by the type and desired design environment for editing [21].

III. IP Library and Component Design

The structure of our library to store and search for an IP component is presented. The following subsections describe the IP component, its architecture, linked information and representation in IP-XACT. Finally, the semi-formal use case format which is used in similarity analysis is elaborated.

A. IP Library

To find the IP component for reuse in the current design we first need to define a library where it can be archived. Apart from being able to store and manage the many different components it also needs to support categorization, classification and description of IP. Within this library searching is necessary to retrieve and select the desired IP.

Therefore, the IP library L is defined as a hierarchical set of categories K.

$$L\{K_1, K_2, K_3, ..., K_m\} \tag{1}$$

Each set of categories K may contain a hierarchy of further (sub)categories K' or components C.

$$K\left\{K'_i, K'_{i+1}, ..., K'_n, C_1, C_2, ..., C_j\right\} \tag{2}$$

To efficiently find and select suitable IP components each set L, K and C is defined as an unique entity.

A graphical user interface is used to communicate with the IP library and to facilitate searching (Fig. 2). The library is implemented as a web-service on a remote server which performs data exchange and IP management. A repository on the same server is used to store the components and avoid concurrent interactions. Since SOAP over HTTP is used for communication the library infrastructures is easily accessible.

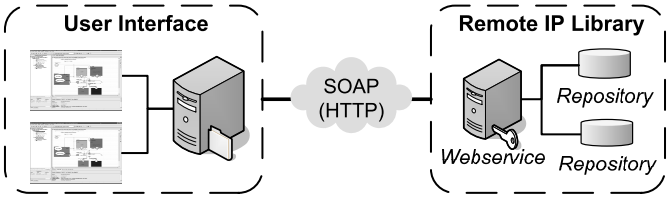

Fig. 2. Interface and Architecture of the IP Library

B. IP Components

Components need a representation that includes functional and non-functional information (e.g. constraints). Functionality is described in the IP component's source-code and its requirements document. Extracting functionality out of source code in different hardware description languages (HDLs) and abstraction levels needs complex semantic analysis. Furthermore, the functionality needs to be classified in order for it to be searchable. This demands a language-independent, additional description of the component's behavior. To reduce complexity and effort we suggest use of the already existing requirements document. However, a purely textual specification contains an arbitrary structure and varied detail. Comparing purely textual requirements may lead to insufficient similarities for significant IP selection (see [16]). Due to its structure and unified representation use case documents are employed for similarity analysis. As a trade-off between natural text and a completely formal structure we express requirements as semi-formal use cases.

Non-functional information is more complex to manage. It may not be available or incomplete since the IP component may be legacy IP, from a third-party or is still under development. This could also be the case for verification. Consequently, the influence of non-functional information on the search results must be carefully considered through weights.

IP is a collection of different information and source-code which managed by a library for archiving and later reuse. We define such a collection of IP and its related data as a component C. However, it is not assumed that each component C is complete since all information may not be available. Consequently, the component C can be defined as a set of documentation, source files, Verification IP (VIP) non-functional data and a description.

$$C\{Doc, Sources, VIP, NFData, Description\} \tag{3}$$

The documents part, Doc, is a collection of files such as the original requirements, application notes, data sheets and source-code documentation.

$Sources$ represents the actual IP core. It is a set of files containing the source-code in one or more hardware description languages (HDLs). Only this part would regularly be covered by the original IP-XACT standard.

Verification IP (VIP) similarly consists of testbenches, test cases and verification results. It also contains a symbolic representation of the IP for our graphical verification tool.

The Non-Functional Data, $NFData$, comprises the power aware design in a power format, results from simulation and power estimation. Moreover, a technology description specifies libraries for synthesis and additional resources for the power

estimation tool RHEiMS. Entries for $NFData$ are optional because some information or results may not be available.

RHEiMS from Neosera Systems Ltd. is used to determine a component's power and energy dissipation at system-level [22]. RHEiMS employs statistical methods and case-based reasoning to estimate power and energy. It compares similarity of input and output of the current component to a similar component with known energy dissipation and calculates power consumption [22]. Thereby, it rapidly estimates the component's power and energy dissipation. The result files are stored in the $NFData$ field of C.

The power design in either power format is stored in $NFData$ because it is specified independently from the HDL source. Also, the power design does not directly contribute to the system's main functionality and is therefore considered non-functional information.

The $Description$ specifies data such as architectural information, a brief abstract and relevant data for IP search.

From this data and the constraints from $NFData$, a summary is automatically generated (subsection III-C). Entries determining representative values for power, energy and timing are stored. The use case document is also included in $Description$. Within UC the component's requirements are stored as a set of semi-formal use cases (subsection III-E).

C. IP Summary

All IP and VIP are linked with related information, results and estimation files. Results from power estimation and timing simulation are stored in separate result files. The verification status of each use case is available in a file. In the file, information about reached coverage goals and testcases which were satisfied or failed is summarized. By automatically evaluating these files and IP parameters a comprehensive summary for each IP is generated. In this summary selected values from properties, constraints and verification status are stored. The summary is used in search for pre-selecting components in the IP library (see subsection IV-A). The summary also contains information about the IP's version and modification history.

D. IP-XACT Extension

Due to its flexibility the IP-XACT format is well-suited to represent our IP components. So far, it only allows description the IP itself. The additional information required to extend the IP-XACT is added a wrapper to the XML schema as illustrated in Fig. 3. This enables compliance with the standard and importation of existing IP-XACT components into our library.

Each field in C is represented as folder with sub-folders and files. Folders and files are relatively linked in the XML file. Parameters for each field are the resource type, properties for visibility and access or suggested tools for editing. Each IP is assigned a unique identifier instead of a name.

E. Use Case Format

We refine requirements into semi-formal use cases as described in [6]. The use cases specify step-wise interactions between a system (or component) and its environment. Thus, a component's functionality is entirely described by the set of all its use cases. The semi-formal use case document may contain both functional and non-functional requirements. They are structured into several sections which contain natural text.

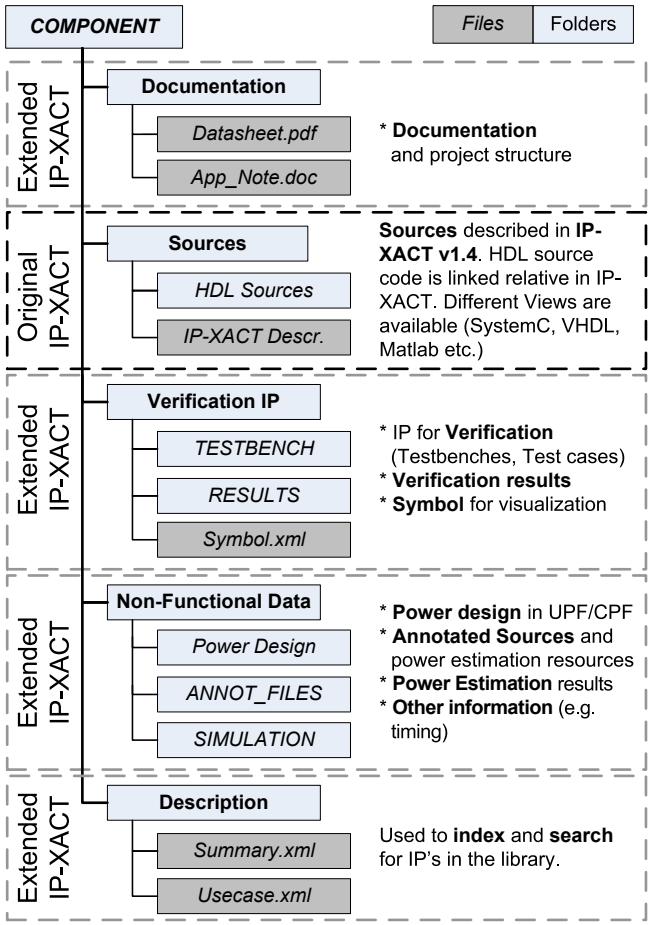

Fig. 3. Extended Component Structure

The most important sections of each use case include the use case name, a brief description, the actor(s), the primary scenario and alternative scenario(s) (see table I). Within the use case document, functional requirements can be expressed at system-level. Later they can be refined to a lower level of abstraction. Due to its structure, the semi-formal use cases can be processed easily which is ideal for searching (see also [6]). The semi-formal use case format also provides a unified representation of all system and component requirements. Our use cases are stored in XML which complements its structure.

IV. Novel Methodology for Component Selection

There are two phases in our component selection methodology. First, it filters all components based on keywords and non-functional information. Second, to use cases of the remaining components similarity analysis is applied. This determines the most suitable IP for the system. Fig. 4 illustrates the concept of our component selection methodology.

TABLE I
HIGH-LEVEL USE CASE - WRITING TO A SRAM

Name: WRITE
Description:
This use case describes a write operation to the static RAM.
Actor: Memory controller
Trigger:
1. The SRAM receives the WRITE_ENABLE_COMMAND.
Primary Scenario:
1. The SRAM receives the WRITE_COMMAND.
2. The SRAM receives the ADDRESS.
3. The SRAM receives the DATA.
4. The SRAM stores the DATA at the ADDRESS.
5. The SRAM sends the ACKNOWLEDGE.
6. The SRAM goes to the OUTPUT_DISABLE usecase.
Alternative Scenarios:
Alternative Scenario 1
1a_1. The SRAM receives the WRITE_DISABLE_COMMAND.
1a_2. The SRAM sends the ACKNOWLEDGE.
1a_3. The SRAM goes to the OUTPUT_DISABLE usecase.
Constants:
ADDRESS=@bit[7:0]
DATA=@bit[15:0]

A. Keyword Search and Constraints Filtering

Since exhaustive similarity analysis over all components is computationally intensive, it is desirable to reduce the search space. The major difficulty is the removal of unsuitable components from the search without inadvertently removing useful ones and this is accomplished by evaluating components based on keyword occurrence and constraints filtering. To avoid exhaustive search in all files and subfolders of C only the IP's summary file is used. The summary's XML structure provides fast access to the individual search criteria.

$$Criteria_{opt}\,\{keywords, architecture, constraints\} \quad (4)$$

$$f(Criteria) : L \rightarrow R \quad (5)$$

$$R\{(Criteria) \in C_i, ..., (Criteria) \in C_j\} \quad (6)$$

For each match in keywords, architecture and constraints a weight is applied to calculate a score. This compensates for IP with incomplete or unavailable data. Additionally, it is possible to change search policies between "strict" and "fuzzy". The first, strictly enforces the search criteria and only returns components that match them. The "fuzzy" policy, does not instantly dismiss an otherwise suitable component because of a barely mismatching constraint. Instead, the component receives a lower score by using different weights.

With either policy, after searching for keywords and filtering by constraints the score of each component is evaluated. The better the component matches the search criteria, the higher its score. Next similarity analysis is applied to the set R.

However, constraints or implementation details may not be known in early design stages. Therefore, each parameter and even the entire set of $Criteria$ is optional. Thus, to search for functionally suitable components without applying keywords or constraints this initial search may be omitted.

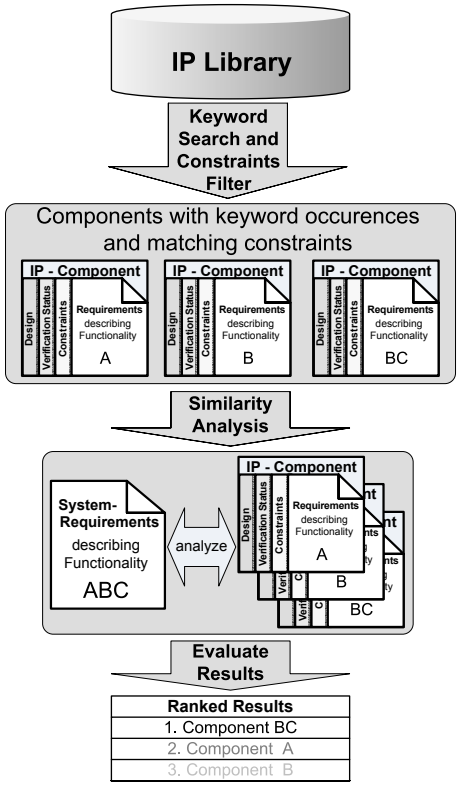

Fig. 4. Component selection methodology

B. Similarity Analysis

A suitable component for the system needs to be able perform some of its functionality. Consequently, functionality of system and component must overlap. The overlapping functionality can be performed by the respective component. The more the functionality of the component corresponds to the system's functionality the better it is suited for the system. In our case, functionality is described in the system's use cases and in the component's use cases. For this reason parts of the system's use cases and the component's use cases are similar.

We exploit this commonality by applying similarity analysis g on the use cases of each component in R. The function g compares all use cases of the system to each use case of the components in R. This produces a list of components with the highest similarity to the system's use cases. The algorithm used for similarity analysis compares entire sentences and is described in [15].

$$g(UC_{system}) : UC_{component} \in R \rightarrow R' \quad (7)$$

After determining the use cases with the highest similarity, a ranking is calculated for the components. The previous ranking of components R is merged with the new information from similarity analysis. The new set R' now contains the final ranked list of best suited components.

V. CASE STUDY

In this section we demonstrate our component search methodology on a case study. Our IP library was filled with

components typical for SoC design. The following subsections describe the system and two scenarios in which our methodology is applied.

A. System-under-Design

As an example for a SoC, a small system for temperature monitoring was taken as a case study. The system is capable of measuring ambient temperature and digitizing the temperature samples. For long term monitoring the system stores the samples into its memory. An active radio frequency identification (RFID) tag could be an application for such a system.

Initially, we analyzed the ISO/IEC 18000-7 protocol [23] for active tags. A series of use cases were specified for the wakeup, collection with universal data block (UDB), read, write and sleep commands. Use cases for the sampling, standby and idle state were also described. When the tag receives a command it performs the corresponding functionality. On a read command the tag sends the contents of the memory at the given address to the reader. The Write use case contains a description of the write operation to the tag's memory. With the write command, the sampling interval of the tag can also be set. When the RFID tag receives a sleep command it enters the standby use case. After an interval set by the RFID reader a timer interrupt is triggered. A sample is taken from the sensor and stored in the tag's memory. Every 2 seconds the tag awakes from standby mode to determine whether the RFID reader has sent a wakeup command. If no wakeup command was detected the tag it goes back to standby. Otherwise it enters idle mode. During idle mode it is ready to receive and process commands from the reader. When the tag receives a sleep command it enters standby mode.

A SoC typically contains a controller, memory, a clock, timer and an external interface. In our system there is also a temperature sensor, analog-digital converter (A/DC) and an RFID transceiver. Our IP library comprises different versions and implementations of the above SoC components. All components consist of different data, constraints, properties and use cases. Information on some components is left partially incomplete to represent legacy or third party IP. Some components are able to perform the same functionality. Not all components are suitable for the system under design.

To select components for the above system we consider two scenarios. In the first scenario we want to select suitable components based on their functionality. The search for suitable IP is conducted without constraining the search space. All use cases of the components in the repository are analyzed for their similarity to the system. The second scenario describes the selection of a specific component. In this case it is a suitable memory component. The search space is reduced by applying the filter to the components in the library. Then the use cases are analyzed to select the best suited memory component based on its functional similarity.

B. Scenario 1 - Functionality Search

In this example we perform a similarity analysis g over all component use cases. This necessitates the comparison of the system's use cases to all use cases of the components in the library.

TABLE II
FUNCTIONALITY SEARCH - RESULTS SIMILARITY ANALYSIS

Rank	Score	Component	Verification	Coverage
1.	68.5%	RFID Transceiver	PASS	63%
2.	65.5%	RFID Controller	PASS	86%
3.	61.5%	Transceiver HL	PASS	40%
4.	56.6%	Digital Temp. Sensor	PASS	67%
5.	55.1%	ROM HL	PASS	51%
6.	54.2%	FLASH Memory	FAIL	10%
7.	53.8%	Timer 10bit	PASS	74%
8.	52.6%	I2C Bus HL	FAIL	46%
9.	51.5%	RAM A	PASS	78%
10.	50.5%	DAC HL A	PASS	55%

Table II contains the ten components with the highest similarity (as %) to the system's use cases. The higher the similarity the more functionality of the IP corresponds to the functionality of the target system. However, the percentage does not reflect the degree of functionality that the IP is able to fulfill of the system. A "FAIL" in Verification means that at least one testcase has failed. The precentage in Coverage shows how extensively the component was tested.

The above list displays the ranking of components. Almost all of the components would be suitable for our system. The RFID Transceiver and RFID Controller receive the highest ranking as expected. These are components conforming to the ISO/IEC 180007 protocol. The transceiver communicates with the RFID reader while the controller processes the commands and coordinates temperature sampling. The Transceiver HL is a high-level, general-purpose implementation of a transceiver. For taking temperature samples the Digital Temperature Sensor is required. This sensor already has a built-in analog to digital converter. Components such as FLASH and RAM are used to store the sampled values. The ROM may contain program code, the tag's manufacturer ID and serial number. The timer enables periodic sampling. Depending on the implementation the I2C bus could be used as well. Only the digital to analog converter (D/AC) would be unsuitable for the system.

Although, similarity analysis of use cases is capable of selecting functionally suitable components it is advisable to narrow the search space, since the exhaustive search is computationally intensive and will increase with the number of use cases and library size. As an effect also false positives can be avoided better.

C. Scenario 2 - Specific Component Search

Our entire methodology was applied to illustrate selection of a specific component for the system-under-design. The component we search in our library is a memory for storing and retrieving the sensor data. Since our RFID tag will have a limited energy source (e.g. battery or energy harvesting device) we want to constrain the memory's energy dissipation.

The search $f('memory', 'energy < 40\mu J')$ returns IP which contains the "memory" keyword and have energy estimation results with less than $40\mu J$. Paired with the results from similarity analysis g the results are ranked. Table III lists the results for the search of a memory component.

TABLE III
RANKED RESULTS FOR STRICT COMPONENT SEARCH

Rank	Score	Component	Verification	Coverage
1.	55.9%	EEPROM	PASS	68%
2.	51.8%	RAM B	PASS	76%
3.	50.1%	RAM C	PASS	43%
4.	44.8%	RAM D HL	PASS	99%
5.	43.9%	ROM A HL	PASS	98%
6.	43.2%	ROM B	FAIL	30%

TABLE IV
RANKED RESULTS FOR FUZZY COMPONENT SEARCH

Rank	Score	Component	Verification	Coverage
1.	55.9%	EEPROM	PASS	68%
2.	53.0%	ROM HL	PASS	51%
3.	51.8%	RAM B	PASS	76%
4.	50.1%	RAM C	PASS	43%
5.	49.9%	FLASH Memory	FAIL	10%
6.	44.8%	RAM D HL	PASS	99%
7.	43.9%	ROM A HL	PASS	98%
8.	43.2%	ROM B	FAIL	30%

From the results the designer may choose the desired component. Rank, verification status and coverage should help in selecting the proper IP. On demand more details from each components are shown to the designer. Since we want a memory to store temperature samples the ROM components can be ignored. However, they match the search criteria and show similarities to the system's use cases and therefore received a score. Based on the verification status either the EEPROM or one of the listed RAMs would be a good choice. Of course it depends on level of abstraction, implementation and whether a coverage below 70% is acceptable.

To demonstrate the "fuzzy" search policy we performed another search with the same criteria. As a result (see table IV) components such as FLASH and ROM HL are not dismissed. Even though their constraints do not exactly match the search criteria they receive a rather high rank due to their similarity in terms of functionality.

The ranking should serve as a guideline for the system designer to support the selection of components. However, the designer has to make the final decision based on suitability and additional information (i.e. verification status, power, energy).

VI. CONCLUSION AND FUTURE WORK

In this paper we presented an approach to extend IP with important additional information. Therefore, crucial parameters for successful selection of IP were identified. With regards to these aspects an IP component and library were designed. Then a method was developed to search and select the best suited components for a given System-on-Chip design. To find IP in the library a novel functionality-based search was introduced. Our methodology exploits the fact that the component's functionality is described in its use cases and the use cases of the target system. Consequently, the use cases show similarities and the matching component can be determined through similarity analysis and constraints filtering.

Future work includes adding more sophisticated security features to our IP library. Moreover we plan to further enhance performance in both component matching and search speed.

REFERENCES

[1] C. Wenwei, Z. Jinyi, L. Jiao, R. Xiaojun, and L. Jiwei, "Study On a Mixed Verification Strategy for IP-Based SoC Design," in *High Density Microsystem Design and Packaging and Component Failure Analysis, 2005 Conference on*, June 2005, pp. 1–4.

[2] G. Hamza-Lup, A. Agarwal, R. Shankar, and C. Iskander, "Component selection strategies based on system requirements' dependencies on component attributes," in *IEEE Systems Conference, 2008 2nd Annual*. IEEE, 2008, pp. 1–5.

[3] W. Kruijtzer, P. van der Wolf, E. de Kock, J. Stuyt, W. Ecker, A. Mayer, S. Hustin, C. Amerijckx, S. de Paoli, and E. Vaumorin, "Industrial IP Integration Flows based on IP-XACT Standards," in *Proc. of the Design, Automation and Test in Europe*, 2008, pp. 32–37.

[4] T. Zhang, L. Benini, and G. De Micheli, "Component Selection and Matching for IP-Based Design," in *Proc. of the Design, Automation and Test in Europe (DATE'01)*. IEEE, March 2001, pp. 40–46.

[5] "IEEE Standard for Design and Verification of Low Power Integrated Circuits," *IEEE Std 1801-2009*, pp. C1–218, 2009.

[6] C. Kirchsteiger, J. Grinschgl, C. Trummer, C. Steger, R. Weiss, and M. Pistauer, "Automatic Test Generation From Semi-formal Specifications for Functional Verification of System-on-Chip Designs," in *IEEE Systems Conference, 2008 2nd Annual*. IEEE, April 2008, pp. 1–8.

[7] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.

[8] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, 2001.

[9] D. Mathaikutty and S. Shukla, "SoC Design Space Exploration through Automated IP Selection from SystemC IP Library," in *International SOC Conference, 2006 IEEE*. IEEE, September 2006, pp. 109–110.

[10] L. Wang and P. Krishnan, "A Framework for Checking Behavioral Compatibility for Component Selection," in *Proc. of the Australian Software Engineering Conference*. IEEE, 2006, pp. 49–60.

[11] L. Reynari, F. Cucinotta, A. Serra, and L. Lavagno, "A Hardware/Software Co-design Flow and IP Library Based of Simulink™," in *Proc. of the 38th Design Automation Conference*. IEEE and ACM, 2001, pp. 593–598.

[12] M. Martinez and A. Toval, "COTSRE: A COmponenTs Selection Method Based on Requirements Engineering," in *Composition-Based Software Systems 2008, Seventh International Conference on*. IEEE, 2008, pp. 220–223.

[13] The Power Forward Initiative (PFI), "A Practical Guide to Low-Power Design - User Experience with CPF," pp. 1–281, 2008.

[14] The Apache Software Foundation, "Apache lucene," http://lucene.apache.org/java/2_3_0/scoring.html, 2006, last visited - 02/02/2009.

[15] C. Collberg, S. Kobourov, J. Louie, and T. Slattery, "SPlaT: A System for Self-Plagiarism Detection," in *Proc. of IADIS International Conference WWW/INTERNET 2003*, 2003, pp. 508–514.

[16] K. Indukuri, A. Ambekar, and A. Sureka, "Similarity analysis of patent claims using natural language processing techniques," *International Conference on Computational Intelligence and Multimedia Applications, 2007*, vol. 4, pp. 169–175, Dec. 2007.

[17] M. Visarius, J. Lessmann, W. Hardt, F. Kelso, and W. Thronicke, "An xml format based integration infrastructure for ip based design," *Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings. 16th Symposium on*, pp. 119–124, Sept. 2003.

[18] IEEE, "Advanced Library Format (ALF) describing integrated circuit (IC) technology, cells, and blocks," IEEE, pp. 1–300, 2005, iEC 62265-2005 First edition 2005-07 IEEE Std 1603.

[19] SPIRIT Schema and ESL Working Group Membership, "IP-XACT Draft/D5: A specification for XML meta-data and tool interfaces," The SPIRIT Consortium, May 2009.

[20] M. Strik, A. Gonier, and P. Williams, "Subsystem Exchange in a Concurrent Design Process Environment," in *Proc. of the Design, Automation and Test in Europe*, 2008, pp. 953–958.

[21] V. Berman, S. Fazzari, C. Ussery, M. Indovina, M. Strik, J. Wilson, O. Florent, F. Rmond, and B. P., "Industrially Proving the SPIRIT Consortium Specifications for Design Chain Integration," in *Proc. of the Design, Automation and Test in Europe*, 2006, pp. 1–6.

[22] Neosera Systems Ltd., "RHEiMS: Rapid Hierarchical Energy Investigation Modelling System," http://www.neosera.com, 2009, 10/7/2009.

[23] International Standardization Organization, "ISO/IEC 18000-7:2008 - Information technology – Radio frequency identification for item management – Part 7: Parameters for active air interface communications at 433 MHz," 2008.

Christoph Trummer received his MSc. degree with distinction in Telematics (Information and Communications Technology) from Graz University of Technology in 2007. Since 2007 he is with the Institute for Technical Informatics at Graz University of Technology where he is working towards his Ph.D in Electrical Engineering. He is IEEE member and has published over ten scientific papers as author and co-author. His main research interests include simulation-based verification, IP exchange, power-aware design and non-functional requirements.

Christoph Ruggenthaler received his BSc. degree in Telematics (Information and Communications Technology) from University of Technology, Graz, Austria in 2007. He is currently working towards his MSc. degree at the Institute for Technical Informatics. His Master's thesis is focused on IP library architectures. This research topic comprises IP representation formats, efficient component selection based on linked verification information and unified IP exchange.

Christoph M. Kirchsteiger received his Bachelor's and Master's degree with distinction in Telematics (Information and Communications Technology) from Graz University of Technology, Austria, in 2006. Since 2006 he has been a PhD student in Electrical Engineering at the Institute for Technical Informatics, Graz University of Technology, Austria. His research interests include design and verification of HW/SW codesigns, especially system-on-chips (SoC). His Ph.D is done in tight cooperation with CISC Semiconductor Design+Consulting GmbH.

Ass.-Prof. Dr.techn. Christian Steger received 1990 the Dipl.-Ing. degree and 1995 the Dr.techn. degree in Electrical Engineering, Graz University of Technology, Austria. Graduated from Export, International Management and Marketing course in June 1993 at Karl-Franzens-University of Graz. From 1990 to 1991 Research Engineer and since 1992 Assistent Professor at the Institute for Technical Informatics, Graz University of Technology. In 2002 he was a visiting researcher at the Department of Computer Science at the University College Dublin (Ireland). He heads the HW/SW codesign group (8 PhD students) at the Institute for Technical Informatics. His research interests include embedded systems, HW/SW codesign, HW/SW coverfication, SOC, power awareness, smart cards, UHF RFID systems, multi-DSPs. He is currently working with industrial partners on heterogeneous system design tools for system verification and power estimation/optimization for RFID systems, smart cards and wireless sensor networks. Christian Steger has supervised and co-supervised over 73 master thesis and co-supervised 8 PhD students, and published more than 70 scientific papers as author and co-author. He is member of the IEEE and member of the ÖVE (Austrian Electrotechnical Association). He was in the organizing committee of the Telecommunications and Mobile Computing Conference 2001, 2003, and 2005.

O.Univ.-Prof. Dr.techn. Reinhold Weiß is Professor of Electrical Engineering (Technical Informatics) and head of the Institute for Technical Informatics at Graz University of Technology, Austria. He received the Dipl.-Ing. degree, the Dr.-Ing. degree (both in Electrical Engineering) and the Dr.-Ing.habil. degree (in Realtime Systems) from the Technical University of Munich in 1968, 1972 and 1979, respectively. In 1981 he was as a Visiting Scientist with IBM Research Laboratories in San Jose, California. From 1982 to 1986 he was Professor of Computer Engineering at the University of Paderborn (Germany). He is author and co-author of about 170 scientific and technical publications in Computer Engineering. For e&i (Elektrotechnik & Informationstechnik, Springer-Verlag) he served several times as a guest editor for special issues on Technical Informatics and Mobile Computing, respectively. In 2001 and 2003 he organized two Workshops on Wearable Computing. His research interests focus on Embedded Distributed Real-Time Architectures (parallel systems, distributed fault-tolerant systems, wearable and pervasive computing). He is a member of the International Editorial Board of the US-journal "Computers and Applications" (ISCA). Further, he is a member of IEEE, ACM, GI (Gesellschaft für Informatik, Germany), and ÖVE (Austrian Electrotechnical Association).

Dr. Markus Pistauer. Degree in Electrical and Electronic Engineering. 1995 Ph.D. degree in Electronic and Control Engineering, Graz University of Technology, Austria. 1988 until 1990 Software Engineer at Siemens AG, Frankfurt. From 1989 until 1991 Software Consultant at SPC Computer Training Ges.m.b.H., Vienna. 1991 Research Engineer at the Department for Sensoric, Joanneum Research Forschungsgesellschaft m.b.H., Graz. 1991 to 1995 Assistent Professor at the Department of Electronics, Graz University of Technology. From 1995 to 1999 Research and Design Application Engineer at Siemens Design Center for Microelectronics in Villach. From 1995 to 1998 Senior Lecturer for Electronic Engineering and Computer Science at University of Applied Sciences Carinthia. Since 1995 leading international and national research projects in the area of CAD/CAE methods for integrated circuit design. Since 1997 judicial approved assessor for electronics and software. Since 1999 heading consultancy office for IT and computer science. 1999 foundation of CISC Semiconductor Design+Consulting GmbH. Since 1999 CEO of CISC Semiconductor. Member IEEE, member of the international program committee for conferences like "Informationstagung für Mikroelektronik", "IASTED International Conference on Modelling, Identification and Control", "International Conference on Computer Systems and Applications", author and co-author of more than 40 publications.

Dr. Damian Dalton Ph.D degree 2000 in Computer Science, University College Dublin. From 1988-2005, he was lecturer in the School of Computer Science and Informatics, UCD and project leader on several European and national research projects and Director of UCD/Xilinx FPGA Design laboratory. Since 2005 he is Senior Lecturer in the same school. His research interests are Advanced Computer Architectures and Parallel Processing, especially in the areas of their application to the hardware acceleration of EDA/ESL simulation and power estimation design tools. He was the designer of the APPLES for gate-level simulation and one of the main architects of the RHEiMS for system-level power analysis and optimisation, and holds 5 international patents for these and related technologies. His research group has produced 8 Ph.D's and over 25 national and international peer reviewed papers from this activity. He is CEO of a campus spin-out company Neosera Systems Ltd, Dublin which he founded in 2002 and which has commercially exploited the output of his University research group. The Company won the All-Island Seedcorn Best Campus Company Start-up in 2004. The Company now has a extensive portfolio of design tools offering solutions to the challenges in System-level simulation and Power Analysis, such as rapid and accurate power estimation of RFID, embedded systems and other portable and ubiquitous computing platforms such as mobile phones and PDA's.

He is also an extern lecturer delivering the Entrepreneurship course on the international computer degree at Fudan University, Shanghai, China, and an extern examiner to several national and international institutes and universities.

Bibliography

[1] C. M. Kirchsteiger, *A Simulation-Based Methodology for Requirements Verification of System-on-Chip Designs.* PhD thesis, Institute for Technical Informatics, Graz University of Technology, 2009.

[2] ChipEstimate.com, "ChipEstimate.com Chip Planning & IP Portal." http://www.chipestimate.com/, 2009. last visited: 10.12.2009.

[3] H. Jian and S. Xubang, "The Design Methodology and Practice of Low Power SoC," in *Embedded Software and Systems Symposia, 2008. ICESS Symposia'08. International Conference on*, pp. 185–190, IEEE Computer Society Washington, DC, USA, 2008.

[4] R. Lissel and J. Gerlach, "Introducing new verification methods into a company's design flow: an industrial user's point of view," in *Design, Automation & Test in Europe, Conference & Exhibition, 2007. DATE '07*, pp. 689–694, IEEE, April 2007.

[5] IEEE, *IEEE Guide for Developing System Requirements Specifications - IEEE Std 1233a-1998.* New York: The Institute of Electrical and Electronics Engineers, Inc., 1998.

[6] A. Varma, *High-Speed Performance, Power and Thermal Co-simulation For SoC Design.* PhD thesis, University of Maryland, College Park, Md., United States, May 2007.

[7] J. Taneja, J. Jeong, and D. Culler, "Design, modeling, and capacity planning for micro-solar power sensor networks," *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pp. 407–418, April 2008.

[8] D. Sunwoo, H. Al-Sukhni, J. Holt, and D. Chiou, "Early Models for System-Level Power Estimation," *Microprocessor Test and Verification, 2007. MTV '07. Eighth International Workshop on*, pp. 8–14, Dec. 2007.

[9] S. Mikami, T. Matsuno, M. Miyama, M. Yoshimoto, and H. Ono, "A Wireless-Interface SoC Powered by Energy Harvesting for Short-range Data Communication," in *Asian Solid-State Circuits Conference, 2005*, pp. 241–244, 2005.

[10] A. Crone and G. Chidolue, "Functional Verification of Low Power Designs at RTL," *Lecture Notes in Computer Science*, vol. 4644, pp. 288–299, 2007.

[11] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual For System-on-Chip Design.* New York: Springer Science+Business Media, 2008.

[12] B. Waldo, D. Stringfellow, J. Pedicone, and G. Maben, "Power Hungry? Strategies to Trim Your Chip's Appetite," tech. rep., Synopsys, inc., 2007.

[13] O. Unsal and I. Koren, "System-Level Power-Aware Design Techniques in Real-Time Systems," *Proceedings of the IEEE*, vol. Vol. 91, No. 7, pp. 1–15, 2003.

[14] B. Wile, J. Goss, and W. Roesner, *Comprehensive Functional Verification the Complete Industry Cycle*. Elsevier/Morgan Kaufmann, 2005.

[15] I. C. Society, *IEEE Standard for Design and Verification of Low Power Integrated Circuits - IEEE Std 1801-2009*. New York: The Institute of Electrical and Electronics Engineers, Inc., 2005.

[16] B. Kapoor, J. Edwards, S. Hemmady, S. Verma, and K. Roy, "Tutorial: SoC Power Management Verification and Testing Issues," in *Proceedings of the 2008 Ninth International Workshop on Microprocessor Test and Verification*, pp. 67–72, IEEE Computer Society, 2008.

[17] W. Nebel, "System-Level Power Optimization," in *Digital System Design, 2004. DSD 2004. Euromicro Symposium on*, pp. 27–34, IEEE, 2004.

[18] R. Pinto, "Power Management Verification-An Evolving Discipline," in *Proceedings of the 2008 Ninth International Workshop on Microprocessor Test and Verification*, pp. 57–60, IEEE Computer Society, 2008.

[19] A. Hunter, A. Piziali, A. Ziv, K. Larson, and S. Hemmady, "Ensuring Functional Closure of a Multi-core SoC through Verification Planning, Implementation and Execution," in *Proceedings of the 2008 Ninth International Workshop on Microprocessor Test and Verification*, pp. 7–13, IEEE Computer Society, 2008.

[20] P. Thaker, "Holistic verification: myth or magic bullet?," in *Proceedings of the 46th Annual Design Automation Conference*, pp. 204–208, ACM/IEEE, 2009.

[21] M. Fukui, S. Iwakoshi, and T. Koyagi, "An Algorithm for Battery Modeling and Life Time Maximization of Small Size Electric Systems," in *Circuit Theory and Design, 2007. ECCTD 2007. 18th European Conference on*, pp. 759–762, IEEE, 2007.

[22] F. Simjee and P. Chou, "Accurate battery lifetime estimation using high-frequency power profile emulation," in *Low Power Electronics and Design, 2005. ISLPED'05. Proceedings of the 2005 International Symposium on*, pp. 307–310, IEEE, 2005.

[23] R. Rao and S. Vrudhula, "Battery optimization vs energy optimization: which to choose and when?," in *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, pp. 438–444, IEEE Computer Society, 2005.

[24] H. Wenzl, I. Baring-Gould, R. Kaiser, B. Liaw, P. Lundsager, J. Manwell, A. Ruddell, and V. Svoboda, "Life prediction of batteries for selecting the technically most suitable and cost effective battery," *Journal of Power Sources*, vol. 144, no. 2, pp. 373–384, 2005.

[25] L. Sarno, R. Wilson, S. Eo, L. Lestringand, J. Goodenough, G. Stark, S. Leef, D. Witt, and C. ARM, "IP Exchange: I'll Show You Mine if You Show Me Yours," in *Design Automation Conference, 2007. DAC'07. 44th ACM/IEEE*, pp. 990–991, ACM/IEEE, 2007.

[26] SPIRIT Schema and ESL Working Group Membership, "IP-XACT Draft/D5: A specification for XML meta-data and tool interfaces." The SPIRIT Consortium, May 2009.

[27] G. Hamza-Lup, A. Agarwal, R. Shankar, and C. Iskander, "Component selection strategies based on system requirements' dependencies on component attributes," in *IEEE Systems Conference, 2008 2nd Annual*, pp. 1–5, IEEE, 2008.

[28] Kirchsteiger, C.M. and Trummer, C., "The SIMBA approach for Systems-on-Chip Verification - Technical Report." Institute for Technical Informatics, Graz University of Technology, 2009.

[29] S. Kajtazovic, *Design Methodology and Framework for Verification of Heterogeneous Embedded Systems*. PhD thesis, Institute for Technical Informatics, Graz University of Technology, 2006.

[30] CISC, "CISC Semiconductor Design+Consulting GmbH." http://www.cisc.at/, 2009. last visited: 02.12.2009.

[31] Dalton, McCarthy, Quigley, and Leeney, "A system-level power evaluation method," 2008. Irish Patent PCT 31498IESP.

[32] Neosera Systems Ltd., "RHEiMS: Rapid Hierarchical Energy Investigation Modelling System." http://www.neosera.com, 2009. last visited: 04.12.2009.

[33] M. Glinz, "On Non-Functional Requirements," in *15th IEEE International Requirements Engineering Conference*, pp. 21–26, IEEE Computer Society, 2007.

[34] S. Churiwala, G. S. Shindaghatta, W. Jiang, and A. K. Pua, "Verification and Generation of Constraints," *Design & Reuse*, 2009.

[35] A. Lachenmann, K. Herrmann, K. Rothermel, and P. Marrón, "On meeting lifetime goals and providing constant application quality," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 4, pp. 36:1–36:17, 2009.

[36] IDENTEC SOLUTIONS. http://www.identecsolutions.com/, February 2009. last accessed - 28/11/2009.

[37] T. Yu, T. Yoneda, D. Zhao, and H. Fujiwara, "Using Domain Partitioning in Wrapper Design for IP Cores Under Power Constraints," in *Proceedings of the 25th IEEE VLSI Test Symmposium*, pp. 369–374, IEEE Computer Society Washington, DC, USA, 2007.

[38] J. Decker, N. Khan, and R. Goering, "Power-Aware Verification Spans IC Design - A Plan-to-Closure Approach Helps Ensure Silicon Success." Cadence Design Systems Inc., http://www.cadence.com/, June 2009.

[39] T. Bahill and J. Daniels, "Using Object-Oriented and UML Tools for Hardware Design: A Case Study," *Systems Engineering*, vol. Vol. 6, pp. 28–48, 2002.

[40] M. Chen and G. Rincon-Mora, "Accurate electrical battery model capable of predicting runtime and I–V performance," *IEEE Trans. Energy Conversion*, vol. 21, no. 2, pp. 504–511, 2006.

[41] J. Rahmé and K. Al Agha, "A state-based battery model for nodes' lifetime estimation in wireless sensor networks," in *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, pp. 337–338, ACM New York, NY, USA, 2009.

[42] Atrenta Inc., "SpyGlass®-Power - Design for Low Power at RTL - Datasheet." http://www.atrenta.com/, 2008.

[43] Cadence Design Systems Inc., "Cadence Incisive III Palladium Dynamic Power Analysis - Datasheet." http://www.cadence.com/, 2008.

[44] T. P. F. Initiative, "A Practical Guide to Low-Power Design - User Experience with CPF." http://www.powerforward.org/, 2008.

[45] F. Bembaron, S. Kakkar, R. Mukherjee, and A. Srivastava, "Low Power Verification Methodology Using UPF," in *Conference on Electronic Systems Design and Verification Solutions, DVCON'09*, pp. 228–233, 2009.

[46] Mentor Graphics Corporation, "Questa - Datasheet." http://www.mentor.com/, 2009.

[47] M. Visarius, J. Lessmann, W. Hardt, F. Kelso, and W. Thronicke, "An XML format based integration infrastructure for IP based design," in *Proceedings of the 16th symposium on Integrated circuits and systems design*, pp. 119–124, IEEE Computer Society, 2003.

[48] Global Semiconductor Alliance, "Global Semiconductor Alliance." http://www.gsaglobal.org/, 2009. last visited: 10.12.2009.

[49] Kobylecky, J. et al., " QIP Quality Metric - User Guide 4.0." Virtual Socket Interface Alliance, 2007.

[50] Design and Reuse, "Design and Reuse - Catalyst of Collaborative IP Based SoC Design." http://www.design-reuse.com, 2009. last visited: 10.12.2009.

[51] M. Wirthlin, D. Poznanovic, P. Sundararajan, A. Coppola, D. Pellerin, W. Najjar, R. Bruce, M. Babst, O. Pritchard, P. Palazzari, *et al.*, "OpenFPGA CoreLib core library interoperability effort," *Parallel Computing*, vol. 34, no. 4-5, pp. 231–244, 2008.

[52] Synopsys, Inc., "Synopsys coreTools - IP Based Design and Verification." http://www.synopsys.com/, 2008.

[53] MAGILLEM The Ontology Company S.A., "Magillem IP-XACT Packager." http://www.magillem.com/, 2009.

[54] MAGILLEM The Ontology Company S.A., "Magillem Platform Assembly." http://www.magillem.com/, 2009.

[55] T. Arpinen, T. Koskinen, E. Salminen, T. Hämäläinen, and M. Hännikäinen, "Evaluating UML2 Modeling of IP-XACT Objects for Automatic MP-SoC Integration onto FPGA," in *Design, Automation & Test in Europe, Conference 2009. DATE '09*, IEEE, 2009.

[56] Accellera Organization, "Verification Intellectual Property (VIP) Best Practices Interoperability Guide." http://www.accellera.org, 2009.

[57] D. A. Matthaikutty and S. S. K., "Mining metadata for composability of IPs from SystemC IP library," *Design Automation Embedded Systems*, vol. 12, pp. 63–94, 2008.

[58] Open SystemC Initiative (OSCI), "SystemC." http://www.systemc.org/, 2009. last visited: 04.12.2009.

[59] EPCglobal Inc., "EPC^{TM} Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz - 960 MHz Version 1.1.0," 2006.

[60] International Standardization Organization, "ISO/IEC 18000-7:2008 - Information technology – Radio frequency identification for item management – Part 7: Parameters for active air interface communications at 433 MHz," 2008.

[61] Tadiran, "MODEL TL-5920 - Datasheet." http://www.tadiranbat.com/. last visited: 18.12.2009.

[62] Texas Instruments Incorporated, "High-Accuracy, Low-Power, Digital Temperature Sensor With SMBus^{TM}/Two-Wire Serial Interface in SOT563 - Datasheet." http://www.ti.com/. last visited: 16.12.2009.

[63] Atmel Corporation, "Two-wire Automotive Serial EEPROMs AT24C128/256 - Datasheet." http://www.atmel.com/. last visited: 16.12.2009.

[64] Tadiran, "http://www.tadiranbat.com/." last visited: 18.12.2009.

[65] Tadiran, "MODEL TL-5903 - Datasheet." http://www.tadiranbat.com/. last visited: 18.12.2009.