



HAL
open science

An electronic system level modeling approach for the design and verification of low-power systems-on chip

Ons Mbarek

► **To cite this version:**

Ons Mbarek. An electronic system level modeling approach for the design and verification of low-power systems-on chip. Other [cs.OH]. Université Nice Sophia Antipolis, 2013. English. NNT : 2013NICE4023 . tel-00837662v2

HAL Id: tel-00837662

<https://theses.hal.science/tel-00837662v2>

Submitted on 18 Nov 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE DE NICE-SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

T H E S E

pour l'obtention du grade de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention Informatique

présentée et soutenue publiquement par

Ons MBAREK

**Une Approche de Modélisation au Niveau Système
pour la Conception et la Vérification de Systèmes
sur Puce à Faible Consommation**

An Electronic System Level Modeling Approach for the Design and Verification of
Low-Power Systems-on-Chip

Thèse dirigée par **Michel AUGUIN**

Laboratoire LEAT, Université de Nice-Sophia Antipolis –CNRS, Sophia Antipolis

soutenue le 29/05/2013

Jury :

M.	Robert DE SIMONE	D.R.	Président
M.	Frédéric ROUSSEAU	Pr.	Rapporteur
M.	Jean-Didier LEGAT	Pr.	Rapporteur
M.	Michel AUGUIN	D.R.	Directeur de Thèse
Mme.	Florence MARANINCHI	Pr.	Examinatrice
M.	Alain PEGATOQUET	M.C.	Examineur

Abstract : a SoC power management solution can be defined by a low-power architecture composed of multiple power domains and a power management strategy for power domains states control. If these two elements are energy-efficient, an energy-efficient solution can be obtained. This approach requires inferring power structural elements and their related behavior in the chip internal logic. A strategy adjusting the power domains states must respect structural and functional dependencies due to the physical power domains composition. This strong relationship between power architecture and its management strategy must be explored at early design stages to find the most energy-efficient solution. Low-power design standards have recently enabled low-power architecture exploration starting from the Register Transfer Level (RTL) by defining semantics to specify power architecture, simulate and check its behavior along with the initial functional one. But, these standards miss semantics for reusable power domain control interface making power management strategies exploration tedious. The RTL-based semantics defined by these standards constrain also their use at Transaction-Level of Modeling (TLM) for fast and easy exploration.

This dissertation proposes extensions to low-power standards to fill these gaps. It provides a complete study of power optimization opportunities based on composition and management of power domains in Transaction-Level (TL) functional models within a common USLPAF framework. USLPAF includes a methodology that combines design and verification of TL low-power models. To apply this methodology, USLPAF incorporates a library of modeling techniques and built-in features.

Keywords: Systems-on-Chip, TLM, Low-Power Design and Verification, Low-Power Design Standards, Power Domains, Energy-Efficient Power Management Solution, Semantics.

Résumé : une solution de gestion de puissance d'un système sur puce peut être définie par une architecture de faible puissance composée de multiples domaines d'alimentation et de leur stratégie de gestion. Si ces deux éléments sont économes en énergie, une solution efficace en énergie peut être obtenue. Cette approche nécessite l'ajout d'éléments structurels de puissance et de leurs comportements. Une stratégie de gestion doit respecter les dépendances structurelles et fonctionnelles dues au placement physique des domaines d'alimentation. Cette relation forte entre l'architecture et sa stratégie de gestion doit être analysée tôt dans le flot de conception pour trouver la solution de gestion de puissance la plus efficace. De récentes normes de conception basse consommation définissent des sémantiques pour la spécification, simulation et vérification d'architecture de faible puissance au niveau transfert de registres (RTL). Mais elles manquent une sémantique d'interface de gestion des domaines d'alimentation réutilisable ce qui alourdit l'exploration. Leurs sémantiques RTL ne sont pas aussi utilisables au niveau transactionnel pour une exploration plus rapide et facile.

Pour combler ces lacunes, cette thèse étend ces normes et fournit une étude complète des possibilités d'optimisation de puissance basées sur la composition et la gestion des domaines d'alimentation pour des modèles fonctionnels transactionnels utilisant un environnement commun USLPAF. USLPAF comprend une méthodologie alliant conception et vérification des modèles transactionnels de faible consommation, ainsi qu'une bibliothèque de techniques de modélisation et fonctions prédéfinies pour appliquer cette méthodologie.

Mots Clés: Systèmes sur Puce, Niveau Transactionnel, Conception et Vérification de Faible Consommation, Normes de Conception Basse Consommation, Domaines d'Alimentation, Solution de Gestion d'Énergie Efficace en Énergie, Sémantique.

Contents

Table of Contents	viii
Acknowledgments	ix
List of Figures	xvi
List of Tables	xvii
Acronyms & Abbreviations	xviii
1' Introduction (In French)	1
1'.1 Problématique	1
1'.1.1 L'optimisation de la consommation d'énergie dans un système sur puce	1
1'.1.2 L'abstraction faible consommation	5
1'.2 Thèse	8
1'.3 Aperçu de la thèse	10
1'.3.1 Contributions	10
1'.3.2 Sommaire	13
1 Introduction	15
1.1 Problem Statement	15

1.1.1	Power Optimization in Systems-on-Chip	15
1.1.2	Low Power Abstraction	18
1.2	Thesis	21
1.3	Overview of Thesis	23
1.3.1	Contributions	23
1.3.2	Outline	26
2	High Level Modeling of Low Power Systems-on-Chip Design: Back- ground & State of Art	28
2.1	Background	29
2.1.1	System-on-Chip Design Flow	29
2.1.1.1	Algorithmic Level (AL)	29
2.1.1.2	Transaction Level of Modeling (TLM)	30
2.1.1.3	Cycle Accurate Level (CAL)	33
2.1.1.4	Register Transfer Level (RTL)	33
2.1.1.5	Gate Level (GL)	33
2.1.1.6	Layout	34
2.1.2	Model Driven Engineering (MDE): Basic Concepts	34
2.1.3	Transaction-Level of Modeling Key Concepts	36
2.1.3.1	TLM Common Concepts	36
2.1.3.2	TLM With SystemC	38
2.1.4	The TLM 2.0 OSCI Standard	41
2.1.4.1	The TLM 2.0 Modeling Features and Mechanisms	41
2.1.4.2	The TLM 2.0 Coding Styles	45
2.1.4.3	The TLM 2.0 Extension Mechanisms	47
2.1.5	Power reduction in Systems-on-Chip	49

2.1.5.1	Dynamic and Static Power	50
2.1.5.2	Low Power Design Techniques	51
2.1.5.3	Power Management Levels	57
2.1.6	Low Power Design Standards	67
2.1.6.1	The Unified Power Format	69
2.1.6.2	UPF Versus CPF: Similarities, Differences and Common Gaps	72
2.2	State of Art	74
2.2.1	State-of-The-Art on High Level Power Modeling, Reduction and Analysis	74
2.2.1.1	Functional/Algorithmic Level	75
2.2.1.2	Cycle-Accurate Level	78
2.2.1.3	Transaction-Level	79
2.2.1.4	Using Model Driven Engineering Approaches	83
2.2.2	State-of-The-Art on Low Power Design Standards Use	85
3	Overview of the USLPAF Framework	88
3.1	The Need for the USLPAF Framework	88
3.1.1	Capturing Power Intent at Transaction-Level	88
3.1.1.1	What if the low power flow is extended to TLM?	88
3.1.1.2	What if a power domain-based reasoning is applied?	93
3.1.2	Power-Aware Modeling Issues at Transaction-Level	99
3.1.2.1	The accuracy problem	99
3.1.2.2	The power/latency trade-off problem	102
3.1.2.3	The synchronization problem	106
3.2	The USLPAF Structure and Features	113

4 USLPAM: A Unified Methodology for System-Level Power-Aware Modeling and Verification	116
4.1 An Overview of the USLPAM Flow	118
4.1.1 The Software Flow Analysis Stage	118
4.1.2 The Power Management Points (PMPs) Identification Stage	119
4.1.3 The Power Intent Specification Stage	122
4.1.3.1 The Main Abstracted UPF Concepts at Transaction-Level	123
4.1.3.2 Inferring the Abstracted UPF Concepts Behavior to TLM	125
4.1.3.3 Power Estimation Models	129
4.1.4 The PMU Modeling Stage	133
4.1.4.1 The Scenario-Based Power Management Strategy	137
4.1.4.2 The Reactive Power Management Strategy	142
4.1.4.3 The Scenario-Tracking Power Management Strategy	146
4.1.5 The Full Power-Aware Simulation Stage	149
4.1.6 The Power-Aware and Simulation-Based Verification Stage	149
4.2 The USLPAM Requirements	154
5 PMPs Specification and Simulation-Based Power-Aware Verification	158
5.1 Identification of Power Management Points Candidates	158
5.1.1 Methodology for PMPs Specification	158
5.1.2 Power-Aware State Modeling of Black-Box and White-Box IPs	165
5.1.2.1 Description of the IP Structure and Behavior:	166
5.1.2.2 Building Power-Aware EFSM Models	169
5.1.2.3 White-Box Vs. Black-Box	170
5.1.2.4 Using PMPs to Locate Power-Aware Checks in the SystemC/TLM IP Code	172

5.2	Dynamic Contracts for Verification of Power-Aware Properties	175
5.2.1	Design Verification Techniques	175
5.2.1.1	Static Verification	175
5.2.1.2	Dynamic Verification	176
5.2.1.3	Assertion Based Verification	177
5.2.1.4	Enabling Design-By-Contract in an Assertion Based Ver- ification Process	178
5.2.2	Verification of Power-Aware Designs	180
5.2.2.1	Structural Bugs	180
5.2.2.2	Control/Sequence Bugs	182
5.2.2.3	Architectural/Coherence Bugs	186
5.2.3	A Modular Power-Aware Verification Flow	187
5.3	Conclusion and Discussion	191
6	The USLPAL Base Utilities	193
6.1	Source Code Instrumentation For the USLPAM Application: A White-Box Based Approach	194
6.1.1	Overview of the White-Box Approach	194
6.1.1.1	The PwARCH Utility Features	195
6.1.1.2	Application on a Case-Study	206
6.1.2	Enhancing the USLPAM Using a Model driven Engineering (MDE) Approach	210
6.1.2.1	The Proposed MDE Approach	212
6.1.3	Concluding Remarks	214
6.2	Power-Aware Wrappers For The USLPAM Application: A Black-Box Based Approach	215
6.2.1	Overview of the Black-Box Approach	215

6.2.1.1	Constraints of the USLPAM application on Black-Box Virtual Platforms	215
6.2.1.2	Power-Aware Wrapper Features	217
6.2.1.3	The PAL Utility For Reuse and Modularity	221
6.2.2	Application on Case-Studies	223
6.2.2.1	Application on an Audio System Virtual Prototype	223
6.2.2.2	Black-Box Versus White-Box Comparison Results	229
6.2.3	Concluding Remarks	232
6.3	The USLPAL Base Utilities for the USLPACom	233
6.3.1	Motivations	233
6.3.2	Power Domain Based Modeling Approach	235
6.3.2.1	Power Domains Layers	236
6.3.2.2	Sourced Power-Aware Communications	236
6.3.2.3	Identifier-Based Addressing and PDMgIF Compliant Components Classification	237
6.3.2.4	PDMgIF Initiator Requirements	238
6.3.2.5	PDMgIF Target Requirements	239
6.3.3	PDMgIF: a Transaction-Level Interface Protocol for Power Domain Management	240
6.3.3.1	Methodology for the PDMgIF Protocol Modeling in TLM 2.0	240
6.3.3.2	Issues of Modeling the PDMgIF Interface Protocol in TLM 2.0	241
6.3.3.3	The PDMgIF Channels and FSMs Definition	243
6.3.3.4	The PDMgIF Protocol Interconnect Structure Behavior Definition	247
6.3.4	Application on a Case-Study	249

6.3.5	Locality and Scalability	253
6.3.6	Concluding Remarks	257
7	Conclusions and Prospects	258
7.1	Summary of Contributions	258
7.2	Prospects	262
7.2.1	Extending the USLPAF Framework With Additional Power-Aware Simulation Semantics	262
7.2.2	Thermal Behavior Analysis and Management Based on Power-Aware Simulation	264
7.2.3	Automating LPDISE	264
7.2.4	A Toolset for PMPs Identification and Off-Line Simulation and Val- idation	265
7.2.5	Complementary Studies on Power-Aware Verification	266
7.2.6	Towards a Standard Structure for Easy Integration and Reuse of IPs' Power Intent and Control Features	267
7.2.7	Validation of System-Level Results at Lower Levels of Abstraction than TLM	268
7.3	Author's Publications Related to This Thesis	269
7'	Conclusions et Perspectives (In French)	270
7'.1	Résumé des Contributions	270
7'.2	Perspectives	274
7'.2.1	L'extension de l'environnement USLPAF avec des sémantiques de simulation supplémentaires orientées consommation d'énergie	275
7'.2.2	Analyse du comportement thermique et de gestion basées sur la simulation orientée consommation d'énergie	277
7'.2.3	Automatiser LPDISE	277

7.2.4	Un ensemble d'outils pour l'identification, la simulation hors ligne et la validation des PMPs	278
7.2.5	Des études complémentaires sur la vérification orientée consommation d'énergie	278
7.2.6	Vers une structure standard pour une réutilisation et intégration facile de l'architecture et du contrôle en énergie d'une IP	280
7.2.7	Validation des résultats obtenus à un niveau d'abstraction inférieur au niveau TLM	281
7.3	Publications de l'auteur liées à cette thèse	282
	Appendix	284
A	Using an MDE Approach for the Enhancement of the USLPAM Simulation-Based Flow	286
A.1	Automatic Generation of "PowerMain" and UPF Codes Using Our MDE-Based Approach	286
A.1.1	Automating "PowerMain" Code Generation	287
A.1.2	Automating UPF Code Generation	290
A.2	Performance Enhancement Results	291
	Bibliography	311

Acknowledgments

Completion of my PhD required countless selfless acts of support, generosity, and time by people in my personal and academic life. I can only attempt to humbly acknowledge and thank the people and institutions that have given so freely throughout my PhD career and made this dissertation possible.

I am deeply grateful to my advisor, Michel Auguin, for the guidance, support, encouragement and invaluable expertise that he has shown me over the last four years. Sincerely, it has been an extreme pleasure and a privilege to work with him and learn from him. He reposed a lot of confidence in me, which let me feel a great responsibility and gave me enormous freedom and challenges in my work. He helped relieve much of the tedium, assuage my apprehensions, boost my self-esteem, and make the thesis work a joy by being readily accessible, letting me have his undivided attention most of the time, offering sound and timely advice and suggesting me corrective measures.

I am extremely thankful to Alain Pegatoquet, my second mentor, for his encouragement, great care and technical guidance during my years at University of Nice-Sophia Antipolis.

I feel honored to have had respected researchers also take the time to serve on my committee. In this regard, thanks are due to Jean-Didier Legat, Frederic Rousseau, Florence Marananchi and Robert De Simone. I am thankful to my entire committee for their feedback on my work and their flexibility in accommodating my requests and respecting my personal constraints when planning my dissertation defense. I must also acknowledge that all these people have greatly inspired me.

This work was supported by the French National Agency of Research (ANR) Arpege project HeLP (High Level Models for Low Power Systems) bearing reference ANR-09-SEGI-006. Many thanks to all the academic and industrial partners of this project for

their valuable comments and advices of expertise on my thesis work during the HeLP meetings: Docea Power Inc., ST Microelectronics Inc., the Verimag Laboratory and the AOSTE team from the INRIA of Sophia-Antipolis. In particular, I am grateful to Florence Marananchi and Matthieu Moy from the Verimag Laboratory for the perfectionists' ideas, remarks, and correction guidelines they gave me during my dissertation. Thanks also to Julien DeAntoni, Carlos Gomez and Robert De Simone from the Aoste team for the collaboration opportunities they provide me to acquire knowledge in the model driven engineering field.

In order to develop and deepen my research ideas during this dissertation preparation, local interactions with the two leading companies, Synopsys Inc. and Texas Instruments Inc, were repeatedly made and led to fruitful exchanges and to the foundation of the new ANR's HOPE project. In this context, I particularly thank the two Synopsys' engineers, Denis Paterson and Xavier Buisson, for the technical support they gave me to rapidly get started with the Innovator tool and use it to validate my approaches. A big thank you also to the design platform CIMPACA staff, in particular Pierre Bricaud and Michel Dubois, for allowing the easy use of several known commercial EDA tools.

Great thanks to all the LEAT laboratory members that helped me on countless occasions especially Daniel Gaffe, Cecile Belleudy, François Verdier and Khurram Bhatti.

Finally, deep gratitude to my parents, Mohamed Mbarek and Radhia Achour, for all their invaluable love, to have always given me so much and taught me the good things that really matter in life. I am also indebted in no small measure to my husband Ameer Sbouï who offered me unconditional understanding, patience and encouragement, and endured a lot during my PhD career. His love and prayers had been like a source of light in my life.

Finally, I am thankful to God Almighty for the turn of events that led to this most valuable and rewarding phase of my life ...

List of Figures

1'.1	Tendances en consommation d'énergie des circuits intégrés selon l'ITRS (International Technology Roadmap for Semiconductors) (Source : Silicon Integration Initiative (Si2), dérivé de l'ITRS 2005)	2
1'.2	Les Opportunités d'optimisation de la Consommation d'Energie à Chaque Niveau d'Abstraction (Source : LSI Logic)	5
1.1	IC Power Trends According to The International Technology Roadmap for Semiconductors (ITRS) (Source: Silicon Integration Initiative (Si2), derived from ITRS 2005 Power Consumption Trends for SoC-PE)	16
1.2	Power Optimization Opportunities at Each Level of Abstraction (Source: LSI Logic)	19
2.1	Typical SoC Design Flow Phases and Abstraction Levels	30
2.2	Model Transformation Process [143]	35
2.3	Example TLM Platform	36
2.4	Example Memory Address Map	37
2.5	Example of SystemC/TLM Architecture and Communication	40
2.6	TLM 2.0 Overview [124]	42
2.7	The TLM 2.0 Default Transaction Fields	42
2.8	The TLM 2.0 tlm_phase Class	44
2.9	A Combined Interface Definition	44

2.10	Message Sequence Chart of a Transaction Between Initiator and Target Using the Loosely-Timed Base Protocol	46
2.11	Message Sequence Chart of a Transaction Between Initiator and Target Using the Approximately-Timed Four-Phase Base Protocol	47
2.12	Example of New Protocol Traits Class With a Non-Ignorable TLM 2.0 Payload Extension	48
2.13	Voltage, Power and Clock Domains for Power Management [15]	52
2.14	High-to-Low Level Shifter in the Destination Domain	53
2.15	Power Management Structure Example Based on Power Domains Partitions [138] [Source: Infineon Diagram With Added Power Domains]	54
2.16	Basic Isolation Cells	55
2.17	Retention Registers	56
2.18	Block Diagram of an SoC with Power Gating	57
2.19	Power Manager Classification	58
2.20	Texas Instruments OMAP3 Block Diagram	60
2.21	Texas Instruments OMAP3 Power Architecture	61
2.22	SPMI System Example [13]	62
2.23	SPMI Slave State Diagram	63
2.24	Reset, Sleep, Shutdown and Wakeup SPMI Command Sequences	64
2.25	ACPI Interface [44]	66
2.26	Functional and Power Intent	67
2.27	Low Power Format Standards Tool Flow Starting from RTL	68
2.28	Example of UPF Defined Concepts	70
2.29	Current Status of All Power Formats [148]	72
3.1	Extending the Low Power Flow to TLM	90
3.2	RTL Functional Code Example	90

LIST OF FIGURES

3.3	UPF Code Example for Retention Strategy Specification	91
3.4	Code Added by The Power-Aware Simulator as Interpretation of UPF Com- mands	91
3.5	Script Code Added in Case of a Non Power-Aware Simulator for Retention Behavior Simulation	91
3.6	Interfaces of a Power-Aware Transaction-Level Component	95
3.7	Relationship between DbC, CBD, and TLVP Approaches	97
3.8	Different Types of Transaction-Level Virtual Platforms	101
3.9	T_{be} Makes the Energy Consumption Equal [45]	102
3.10	Impact of the power domain management strategy on handling time and energy overheads in case of depending power domains	105
3.11	Example TLM platform and corresponding power architecture	107
3.12	Impact of added power management latencies on timing-dependent func- tional synchronization	108
3.13	Impact of added power management latencies on timing-independent func- tional synchronization	111
3.14	Impact of functional synchronization mechanisms on power management opportunities	112
3.15	The Unified System-Level Power-Aware Framework (USLPAF)	114
4.1	The General USLPAM Flow	117
4.2	Example of PMPs Specification	121
4.3	Abstract UPF Semantics For Power Intent Specification at Transaction-Level	124
4.4	Inferring Power Gating Behavior to RTL Using UPF Semantics	127
4.5	Example of the power-aware internal interfaces use during power-gating . .	128
4.6	Comparison of Energy Consumed With/Without Power Gating	132
4.7	The PMU Features	134
4.8	Hookup and Power-up/Power-down Sequencing of a Domain Power Controller	135

4.9	Example of a Scenario-Based Power Management Strategy Use	138
4.10	Pseudo-code of the Power Manager Module	141
4.11	Pseudo-code of the <i>PM Commands Dispatcher</i> Process	142
4.12	Handling Dependencies in a Reative Power Management Strategy	144
4.13	A PDMgIF Bus Interface for Inter-Power Domain Communication	145
4.14	A Functional SoC Example	147
5.1	Building an EFSM-Based Behavioral Model of a TL Component	163
5.2	An Example of a Slave/Master SystemC-TLM White-Box IP block: Inter- face and Structure	166
5.3	Example of the EFSM-Based Methodology Application on the White-Box IP Version	167
5.4	State Transition Diagram of an EFSM Modeling The Power Managed Func- tional Behavior of a Black-Box IP	170
5.5	Using PMPs for Checking Power-Aware Specifications in a SystemC/TLM IP Code	173
5.6	Redundant Isolation [137]	181
5.7	A Specification Example of Allowed Power States Sequences	184
5.8	Example of Class 1 Contract-Based Assertions Inserted in A PowerSwitch Class[137]	188
5.9	Using AOP and Callbacks of Monitors for a Modular Power-Aware Verifi- cation Framework	190
6.1	Using the PwARCH Utility within the USLPAM Simulation-Based Flow	195
6.2	PwARCH General Class Structure	196
6.3	Partial Class Diagram for Concepts in PwARCH: Purposes and Relationships	197
6.4	The Instrumentation-Based Approach	198
6.5	The Power Domain Management Interface in PwARCH	203

LIST OF FIGURES

6.6	The Case-Study: Architecture and Transaction Flow Analysis	206
6.7	Power-Aware Architecture Alternatives	208
6.8	Application of the Power Intent Specification Stage	209
6.9	Power Domains Hierarchy and Characteristics in Each Power Domain Partitioning Alternative	209
6.10	MDE Approach Integration in the USLPAM Simulation-Based Flow	212
6.11	Structure and Behavior of a slave/master IP's Power-Aware Wrapper	218
6.12	The Pw_Prefs Class of the PAL Library	221
6.13	The Wrapper_Factory Class of the PAL Library	222
6.14	The Wrapper_Factory_Support Class of the PAL Library	223
6.15	The Audio Virtual Platform Block Diagram	224
6.16	Excerpt of the Transaction Flow During the Record Scenario Using Platform Analyzer Tool	225
6.17	Developing Power Wrappers Using the Innovator Tool	226
6.18	The Considered Power-Aware Architecture Alternatives	227
6.19	A Power-Aware Architecture Alternative	230
6.20	Layering the Power Domain Management TL Structure on Top of the Functional TL Model	235
6.21	A Generic Example Showing the Internal Structure of the AO_PD Power Domain	238
6.22	Overview of the General Modeling Methodology	241
6.23	The PDMgIF Protocol Phase Sequences	244
6.24	Mapping Channels' FSMs to Initiator and Target Finite State Machines	246
6.25	The Internal Structure and Behavior Modeling of the PDMgIF Interconnect Using the TLM 2.0 Standard Transport Interfaces	248
6.26	The Considered Power Architecture Alternatives	250

6.27	Energy savings, modeling effort savings and simulation time for the various power management strategies and power architecture alternatives	252
6.28	Using the PDMgIF Interface in a Hierarchical Power Domain Management Structure	254
6.29	Example of Three-Level Hierarchical Power Domain Management Tree Structure	256
A.1	Generation and Integration process	287
A.2	The Power Intent (PI) Metamodel	288
A.3	Relationships Between UPF Standard, PwARCH and PI Metamodel	289
A.4	Comparison of Results Between Manual Writing and Automatic Generation of "PowerMain" Codes	292

List of Tables

- 4.1 An Overview on The Different Classes of Contracts Involved in The Power-Aware Verification Process 151

- 5.1 White-Box Vs. Black-Box 171

- 6.1 Excerpts of Power-Aware Verification Results 210
- 6.2 Power State Table for Alternative (a) 227
- 6.3 Energy Savings for the Different Power Intent Alternatives According to the Play & Record Software Scenario 228
- 6.4 Comparing the Black-Box Platform Performances With Those of the White-Box Platform 231
- 6.5 Attributes and Timing Points of Each Channel 242
- 6.6 An Example of a PST for the Power Architecture Alternative 1 251
- 6.7 A Power State Table Attached to PD_Top 255
- 6.8 A Power State Table Attached to PD3 255

- A.1 Required Effort to Perform Generation Process 293
- A.2 Analogy Between Some Code Lines of the PI(b) "PowerMain" and the Corresponding UPF Commands 295

Acronyms & Abbreviations

ACPI	Advanced Configuration and Power Interface
AT	Approximately Timed
CA	Cycle Accurate
CPF	Common Power Format
DE	Design Element
DPC	Domain Power Controller
DPM	Dynamic Power Management
DVFS	Dynamic Voltage and Frequency Scaling
EFSM	Extended Finite State Machine
HFSM	Hierarchical Finite State Machine
IID	Initiator Identifier
LPDISE	Low Power Design Intent Space Exploration
MDE	Model Driven Engineering
NRCT	Non Request Capable Target
PAL	Power Aware Layer
PCI	Peripheral Component Interconnect
PCIe	Peripheral Component Interconnect express
PD	Power Domain
PDID	Power Domain Identifier

PDMgIF	Power Domain Management InterFace
PST	Power State Table
PSTrans	Power State Transition
PSw	Power Switch
PI	Power Intent
PM	Power Manager
PME	Power Management Event
PMP	Power Management Point
PMU	Power Management Unit
PRCM	Power Reset, and Clock Manager
PV	Programmer View
PVT	Programmer View with Timing
PwARCH	Power ARCHitecture
PwCTr	Power Ccontrol Transaction
PwE	Power Event
RCT	Request Capable Target
RTL	Register Transfer Level
SPMI	System Power Management Interface
SSM	System Synchronization Point
SSP	System Synchronization Mechanism
TID	Target Identifier
TL	Transaction Level
TLM	Transaction Level of Modeling
UPF	Unified Power Format
USLPAF	Unified System Level Power Aware Framework
USLPAM	Unified System Level Power Aware Methodology
USLPAL	Unified System Level Power Aware Library
USLPACom	Unified System Level Power Aware Communication
VP	Virtual Platform

Chapitre 1'

Introduction (In French)

1'.1 Problématique

1'.1.1 L'optimisation de la consommation d'énergie dans un système sur puce

DE nos jours, la consommation d'énergie est devenue la question la plus critique dans la conception d'un système sur puce (SoC). Avec la technologie de processus évolutif et la croissance explosive des domaines du sans fil et de la communication mobile ainsi que de l'électronique à domicile vient la demande du calcul intensif et des fonctionnalités complexes pour des raisons de concurrence. Les appareils portables d'aujourd'hui, sont sensés non seulement avoir une petite taille et être léger, mais aussi fournir une batterie de longue durée. Même les systèmes de communication filaires doivent accorder une attention à la chaleur, à la densité de la consommation et aux exigences de faible puissance. Figure 1'.1 illustre l'évolution de la densité de la consommation par rapport aux exigences de la conception de la consommation d'énergie pour les systèmes sur puce modernes.

Comme il est décrit dans la Figure 1'.1, le large écart représente le défi le plus critique rencontré de nos jours. Pour relever ce défi, les concepteurs du SoC changent de l'approche monolithique traditionnelle, où une source unique d'alimentation est utilisée pour toutes les portes internes d'une conception, à une architecture ayant de multiples alimentations, où les différents blocs fonctionnent à différentes tensions selon leurs exigences

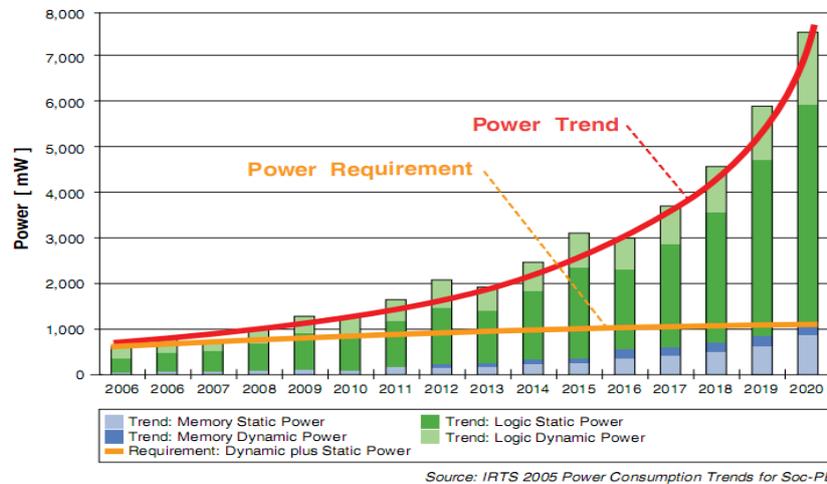


FIGURE 1'1 – Tendances en consommation d'énergie des circuits intégrés selon l'ITRS (International Technology Roadmap for Semiconductors) (Source : Silicon Integration Initiative (Si2), dérivé de l'ITRS 2005)

fonctionnelles. Dans certains cas, les concepteurs utilisent la technique de tension à échelle ("voltage scaling") pour changer la tension (et la fréquence d'horloge) d'un bloc critique selon sa charge de fonctionnement. Avec cette nouvelle approche de conception des SoCs modernes, les différents blocs ont des contraintes et objectifs de performance différents. La forme la plus basique de cette approche est de partitionner la logique interne de la puce en plusieurs zones de tension ou de domaines d'alimentation, chacun ayant son propre arbre d'alimentation. Une fois les alimentations sont séparées, des stratégies de consommation d'énergie plus efficaces peuvent être appliquées. Ils incluent notamment des stratégies multi-tension pour la technique "voltage scaling" lorsqu'une haute performance n'est pas nécessaire pour certains blocs du SoC, ainsi que la stratégie d'alimentation périodique ou "Power gating" dans le cas où les domaines d'alimentation seront carrément forcés à un voltage nul. Toutefois, la mise en oeuvre de ces stratégies présente certains défis aux concepteurs. Il s'agit notamment de quatre principaux et qui sont :

- **La conception et la vérification du réseau d'alimentation et des interfaces de gestion d'alimentation supplémentaires sont nécessaires :** Selon les stratégies de faible consommation d'énergie, le réseau d'alimentation, y compris les interrupteurs et les sources de courant, doit être défini d'une manière adéquate. En outre, les interfaces de chaque domaine d'alimentation doivent être soigneusement conçues et vérifiées. Ces

interfaces comprennent des éléments structurels tels que le décalage de niveau logique et les cellules d'isolement qui sont requis pour passage entre des domaines ayant des alimentations primaires différentes. Chacune de ces interfaces ainsi leurs arbres d'alimentation doivent être gérés dans un ordre précis. Pour cela, une interface de contrôle entre chaque domaine d'alimentation et son contrôleur de tension doit être ajoutée. Les commandes fournies par les contrôleurs de puissance définissent le comportement en consommation d'énergie d'un SoC à multiples domaines d'alimentation et dépendent du choix du réseau d'alimentation. Un tel choix peut compliquer le routage de l'arbre d'alimentation de la puce et conduit, non seulement à une grande surface de silicium (donc un grand coût) du circuit final, mais aussi à de complexes contrôleurs de consommation d'énergie. Incontestablement, une grande complexité dans la phase de vérification est introduite concernant notamment l'intégrité du réseau d'alimentation, la connectivité entre les domaines d'alimentation et les séquenceurs de contrôle de la consommation d'énergie.

- **L'augmentation des états de consommation d'énergie :** Le réseau d'alimentation d'un SoC permet de définir l'ensemble les états de consommation d'énergie de chaque domaine d'alimentation. Les SoCs d'aujourd'hui sont grands et supportent un grand nombre d'applications logicielles embarquées. Donc, ils ont divers états logiciels, chacun d'eux se caractérise par une charge spécifique de travail. En conséquence, de nombreux domaines d'alimentation sont alors nécessaires afin de correspondre à toutes les charges applicatives potentielles de travail demandées par l'utilisateur final. Parce que la définition des limites des domaines d'alimentation est si étroitement liée aux exigences de consommation des différentes applications embarquées, le nombre élevé d'états logiciels rend le partitionnement en domaines d'alimentation une tâche difficile. En outre, un nombre élevé de domaines d'alimentation engendre un nombre croissant d'états de consommation d'énergie ce qui complique encore plus la tâche de vérification.

- **Des compromis entre la réutilisation et l'efficacité énergétique doivent être considérés :** En fonction de la charge de travail logicielle requise, les états des domaines d'alimentation sont ajustés. Un tel changement d'états engendre une pénalité en énergie qui peut influencer sur les économies d'énergie réalisable. En règle générale, la technique du "Power Gating" ajoute des retards considérables pour entrer et sortir en sécurité des différents modes de consommation d'énergie. Par conséquent, allumer et éteindre un domaine d'alimentation fréquemment dans le temps peut gaspiller plus d'énergie en rechargeant l'état enregistrée à chaque réveil. Prenant en compte les besoins logiciels en énergie, le

concepteur doit donc définir et partitionner le SoC en domaines de faible consommation d'une manière à ce que l'infrastructure en énergie permette une meilleure économie d'énergie. Cependant, une telle infrastructure doit également être conçue pour donner un compromis raisonnable entre sa réutilisation et son efficacité énergétique. En effet, comme une conception à faible consommation reste inchangée une fois implémentée, elle doit garder son efficacité lors de l'ajout de nouvelles applications avec de nouvelles exigences d'alimentation sur le même SoC.

- **La corrélation entre les dépendances fonctionnelles et en énergie doit être soigneusement gérée :** D'un côté, une infrastructure de gestion d'énergie peut créer des dépendances structurelles entre les états des domaines d'alimentation. De l'autre côté, les fonctions et les états de certains blocs matériels peuvent nécessiter des états ou des fonctions bien spécifiques d'autres blocs. Cela crée des dépendances fonctionnelles entre les états des domaines d'alimentation. Par conséquent, une implémentation à faible consommation doit tenir compte de ces possibles dépendances fonctionnelles. Par ailleurs, une politique de gestion d'énergie doit respecter les dépendances fonctionnelles et structurelles. En d'autres termes, un SoC final doit combiner une ces deux types de dépendances d'une façon à ce qu'aucun conflit entre eux puisse se produire. Cette contrainte doit être prise en compte très tôt dans le flot de conception ciblant une faible consommation. Elle doit aussi être intégrée dans la dernière phase de vérification du SoC.

La complexité ajoutée lors de la conception et de la vérification des SoCs de faible consommation est un problème commun soulevé par cet ensemble de défis. En outre, ces défis invoquent une relation forte entre les aspects fonctionnels et de consommation d'énergie dans un SoC. Comprendre cette relation permet de prendre des décisions efficaces de gestion d'énergie et atteindre les objectifs de ces différents défis. Face aux défis de la réalisation d'une architecture faible en consommation et moins erronée, de la réutilisation et la modularité d'un design faible consommation, ainsi que de la gestion du problème d'explosion d'états d'énergie, les questions cruciales suivantes sont encore sans réponse : *Quelle est l'architecture et la politique de gestion d'énergie à appliquer ? Quelles sont les stratégies de faible consommation à utiliser et à quel endroit du SoC doivent être appliquées ?*

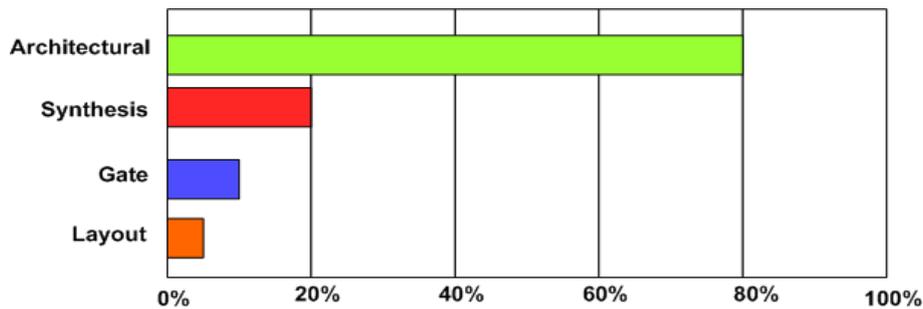


FIGURE 1'.2 – Les Opportunités d'optimisation de la Consommation d'Énergie à Chaque Niveau d'Abstraction (Source : LSI Logic)

1'.1.2 L'abstraction faible consommation

Jusqu'à présent, la plupart des efforts d'optimisation de consommation ont été portés à un niveau de conception proche des registres, portes logiques ou encore niveau "layout". Cependant, travailler directement sur une liste de portes logiques ("Netlist") pour ajouter des composants de gestion de consommation d'énergie engendre à la fois une lente simulation et des difficultés de débogage. Par conséquent, les spécifications ciblant une faible consommation, conçues dès le niveau "RTL" (Register Transfer Level) garantissent la validation de ces composants à ce niveau RTL, et seront par la suite synthétisés, placés et routés correctement dans l'implémentation matérielle du SoC. Cela nécessite un format unique de spécification de l'infrastructure de consommation d'énergie supporté par tous les outils de conception des SoCs à n'importe quel niveau d'abstraction. Un tel format devrait faciliter l'implémentation, la validation et les raffinements incrémentaux de modèles de faible consommation tout en adressant la réutilisation des spécifications fonctionnelles.

Le standard "CPF" (Common Power Format) [29] et la norme IEEE 1801-2009 de "Accelera" [30], connu sous le nom de «UPF» (Unified Power Format), définissent les deux un langage et une sémantique de simulation permettant de spécifier comment les alimentations doivent être fournies, distribuée et dynamiquement gérées dans un système numérique à faible consommation. Ces normes ont déplacé la spécification faible consommation vers le niveau RTL et fourni les moyens de base pour spécifier les éléments de faible consommation et les informations de leurs contrôles nécessaires pour adapter les modules RTL à leurs exigences en faible consommation. Ces caractérisations sont décrites dans

un format portable pour l'utilisation durant la simulation, la synthèse et le placement et routage. La portabilité est renforcée par la méthodologie utilisée par ces standards, qui se base sur la séparation entre les aspects fonctionnels et d'énergie. Ceci est réalisé en fournissant une spécification de faible consommation dans un fichier séparé du code de la spécification fonctionnelle.

Une telle méthodologie a été utilisée pour diverses raisons. Tout d'abord, elle ne nécessite ni la mise à jour de la spécification fonctionnelle RTL lorsque la description en consommation d'énergie est ajoutée ni vérifier cette dernière s'il y aura un changement dans le code RTL du module. En plus, le lien étroit entre l'aspect fonctionnel et la spécification énergétique du module n'est pas obligatoire. En outre, comme les plus importants aspects de la spécification en consommation d'énergie sont liés à la technologie utilisée, ils seront généralement modifiés plus souvent que la spécification fonctionnelle RTL.

Malheureusement, en utilisant ces standards, seules les spécifications fonctionnelles dès le niveau RTL peuvent être superposées avec la sémantique orientée consommation (y compris des éléments structurels de gestion de l'alimentation et les aspects comportementaux). Afin d'appliquer ces sémantiques orientées consommation, au niveau système ou "ESL" (Electronic System Level), elles doivent être adaptées à ce dernier niveau. En réalité, les possibilités d'optimisation de la consommation d'énergie sont meilleures au niveau ESL, lorsque l'architecture est en cours de développement.

Selon une étude faite par "LSI Logic" que montre la Figure 1.2, plus une description du système se déplace à un niveau d'abstraction plus bas, moins les techniques d'optimisation d'énergie pourraient être appliquées. La Figure 1.2 montre que les techniques disponibles à la phase de synthèse RTL ont la capacité de réduire la consommation par 20 pourcent. Celles qui sont au niveau porte logique offrent une réduction de 10 pourcent, tandis que celles au niveau de "Layout" peuvent réduire la consommation seulement de 5 pourcent. En attendant le code RTL pour commencer à optimiser la consommation est une occasion ratée car la consommation en énergie peut être réduite de 80 pourcent si elle a été modélisée au niveau ESL.

L'optimisation d'énergie doit plutôt commencer par l'analyse architecturale, l'exploration et l'optimisation de la consommation au niveau ESL. Ce niveau d'abstraction permet une simulation plus rapide et des modèles d'exécution plus simples, d'où une vérification simple et rapide. En outre, il est très important de simuler la plateforme avec le logiciel

d'application finale afin d'identifier les opportunités d'optimisation de la consommation en fonction de la charge du travail du système. Tel qu'il est mentionné dans la section précédente, la corrélation de la consommation d'énergie avec le travail réel effectué par le système fournit une plus grande opportunité de gérer l'énergie. Au niveau ESL, le logiciel embarqué final est disponible au début du flot de conception du SoC et peut être rapidement validé sur une plateforme matérielle de référence. Pour toutes ces raisons, adresser la gestion d'énergie au niveau ESL contribue à la réalisation de différents compromis mentionnés dans la section précédente tout en optimisant considérablement l'énergie dissipée du SoC.

Les prototypes virtuels au niveau transactionnel sont l'une des principales méthodologies de conception ESL. En premier lieu, ils ont été développés pour accélérer la validation des logiciels embarqués. Un modèle au niveau transactionnel ou TLM (Transaction Level of Modeling) [124] exclut certains détails du niveau signal du modèle du système afin de se contenter de son aspect comportemental. Il utilise la notion de transaction pour modéliser les communications entre les composants du système. Par conséquent, moins d'effort est nécessaire pour concevoir un modèle au niveau des transactions et ce modèle est disponible bien avant le modèle RTL. Pour écrire des spécifications fonctionnelles au niveau transactionnel, le standard SystemC TLM 2.0 [124] propose des règles de codage et des mécanismes de modélisation qui permettent le raffinement au niveau TLM, de modèles non temporisés vers des communications à cycle d'horloge près. Cependant, cette norme n'a pas encore défini de sémantiques pour la modélisation et l'optimisation de la consommation d'énergie et pour le couplage des spécifications fonctionnelles et d'énergie. Dans ce contexte, beaucoup de questions cruciales se posent : *Quelles sont les sémantiques des normes existantes pour la conception faible consommation qui doivent être adoptées et abstraites au niveau TLM ? Y a-t-il des contraintes ou des extensions des standards requises pour appliquer la simulation orientée consommation au niveau TLM ? Quels sont les mécanismes nécessaires pour modifier le comportement du module matériel afin de refléter le changement d'états d'énergie ? A quel point peut-on appliquer la séparation entre les aspects fonctionnels et d'énergie adoptée par ces standards au niveau TLM ? Comment un système faible consommation conçu et évalué au niveau TLM peut être réutilisé dans le reste des étapes du flot de conception du SoC ?*

1'2 Thèse

Cette thèse tente de résoudre les questions soulevées dans les sections précédentes. Elle consiste à procurer une étude complète des opportunités d'optimisation de consommation d'énergie basées sur la composition et la gestion des domaines d'alimentation au niveau TLM. Ce travail utilise la notion et le mot clé du domaine d'alimentation pour décrire un groupe de blocs fonctionnels qui partagent le même réseau et support d'alimentation, donc qui a son propre ensemble de modes d'énergie et peut être contrôlé individuellement. Une attention particulière dans cette étude a été portée au déplacement du niveau d'abstraction de la description de la consommation au niveau TLM. En conséquence, les sémantiques de simulation et de vérification pertinentes qui sont définies dans les standards existants de conception faible consommation seront également transférées au niveau TLM.

Une autre préoccupation de cette étude est d'explorer les relations entre les concepts orientés énergie et ceux purement fonctionnels. En raison d'incohérence éventuelle entre ces deux aspects, le comportement d'une infrastructure de gestion d'énergie peut affecter la fonctionnalité initiale du système. En dépit de cette relation étroite, nous proposons tout au long de cette thèse des solutions d'étendre les spécifications fonctionnelles au niveau TLM avec des sémantiques de consommation d'énergie. Comme les modèles TLM sont d'abord développés pour valider les logiciels embarqués, l'ajout des fonctionnalités orientées énergie, doit uniquement être activé à des fins d'analyse de consommation, sinon désactivé.

Un deuxième type de relations entre les concepts orientés énergie et ceux purement fonctionnels se résume dans les interactions basées sur l'activité entre les domaines d'alimentation. Comme un bloc fonctionnel dans un domaine d'alimentation peut interagir avec un bloc dans un autre domaine d'alimentation, les transactions aux limites des deux domaines d'alimentation peuvent entraîner ou nécessiter un changement d'état d'énergie d'un sous-système. Ces transactions représentent les interactions orientées consommation et doivent être soigneusement analysées. Typiquement, un système faible consommation comprend une unité de gestion d'énergie par domaine d'alimentation. La capture d'interactions entre domaines d'alimentation est utile pour une telle unité spécialisée pour prendre de bonnes décisions lors d'une gestion dynamique de l'alimentation.

En réalité, l'analyse des relations entre la partie fonctionnelle et celle orientée consommation d'énergie au niveau TLM contribue à explorer à la fois une architecture économe

en énergie et une politique de gestion des domaines d'alimentation pour un SoC faible consommation. Afin de faciliter et d'accélérer l'exploration, une interface commune et générique de gestion de domaine d'alimentation est nécessaire. Ainsi, l'architecture de gestion de l'alimentation peut être implémentée indépendamment des domaines et de l'infrastructure de faible consommation. En d'autres termes, le choix de l'architecture de l'unité de gestion de l'alimentation et de la stratégie ne devrait pas exiger une nouvelle conception des domaines d'alimentation. De même, la modification de l'infrastructure à faible consommation ne doit ni contraindre la structure en énergie ni le comportement de son unité de gestion. Dans ce travail, nous étendons le standard TLM pour créer un modèle de simulation d'une interface de protocole de gestion des domaines d'alimentation.

Afin de traiter le problème de d'explosion de l'espace à explorer des états d'énergie et de réduire l'effort de modélisation et de vérification de l'unité de gestion d'énergie, nous pensons qu'une structure distribuée de gestion des domaines d'énergie serait plus fiable qu'une seule grande unité centralisée. En outre, les grands SoCs comprennent généralement des sous-systèmes de gestion d'énergie fournis avec leurs propres contrôleurs d'alimentation. Dans une structure hiérarchique de gestion de domaine d'alimentation, chacun de ces contrôleurs représente une unité de gestion d'énergie locale qui gère les états d'énergie de son sous-système sous le contrôle d'une unité de gestion d'énergie globale. Une structure hiérarchique des unités de gestion d'énergie d'un SoC nécessite une synchronisation entre les unités de gestion d'énergie locales et l'unité globale tout en respectant les dépendances entre les états des domaines d'alimentation. Ces exigences doivent être prises en compte par l'interface de protocole de gestion des domaines d'alimentation proposée.

Au meilleur de notre connaissance, ce travail est la première étude complète sur le sujet de la conception et la vérification faible consommation au niveau TLM. L'objectif principal est de réduire la consommation d'énergie tout en répondant aux exigences de performance. Ainsi, la conception d'un système à faible consommation à partir du niveau TLM vise d'abord à une prise de décision tôt et rapide d'une solution d'implémentation efficace en énergie incluant une architecture et une stratégie de gestion des domaines d'alimentation pour un système fonctionnel donné. Le résultat est une description de référence d'une conception à faible consommation pré-vérifiée et à rendement énergétique haut, utilisée par les équipes de conception RTL et comme entrée pour des outils au niveau RTL (lors du raffinement du modèle TLM au niveau RTL).

1'3 Aperçu de la thèse

1'3.1 Contributions

Une contribution principale de cette thèse concerne l'étude des concepts de conception à faible consommation d'énergie pour un modèle fonctionnel au niveau transactionnel dans un environnement commun, appelé USLPAF. L'USLPAF, se référant à "Unified System-Level Power-Aware Framework", offre une méthodologie connue sous "USLPAM" (Unified System-Level Power-Aware Methodology) qui combine la conception et la vérification orientées faible consommation au niveau transactionnel dans un flot de conception unifié. L'USLPAF fournit également une librairie nommée "USLPAL" (Unified System-Level Power-Aware Library) comprenant un ensemble de techniques de modélisation et d'utilitaires permettant d'appliquer facilement et rapidement la méthodologie USLPAM. Sur la base de cet environnement, ce travail contribue à :

- **Une Méthodologie orientée consommation d'énergie au niveau système :**

Cette méthodologie permet d'ajouter des capacités de spécification et de gestion d'une infrastructure à faible consommation à des modèles fonctionnels au niveau transactionnel d'une manière bien structurée. Une vérification basée sur la simulation et orientée consommation d'énergie intègre également le flot de la méthodologie proposée. Les sémantiques de simulation et de vérification ainsi que la méthodologie de séparation des aspects fonctionnels et d'énergie définies par le standard UPF ont été utilisées comme support par notre méthodologie USLPAM. L'objectif principal de cette méthodologie est de permettre d'explorer à l'avance différentes architectures à faible consommation d'énergie et alternatives de gestion des domaines d'alimentation afin d'évaluer les effets des techniques de gestion d'énergie sur les performances d'un système et sa fonctionnalité. La méthodologie USLPAM assure la connexion avec le flot de conception à faible consommation au niveau RTL et ce en fournissant une solution de gestion de la consommation, pré-vérifiée et le plus économe en énergie, composée d'une spécification UPF et d'un modèle de référence pour le gestionnaire d'énergie.

- **Contrats basés sur des assertions pour la vérification orientée consommation d'énergie**

La gestion des domaines d'alimentation affecte profondément et complique la tâche de vérification fonctionnelle du SoC. Un processus de vérification orienté consommation d'énergie

a été défini tout au long du flot de méthodologie USLPAM pour vérifier un ensemble de propriétés dans un ordre prédéterminé. Nous supposons que le modèle initial fonctionnel au niveau transactionnel est valide et que son comportement correct est assuré. Ainsi, les propriétés d'énergie définies dans ce travail sont liées à la structure à faible consommation et ses effets sur le fonctionnement normal du modèle initial. Ces propriétés sont définies pour s'adapter à une modélisation au niveau transactionnel. Certaines d'entre elles sont dérivées des spécifications du standard UPF, tandis que d'autres sont déduites des interactions entre les modèles fonctionnels et ceux dédiés faible consommation. Le principe de "DBC" (Design by Contrat) est utilisé pour identifier les propriétés orientées énergie et les classer dans des catégories de contrats. Le test des contrats est effectué en utilisant des expressions d'assertions ajoutées dans le modèle SystemC/TLM.

- **Une méthode pour l'identification des PMPs**

Les emplacements dans un modèle fonctionnel au niveau transactionnel où un changement dans l'état d'énergie du système peut se produire sont appelés points de gestion d'énergie ou (PMPs) (Power Management Points). Déterminer un PMP repose sur la façon dont le logiciel utilise le matériel et comment la consommation d'énergie est impactée. Elle représente la première étape dans le flot de la méthodologie USLPAM et vise l'établissement d'une solution cohérente et efficace de gestion des domaines d'alimentation. Selon les PMPs identifiés, une infrastructure à faible énergie est spécifiée, une stratégie de gestion de l'alimentation est décidée et des propriétés spécifiques orientées énergie sont ajoutées dans le code SystemC/TLM sous la forme d'assertions.

- **Une méthode d'instrumentation du code source pour l'application de USLPAM**

Les prototypes virtuels au niveau transactionnel sont généralement construits par assemblage de propriété intellectuelle (IPs) décrites en SystemC/TLM. Ces IPs peuvent être des boîtes blanches, ayant un code source accessible, ou comme étant des boîtes noires, qui sont déjà préconçus, précompilés et pré-vérifiés. Avoir accès à des modèles des IPs de type boîte blanche ne contraint aucune étape du flot de la méthodologie USLPAM et donne même plus d'opportunités pour réduire la consommation. Nous démontrons comment cela est réalisable grâce à l'instrumentation du code source d'une plateforme virtuelle d'IPs avec des informations sur la gestion de la consommation d'énergie. Une telle méthode basée sur l'instrumentation repose sur l'utilisation de l'utilitaire PwARCH de la librairie USLPAL. Ayant comme objectif principal l'exploration précoce et rapide, PwARCH fa-

cilite chaque étape tout au long du flot de cette méthodologie. En particulier, PwARCH permet une spécification de conception de faible consommation semblable à celle de l'UPF où les changements d'état d'énergie du système sont effectués via des appels aux fonctions spécifiques à la librairie PwARCH.

- **Une méthode orientée couche d'énergie pour l'application de USLPAM**

L'ensemble des points de gestion de consommation obtenus avec une plateforme virtuelle se basant sur des IPs ouvertes peut être différent pour une même plateforme comportant des IPs à code source fermé. Ceci est principalement dû à l'observabilité limitée de changements d'états internes d'une IP à code source fermé. Les principales contraintes de l'application de USLPAM sur ce type de plateformes consistent dans la spécification et la simulation du comportement des mécanismes de rétention de l'état, ainsi que des contrats de contrôle orientés énergie. Une nouvelle méthode qui gère ces contraintes est proposée comme une alternative de la méthode d'instrumentation du code source. Cette méthode est basée sur la superposition des capacités de simulation et de vérification orientées énergie au-dessus de chaque bloc fonctionnel à code source fermé. Par la construction de ces couches dédiées consommation d'énergie, une séparation d'aspects est effectuée semblablement à UPF. L'utilisation de l'utilitaire PAL fourni par la bibliothèque USLPAL aide à personnaliser le comportement requis de chaque couche.

- **Séparation des communications orientées consommation de celles fonctionnelles dans le modèle TLM**

L'ajout de fonctionnalités orientées énergie à une plateforme de simulation fonctionnelle existante modélisée en TLM est le point de départ pour notre méthodologie de USLPAM. Pour cela, les deux méthodes (celle en boîte blanche et celle en boîte noire) doivent adopter la séparation des aspects définie par les normes existants. Cependant, les communications orientées énergie y compris les messages pour le contrôle des états des domaines d'alimentation dépendent encore de deux facteurs : l'infrastructure pour une faible consommation spécifiée et l'architecture de gestion de l'alimentation et la stratégie utilisée. Du moment où une adaptation de la structure de gestion d'énergie est nécessaire quand de ces facteurs change, une telle dépendance rétrécit l'exploration de solutions de gestion de consommation. En outre, contrairement aux communications fonctionnelles basées sur les transactions en lecture et écriture dans/de la mémoire, les communications orientées énergie ont besoin de sémantiques supplémentaires et de mécanismes de synchronisation. Ces communications se produisent également entre les domaines d'alimentation qui sont

plutôt des groupes de composants fonctionnels ayant des caractéristiques communes de consommation d'énergie.

Une contribution de ce travail est une nouvelle technique de modélisation qui sépare les communications orientée énergie de celles fonctionnelles. Au coeur de cette technique de modélisation réside la spécification d'une nouvelle interface de protocole de gestion d'énergie qui unifie les communications entre les domaines d'alimentation indépendamment de l'architecture et de la stratégie de gestion utilisées. Les caractéristiques générique de base de cette interface, appelée PDMgIF, représentent la partie utilitaire USLPACom de la librairie USLPAL.

Afin de réduire la complexité de la modélisation et de la vérification engendrée par l'utilisation d'une unité unique de gestion de domaine d'énergie centralisée, l'interface PDMgIF peut être utilisée pour construire une architecture hiérarchique des unités de gestion de domaines d'alimentation. Dans le cas général, une telle structure représente une bonne solution pour réduire la complexité de la modélisation et de la vérification induite par une structure unique et centralisée de gestion des domaines d'alimentation. Néanmoins, un contrôle de domaines d'alimentation hiérarchique nécessite une manipulation soigneuse des interactions entre les unités locales de gestion d'énergie et celles globales, ainsi des dépendances entre eux. Dans ce contexte, nous discutons l'évolutivité et la modularité de l'interface PDMgIF dans le cas complexe de gestion hiérarchique de domaines d'alimentation.

Toutes les techniques de modélisation proposées dans ce document ont été validées sur des plateformes fonctionnelles modélisées au niveau transactionnel.

1'.3.2 Sommaire

Chapitre 2 commence par une présentation des différents défis en matière de modélisation de haut niveau orientée faible consommation d'énergie pour les SoCs. Tout au long de ce chapitre, nous présentons une étude sur les techniques de gestion d'énergie et la modélisation au niveau transactionnel, ainsi qu'une bibliographie sur la modélisation d'énergie au niveau système ou "ESL" et l'utilisation des standards de conception à faible consommation d'énergie.

Chapitre 3 s'adresse au besoin d'un environnement commun pour la conception et vérification orientée consommation d'énergie au niveau transactionnel et expose les soucis liés à ce genre de modélisation au niveau TLM. Les objectifs, les caractéristiques clés et la composition de notre environnement "USLPAF" pour la la modélisation au niveau TLM de SoCs à faible consommation sont ensuite introduits.

Chapitre 4 présente le flot et les exigences de la méthodologie "USLPAM". Il détaille également le processus de vérification basé sur les contrats et donne des exemples de contrats impliqués dans ce processus de test.

Chapitre 5 aborde le problème de la simulation des états de rétention au niveau transactionnel et explique la méthode proposée pour identifier les points de gestion d'énergie (PMP) basée sur le comportement d'un modèle TLM. Il souligne également l'utilité de ces PMPs pour identifier les emplacements dans le code fonctionnel SystemC/TLM où les contrats d'énergie doivent être ajoutés.

Chapitre 6 couvre les principales utilités de la librairie "USLPAL" utilisés pour faciliter la mise en oeuvre de la méthodologie "USLPAM" sur les différents types de prototypes virtuels au niveau TL. Premièrement, il présente la méthode d'instrumentation du code source ciblant l'application de la méthodologie "USLPAM" sur une IP de type boîte blanche. Il explique les principales caractéristiques de l'utilitaire "PwARCH" fourni par la librairie "USLPAL" pour faciliter l'implémentation de cette méthode.

Deuxièmement, il présente la méthode à base de "Wrapper" proposée pour l'application de la méthodologie USLPAM sur une IP fermée ou en boîte noire. Il explique les principales fonctionnalités de l'utilitaire "PAL" fourni par la librairie "USLPAL" en détaillant ses principaux services.

Il aussi souligne la nécessité d'une interface adaptative de protocole de gestion des domaines d'alimentation au niveau TLM. Une approche de modélisation qui gère la séparation des communications fonctionnelles et d'énergie est présentée avec une nouvelle spécification d'interface de protocole PDMgIF. Ce chapitre explique également la méthodologie utilisée pour modéliser l'interface de protocole PDMgIF au niveau transactionnel et discute la gestion hiérarchique des domaines d'alimentation.

Enfin, le **chapitre 7** conclut cette thèse et identifie des directions pour des travaux futurs.

Chapter 1

Introduction

1.1	Problem Statement	15
1.1.1	Power Optimization in Systems-on-Chip	15
1.1.2	Low Power Abstraction	18
1.2	Thesis	21
1.3	Overview of Thesis	23
1.3.1	Contributions	23
1.3.2	Outline	26

1.1 Problem Statement

1.1.1 Power Optimization in Systems-on-Chip

POWER is emerging as the most critical issue in system-on-chip (SoC) design today. With the evolving process technology and the explosive growth of personal, wireless, and mobile communications, as well as home electronics, comes the demand for high-speed computation and complex functionality for competitive reasons. Today's portable devices are expected not only to be small, cool, and lightweight, but also to provide long battery life. Even wired communications systems must pay attention to heat, power density, and low-power requirements. Figure 1.1 illustrates the power density trend versus power design requirements for modern SoCs.

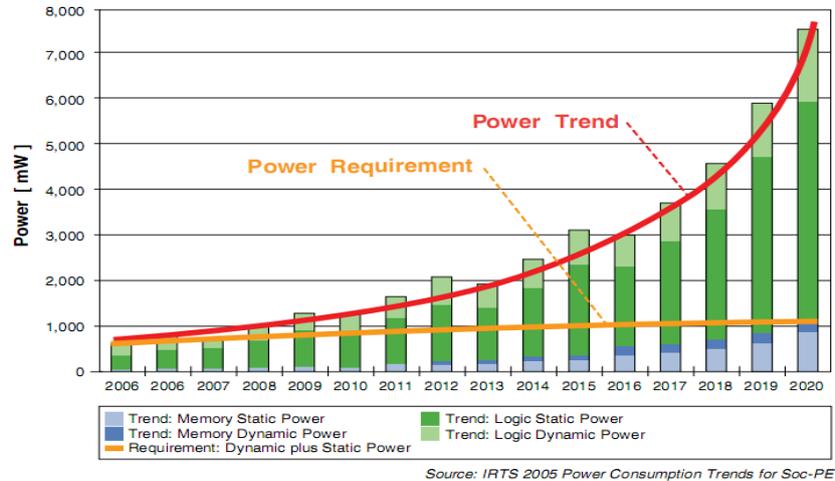


Figure 1.1: IC Power Trends According to The International Technology Roadmap for Semiconductors (ITRS) (Source: Silicon Integration Initiative (Si2), derived from ITRS 2005 Power Consumption Trends for SoC-PE)

As depicts Figure 1.1, the widening gap represents the most critical challenge faced today. To address this challenge, SoC designers are moving from the traditional monolithic approach, where a single supply voltage is used for all the internal gates of a design, to a multiple supply architecture, where different blocks are run at different voltages, depending on their individual requirements. In some cases, designers are using voltage scaling techniques to change the supply voltage (and clock frequency) of a critical block depending on its workload. With this new approach of modern SoC design, different blocks have different performance objectives and constraints. The most basic form of this approach is to partition the internal logic of the chip into multiple voltage regions or power domains, each having its own supply net. Once having separate supplies, more efficient low power strategies can be applied. These include multi-voltage strategies for scaling voltage when full performance is not needed in specific blocks of a design, as well as power gating strategy where power domains are downright powered-down through dropping their supply voltage to zero. However, implementing these strategies presents certain challenges to designers. These include four major ones:

- **Design and verification of the power network and additional power management interfaces are required:** Depending on the applied low power strategies, the power network including power switches and supply nets must be appropriately defined.

Moreover, interfaces of each power domain must be carefully designed and verified. These interfaces include structural elements such as level shifters and isolation cells which are required for the safe inter-power domain communication. Each of these interfaces as well as the power network has to be managed in a specific order. For that, a signaling control interface between each power domain and its power controller is added. Controls provided by power controllers define the power-aware behavior of a multi-power domain chip and depend on the power infrastructure choice. Such a choice can complicate the chip power routing and lead not only to higher silicon area (i.e. cost) of the end-product, but also to complex power controllers. Unquestionably, a significant complexity in the verification process is introduced including the power network integrity, the connectivity between power domains and the power control sequencers.

- **Power state design space explosion increases:** The supply network of a chip helps defining the power states set of each power domain. Today's systems-on-chip are large and support a high number of embedded software applications. So, they have various software states such that each state refers to a specific application workload. As a consequence, many power domains are then required in order to match all the potential application workloads demanded by the end-user. Because defining power domains boundaries is so closely tied to power requirements of the different embedded application workloads, the high number of software states in a chip makes the partitioning into power domains harder. In addition, such a high number of power domains implies an increasing number of power states that complicates even more power verification.
- **Tradeoffs between reuse and energy efficiency must be considered:** Depending on the required software workloads, power domains states are adjusted. Changes in power states almost incur an energy penalty that can impact the achievable energy savings. In general, power gating adds significant time delays to safely enter and exit power gated modes. Therefore, turning on and off a power domain frequently in time can waste more energy in reloading state than that saved when power gated. Taking into account the software power requirements, the designer must hence define and partition the low power design such that the power management infrastructure allows a high energy savings. However, such an infrastructure must also be designed to give a reasonable tradeoff between its reuse and its energy efficiency. Indeed, as a low-power design remains unchanged once implemented, it must remain as much energy-efficient as possible when running new applications with new power requirements on the same chip.

- **Correlation between power and functional dependencies must be carefully handled:** On the one side, a power management infrastructure can create structural power dependencies between power domains states. On the other side, functions and states of some hardware blocks may require specific states or functions of other blocks. This creates functional dependencies between power domains states. Therefore, a low power design must take into account these possible functional dependencies. Moreover, a power management policy must respect both functional and structural dependencies. In other words, a power and functional managed final system has to combine both dependencies such that no conflicts between them can occur. This constraint must be taken into account early in the low power design flow. It has to be integrated into the final system verification task as well.

Complexity added when designing and verifying low power SoCs is a common issue raised by this set of challenges. In addition, these challenges commonly invoke a strong relationship between functional and power concerns in a SoC. Understanding this relationship helps taking efficient low power management decisions and reaching the goals of these different challenges. Face to the challenges of achieving the most energy-efficient and the least erroneous design, preserving low power design reuse and modularity and handling power state explosion, the following critical questions are still unanswered: *What is the power management policy and architecture to apply? What are the low power strategies to use and on which sections of the chip they must be applied?*

1.1.2 Low Power Abstraction

So far, most of the power optimization effort has been focused at the low levels of the design flow (the register, gate, or layout levels). However, operating at the gate-level netlist to add low power management components and behaviors implies slow simulation times and difficulties for debugging and problem resolution. Therefore, low power specifications starting from the Register Transfer Level (RTL) ensure that correct power management components are implemented at the RTL, inferred correctly during synthesis, and placed-and-routed efficiently and accurately in the physical design. This requires a single power format accepted by all the tools in the flow at any given abstraction level. Such a power format facilitates implementation, early validation and incremental refinements of low power designs while addressing reusability of functional specifications.

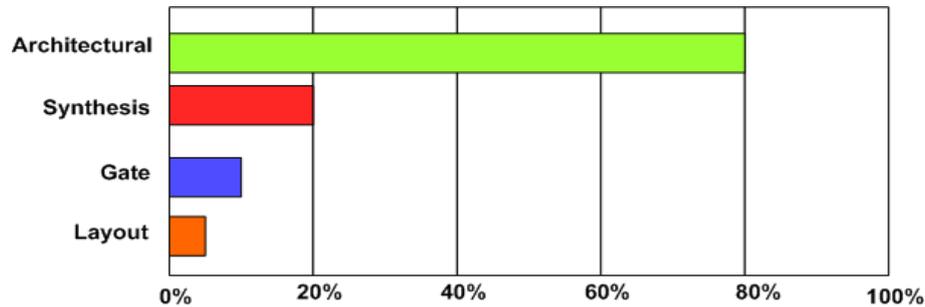


Figure 1.2: Power Optimization Opportunities at Each Level of Abstraction (Source: LSI Logic)

The Si2's Common Power Format (CPF) [29] and the IEEE 1801-2009 Accelera's standard [30], known as the Unified Power Format (UPF), define both a language format and simulation semantics for specifying how power is to be supplied, distributed, and dynamically managed in a low power digital system. These standards have moved the low power specification to register transfer level and provide the means for specifying the low power infrastructure and control information that are necessary to adapt the digital RTL according to low power requirements. These features are captured in a portable form for use in simulation, synthesis, and routing. Portability is enforced by the methodology which is used by these standards based on the separation of functional and power concerns. This is achieved by providing the low power specification in a side file separately from the functional specification code. Such a methodology has been used for various reasons. First, it does not require neither updating the RTL functional specification when power information is added nor re-verifying a module when its RTL code is changed. Second, a tight coupling between the design functionality and the low power design is not mandatory. Moreover, as significant aspects of the low power infrastructure are related to the technology implementation, it is usually modified more often than the RTL functional specification.

Unfortunately, by using these standard power formats, only functional specifications starting from RTL can be overlaid with power-aware semantics including structural power management elements and behavioral aspects. In order to apply these power-aware semantics at the Electronic System Level (ESL), they need to be abstracted and adapted to ESL models semantics. Actually, opportunities for optimizing a design for power efficiency are better at the ESL, when the architecture is being developed.

According to a LSI Logic study shown in Figure 1.2, the further a design moves downstream the less power optimization techniques could be applied. The Figure 1.2 shows that techniques available at the RTL synthesis phase have the ability to reduce power by 20 percent. Those at the gate level offer a 10 percent reduction, while those at the layout level can reduce power by only 5 percent. Waiting for the RTL code to start power optimization is a wasted opportunity because power usage can be reduced by 80 percent at the ESL. Power optimization must rather begin with architectural analysis, exploration, and optimization of power at the ESL. This abstraction level provides high simulation speed and simpler executable models, hence an easy and fast verification. Furthermore, it is very important to simulate the platform with the final application software in order to identify power optimization opportunities based on the system workload. As mentioned in the previous section, correlating power with the actual work performed by the system provides the largest opportunity for optimizing power. At the ESL, the final embedded software is available early in the design flow and can be rapidly validated on a reference hardware platform. For all these reasons, addressing low power management issues at the ESL helps achieving the various tradeoffs mentioned in the previous section while optimizing power significantly.

Transaction-Level virtual prototypes are one of the key ESL design methodologies. They have been initially developed to speed-up the validation of the embedded software. A transaction-level model [124] excludes some of the signal-level details of the digital model in order to focus on the system-level behavior. It uses the notion of transaction to model both units of communication among system components and units of computation within system components. Therefore, less effort is required to build a Transaction-Level model and this model is available far before the RTL in the design flow. To write Transaction-Level functional specifications, the SystemC TLM 2.0 standard [124] proposes coding styles and modeling mechanisms which enable the refinement of transaction-level models from untimed down to cycle-accurate communication. However, this standard still lacks semantics for power modeling and optimization, as well as coupling low power and functional design specifications at this level of modeling. In this context, lots of critical questions arise: *Which semantics of the existing low power standards need to be adopted and abstracted at the Transaction-Level of Modeling (TLM)? Are there any constraints or required standards extensions to apply power-aware simulation at Transaction-Level? What are the mechanisms needed to modify*

the design behavior in order to reflect the specified low power design state changes? To what extent the separation of power and functional concerns adopted by the low power standards can be applied in the TLM context? How a low power design which has been evaluated at the Transaction-Level can be reused in the rest of the downstream design flow steps?

1.2 Thesis

This thesis addresses and attempts to resolve questions raised in the previous sections. It consists in a complete study of power optimization opportunities based on composition and management of power domains in Transaction-Level models. This work uses the key term power domain to describe a group of functional blocks which shares the same supply network, hence has its own set of power modes and can be controlled individually. A dedicated focus of this study has been to shift the low power abstraction level to the Transaction-Level of Modeling. As a consequence, the relevant semantics which are defined by the existing power format standards across simulation and verification have also been shifted to this level of abstraction.

Another concern of this study has been to explore relationships between non-power aware functionality and power-aware one. Unquestionably, a low-power design behavior may impact the original system functionality due to incoherence between both designs. In spite of this close relationship, we propose throughout this thesis solutions to extend Transaction-Level functional specifications with power-aware semantics, including specification, behavior and constraints. As Transaction-Level models are first developed to validate embedded software, added power-aware features should be enabled only for power analysis purposes, otherwise disabled. Hence, a separation of concerns methodology used by the power format standards is required at this level of abstraction.

A second type of relationships between non-power aware functionality and power-aware one consists in activity-based interactions between power domains. As a functional block in a power domain can interact with a block in another power domain, transactions at power domains boundaries may incur or require a change in a sub-system power state. Such transactions represent power-aware interactions and must be carefully analyzed. Typically, a power-aware system includes a power management unit in charge of managing

the power domains states based on a specific power management strategy. Capturing power domains interactions is helpful for such a specialized unit to take good power management decisions dynamically.

Actually, analyzing relationships between non-power aware functionality and power-aware one at the Transaction-Level helps exploring both an energy-efficient architecture and a power domain management policy for a given low power design. In order to facilitate and accelerate exploration, a common and generic power domain management interface is required. So, the power management architecture can be implemented independently from the domains and the low power infrastructure. In other words, the choice of the power management unit architecture and strategy should not require the redesign of power domains. Similarly, modifying the low power design should neither constrain the structure nor the behavior of the power management unit. In this work, we extend the scope of TLM standard to create a simulation model for power domains management protocol interface.

In order to handle the power state space explosion problem and reduce the effort of the power management unit modeling and verification, we believe that a distributed power domain management structure would be more reliable than a huge centralized one. Moreover, large systems-on-chip usually include sub-systems provided with their own power controllers. In a hierarchical power domain management structure, each of these power controllers represents a local power management unit that handles power domains states of the underlying sub-system under the control of a global power management unit. A hierarchical organisation of SoC power management units requires a careful synchronization handling between the local power management units and the global one with respect to dependencies among power domains states. These requirements need to be taken into account by the proposed power domain management protocol interface.

To the best of our knowledge, this work is the first complete study on the subject of low power design and verification at Transaction-Level. In a low power design, the main goal is to minimize power consumption while still meeting performance requirements. So, designing a low-power system starting from the Transaction-Level aims first at an early and rapid decision for the most energy-efficient low power infrastructure as well as the most energy-efficient power management architecture and strategy for a given functional system. The result is a golden description of an energy-efficient and pre-verified

low power design including power infrastructure as well as power domain management strategy, architecture and protocol interface. Such a description can be used as a reference specification by RTL design teams and even be an input for RTL tools when the TL design is refined to RTL.

1.3 Overview of Thesis

1.3.1 Contributions

A main contribution of this thesis concerns a study on low power design concepts for a functional Transaction-Level model within a common framework, called USLPAF. USLPAF, referring to the Unified System-Level Power-Aware Framework, provides an effective Unified System-Level Power-Aware Methodology (USLPAM) that combines design and verification of Transaction-Level low-power models within a unified design flow. USLPAF provides also a Unified System-Level Power-Aware Library (USLPAL) including a set of modeling techniques and utilities that enable many built-in features for easily and rapidly apply the USLPAM methodology. Based on this framework, this work contributes to:

- **A Unified System-Level Power-Aware Methodology**

This methodology allows adding low power design and management capabilities to functional Transaction-Level models in a well-structured manner. A simulation-based power-aware verification process incorporates also the proposed methodology flow. The simulation and verification semantics as well as the separation of concerns methodology defined by the Unified Power Format (UPF) standard have been used as a support by our USLPAM methodology. The main goal of this methodology is to enable early exploration of different low power design and management alternatives to evaluate the effects of low-power techniques on system performance and functionality.

The USLPAM methodology ensures the connection to the RTL low power design flow by providing the most energy-efficient pre-verified power domain management solution composed of an RTL-based UPF specification and a reference model for the corresponding power management strategy and structure.

- **Assertion-Based Contracts for Power-Aware Verification**

Multi-power domain management deeply impacts and complicates SoC functional veri-

fication. A power-aware verification process has been defined throughout the USLPAM methodology flow to check for a set of power-aware properties in a predetermined order. We assume that the initial Transaction-Level functional model is valid and that its correct behavior is ensured. Thus, the defined power-aware properties in this work are related to the low-power structure and its effects on the normal operation of the initial model. These properties are defined to fit the Transaction-Level abstraction modeling. Some of them are derived from the UPF standard specifications while others are deduced from interactions between functional and low-power models. The Design by Contract (DbC) principle is used to identify power-aware properties and classify them into classes of contracts. Contracts checking is performed using assertion expressions added in the SystemC TLM model of the system.

- **A Method for Power Management Points Identification**

Locations in a Transaction-Level functional model where a change in a system power state can occur are called Power Management Points (PMPs). Determining PMPs relies on how the application software utilizes the hardware and how power consumption is impacted. It consists in the first step in the USLPAM methodology flow towards establishing a coherent and efficient power management solution. According to the identified PMPs, a low-power infrastructure is specified, a power management strategy is decided and specific power-aware properties are added into the SystemC code in the form of assertions. In this work, different types of Power Management Points (PMPs) are defined. In addition, a method for specifying alternatives of these points based on a given SystemC TL model description is proposed. This method allows the conversion of a functional Transaction-Level model to a form more suitable for low power management and validation.

- **A Source Code Instrumentation Method for the USLPAM Application**

Transaction-Level virtual prototypes are generally constructed through assembling SystemC Transaction-Level (TL) Intellectual Property (IP) cores. These cores can be either white-box IPs with accessible source codes or black-box ones already pre-designed, pre-compiled and pre-verified. Having access to white-box IP models does not constrain any step in the USLPAM methodology flow and even gives larger power reduction opportunities. We demonstrate how this is achievable through instrumenting the source code of a white-box virtual platform with required low power management information. Such an instrumentation-based method relies on using the PwARCH utility of the USLPAL library. Having as a main goal an early and rapid exploration, PwARCH eases each step

throughout the methodology flow. In particular, PwARCH allows a UPF-like low-power design specification where system power state changes are performed via calls to specific PwARCH library functions.

- **A Power-Aware Layering Method for the USLPAM Application**

The set of power management points obtained with a white-box virtual platform may be different with a same platform including black-box IP cores instead. This is mainly due to the limited observability of internal state changes of a black-box IP. The major constraints of the USLPAM application on this kind of TL platforms consist in specification and behavior simulation of state retention mechanisms as well as power-aware contracts checking. A novel method that handles these constraints is proposed as an alternative of the source code instrumentation. This method is based on layering the power-aware simulation and verification capabilities on top of each black-box functional block. By building such power-aware layers, a UPF-like separation of concerns is performed. The use of the PAL utility provided by the USLPAL library helps customizing the required behavior of each power-aware layer. This eases the method application and enforces its modularity.

- **Separation of Power-Aware Communications from Functional Communications for Transaction Level Models**

Adding power-aware capabilities to an existing functional TL simulation platform is the starting point of our USLPAM methodology. For that, both white-box and black-box methods adopt the separation of concerns methodology defined by the existing low-power format standards. However, power-aware communications including messages for power domains state management still depend on two factors: the specified low power infrastructure and the power management architecture and strategy. Since an adaptation of the low power management structure is required as long as one of these factors changes, such a dependency slows down the exploration of low power management solutions. Moreover, unlike functional communications based on read and write transactions to memory and block registers, power-aware communications need additional semantics and synchronization mechanisms. They also occur between power domains which are mainly groups of functional components with common low power features.

A contribution of this work is a new modeling technique that separates power-aware communications from functional ones. At the heart of this modeling technique is the specification of a new power domain management protocol interface that unifies commu-

nications between power domains independently of the power management architecture, strategy and low power infrastructure. The basic and generic features of this interface, called PDMgIF, represent the USLPACom utilities part of the USLPAL library. The coupling between the initial design functionality and the low power based activity is also ensured through adding power-related modeling details while preserving separation of concerns.

In order to reduce modeling and verification complexity implied by a single centralized power domain management unit, this PDMgIF interface can be used to construct a hierarchical architecture of power domain management units. In the general case, such a structure allows divide and conquer principle use. Indeed, it represents a good solution to reduce modeling and verification complexity implied by a single centralized power domain management structure. Nevertheless, a hierarchical power domain control requires a careful handling of interactions between local power management units and the global one, as well as dependencies between power domains. In this context, we discuss the scalability of the PDMgIF interface protocol in terms of complex and hierarchically organized power domain managers handling. We also suggest extensions of this protocol to best handle interactions between distributed power domain managers at different levels of hierarchy.

All the modeling techniques proposed in this document have been designed and validated using experiments with corresponding Transaction-Level simulation models.

1.3.2 Outline

Chapter 2 starts with a presentation of the different challenges in high-level modeling of low power Systems-on-Chip. Throughout this chapter, a background on low power design techniques, and Transaction-Level Modeling as well as, a bibliography on power modeling at the Electronic System Level (ESL) and low power design standards use is given.

Chapter 3 addresses the need for a common framework for low power design and verification at Transaction-Level and exposes related modeling issues at this level of abstraction. Objectives, key features and composition of our proposed Unified System-Level Power-Aware Framework (USLPAF) for building Transaction-Level low-power System-On-Chip models are then introduced.

Chapter 4 presents the Unified System-Level Power-Aware Methodology (USLPAM) flow

and requirements. It also details the contract-based verification process incorporated in this flow and gives examples of contracts involved in this checking process.

Chapter 5 addresses the problem of state retention simulation at Transaction-Level of Modeling and explains the method proposed to identify Power Management Points (PMPs) based on a Transaction-Level model behavior. It also outlines the utility of these PMPs in identifying locations in the functional SystemC/TLM user code where power-aware contracts must be added.

Chapter 6 covers the main utilities of the Unified System-Level Power-Aware Library (USLPAL) used to ease the USLPAM methodology implementation on the different types of TL virtual prototypes. First, it presents the source code instrumentation method proposed for the USLPAM methodology application on white-box IPs. It explains the main features of the PwARCH utility provided by the USLPAL library to ease this method implementation. Second, it presents the wrapper-based method proposed for the USLPAM methodology application on black-box IPs. It explains the main features of the PAL utility provided by the USLPAL library to ease this method implementation. Finally, it presents the main utilities of the USLPAL library which are built on top of the USLPAM methodology to enable Unified System-Level Power-Aware Communications (USLPACom). In this context, it addresses the need for a common and adaptive Transaction-Level power domain management protocol interface. A modeling approach that manages the separation of functional and power communications is presented along with a new PDMgIF protocol interface specification. This chapter also explains the methodology used to model the PDMgIF protocol interface at the Transaction-Level and discusses the hierarchical composition and management of power domains.

Finally, **chapter 7** concludes this dissertation and identifies areas for future works.

Chapter 2

High Level Modeling of Low Power Systems-on-Chip Design: Background & State of Art

2.1	Background	29
2.1.1	System-on-Chip Design Flow	29
2.1.2	Model Driven Engineering (MDE): Basic Concepts	34
2.1.3	Transaction-Level of Modeling Key Concepts	36
2.1.4	The TLM 2.0 OSCI Standard	41
2.1.5	Power reduction in Systems-on-Chip	49
2.1.6	Low Power Design Standards	67
2.2	State of Art	74
2.2.1	State-of-The-Art on High Level Power Modeling, Reduction and Analysis	74
2.2.2	State-of-The-Art on Low Power Design Standards Use	85

THIS chapter introduces the basic concepts and notions required for the understanding of this thesis. It also presents and discusses, on the one hand state-of-the-art methods and tools that have used low power design standards as well as those that have targeted power modeling, reduction and analysis at the Electronic System Level (ESL) and on the other hand state-of-the-art works related to the low power design industry-standards use.

2.1 Background

2.1.1 System-on-Chip Design Flow

The current System-on-Chip (SoC) design flow typically relies on six different levels of abstraction. As depicts Figure 2.1, these levels range from the Algorithmic Level (AL), which is classified as the most abstract and less precise level, to the layout level, which conversely presents the most precise and realistic model. Throughout this flow, a model of the same chip with less or more details is provided at each abstraction level.

This dissertation study focus on the Electronic System Level (ESL) which gathers different levels of abstraction above the Register Transfer Level (RTL) as depicts Figure 2.1. ESL is the most adapted level to create a behavioral description of the system and play "what-if" games with system partitioning (parts will ultimately be implemented either in hardware or in software). Indeed, the relative absence of implementation details at ESL enables simulations to run significantly faster than they would at RTL and downstream design stages, enabling the design team to quickly evaluate a large number of implementation alternatives. In the following, main features of each level in the SoC design flow are briefly described.

2.1.1.1 Algorithmic Level (AL)

At this level, the application is described in an algorithmic form based on a standard specification or an existing documentation. Models at this level are described using a high-level description language such as Matlab, C or C++. They are then functionally analyzed and checked in order to efficiently partition the application into hardware and software tasks.

In addition, model based engineering techniques (e.g. Mealy/Moore machines and meta-modeling) and languages (e.g. the Unified Modeling Language (UML)[19] and Architecture Analysis and Design Language (AADL)[126]) are widely used at the algorithmic level in order to specify and validate an application. As depicts Figure 2.1, these techniques and languages are also used to model both the software and hardware parts after the HW/SW partitioning phase. The full system functionality is then validated by running both models together.

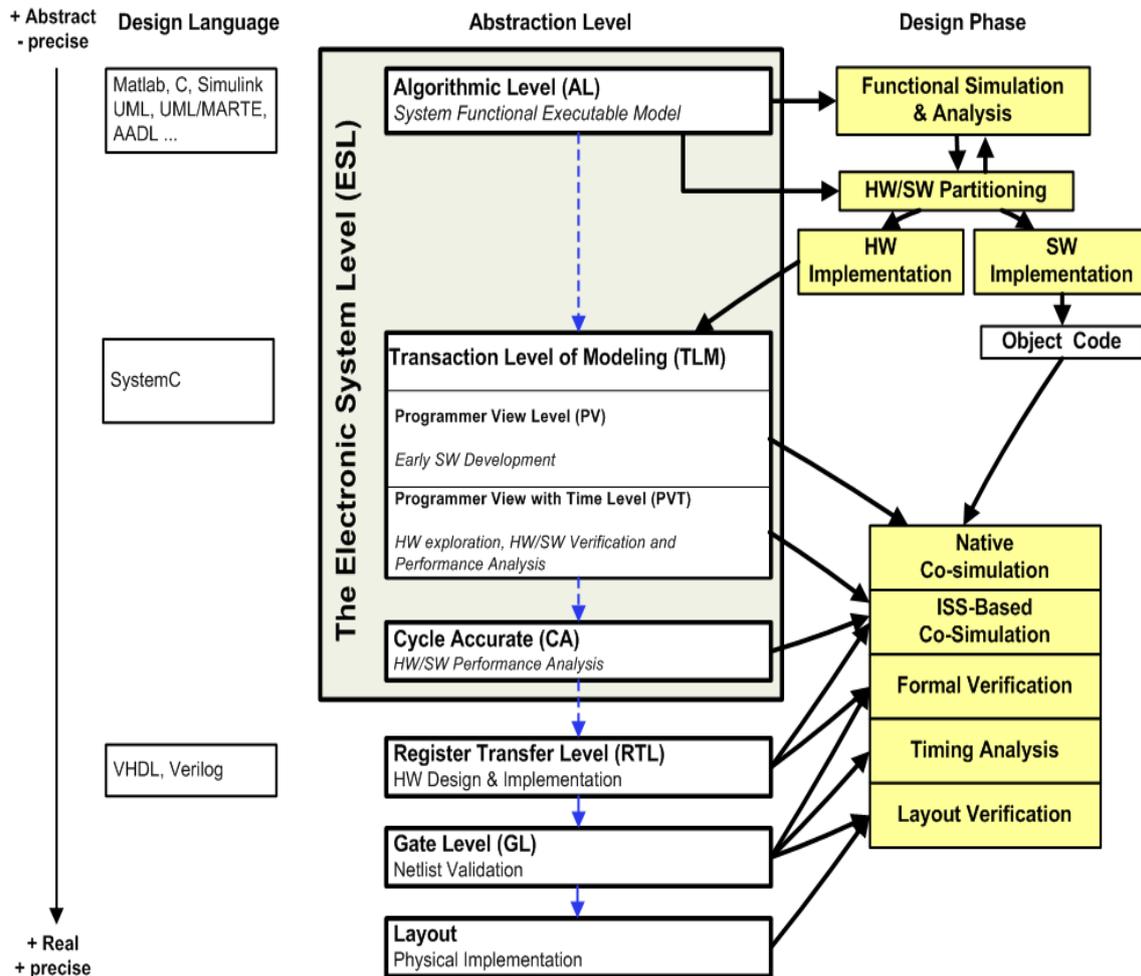


Figure 2.1: Typical SoC Design Flow Phases and Abstraction Levels

2.1.1.2 Transaction Level of Modeling (TLM)

TLM is a high-level modeling approach founded on high-level programming languages such as SystemC [92] to describe a virtual prototype (VP) of the hardware design part. Several definitions and classifications of various TLM levels have been presented in the literature [74] [55] [77] [68] [87]. This proves the lack of common understanding on the definition of what a TL model precisely is. Nevertheless, all these proposals have the following points in common. First, a transaction in TLM context refers to the exchange or synchronization of structures of data and/or control information between two components [54] [81]. Transactions passing is simplified by using specific methods of communication called via channels [32]. Second, communication and computation aspects are separated.

Finally, TLM is presented as a taxonomy of several sub-levels.

As depicts Figure 2.1, we adopt the Programmer View (PV) and the Programmer View with Time (PVT) levels as a classification of TLM levels since these levels are inline with the type of models interesting us in this work. These two levels take into account the system architecture. The main difference between them is on timing accuracy.

- **Programmer View (PV) Level:**

The PV level has no timing information, but enough synchronization to enable correct functionality. Therefore, it is mainly used to early validate the full system by executing the final software application on the Transaction-Level architecture model. Non-functional properties such as execution (or computing) time and power consumption are either omitted or coarse-grained approximated. Some architectural details such as the bus arbitration and the cache activity are not modeled at the PV level. Such details have a great influence on the accuracy of the simulation model and performance measurements but they slow down the simulation. This would constrain a rapid SW functional validation targeted by the PV level. These details are rather modeled at the PVT level.

- **Programmer View with Time (PVT) Level:**

The PVT level is the same as PV in functionality, but with timing added. It is often referred to the PV version with timing annotations. Operations will take the correct number of clock cycles to execute, although within atomic operations not every clock tick needs to be considered. At this level, different architecture details are added for both processing and communication parts. The interconnect is said to be fully fixed and modeled, and some arbitration of the communication is applied. Therefore, it is more precise than the PV level and rather adopted for early design stages performance evaluation including design architecture exploration and verification.

Once HW/SW partitioning is performed at the Algorithmic-Level, both HW and SW parts can be rapidly and easily integrated together and validated at TLM. A TL hardware model describing sufficiently the functionality is first developed so that the software team can use it as a development platform for the final embedded software. The final embedded system would have the same behavior as the simulation TL model (composed of the embedded software and hardware models developed at TLM) if the TL virtual prototype is faithful to the final hardware platform.

As depicts Figure 2.1, to execute the embedded software on the TL hardware virtual prototype, engineers in the industry rely on two approaches: **native wrappers** and

ISS-based simulation. The first approach relies on wrapping the embedded software into a piece of code in order to intercept the communications of the software with the hardware components. The wrapper is part of the processor model. It offers to the software developer primitives related to the communication with the hardware. To simulate the whole system, the wrapped software code together with the hardware virtual prototype are compiled into a binary code that is directly executed by the host machine.

On the other hand, an Instruction Set Simulator (ISS) emulates the behavior of a specific processor when executing a binary code. An ISS is able to interpret the complete instruction set of a processor, and to maintain a set of variables that corresponds to the registers of the processor. Contrary to native wrapper simulation, ISS-based simulation requires the software to be cross-compiled into the binary code of processor of the hardware platform. The resulting binary code, is given as input to the model of the CPU in the virtual prototype. The CPU model is in fact the ISS. During the simulation, the model of the hardware (including the ISS) is executed by the host machine. The ISS interprets the binary code of the software and reflects its behavior on the hardware model. On the other hand, an Instruction Set Simulator (ISS) emulates the behavior of a specific processor when executing a binary code. An ISS is able to interpret the complete instruction set of the processor, and maintains a set of variables that corresponds to the registers of the processor. Contrary to native wrapper simulation, ISS-based simulation requires the software to be cross-compiled into the binary code of processor of the hardware platform. The resulting binary code, is given as input to the model of the CPU in the virtual prototype. The CPU model is in fact the ISS. During the simulation, the model of the hardware (including the ISS) is executed by the host machine. The ISS interprets the binary code of the software and reflects its behavior on the hardware model.

Moving from an Algorithmic-Level description to a Transaction-Level description as well as from Transaction-Level to downstream stages of the design flow (typically to Cycle-Accurate or Register Transfer Level) is usually done manually as illustrated by Figure 2.1. Nevertheless, few ad-hoc tools and methods exist for the automation of translation between these levels. For instance, to automatically move from the AL to TL, Model-Driven Engineering based code generation approaches have been recently proposed [103] [99]. Few EDA tools and interfaces such as CatapultC and SystemC Studio have also enabled the translation of SystemC Transaction Level Synthesis. But, their generated output files still almost need an additional effort to manually add missing behavioral code

and validate it.

A more detailed presentation of the SystemC/TLM modeling principles is developed in section 2.1.3.

2.1.1.3 Cycle Accurate Level (CAL)

At Cycle-Accurate level, the model is described precisely from the execution time point of view. A CA component behavior is sensitive to whatever happens at the interval of each clock cycle. Its bounds are the same wires as RTL and exhibit the same value at each clock cycle. The internals of the component are left free to the designer. A CA internal behavior generally implements and computes the various outputs depending on the current and past inputs using standard programming language (such as C or SystemC). At the processing level, a description of the internal micro-architecture of the processor (pipeline, branch prediction, cache ...) is performed, whereas at the communication level, a precise bit-accurate communication protocol is adopted.

Such CA models precision improves the accuracy of early performance estimation. However, these models exhibit a limited speed (generally one order of magnitude faster than the Register Transfer Level (RTL)) and require a significant modeling effort while they do not provide any synthesizable description.

2.1.1.4 Register Transfer Level (RTL)

At RTL, the physical implementation of a system is described using registers and a data-flow description of the transfers between them. Still, each wire is represented, but its precise value is known only at each clock tick. Hardware description languages (HDL) such as VHDL, SystemVerilog, or Verilog are used for writing models at this level. The translation to gate level is done by EDA synthesis tools that allow automatic optimization of the circuit with respect to its surface, power and timing.

2.1.1.5 Gate Level (GL)

The gate level abstracts away a lot of details by focusing only on describing the logical gates (AND, OR, flip-flops ...) and their connections. Such a description forms the output

netlist from the RTL synthesis tool usually encoded using Verilog gate level primitives and is then imported into a place and route EDA tool.

2.1.1.6 Layout

This is the most precise description of a chip in which the location and design of each transistor is precisely known and carefully checked. When verification is complete, the data is translated into an industry standard format, typically GDSII, and sent to a semiconductor foundry. Ultimately, the foundry converts the data patterns representing the transistors and their connections into so-called masks. Modern IC layout is done with the aid of IC layout editor software, mostly automatically using EDA tools, including place and route tools or schematic driven layout tools.

As masks are very costly (estimated that a set of masks at 65 nm technology can cost up to 3 Millions dollars), design errors in the hardware can prove economically disastrous to fix because of the need for rebuilding the masks.

2.1.2 Model Driven Engineering (MDE): Basic Concepts

In general, the MDE methodology is based on three main strongly related concepts: meta-models, models and model transformations.

- **Meta-models:** a meta-model reflects the domain concepts and relationship between them and is defined using a model description language such as the Object Management Group's (OMG) Unified Modeling Language (UML). It allows designers to specify their own domain-specific languages in which models can be instantiated.
- **Models:** a model is defined according to a specific meta-model to which it conforms, hence representing an instance of it. It can be observed from different abstract points of view (views in MDE). The abstraction mechanism avoids dealing with details and separating concerns in different reusable models.
- **Model Transformations:** a model transformation (MT) [131] is a compilation process that allows moving from an abstract model to a more detailed target model containing additional implementation information as illustrated by Figure 2.2. A MT is based on a set of transformation rules that help to identify concepts in a source model in order to create enriched concepts in the target model. Each MT is performed using a transforma-

tion engine based on a source model and transformation specification rules to generate a target model as shown in Figure 2.2. A key characteristic in MDE approaches is that the specified transformation rules can be modified or extended allowing the definition of a new MT targeting a different model. Hence, several MTs can be defined based on the same high-level abstraction model but generating different target models.

Model transformations can be either **unidirectional** or **bidirectional**. For unidirectional MTs, only source model can be modified and target model is regenerated automatically. For bidirectional MTs, target model is also modifiable requiring the source model to be modified in a synchronized way and possibly leading to a model synchronization issue [139].

Additionally, two basic techniques of model transformations can be distinguished: **Model-to-Model (M2M)** and **Model-to-Text (M2T)**. The distinction between the two categories is that, while a model-to-model transformation creates its target as an instance of the target metamodel, the target of a model-to-text transformation is just strings. In M2M transformation, Czarnecki et al. [66] define direct-manipulation approaches, relational approaches, graph-transformation-based approaches, structure-driven approaches, hybrid approaches and some other M2M approaches. In the M2T category, Czarnecki et al. [66] define visitor-based and template-based approaches which are useful for generating both code and non-code artifacts such as documents.

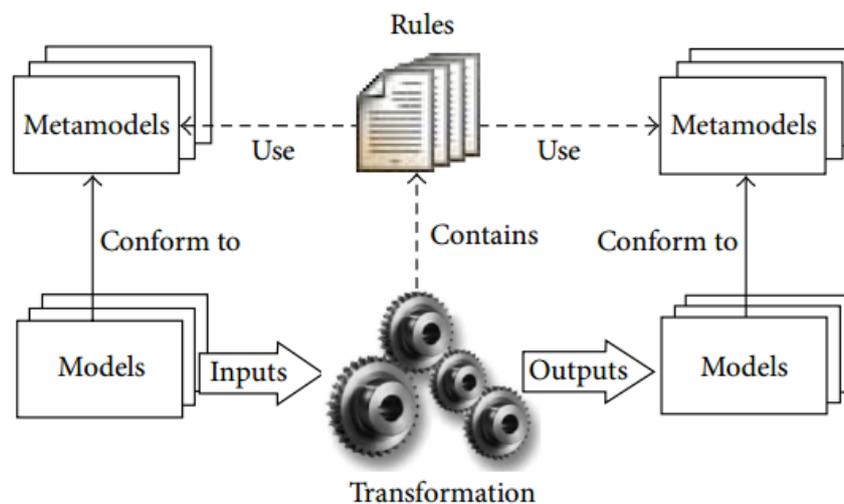


Figure 2.2: Model Transformation Process [143]

On the one side, many different kinds of transformation languages exist to express M2M transformations such as graph transformation languages like MOLA [95] or languages based on the OMG standard Query/View/Transformation (QVT) [16]. QVT principles have been implemented in several languages, such as the object-oriented language Kermeta or the declarative language ATL (ATLAS Transformation Language [35]) that is currently the most widely used.

On the other side, M2T transformations rely either on graphical languages based on existing parsers like TrML XML XSLT-based languages, or on languages based on a programming language (for instance, JMI that expresses Java-like transformations or the Java API of the Eclipse Modeling Framework (EMF)), or on transformation templates such as the JET component or the ACCELEO code generation tool used by EMF. A template usually consists of target text containing splices of meta-code to access information from the source and to perform code selection and iterative expansion.

2.1.3 Transaction-Level of Modeling Key Concepts

2.1.3.1 TLM Common Concepts

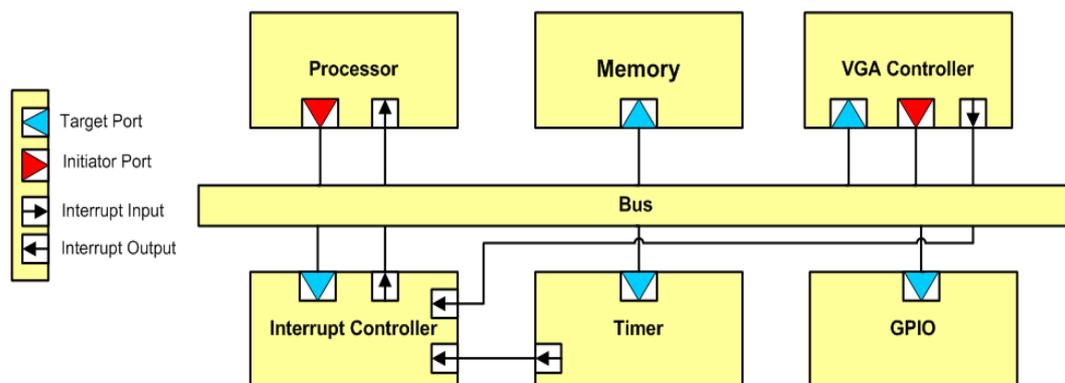


Figure 2.3: Example TLM Platform

Figure 2.3 represents an example of a TLM platform. The platform is composed of different components linked by connections between typed ports. Some of these components play the role of communication channel, that is they are directly involved in the communication between other components. For instance, this is the case of the bus model in Figure 2.3.

Depending on their input and output TLM interface ports, three types of components can be involved in a TLM communication.

An initiator (master) component can initiate transactions through an **initiator port** (e.g. the processor component in Figure 2.3 is an initiator component).

A target (slave) component can receive transactions through a **target port** (e.g. the GPIO component in Figure 2.3 is a target component).

An initiator and target (master/Slave) component has at the same time initiator and target ports as interface. This is the case of the VGA controller in Figure 2.3 which receives controls from the processor through its target port and does data transfers to memory using its initiator port.

Transactions transmitted from an initiator to a target component pass through a channel component that routes them to their final destination depending on their address and according to its defined bus protocol rules. For that, a channel component uses a **global memory address map** which associates a memory range to each target port. For each memory range, the relative address allows accessing the local memory space for memories or registers of the hardware block. Figure 2.4 shows an example of memory map corresponding to the platform of Figure 2.3.

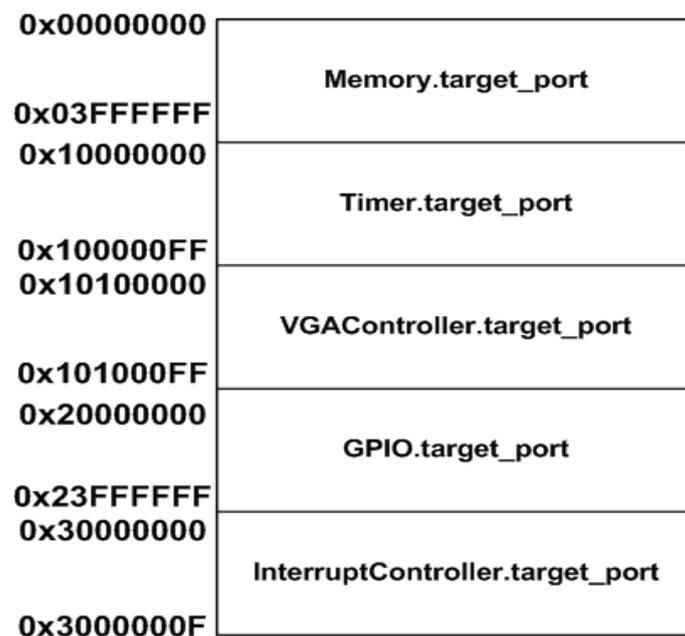


Figure 2.4: Example Memory Address Map

Information exchanged via a transaction depends on the bus protocol. Commonly, such information include the **type** of the transaction, its destination **address**, **data** communicated to the target as well as a return status and transaction latency. The transaction type determines the intention of the transfer. It is generally intended to read or write from or to a register or internal component memory. The address defines the register or memory location of the target component.

Only **memory-mapped** registers can be accessed within a target component from an initiator component. By memory-mapped, we mean it has been assigned an address range in the address memory map. In general, memory-mapped registers correspond to status and control registers of target components, noted in the following **CSR** (**C**ontrol and **S**tatus **R**egisters). Component models can have additional internal registers (such as internal buffers) required for the internal behavior of the component but they are non-accessible from outside this component. We call these registers **non-memory mapped** as they do not have any entry in the global address memory-map.

In order to correctly execute the embedded software, the address map and the registers offset must be the same as in the final chip (**register accuracy**). Registers offsets and access types restrictions [7] (read-only, write-only, read-writeOnce ...) must also be declared in the component model and respected by the modeled behavior of this component. Moreover, the data produced and exchanged by the components must also be the same as in the final chip (**data accuracy**).

As it can be seen in Figure 2.3, the **interrupt** is another type of TLM communication. Interrupt refers to a unidirectional data exchange between components through a point-to-point connection. Classically, the term interrupt refers to a wire whose state changes to communicate an asynchronous event and does not require additional protocol signals. As depicts Figure 2.3, two kinds of ports for interrupts may exist: input and output ports. Interrupts modeled in the TL virtual prototype have to logically correspond to the ones used in the final chip.

2.1.3.2 TLM With SystemC

SystemC [92] is a C++ library that has been gaining a large popularity in the industry for modeling SoCs above RTL, from cycle accurate to purely functional models. This standard offers a set of primitives for the description of parallel activities representing the

physical parallelism of the hardware blocks. It also offers an entire simulation environment with a non-deterministic and non-preemptive scheduler allowing early simulation-based validation of a SoC model.

In this section, we present the main concepts and mechanisms provided by the SystemC standard to model TLM platforms. We were quite inspired by authors' point of view in [62] and [50] on the general architecture and control flow of a SystemC TLM model. The different concepts presented in this section are relevant to this thesis and will help the reader to understand the EFSM-based approach presented in the 5.

Components of a TLM platform are modeled as SystemC modules that expose ports and represent some physical entities that behave in parallel upon the execution of the embedded software to reflect the final system expected functionality. Figure 2.5 depicts an example of a generic and simplified communication between two transaction-level initiator and target (master/slave) SystemC modules. The register structure of module 1 is composed of memory-mapped registers: two control registers, *CReg0* and *CReg1*, and one status register *SReg1*. This set of registers can be read or written from outside this module via bus transactions sent through its *p1* target port. Moreover, there is an internal register, called *internal_buffer*, which is non-memory mapped, hence that cannot be accessed from outside the module. The register structure of module 2 component is composed of only two memory-mapped registers accessed from outside via *p2* target port of module 2: a control register *CReg2* and a status register *SReg2*.

The behavior of a module is modeled by a set of threads that may execute concurrently (represented by curved lines in Figure 2.5) and a set of methods (represented by straight lines in Figure 2.5), both programmed in C++. Module 1 has two threads *T1* and *T2*, and one method *M1*, while module 2 has a single thread *T3* and two methods *M3* and *M4*. Threads are active code scheduled by the global SystemC scheduler while methods are passive code offered to other components, and called from a thread. Each method is attached to a target port of the module (e.g. *M1* is attached to *p1* in module 1, *M3* is attached to *p2* in module 2 and *M4* is attached to *p3* in module 2). Methods attached to target ports implement the read and write methods declared as pure virtual methods in the target port interface (i.e. abstract base class in C++).

Synchronization between behaviors of different modules as well as synchronization between internal processes of a single module is mainly ensured by SystemC events that

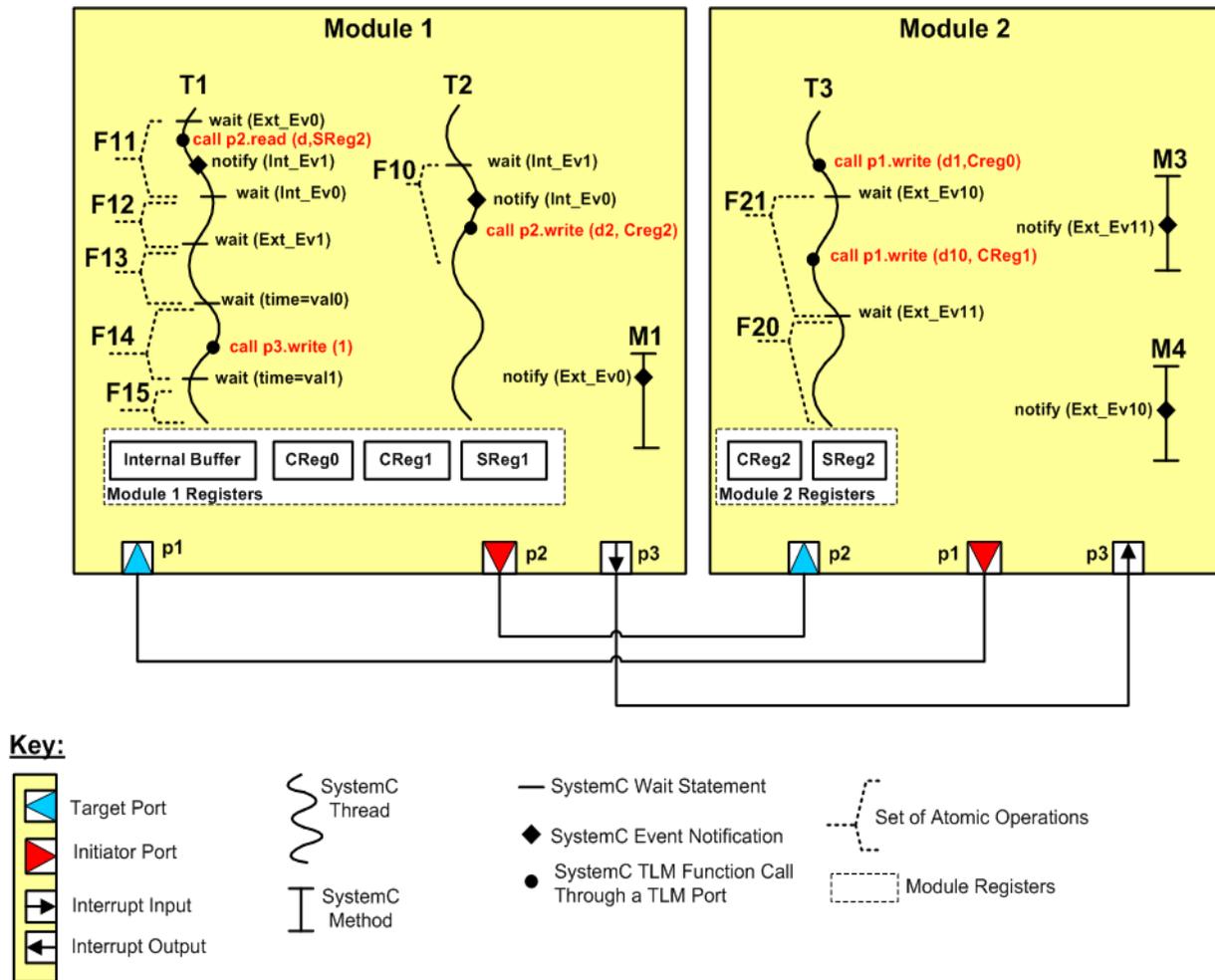


Figure 2.5: Example of SystemC/TLM Architecture and Communication

can be notified or waited for. We distinguish between **internal events** and **external ones**. Internal events, denoted by *Int_Evi*, are defined as events used to synchronize threads within an IP. External events, denoted by *Ext_Evi*, are defined as events used to synchronize behaviors of different components (the *i* subscript is only used for enumeration purposes). This kind of events is further notified to a communication from outside the IP (i.e. upon receiving a transaction or an interrupt via a target port or an input interrupt port).

All the threads are globally managed by the non-deterministic SystemC scheduler. As this scheduler is non-preemptive, a running thread has to yield back control to the scheduler by performing a wait either on an internal or an external event or on time.

The scheduler elects then another ready thread to run. For instance, the thread $T2$ of module 1 yields only on $wait(Int_Ev1)$ statement. The remaining code of $T2$, including the Int_Ev0 notification and writing to the $CReg2$ register of the module 2, is executed in an atomic (i.e. non-interruptible) way. In general, between two wait statements in a module's thread, there is a set of atomic operations denoted by Fi on Figure 2.5.

Communications between an initiator and a target is ensured via transactions. In Figure 2.5, the thread $T3$ of module 2 initiates a transaction on its port $p1$ ($p1.write(d1,CReg0)$ method). It writes data $d1$ into the control register $CReg0$ of module 1. As the initiator port $p1$ of module 2 is connected to the target port $p1$ of module 1, this is actually a call to the method $M1$ in module 1 (which is attached to the target port $p1$ of module 1). When the call is executed ($T3$ being running), the control flow is transferred to module 1. In $M1$, the implementation of the write method would notify Ext_Ev0 external event of module 1 which makes the thread $T1$ ready to execute. When $M1$ terminates, the control flow returns to module 2, and the execution continues until the next yielding point ($wait(Ext_Ev11)$ in the example). The scheduler would hence give execution control to the ready thread $T1$ of module 1. $T1$ executes until reaching the $wait(Int_Ev0)$ and the control flow is transferred to the $T2$ thread ready since $T1$ has notified the Int_Ev1 event. This is an example of internal processes synchronization inside a single module.

2.1.4 The TLM 2.0 OSCI Standard

2.1.4.1 The TLM 2.0 Modeling Features and Mechanisms

Due to the absence of standards, the different TLM approaches and proprietary solutions for TL virtual platforms were introduced by several companies. Therefore, a common standard that models interoperability and provides a high simulation speed was a necessity to maintain and grow a healthy TL virtual prototyping industry. In order to address this requirement, the OSCI TLM Working Group has developed the TLM 2.0 OSCI standard. This standard focuses mainly on on-chip memory-mapped buses modeling but offers extension mechanisms to model either memory-mapped or non-memory-mapped protocol-specific interconnects.

Figure 2.6 shows a diagram of how the TLM 2.0 classes are layered on top of the SystemC class library and include those of its former TLM 1.0 standard. Indeed, the

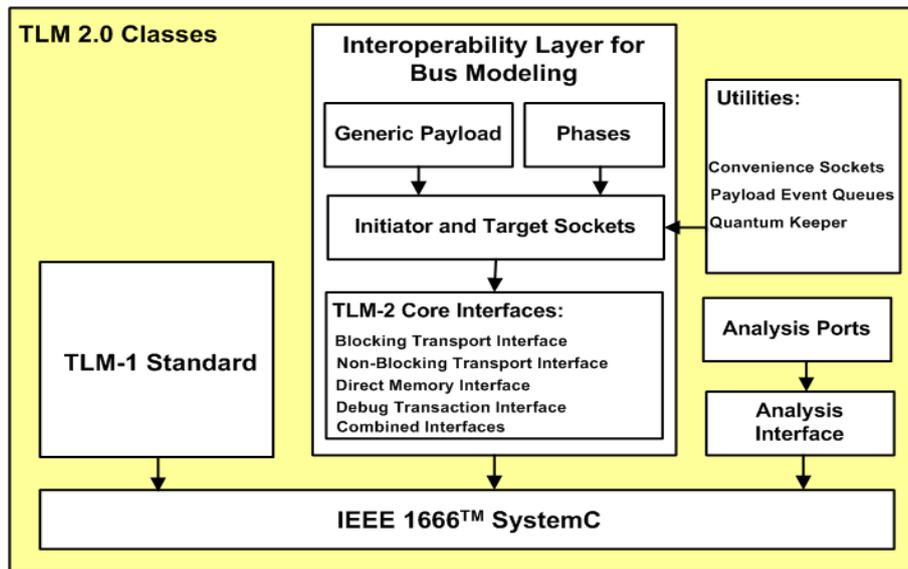


Figure 2.6: TLM 2.0 Overview [124]

OSCI TLM 2.0 standard have addressed several of the shortcomings of the TLM 1.0 standard with respect to busses modeling such as the absence of a standard transaction class as well as a standard way of communicating timing information between models. In addition to utility classes and analysis interfaces and ports, the TLM 2.0 layered structure involves an interoperability layer specific for bus modeling (Figure 2.6). This layer consists of the generic payload, the base protocol phases, initiator and target sockets and the TLM 2.0 core interfaces.

```
// Default tlm_generic_payload class constructor
tlm_generic_payload()
: m_address(0)
, m_command(TLM_IGNORE_COMMAND)
, m_data(0)
, m_length(0)
, m_response_status(TLM_INCOMPLETE_RESPONSE)
, m_dmi(false)
, m_byte_enable(0)
, m_byte_enable_length(0)
, m_streaming_width(0)
, m_extensions(max_num_extensions())
, m_num(0)
, m_ref_count(0)
{
}
```

Figure 2.7: The TLM 2.0 Default Transaction Fields

The generic payload is a transaction object that supports the modeling of simple abstract memory-mapped buses. The default transaction type for the socket classes, implied in the absence of any template arguments, is **tlm_generic_payload**. Each generic payload transaction instantiated from the `tlm_generic_payload` class has a standard general-purpose set of bus attributes: command, address data, byte enables, streaming width, and response status. Figure 2.7 depicts the default settings of a TLM 2.0 standard transaction's fields. It is worth mentioning that the generic payload command field supports only two commands, read and write. Therefore, transactions generated by an initiator component and passed through TLM 2.0 standard initiator socket will only read from or write to the components internal memory or memory-mapped registers.

The TLM 2.0 core interfaces involve blocking and non-blocking transport interfaces, a direct memory interface (DMI) and a debug transport interface. The transport interfaces are the main interfaces used to transport transactions between initiators, targets and interconnect components. Both the blocking and non-blocking transport interfaces support timing annotation and temporal decoupling.

A non-blocking transport call corresponds to either **nb_transport_fw**¹ method calls to transmit a transaction on the forward path from an initiator to a target, or to **nb_transport_bw**² method calls to transmit a transaction on the backward path from a target to an initiator. Only non-blocking transport interfaces support multiple phases within the lifetime of a transaction. Blocking transport calls correspond to **b_transport**³ method calls. They do not have an explicit phase argument.

The rules governing memory management of the transaction object (i.e. generic payload), transaction ordering, and the permitted function calling sequence depend on the specific transaction type passed as a template argument to the transport interface, which in turn depends on the protocol traits class passed as a template argument to the socket.

In order to ensure maximal interoperability between transaction level models of components that interface to memory-mapped buses, the TLM 2.0 standard defines a de-

¹`tlm::tlm_sync_enum nb_transport_fw (tlm::tlm_generic_payload trans, tlm::tlm_phase phase, sc_core::sc_time t)`

²`tlm::tlm_sync_enum nb_transport_bw (tlm::tlm_generic_payload trans, tlm::tlm_phase phase, sc_core::sc_time t)`

³`void b_transport (tlm::tlm_generic_payload trans, sc_time delay)`

```

namespace tlm {

enum tlm_phase_enum {
    UNINITIALIZED_PHASE=0, BEGIN_REQ=1, END_REQ, BEGIN_RESP, END_RESP };

class tlm_phase{
public:
    tlm_phase();
    tlm_phase( unsigned int );
    tlm_phase( const tlm_phase_enum& );
    tlm_phase& operator= ( const tlm_phase_enum& );
    operator unsigned int() const;
};

```

Figure 2.8: The TLM 2.0 tlm_phase Class

```

namespace tlm {

//The default protocol traits class
struct tlm_base_protocol_types
{
    typedef tlm_generic_payload tlm_payload_type;
    typedef tlm_phase          tlm_phase_type;
};

//The combined forward interface
Template<typename TYPES = tlm_base_protocol_types>

class tlm_fw_transport_if
    : public virtual tlm_fw_non_blocking_transport_if <typename TYPES::tlm_payload_type,
                                                    typename TYPES::tlm_phase_type>
    , public virtual tlm_blocking_transport_if <
        typename TYPES::tlm_payload_type>
    , public virtual tlm_fw_direct_mem_if<
        typename TYPES::tlm_payload_type>
    , public virtual tlm_transport_dbg_if<
        typename TYPES::tlm_payload_type>
    {};

//The combined backward interface
class tlm_bw_transport_if
    : public virtual tlm_bw_nonblocking_transport_if <typename TYPES::tlm_payload_type,
                                                    typename TYPES::tlm_phase_type>
    , public virtual tlm_bw_direct_mem_if
    {};

} //namespace tlm

```

Figure 2.9: A Combined Interface Definition

fault set of rules and transaction ordering for a basic and generic bus protocol called **the base protocol**. This protocol is represented by the pre-defined protocol traits class **tlm_base_protocol_types** that contains two type definitions: the default generic payload (`tlm_generic_payload` class) and the default phase types (`tlm_phase` class shown in Figure 2.8) used by the non-blocking transport interface class templates as well.

This protocol requires the use of the TLM-2.0 interoperability layer socket classes (which are **tlm_initiator_socket** class and **tlm_target_socket** class or classes derived from them) and parametrize the transport interfaces. As depicts Figure 2.9, templates of the combined forward and backward interface [124] grouping all the TLM 2.0 interfaces are parameterized with a protocol traits class that defines the types used by the forward and backward interfaces, namely the payload type and the phase type. Here, the protocol traits class is associated by default with the **tlm_base_protocol_types** class as shown in Figure 2.9.

The default initiator and target TLM2.0 sockets are templated on the base protocol (`tlm::tlm_base_protocol_types` class) as well and define the full sequence of phase transitions for a given transaction through each socket type.

2.1.4.2 The TLM 2.0 Coding Styles

The TLM 2.0 defines two main coding styles: loosely-timed (LT) and approximately-timed (AT). Each consists in a set of guidelines for using TLM 2.0 features to create models with a certain degree of communication timing accuracy and fitting a specific range of abstraction details.

- **The Loosely Timed (LT) Coding Style:** this coding style uses the blocking transport interface to perform the transactions that are being sent from an initiator module to a target module. Each transaction that is made through this interface has two timing points. The first timing point is the transport call from the initiator to the target and the second timing point is the return of the transport function from the target back to the initiator as depicts Figure 2.10. These timing points are typically associated with the beginning of the request and response phases of the transaction. According to LT coding style, a transaction is completely transmitted in a single call of the `nb_transport` method and its initiator is blocked until it receives the return from this method.

With these two timing points, the loosely timed coding style allows only modeling the

overall transaction latency (i.e. delay between the start and the end of a transaction). Given such limited timing details are sufficient to model simple timers and interrupts that are needed to boot an operating system and run software on a virtual platform model. In other words, the LT coding style coincides with the programmer's view (PV) TLM sub-level (shown in Figure 2.1) that is best suitable for software development and validation use cases.

- **The Approximately Timed (AT) Coding Style:** conversely to the LT coding style, the AT coding style adds more timing details to the transactions that are sent between the components of a system by using multiple timing points (phases) for each transaction. Therefore, it can be used for more detailed hardware architecture analysis, verification and performance analysis. So, when referring again to Figure 2.1, the AT coding style coincides with the programmer's view with time (PVT) TLM abstraction sub-level.

For the AT coding style, the TLM 2.0 non-blocking transport interface is used. The non-blocking transport interface differs from the blocking transport interface that is used in the LT coding style in several ways. First, each transaction transmitted through this interface is split in different sequences before it completes. Each sequence corresponds to a specific phase sent with the non-blocking transport call to indicate the current state of the transaction. Indeed, the base protocol for the AT TLM 2.0 coding style defines four timing points for each transaction, which mark the begin request phase, the end request phase, the begin response phase and the end response phase. Figure 2.11 shows the sequencing of the four base protocol phases while modeling the request and response accept delays and the latency of the target.

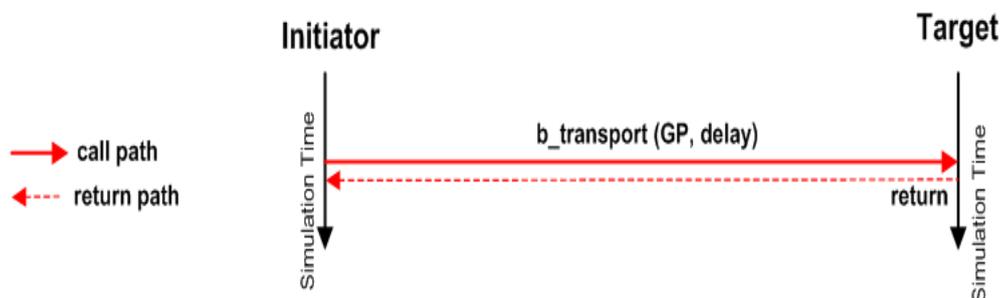


Figure 2.10: Message Sequence Chart of a Transaction Between Initiator and Target Using the Loosely-Timed Base Protocol

As it can be seen in Figure 2.11, the AT coding style also enables bi-directional communication through the non-blocking transport interface. There is a forward path for transfers from the initiator to the target and there is a backward path for transfers from the target to the initiator. Thus, each component can be at the same time an initiator and a target of the same transaction. The non-blocking transport interface is particularly suited for modeling pipelined transactions. In other words, the same module can initiate separate transactions through `nb_transport_fw` method calls without having to wait for the first transaction to complete. This second important feature of the AT TLM 2.0 standard coding style has been particularly exploited in Chapter 6 of this thesis.

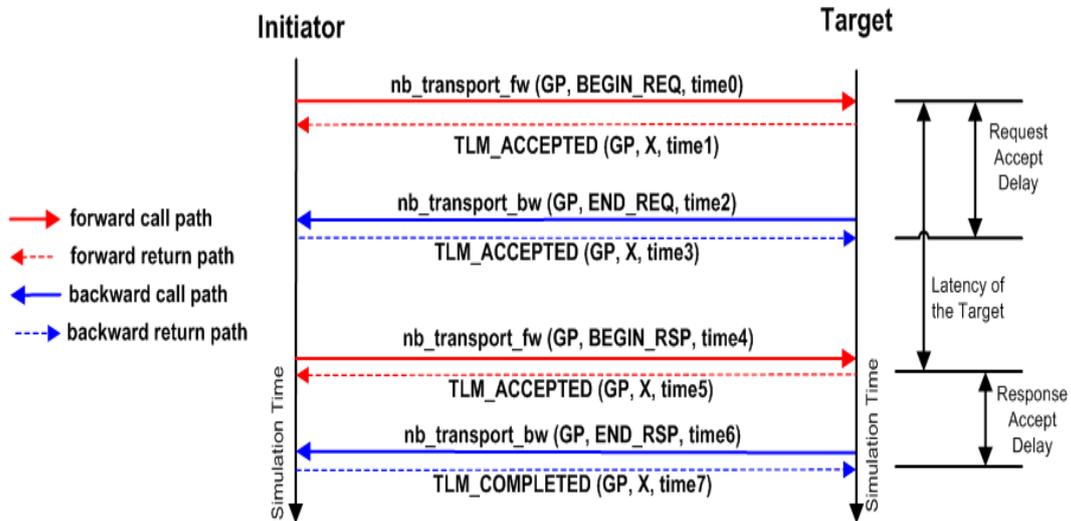


Figure 2.11: Message Sequence Chart of a Transaction Between Initiator and Target Using the Approximately-Timed Four-Phase Base Protocol

2.1.4.3 The TLM 2.0 Extension Mechanisms

When the TLM 2.0 standard sockets, generic payload and the base protocol phases are inappropriate to model protocol-specific communications other than the base protocol, the TLM 2.0 standard offers the possibility to extend either the TLM 2.0 generic payload attributes (`tlm_generic_payload`) or the TLM 2.0 generic protocol phases (`tlm_phase`) or both.

On the one hand, the TLM 2.0 allows defining an extension of the TLM 2.0 generic

payload as an object of a type derived from the TLM 2.0 standard class `tlm_extension`. By using this mechanism, two different types of generic payload extensions can be modeled: ignorable and non-ignorable extensions. Ignorable ones may be added to the generic payload extension array by initiators and in case the target does not care about this extension, the communication is not negatively affected. Non-ignorable ones are added by the initiator as well, but this time the target has to make use of it otherwise the communication will fail or significantly misbehave. When using a non-ignorable extensions the user has to define his own traits class on which the socket has to be templated, so that it cannot be bound to sockets that have been templated on a the TLM 2.0 standard traits class.

```
//the new protocol traits class with a non-ignorable extension in the tlm
generic payload

struct priority_extended_gp {

typedef tlm::tlm_generic_payload    tlm_payload_type;
typedef tlm::tlm_phase              phase_type;

}
```

Figure 2.12: Example of New Protocol Traits Class With a Non-Ignorable TLM 2.0 Payload Extension

Figure 2.12 represents a user-defined traits class with the normal transaction types (`tlm_generic_payload` and `tlm_phase`), but here the generic payload (denoted GP) has been extended with a non-ignorable extension that defines the priority of a request. By defining this new traits class and using rather sockets templated on this class, the user imposes that a target deals with the extension as soon as it receives such a transaction type, while the TLM 2.0 defined rules and transaction transitions of the base protocol are still applied.

Defining an alternative custom transaction type (i.e. payload) with the core interfaces and sockets without using the `tlm_extension` mechanism offered by the TLM 2.0 standard is also possible. But, this will significantly restrict the interoperability of the models.

On the other hand, the TLM 2.0 standard enables extending the set of the four phases provided by `tlm_phase` class using the `DECLARE_EXTENDED_PHASE` macro. Sim-

ilarly to generic payload extensions, phase extensions can be ignorable or non-ignorable. By using this TLM 2.0 extension mechanism, the base protocol phases sequencing rules must always be respected. Alternatively, when pre-defined phases and rules of the base protocol do not match with a target protocol model, a phase extension can be done through defining a new phase class. Similarly to the non-ignorable phase extension case, this current case requires defining a new protocol traits class that is rather used to define the transport core interfaces and custom sockets, restricting hence the model interoperability. The user-defined initiator and target sockets would define the new transaction type state transitions and ordering rules respectively on the initiator and target sides.

TLM 2.0 extension mechanisms have been widely used by industrial and academics for different modeling requirements and use cases. For instance, in [88], Robert Gunzel of GreenSocs proposes the GreenSocket approach based on another classification of TLM 2.0 extensions. This approach is a set of an API and a methodology to build TLM 2.0 convenience sockets that improve the TLM 2.0 interoperability by providing automatic memory management of transactions and extensions. In addition, in [33], the GreenSocs initiative proposes a TLM 2.0 based Asynchronous Serial communication protocol which can be used to model industry standard serial interfaces such as UART Model. The main contribution of this work is to show how TLM 2.0 extension mechanisms can be used to model even non-memory mapped based protocols. In [145], authors have proposed a well-structured implementation methodology to model protocol-specific Bus Cycle-Accurate (BCA) TLM 2.0 interfaces and transactors based on the TLM 2.0 standard extension mechanisms. In [67], Damm et al. have used the TLM 2.0 standard generic payload extensions to model wireless communication within a wireless sensor network simulation where neither dedicated buses nor routers nor memory-map are used. Still in the context of non-memory mapped protocol-specific Transaction-Level modeling such as [33] and [67], we will present in 6 a TLM modeling approach of inter-power domain communications as a new use case of TLM 2.0 extension mechanisms.

2.1.5 Power reduction in Systems-on-Chip

This section first describes the relevant state-of-the-art power management techniques that can be exploited to design low-power systems and outlines the architectural blocks needed to support each of these techniques. Then, this section depicts state-of-the-art

levels of implementing such power management techniques and describes in particular the existing works and standardization initiatives addressing power management interfaces at each level. Quite a few concepts from power management interfaces and architecture basis have been used in this thesis.

2.1.5.1 Dynamic and Static Power

The total power consumption for a SoC design consists of dynamic power and static power. Dynamic power is the power consumed when the device is active; that is when signals are changing values. Static power is the power consumed when the device is powered up but no signals are changing value. Equation 2.1 and Equation 2.2 describe respectively the instantaneous dynamic ($P_{dynamic}(t)$) and static ($P_{static}(t)$) power consumption (i.e. at a time t) of a device.

$$P_{dynamic}(t) = C' \cdot V^2(t) \cdot f_{clock}(t) \text{ (in Watt)} \quad (2.1)$$

$$P_{static}(t) = V(t) \cdot I_{leakage} \text{ (in Watt)} \quad (2.2)$$

C' is the switching activity multiplied by the effective load capacitance. $V(t)$ is the supply voltage at time t . $f_{clock}(t)$ is the frequency of the system clock at time t and $I_{leakage}$ refers to the leakage current. Actually, C' , f_{clock} and $I_{leakage}$ are technology-dependent constant parameters that characterize a functional block implementation. These parameters may come either from technical datasheets or measured at low design stages (typically the Register Transfer Level or the Gate Level) using dedicated EDA tools or on the real board.

On the one hand, the first and primary source of dynamic power consumption is switching power, the power required to charge and discharge the output capacitance of a gate. Because of the quadratic dependence of power on voltage, decreasing the supply voltage is a highly leveraged way to reduce dynamic power. As it will be explained in the next section, several state-of-the-art power management techniques such as voltage scaling techniques take advantage from this approach. Another approach for reducing dynamic power is clock gating. Driving the frequency to zero drives the dynamic power to zero.

On the other hand, static power consumption in CMOS devices is due to leakage. As lowering the dynamic power results in raising leakage current, with 90nm and deeper submicronic technologies, we are getting to the point where static power represents a big problem as dynamic power, and techniques for reducing static power consumption are strongly needed. Among the state-of-the-art techniques for reducing leakage current is to shut down the power supply to a block of logic when it is not active. This approach is known as power gating and is discussed in more details in the next section.

2.1.5.2 Low Power Design Techniques

a. Clock Gating

A significant fraction of the dynamic power in a chip is in the clock distribution network. Up to 50% or even more of the dynamic power can be spent in the clock buffers incurring time delays. The flops receiving the clock also dissipate some dynamic power even if the input and output remain the same. The most common way to reduce this dissipated power is to turn clocks off when they are not required. This approach, known as clock gating, is supported through clock domains architectural blocks. A clock domain is a group of modules (or subsystems) fed with the same gated clock signal (see Figure 2.13). This concept enables the control of dynamic power consumption of a device by gating the clock to this device clock domain as long as all modules of this domain are inactive.

b. Multi-Voltage Scaling

Since dynamic power is proportional to V^2 and static power is proportional to V , lowering V on specific blocks helps reducing the overall system power significantly. In this context, multi-voltage scaling power management technique relies on moving away from the traditional approach of using a single and fixed supply rail for all the internal logic of the chip. According to this technique, different blocks may have separate power supplies such that each block can run at the lowest voltage while meeting system timing constraints and performance objectives.

For instance, a processor requires a relatively high supply voltage as it may need to run as fast as the semiconductor technology will allow. Conversely, a lower supply rail may be sufficient for a USB block to meet timing constraints since it runs

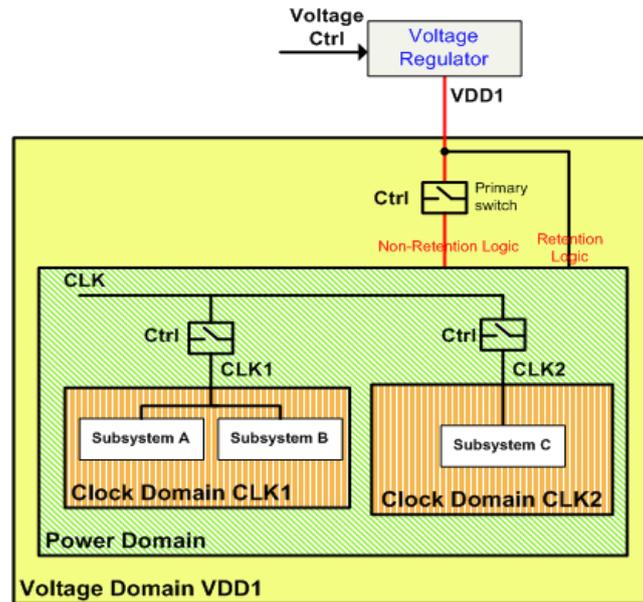


Figure 2.13: Voltage, Power and Clock Domains for Power Management [15]

rather at a fixed lower frequency dictated more by the protocol than the underlying technology. As a consequence, the CPU and the USB are put in different voltage domains, each with its own supply. A voltage domain is a group of modules supplied by the same voltage regulator used to control this group voltage independently (see Figure 2.13). Here, assigning a lower power supply to the USB block means that its dynamic and static power will be lower and hence significant power savings would be obtained.

Depending on the power supply voltage assigned to a voltage domain and how its voltage is controlled, multi-voltage scaling techniques can be classified as follows [125] [96]:

- **Static Voltage Scaling (SVS):** different blocks or subsystems are given different fixed supply voltages.
- **Multi-level Voltage Scaling (MVS):** an extension of the static voltage scaling case where a block or subsystem is switched between two or more fixed and discrete voltage levels.
- **Dynamic Voltage and Frequency Scaling (DVFS):** an extension of MVS where the operating voltage of a group of blocks is dynamically switched to an optimal level in order to follow changing workloads while meeting performance con-

straints. Such an operating voltage is changed between a larger number of discrete voltage levels named operating performance points (OPP). Each OPP is composed of a voltage and frequency pair.

- **Adaptive Voltage Scaling (AVS):** an extension of DVFS where a dynamic voltage control loop regulates the voltage and the clock based on the performance level.

Though multi-voltage scaling techniques help achieving system energy-efficiency, they add complexity to the design and verification process. For instance, even the simplest multi-voltage design requires to choose and to place carefully level shifters that consist in specific buffers that translate the signal from one voltage swing to another. Figure 2.14 depicts an example of two voltage domains embedded in a third voltage domain. Here, a high-to-low level shifter is placed in the destination domain of the output signal crossing different voltage domains and uses the voltage rail from the lower power domain. As level shifters do not affect the functionality of the design and from a logical perspective they are just buffers, recent tools can automatically insert level shifters where they are needed. Such tools do not change the RTL and only require a level shifter placement strategy specification of which blocks require level shifters, where to place the low-to-high level shifters in the lower domain, the higher domain, or between them, and may be a minimum voltage difference that requires level shifter insertion.

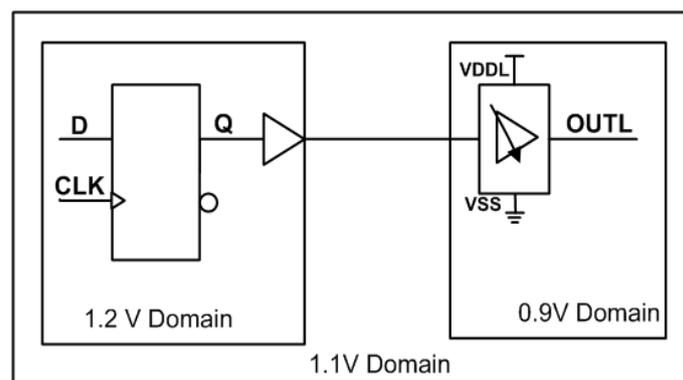


Figure 2.14: High-to-Low Level Shifter in the Destination Domain

c. Power Gating

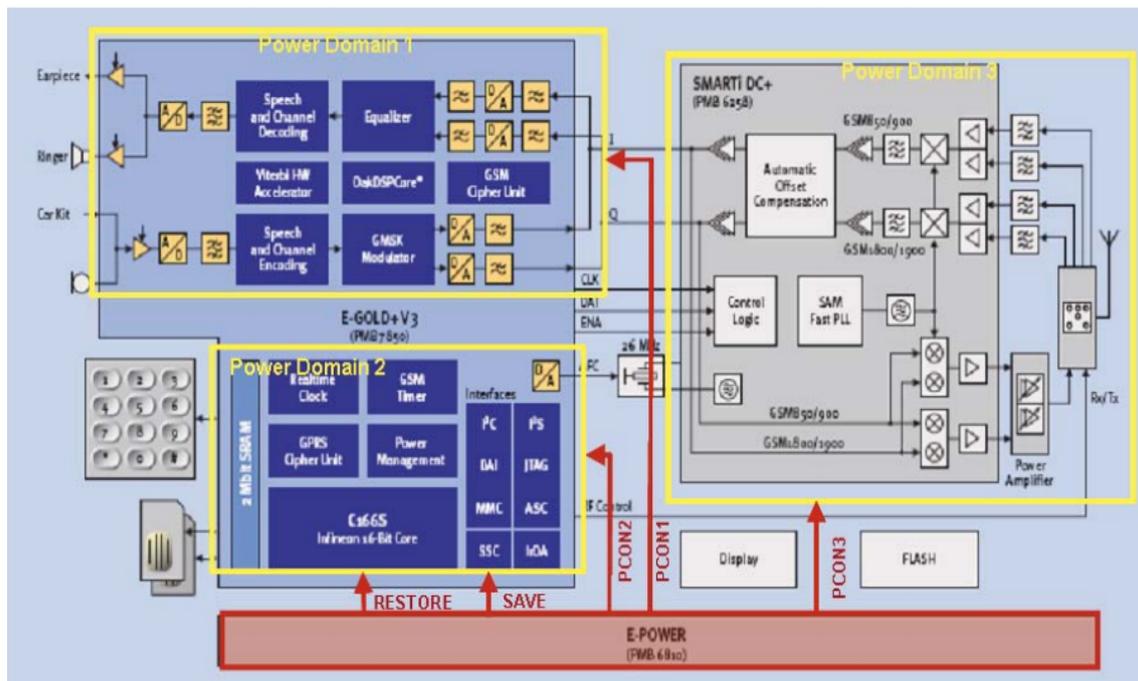


Figure 2.15: Power Management Structure Example Based on Power Domains Partitions [138] [Source: Infineon Diagram With Added Power Domains]

To reduce the overall leakage power of the chip, it is highly desirable to add mechanisms to turn off blocks that are not being used. This technique is known as power gating. The basic strategy of this technique is to provide two power modes: a low power mode and an active mode. The goal is to switch between these modes at the appropriate time and in the appropriate manner to maximize power savings while minimizing impact on performance. Actually, this technique is more invasive than clock-gating or voltage scaling in that it affects inter-block interface communication and adds significant time delays to safely enter and exit power gated modes. Therefore, the achievable savings through applying power gating are compromised to some extent.

In order to apply power gating, the internal logic of the chip must be split into power domains as illustrated by Figure 2.15. Each power domain is a group of the chip devices or subsystems that share the same primary and independent power rails. Thereby, it can be turned on/off without affecting the other parts of the chip. As illustrated by Figure 2.13, a power domain is supplied by one or more voltage

domains that can be scaled down or switched off to save power. A power domain that is never scaled down or switched off is called an always on power domain. It is supplied with a fixed supply voltage.

To switch-off a power domain for a short time, internal power switches are used to control power to this domain blocks. This method is called on-chip power gating. Conversely, off-chip power gating turns off the supply voltage to the power-gated domains with a switchable voltage regulator on board. This approach suits long-term power shut-off because it may take a long time to restore power to the gated blocks.

Figure 2.18 shows a simplified example of a SoC that uses on-chip power gating. Unlike a block that is always powered on, the power-gated block receives its power through a power-switching network. This network switches either VDD or VSS to the power gated block. In this example, VDD is switched and VSS is provided directly to the entire chip. The power-switching network typically consists of a large number of CMOS switches distributed around or within the power gated block.

Among the main critical issues in designing power gating is the design of inter-power domain communication interfaces. The additional interfaces consist in **retention flops** and **isolation cells** as it can be seen on Figure 2.18. On the one hand, **isolation cells** are required to be placed between the outputs of the power gated block and the inputs of the always on block in order to prevent crowbar (i.e. short circuit) currents in the always powered on block as long as the control isolation cell is off. The basic approach for controlling outputs of powered down blocks is to use an isolation cell to clamp outputs of a gated power domain to an inactive state. When using active high logic, the most common approach is to clamp the value to

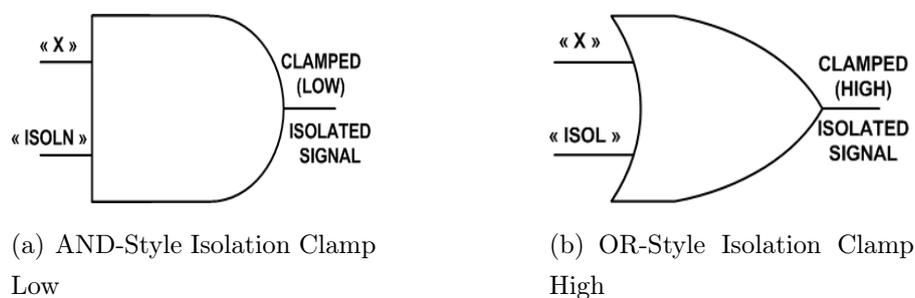


Figure 2.16: Basic Isolation Cells

"0". An AND-gate function accomplishes this. Figure 2.16(a) shows an AND-style isolation clamp low. When the active low isolate signal "ISOLN" is high, the signal passes to the output and when "ISOLN" is low, the output is clamped low. With active low logic, an OR-gate function parks the output at logic "1" (Figure 2.16(b)).

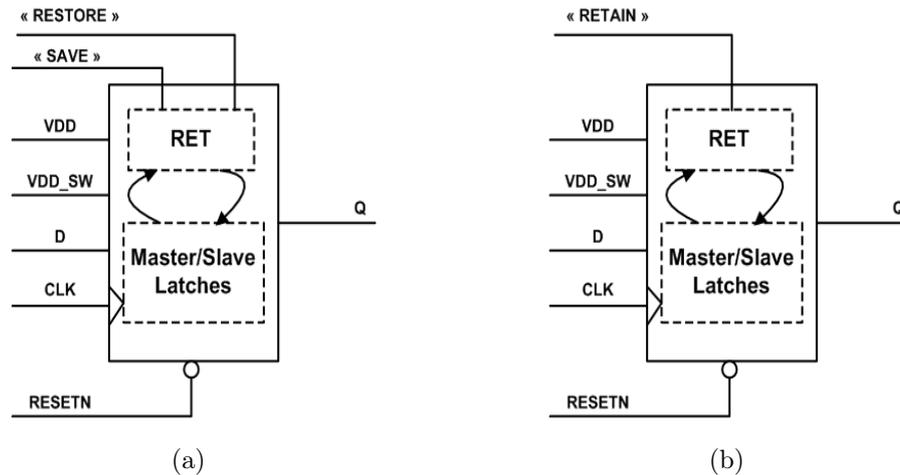


Figure 2.17: Retention Registers

On the other hand, when powering down a power domain, its internal memory and logic states are lost. So, to resume its operation on power up, the gated power domain must either have its state restored from an external source or build up its state from the reset condition. In order to save the time and power required to restore its state, a retention strategy must be employed. A commonly used and efficient approach to implement retention strategies inside power domains is to replace ordinary flip-flops with **retention registers**. Figure 2.17 gives examples of retention flip-flops.

A retention flip-flop typically has an auxiliary or shadow register ("RET" in Figure 2.17) that is slower than the main register (the master and slave latches of the flop in Figure 2.17) but has much less leakage current. The main register is powered by the switched power rail ("VDD_SW" in Figure 2.17). The clock ("CLK" in Figure 2.17), D and reset ("RESETN" in Figure 2.17) pins operate on the main register, which drives the Q output. The shadow register is always powered up, and stores the contents of the main register during power gating. A retention register needs to be

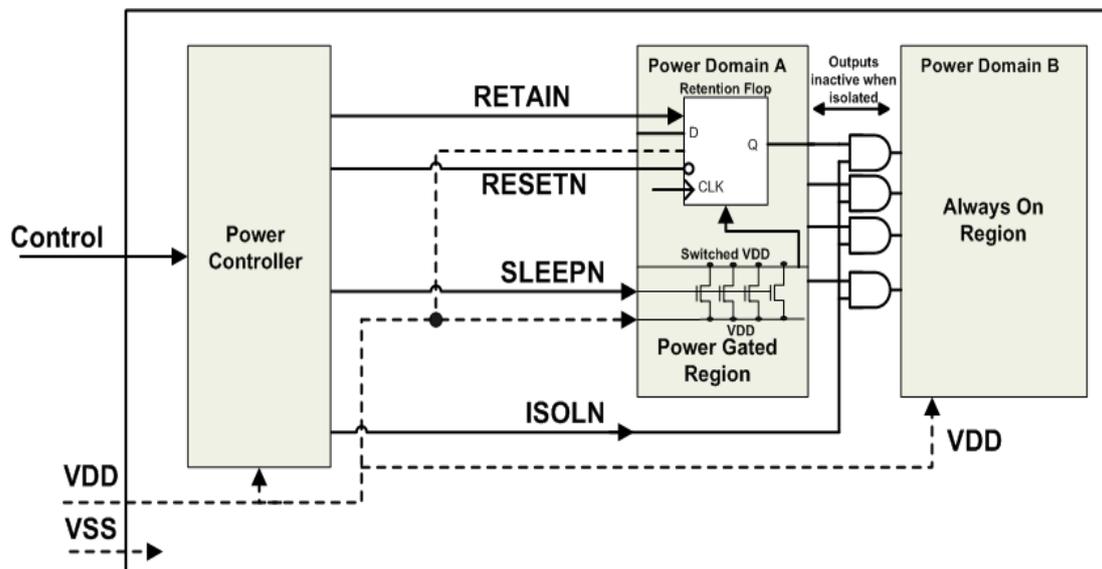


Figure 2.18: Block Diagram of an SoC with Power Gating

told when to store the current contents of the main register into the shadow register and when to restore the value back to the main register. For instance, in Figure 2.17, the state of the main register is loaded into the shadow register when "SAVE" is asserted in Figure 2.17(a) or when "RETAIN" goes high in Figure 2.17(b). When "RESTORE" is asserted in Figure 2.17(a) or "RETAIN" goes low in Figure 2.17(b), the content of the shadow register is loaded back into the main register. Similarly to control signals of isolation cells, retention control signals are provided by the power domain's power controller as depicts Figure 2.18.

2.1.5.3 Power Management Levels

In order to implement the aforementioned power reduction techniques in systems on chip, a system power manager (PM) that coordinates activities of the different components and schedules their power states according to a specific control procedure is required. A control procedure is usually called policy; the timeout policy is a typical example that shuts down a component after a fixed inactivity time delay.

Given a policy, a power manager typically requires information on the usage of each hardware component in order to have an up-to-date control on their power state. There-

fore, Benini et al. in [44] have pointed out that "standardization between the PM and system is an important feature for decreasing design time". In the same context, Bergeron in [47] has highlighted the standardization trend for power management interfaces and emphasizes on the need for a common SoC power management protocol and interface to increase interoperability of SoC Intellectual Property (IP) cores.

Nevertheless, implementing such a common interface would depend on the control procedure level (component, system, hardware, software, etc.) and the physical realization style of the power manager. Figure 2.19 illustrates our classification for power managers. Indeed, a power manager may be power-controller (PC) directed power manager that is simply initiated by hardware timers or using a hardwired specialized controller unit. Alternatively, it may be operating system (OS) directed that is initiated by a control software routine as part of the components drivers or the operating system tasks. A hybrid hardware-software power manager in which the PM functionality is implemented as a firmware running on a CPU is also possible.

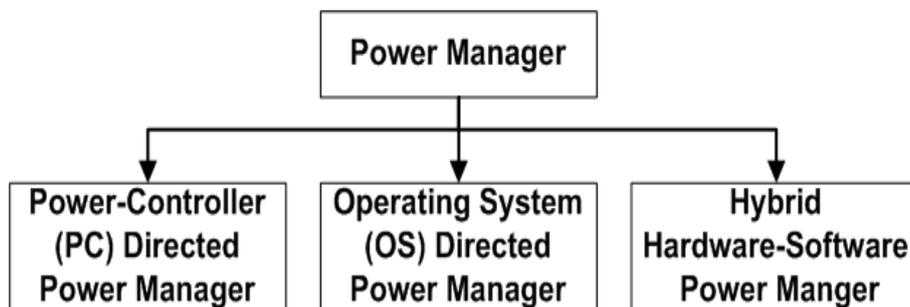


Figure 2.19: Power Manager Classification

For each PM embodiment, the system power states control and the data collection process are differently implemented by using either a software or hardware controllable or both power management interfaces. The following sections detail the fundamental characteristics of PC-directed and OS-directed power managers. Some interesting industrial power manager design models and state-of-art power management interfaces are discussed in this part. We have used quite a few concepts from the mentioned power management design approaches in this thesis. These sections will help the reader to understand our approach discussed in the 6.

a. Power Controller Directed Power Management

In this type of power management, the control of the devices power states is assigned to a specialized hardware unit called Power Controller (PC) that manages hardware control signals added for power management purposes. On the one hand, the Power, Reset and Clock Manager (PRCM) [15] integrated in the OMAP3 platforms of the Texas Instruments Company (TI) is a typical example of PC-directed industrial power management solution that addresses the control by hardware of a power architecture partitioned into different power, voltage, clock and reset domains.

On the other hand, there are recent protocol interfaces that standardize PC-directed power management communications. Examples of such power management protocol interfaces are the PMBus open-standard protocol [24] and the SPMI bus [13] specified by the MIPI Alliance System Power Management Working Group. Both of them defines an enhanced I2C serial interface. The PMBus focuses on the transport and physical layer as well as on command language to communicate with power converters. The SPMI bus defines a command set and a protocol for power management and traffic control between Power Controllers (PCs) of SoC processors and peripheral devices. Although both protocol interfaces specifications use dedicated hardware-triggered control signals to change the power state of a device, these two buses enable only the control of system devices power states without considering any power architecture features. Even though they offer some semantics that can be adopted in a power domains management context, new semantics are still required.

In [134] and [133], authors have recently proposed a PC-directed interface in the form of a session-based Domain Power Interface (DPI). This interface defines the protocol and signals involved in power management communication between power domains and their PC-directed power manager. Their interface specification targets a dedicated wireless sensor network node protocol processor and remains so close to the power-managed system architecture proposed in their work.

More details on the PRCM and the MIPI's SPMI bus are given in the following.

- **Texas Instruments Power, Reset and Clock Manager (PRCM):**

The PRCM block is an example of a PC-directed power manager integrated in the TI OMAP3 platforms. As depicted in Figure 2.20, the PRCM (outlined in red) is a hardwired power manager in charge of implementing all the control in the whole system to perform power state transition according to the functionalities activated

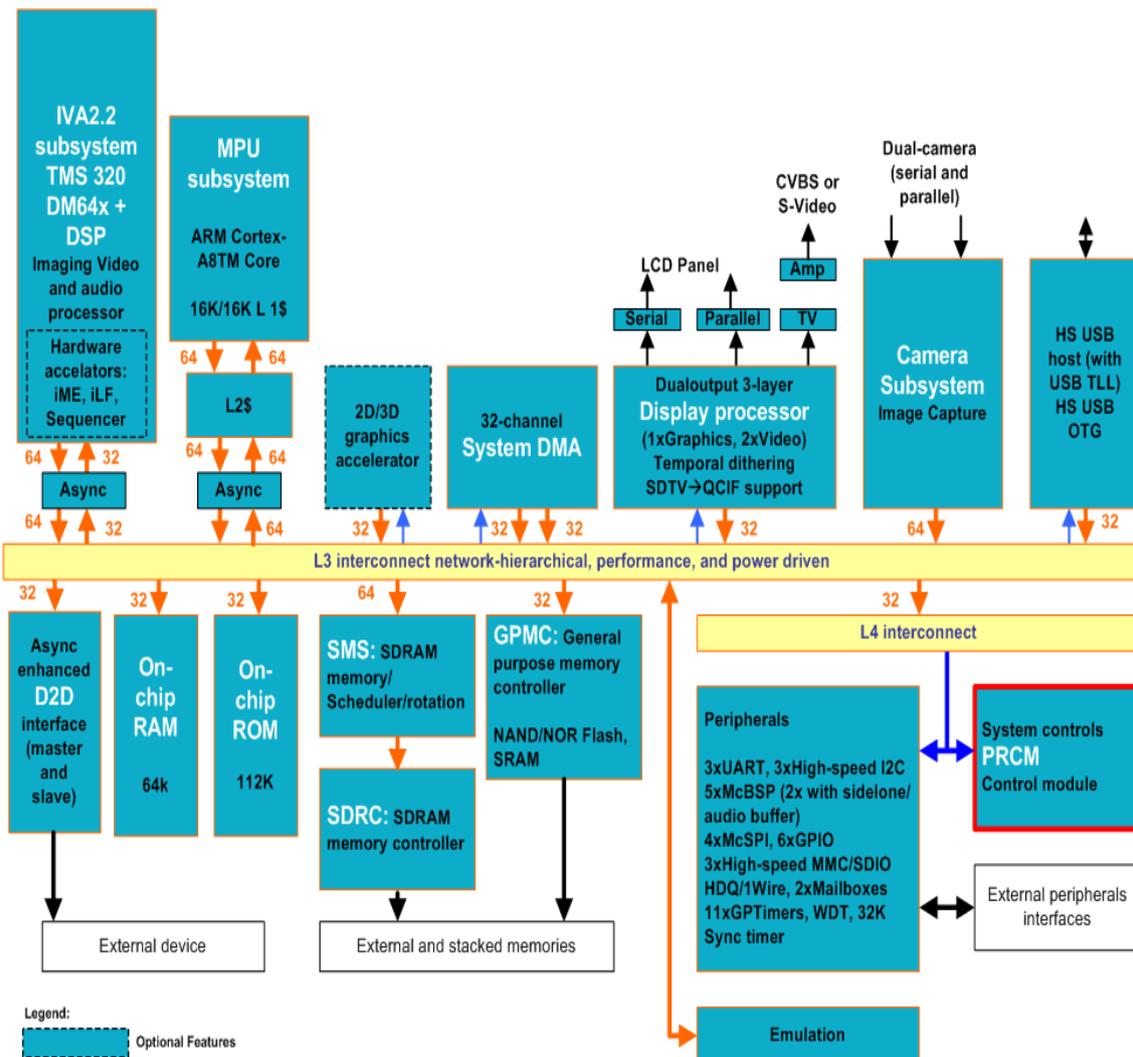


Figure 2.20: Texas Instruments OMAP3 Block Diagram

by the end user.

The OMAP3 platform is partitioned into power, voltage, clock and reset domains as depicted by Figure 2.21. The PRCM provides the Application Programming Interface (API) for controlling the states of all these domains as well as state dependencies between them. Indeed, control of the different domain states can be either software-controlled by adequately setting appropriate memory-mapped registers of the PRCM or hardware-triggered using dedicated hardware control signals. For instance, transitions of each domain state can be controlled by software through a set of dedicated status registers that allow the configuration of the power state

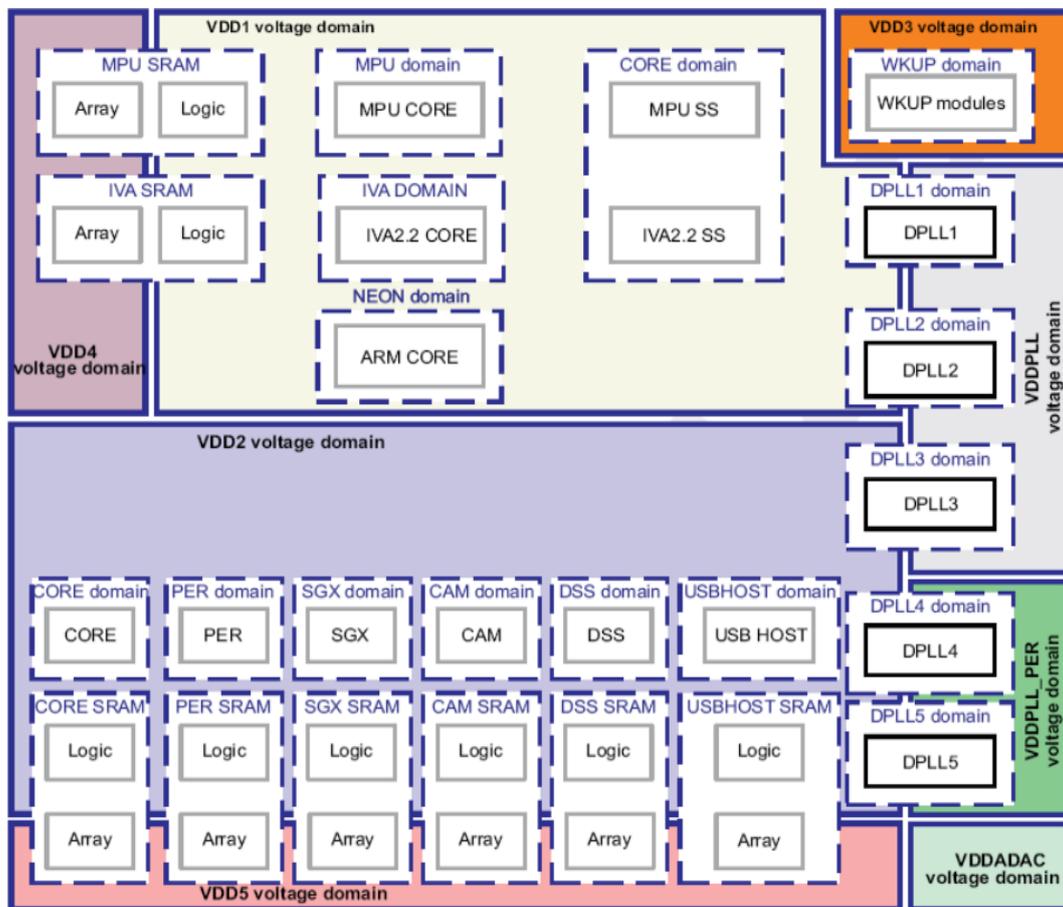


Figure 2.21: Texas Instruments OMAP3 Power Architecture

into which the domain enters after completing a transition. These status registers are used to check the current state of the logic and memories in a domain and to learn about any ongoing state transition, so to inform about the activity profile of domain’s components. Similarly, some dependencies between domains are programmable by software while others are hardwired.

Due to the large number of the PRCM software-configurable registers as well as the huge number of power, voltage, clock and reset domains, the TI PRCM represents a very complex solution that lacks modularity and is hard to use and debug. Targeting higher performances, the OMAP4 platform integrates more than 300 IP blocks in a single chip. The fact that these IPs are all under the control of a single hardwired and centralized PRCM which has the strategic role of implementing efficient power techniques, in a strong coherency with the execution of the functionalities

such as power consumption is optimized, illustrates more this uncontrollable complexity. It is also worth noting that the PRCM reference guide is more than 400 pages (for OMAP3-based platforms) [15].

- MIPI's System Power Management Interface (SPMI):

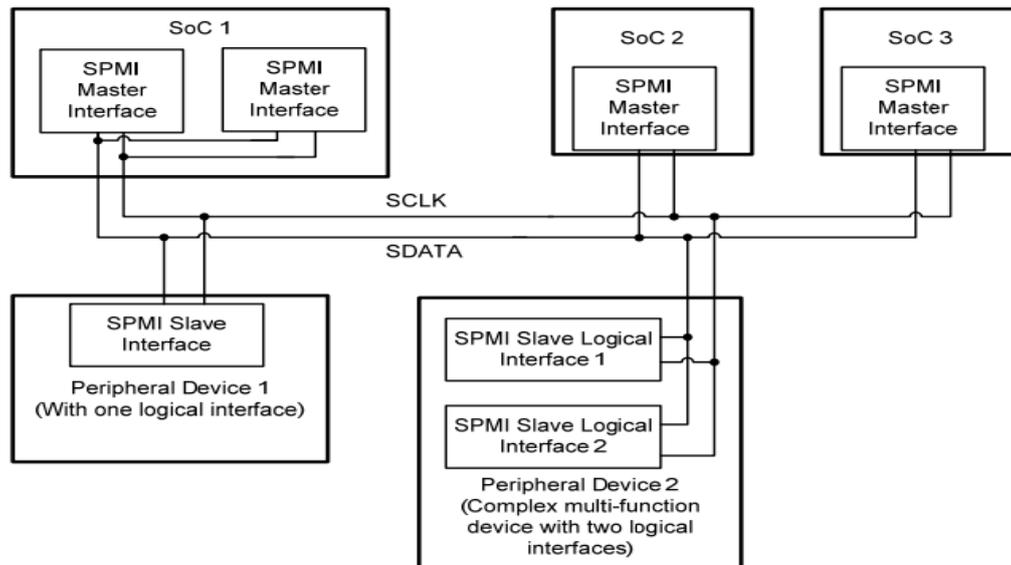


Figure 2.22: SPMI System Example [13]

The MIPI alliance System Power Management (SPM) working group has delivered in 2008 the System Power Management Interface (SPMI) standard specification [13]. This standard represents the first serious industrial initiative of power management interfaces standardization that enables a rapid deployment of advanced power management techniques. More precisely, SPMI specifies a standard hardware power management interface between baseband or application processors and peripheral components. It enables systems to dynamically adjust the supply and substrate bias voltages of the voltage domains inside the SoC using a single SPMI power management bus specified as an enhanced I²C bus, a two-wire serial interface (SCLK (SPMI clock) and SDATA (SPMI data) as illustrated by Figure 2.22).

The SPMI specification defines the SPMI devices operating states, the command set, communication sequences, I/O structures/physical layer, and the low-level protocol

for data communication between SPMI devices on a SPMI bus. A first fundamental feature of SPMI bus interface is the identifier-based addressing of the SPMI devices. A SPMI system may have up to four Master devices and up to sixteen Slave devices. In order to communicate on SPMI bus, each SPMI Master or Slave device needs a unique identifier. Additionally, group slave identifier numbers can be used to identify groups of SPMI slave devices enabling hence communication to single or multiple slaves at a time.

In addition, the SPMI bus protocol is a sequence-based protocol where each sequence transmitted on the bus is composed of individual bits. Sequences comprise the following events that occur in order: bus arbitration, transmission of Sequence Start Condition (SSC), transmission of frames (a command frame and possibly one or more data frames) and finally transmission of a Bus Park Cycle.

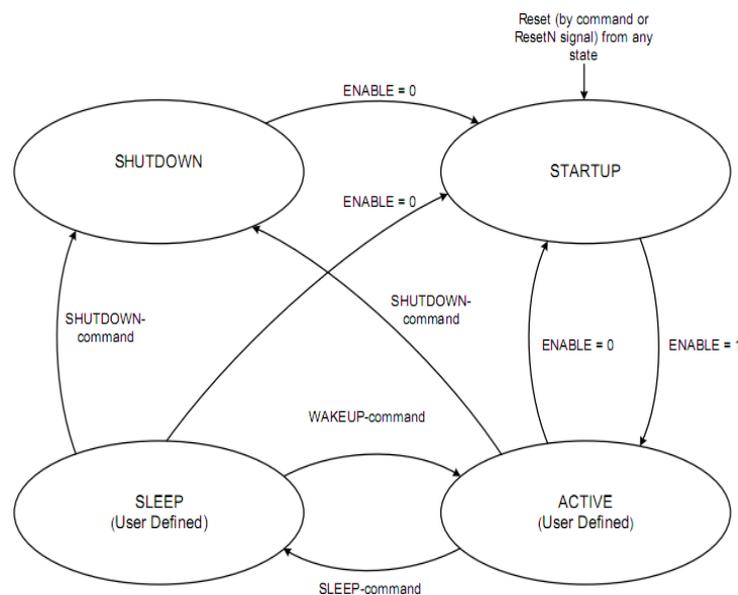


Figure 2.23: SPMI Slave State Diagram

In order to allow independent power modes on SPMI devices, each SPMI slave shall have four operating states: ACTIVE, SLEEP, SHUTDOWN and STARTUP as depicted by Figure 2.23. The ACTIVE state represents a user-defined normal operating state of a slave after the power-on sequence (STARTUP state) while the SLEEP state represents a user-defined lower power state other than the SHUTDOWN state

(where all output voltages go to 0). As shown by the state machine of Figure 2.23, these different states may be automatically entered by triggering external hardware control signals (such as the ENABLE and RESETN SPMI slave device input signals) or transmitting specific SPMI command sequences on the SPMI bus (such as RESET, SLEEP, SHUTDOWN or WAKEUP commands).

Figure 2.24 depicts an example of power mode transition request command sequence. As it can be seen, such a sequence starts with the SSC followed by a command frame, which is unique for each command, and ends with a Bus Park Cycle. Different other SPMI command frame payload have been defined such as authenticate and transfer bus ownership required for bus arbitration and register write and SPMI control read and write registers.

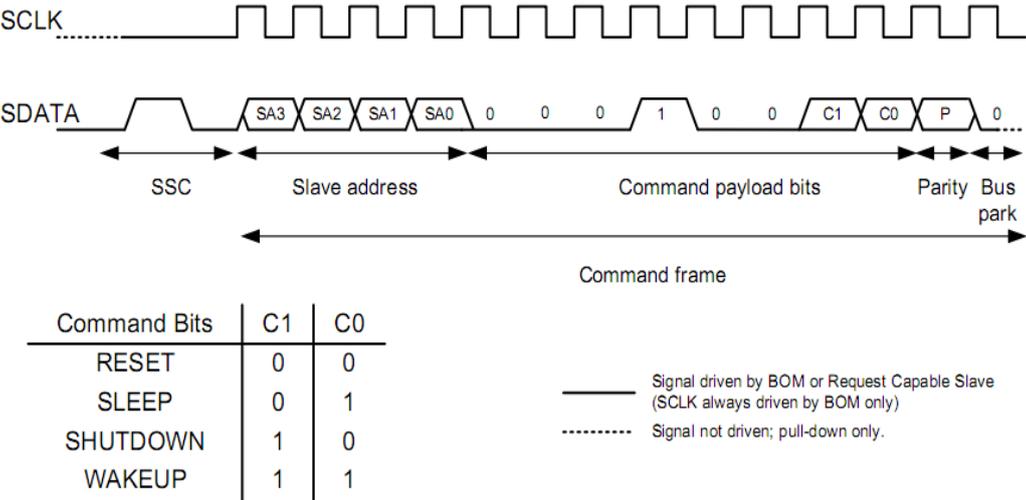


Figure 2.24: Reset, Sleep, Shutdown and Wakeup SPMI Command Sequences

Another interesting and original feature of SPMI bus is bus arbitration. Indeed, the SPMI bus is shared between multiple master and slave devices allowing direct Master-to-Master, Master-to-Slave, Slave-to-Slave or Slave-to-Master communications via the SPMI bus. In particular, the concept of Request Capable Slave (RCS) device defined in SPMI makes Slave-to-Slave or Slave-to-Master communications on SPMI bus possible. In fact, a RCS device is a slave device that can arbitrate for SPMI bus to initiate Sequences on it. Conversely, a slave device that can not initiate Sequences is called a Non-Request Capable Slave (NRCS) device.

Having multiple potential sequences initiators on the SPMI bus, four different bus arbitration levels have been defined. Such a definition allows the transmission of timing critical and urgent Sequences on SPMI bus with minimal latency while the transmission of sequences that can tolerate more latency occurs when unused bus bandwidth is available. Priority sequences from Request Capable Slave devices are arbitrated at the highest bus arbitration level (level 1) using the Alert bit (A-bit) while secondary sequences from these devices are arbitrated at bus arbitration level 3 using the Slave Request bit (SR-bit) instead. By using a Round-Robin algorithm to change the Master Priority Level (MPL) of a master device, priority sequences from master devices are arbitrated at bus arbitration level 2 while secondary ones for these types of devices are arbitrated at the lowest bus arbitration level (level 4). Note that the SPMI specification does not consider physical power management architecture features such as multiple and hierarchical distribution of power domains and supply networks. While it offers some rules and semantics for hardware-oriented power management, it does not propose mechanisms and concepts for the control of a power architecture. Nevertheless, we will see in 6 of this thesis how different SPMI concepts can be useful to define a specialized power domain management bus interface that fills SPMI gaps.

b. Operating System Directed Power Management

In an OS-directed power management, the OS implements a global power management strategy to control the devices power states independently of each other. For that, the hardware resources need to be interfaced with the OS-oriented software power manager and both the hardware resources and the software application programs need to be designed so that they cooperate with the OS power manager. Actually, the abstract power-management interface between the OS and the hardware platform defines the global system and devices power states as well as the hardware registers for power management control. Through such an interface, devices expose specific power management capabilities to supply the OS with their activity information.

ACPI (Advanced Configuration and Power Interface) is an example of abstract interfaces that enable OS-directed Power Management [4]. The software and hardware components relevant to ACPI are shown in Figure 2.25. Applications interact with

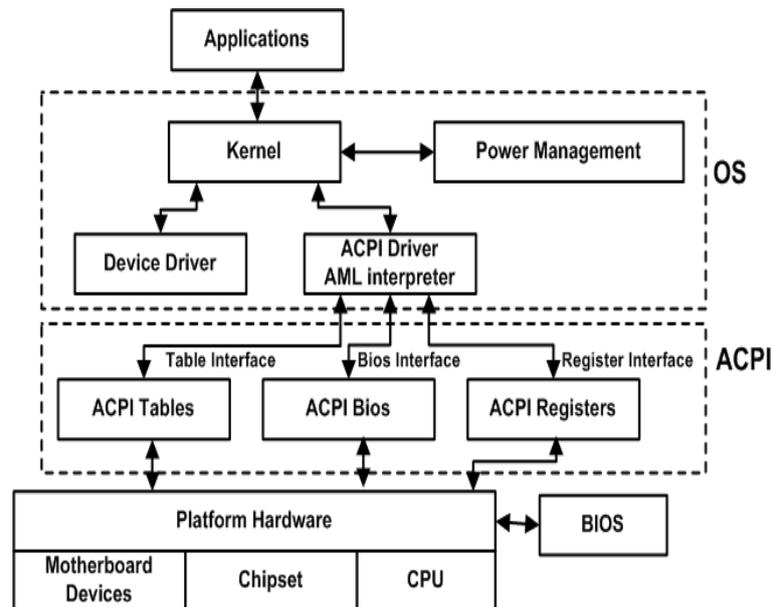


Figure 2.25: ACPI Interface [44]

the OS kernel through Application Programming Interfaces (APIs). A module of the OS implements the power management policies. The power management module interacts with the hardware through kernel services (system calls). The kernel interacts with the hardware using device drivers. The ACPI driver is used to map kernel requests to ACPI commands, and ACPI responses/messages to kernel signals/interrupts. Note here that ACPI-compliant hardware devices must provide a mechanism to inform the power manager about their power state or to request a change. For instance by using the Peripheral Component Interconnect (PCI) [22] and the Peripheral Component Interconnect Express (PCIe) [23] bus power management interface specifications, any PCI-based component can communicate with the OS-level power manager through asserting Power Management Event (PME) signals. The ACPI allows PCI devices control at the OS-level through mapping the PCI device states and registers into those of the ACPI interface.

Although OS-directed power managers are easy to write and to reconfigure, they specify neither how to implement hardware devices nor how to realize power management in the operating system. In addition, they do not define a standard way to interface with a power architecture composed of multiple power domains. They define how to handle the global system device by controlling the system devices inde-

pendently of each other. However, they do not specify how to control the local states of power domains (group of devices with common power features) and interactions between them. Indeed, the power management aspects handled by OS-directed PM interfaces are only appropriate for an OS-level power state control using specific software-configurable registers.

2.1.6 Low Power Design Standards

For designs without advanced power management techniques, only the power net and the ground net were traditionally defined and implemented in the layout phase since they did not have functional impact on the chip. Now, with the use of multiple power and voltage domains partitions and the increased complexity of Intellectual Property (IP) designs in a SoC, several power and ground nets are being used to supply parts of the chip and their state define the chip behavior. Given such a strong dependency between a chip power management architecture and functionality, the power distribution (supply nets and power switches) and its state change behavior must be defined and validated early in the design flow.

Nonetheless, neither the Register Transfer Level (RTL) traditional hardware description languages (HDL) nor the logical views for basic library elements (leaf cells) have implicit representation of power design nets. Furthermore, a special handling and global connection of the power and functional designs in the back-end phase is tedious and error-prone.

Recently, two competitive industry-standard power intent formats have emerged as a solution for designing low power SoCs at early stages of the design flow. The Unified Power Format (UPF) standard has been initially released as the UPF 1.0 specification version



Figure 2.26: Functional and Power Intent

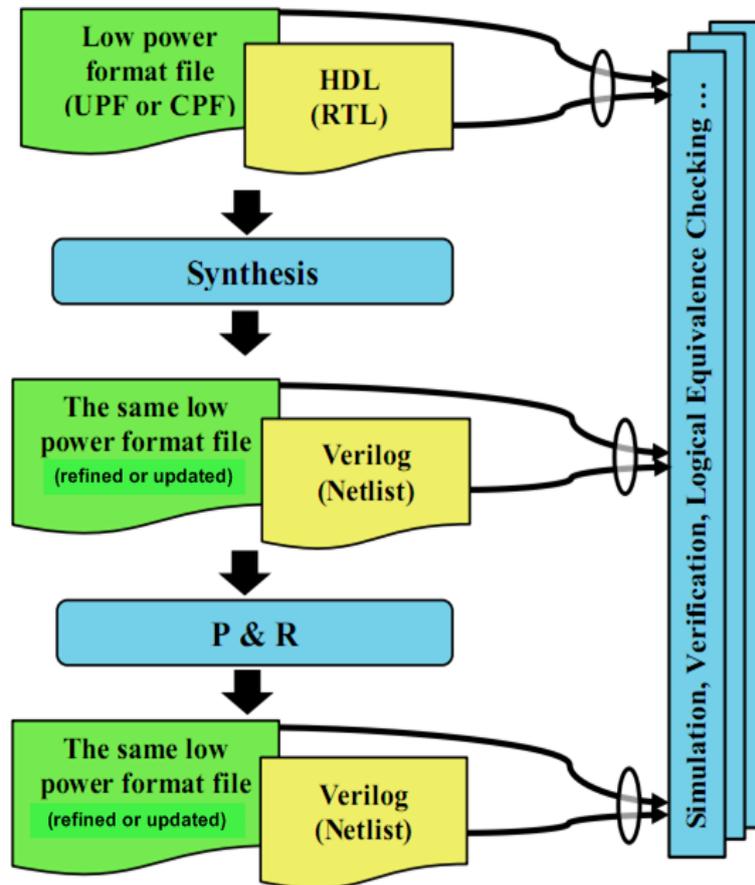


Figure 2.27: Low Power Format Standards Tool Flow Starting from RTL

in January 2007 by Accellera, and recently released in March 2009 as the IEEE Standard for Design and Verification of Low Power Integrated Circuits (IEEE-1801 standard [30]), also called UPF 2.0 specification version. The second one is the Common Power Format (CPF) 2.0 [29] which is rather managed by Silicon Integration Initiative (Si2)'s Low Power Coalition.

These low power format standards are based around TCL, the Tool Control Language embedded in most EDA tools. Although each standard has its own syntax, both formats can be seen as a set of TCL procedure definitions rather than a new language. This definition enables delivering a power intent specification, that includes the main features of a power management architecture, separately from the functional specification. Thereby, a design specification is captured as a power intent specification and a functional specification pair as illustrated by Figure 2.26.

This separation of power and functional specifications employed by both standards avoids a direct specification of the power semantics in an HDL code that would tie the logic specification directly to a constrained power implementation. Conversely, this approach facilitates the reuse of a golden functional specification to explore different power architectures starting from RTL without a need to understand the details of power domains implementation. It also ensures that changes to the power intent do not require rewriting and re-verifying the HDL, and vice versa.

The ability to use the same power intent specification file throughout the SoC design flow represents another important feature of these standards. As depicts Figure 2.27, the same power format file (either UPF or CPF) used to describe power intent, is first combined with the HDL. Then, as power format standards define also consistent semantics across verification and implementation, this power format file would represent an additional input to the different tools used throughout the flow (e.g. simulation, synthesis and formal verification tools). It can also be incrementally updated and refined throughout the design flow as depicted by Figure 2.27.

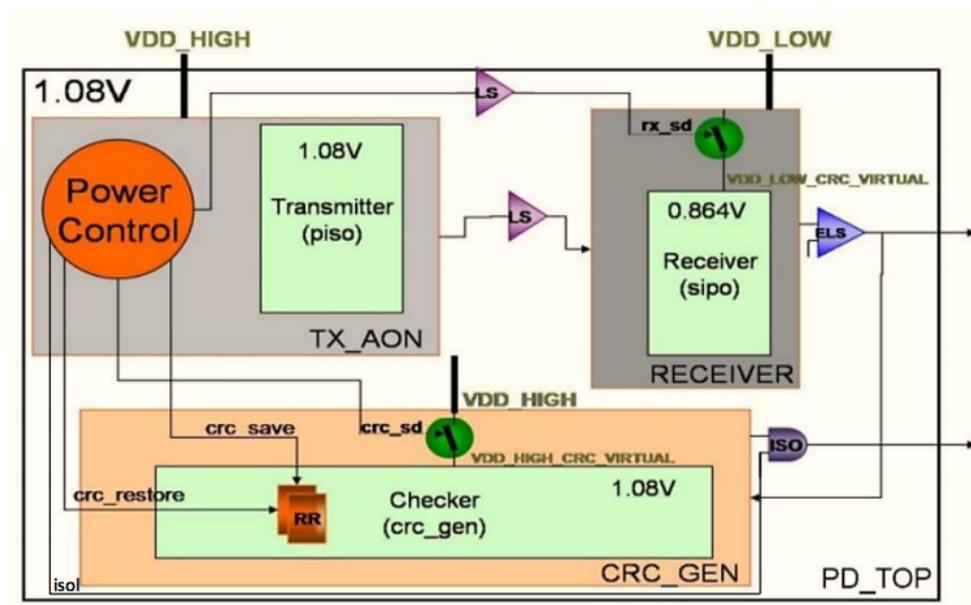
Obviously, each tool in the design flow is required to understand and interpret the power intent specification semantics in a power format file. For that, different EDA tool vendors such as Magma, Mentor and Synopsys added specific features to their tools in order to support UPF or CPF. So, these tools are able to understand the power intent file semantics and produce changes to the output file by adequately inferring required low power elements and behavior inside the functional description. To do this, these tools are able to automate many manual tasks such as the insertion of level shifters or isolation cells given only a simple strategy specified in the power intent file.

In the following sections, the fundamental concepts and features of the Unified Power Format are explained using an example. Then, similarities and differences between UPF and CFP standards as well as common gaps in both standards are delineated.

2.1.6.1 The Unified Power Format

In the following, we will use Figure 2.28 to exemplify the main power elements used by the UPF semantics to specify a power design intent. First, the power-domain concept in the UPF standard is defined as a group of elements from the logic hierarchy that share the same primary supply nets. In other words, each power domain is supplied by at least a

power net and a ground net and overlays at least one functional block. Each power domain can therefore be controlled individually. For instance, Figure 2.28(a) shows four different power domains. The top power domain (PD_TOP) has two primary power nets, each furnishes a different voltage value, (VDD_HIGH and VDD_LOW) and includes three nested power domains (TX_AON , $RECEIVER$ and CRC_GEN). The CRC_GEN power domain overlays for instance the Checker functional block. Note that the voltage domain concept previously introduced in section 2.1.4.2 is the same as the power domain concept defined by UPF.



(a) A Power Distribution Example

	VDD_HIGH	VDD_LOW	VDD_HIGH_CRC_VIRTUAL	VDD_LOW_CRC_VIRTUAL
PRE_BOOT	High_Voltage	Low_Voltage	CRC_OFF	RX_OFF
CRC_ON	High_Voltage	Low_Voltage	High_Voltage	RX_OFF
RX_ON	High_Voltage	Low_Voltage	CRC_OFF	Low_Voltage
ALL_ON	High_Voltage	Low_Voltage	High_Voltage	Low_Voltage

(b) A Power State Table (PST) Example

Figure 2.28: Example of UPF Defined Concepts

UPF also defines commands to specify power switches components in charge of shutting down or powering up power domains, as well as their controls required to change a power domain state. For instance, the primary power net of the *CRC_GEN* power domain represents the *VDD_HIGH_CRC_VIRTUAL* supply net which is an output supply net of a power switch. Thereby, the *CRC_GEN* power domain state depends on this power switch state defined by its control signal *crc_sd*.

Retention and isolation strategies can also be specified in UPF. In this context, a strategy is a general rule on how to implement these low power design functions. UPF also supports the specification of level shifter strategies, including voltage tolerance threshold, whether the strategy applies to up-shift, down-shift, or both, and it allows the designation of whether a strategy applies to input or output mode ports. As it can be seen on Figure 2.28(a), retention registers (named *RR* in Figure 2.28(a)) controlled by the *crc_save* and *crc_restore* control signals have been assigned to the *CRC_GEN* power domain in order to save the internal logic state of the Checker functional block when *CRC_GEN* is switched off and the active high control signal *crc_save* is high. Isolation cells (named *ISO* in Figure 2.28(a)) controlled by the *isol* control signal have been specified at the output of the power-gated *CRC_GEN* domain in order to avoid undefined signal values during power-down. Level shifters (named *LS* in Figure 2.28(a)) have been specified between *TX_AON* and *RECEIVER* power domains since they operate at different voltage levels (respectively 1.08 V and 0.864 V).

Among the main concepts of UPF, we find the power state table (PST) defining a static system power-management strategy in terms of the power domains' supply nets states. This concept ensures the integration of the system functional design with the low power design. Figure 2.28(b) depicts an example of a PST for the power distribution architecture of Figure 2.28(b). Columns of a PST represent local states of power domains in terms of their power supply net states, while lines represent the different system power modes. Each line corresponds to one legal combination of specific power-domain states. In general, a system power mode (line of a PST) refers to a set of activated functionalities matching a specific system scenario. For instance, the *RX_ON* power mode in Figure 2.28(b) corresponds to the receiving with disabled cyclic redundancy check (CRC) checking scenario. Recently, the IEEE 1801 standard also allows the specification of legal and illegal transitions between the different system power modes specified in a PST.

It is worth noting that the UPF language implicitly imposes a set of composition dependency rules between all these concepts to define a design power intent in a well-structured way. For instance, to be applied to a specific power domain, retention strategies must be defined in the context of this power domain. Similarly, a PST can only be specified after defining active and inactive states for each specified primary supply net.

As it can be seen in Figure 2.28(a), a power controller must be defined as an HDL functional block that uses the PST and potentially the legal power modes transitions in order to control states of all the UPF-defined power elements through their control signals (e.g. *crc_sd* control signal in Figure 2.28(a) is used to control the state of the *CRC_GEN* power domain's power switch). Indeed, UPF promotes a power-controller oriented power management since it defines a hardware power management interface that controls the state of each power domain. However, structure and behavior of such a power controller unit are still outside the scope of UPF and it is up to the designer to define them.

2.1.6.2 UPF Versus CPF: Similarities, Differences and Common Gaps

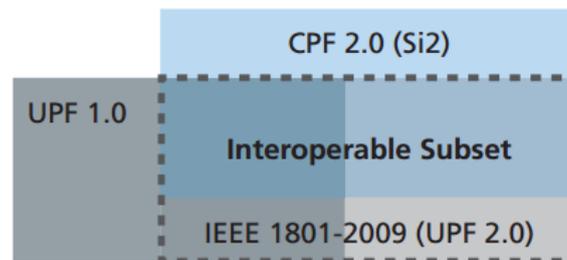


Figure 2.29: Current Status of All Power Formats [148]

Figure 2.29 delineates the current status of the different power standards. First, note that this figure depicts the existence of a common part between UPF and CPF standards. Indeed, based both around TCL, CPF and UPF standards cover 90% of the same concepts. Using the common low-power concepts namely power domains, supply nets, retention registers, power switches, isolation cells and level shifters, low-power designers can represent their power intent in any format. Nonetheless, UPF and CPF use completely different syntax. Each file would have a different number of commands and options to each command to capture the same power intent.

Among the other important differences between the two formats, UPF requires .lib (Liberty) files to define library cells such as level shifters or Retention registers. Conversely to CPF that provides syntax to define these library attributes, UPF does not define these attributes as it assumes that some other library formats exist to capture this information instead (e.g. Liberty "Synopsys dot lib").

As illustrated by Figure 2.29, the recent IEEE 1801-2009 standard (UPF 2.0) has come up with new constructs to close some of the methodology differences between UPF 1.0 and CPF 2.0. For instance, similarly to CPF semantics, the IEEE 1801-2009 introduces the concept of the supply set which allows the supplies to power domains to be specified more abstractly and provides an improved way of specifying power states. Unlike the approach used in UPF 1.0 and similarly to CPF, an RTL designer does no longer require to have the complete physical power network information in order to describe power intent.

Another important feature supported by the CPF and the UPF 2.0 constructs, which have not been defined by UPF 1.0, is the formal hierarchical power intent design approach including macro modeling for hardened low-power intellectual property (IP) [100]. This feature relies on the use of virtual ports and virtual power domains to simplify rules specification for design objects that will later appear lower in the hierarchy, when the design implementation is refined. The designer is hence able to code the block-level power intent and integrate this low-power block in multiple situations that require different uses of the block's internal power intent capabilities.

However, as it can be seen in Figure 2.29, UPF 1.0 is still a subset of IEEE 1801-2009. As a result, within the same standard there are two radically different methodologies to describe the same power intent which could create confusion for users.

The interoperable subset in Figure 2.29 gathers similarities between the different formats. These similarities have been exploited by some EDA tool vendors to offer solutions enabling mixed CPF-enabled and UPF-enabled tool flow interoperability. For instance, customers using UPF can benefit from the CPF-enabled Cadence low-power complete solution tools by using the Cadence Encounter Conformal Low Power tool to import their UPF file and export a semantically equivalent CPF file.

Even so, there are still some areas in which both formats need to improve and evolve. Among these areas, we mention for instance:

- The explicit definition of power and control structure for clock and voltage domains

and their related constraints and dependence relationship with the defined power domains features defined by the low power format.

- The definition of a common protocol interface for power domains state management reusable with different power domain partitioning and management strategies.
- Still complementary to the previous raised point, a common structure and functionality of a power management block in charge of controlling the power domains states is required. Such a block needs a well-defined power management interface to operate. These two design elements should be reusable whatever the organization of the power domain controllers and power domains (either flat or hierarchical).

In this thesis, we have developed solutions to fill the last two mentioned gaps. These solutions can be seen as potential extensions of the existing low power formats.

2.2 State of Art

2.2.1 State-of-The-Art on High Level Power Modeling, Reduction and Analysis

Many ad-hoc approaches and tools have addressed power modeling and estimations at the ESL starting from the Algorithmic/Functional level to the Cycle-Accurate one (Figure 2.1). Finding the best trade-off between speed and accuracy is the concern of almost all researches in this area.

While power can be accurately estimated after RTL synthesis, power characterization at higher levels of abstraction than RTL is a crucial task and has been extensively investigated. Since there is no standard way to create system-wide or IP cores ESL power models, approaches dealing with this issue support different criteria to probe power profiles in an ESL context. These criteria can be generally classified into spread sheet based approaches, power model and macro-model based approaches that often use low level simulations (RTL or Gate level). Actually, the proposed power models range from component-centric to transaction-based. Each of these types may in turn be either instruction-based or function-based or state-based. These power models are usually evaluated at the ESL using either simulation or simulation dump processing or post-processing techniques that rely on the useful data and properties extraction from a functional simulation. While tech-

niques for adding power information based on a separation of power and functional concerns methodology have recently emerged, annotation-based techniques that instrument functional models with power are considered as the widely used industrial methodology.

In the following sections, we present a synthesis of relevant research works and available tools and EDA methodologies at each ESL sub-level beyond RTL that deal with high-level power modeling, analyzing and optimizing issues. We will show how the major works have notably focused on finding reliable power estimation methods to mainly explore different hardware architecture parameters and configurations and to early determine the best balance between performance and power consumption. As far as we know, only very few works have targeted the exploration of power management solutions early in the design flow and there is no work that has targeted efficient low power architecture exploration in relation with a system power management strategy that controls this architecture at the ESL.

2.2.1.1 Functional/Algorithmic Level

Tiwari et al. [142] have proposed the first power consumption estimation method of a program. This method has been a reference for processor power modeling and is applicable to all types of processors (general purpose processor such as Pentium or PowerPC and dedicated processors such as DSP). They have introduced the concept of **Instruction Level Power Analysis (ILPA)**. They associate a power consumption model with instructions or instruction pairs. The power consumed by a program running on the processor can be estimated using an Instruction Set Simulator (ISS) to extract instruction traces, and then adding up the total power cost of instructions.

Although accurate, this method suffers from the high number of experiments required to obtain the power model and the need for an ISS of the target processor. Thereby, characterization of power instruction based power model can be very time consuming and may take several months especially for processors with complex instructions set.

JouleTrack [135], a tool for software energy estimation, is an example of an instruction-based environment that computes the energy consumption of a given software based on the approach of Tiwari et. al. The model of power dissipation has been derived from experimental measurements of the supply current of the processor while executing different instructions. It has been applied to StrongARM SA-1100 and Hitachi SH-4 microproces-

SORS.

Sinha et al. [135] show that for a simple processor model, taking into account only its voltage and frequency, this tool can give relatively accurate results. Therefore, it can be applied to estimate the software consumption of a simple RISC processor. Once the processor architecture becomes more complex, this approach is no more interesting since it generates significant estimation errors.

Several extensions were proposed for the Tiwari works in order to handle the case of complex processors and overcome the modeling time drawback [101] [102] [94]. They are based on a **Functional Level Power Analysis (FLPA)** methodology which relies on the identification of a set of functional blocks that influence the power consumption of the target component. The model is represented by a set of analytical functions or a table of consumption values which depend on functional and architectural parameters. Once the model is build, the estimation process consists of extracting the appropriate parameter values from the design, and inject them into the model to compute the power consumption. Based on this methodology, SoftExplorer [72] has been developed and included in the recent CAT [73] toolbox. It includes a library of power models from simple to complex processors. Only a static analysis of the code, or a rapid profiling is necessary to determine the input parameters for the power models. However, when complex hardware or software components are involved, some parameters may be difficult to determine with accuracy. This lack of precision may have a non-negligible impact on the final estimation.

In order to perform power profiling for a full SoC, HW Intellectual Properties (IP) must also be modeled. The instruction-based estimation can be extended to peripherals, using a functional IP model. In [83], authors propose to split the IP into an orthogonal instruction set, covering all its functionalities. Their core power evaluation technique relies on dividing the function of the cores into instructions and performing estimation using instruction level power models.

Bus system power modeling has particularly gained a great interest of several works on macro-model-based power estimation at this level [56] [120] [48] [106] [57]. A macro-model consists in an abstract model that encapsulates factors having a strong correlation with energy consumption for a given component and obtained through measurements on existing implementations with the help of low-level (RTL or gate-level) methods or tools.

It can also encapsulate factors having a strong correlation to energy consumption for a given component.

Caldari et al. [56] considered a simple AMBA AHB bus and decomposed it into the following components: an arbiter, a decoder, and multiplexing logic. Macro-models were created for these components using gate level analysis. These models were used to create a higher level instruction model for AHB power consumption. Four main activity modes were identified on the bus: IDLE, READ, WRITE, and IDLE with bus handover. An instruction set was created from all possible transitions between one of these states to another. Only dynamic energy is accounted by the macro-models created by Caldari et al. [56]. Leakage and clock energy consumption is ignored.

Function-based power estimation methods capture the inter-instruction effects and take into account user-defined functions available only when the software package is known. Therefore, they are considered as a good alternative of instruction-based methods. In [128], a function based power estimation method was presented for embedded software executing on microprocessors. For a given microprocessor core, authors build the "power data bank", which stores the power information of library functions and basic instructions. This phase is done using a power estimation tool that takes the user's program and test data as input, and predicts the power behavior of the execution of such program on the given microprocessor core. The power simulator can be at any level from transistor level to RTL. Then, to estimate the average power of an embedded software on this core, they use the execution information of the target software from program profiling/tracing tools. The total energy consumption and execution time are consequently evaluated based on the "power data bank".

By building a power state machine from the power profile, states can be considered rather than instructions or functions [46] [86]. A timed simulation is required to determine elapsed time in each state. Thereby, inactive phases as well as static power consumption can be taken in account.

Whereas all the mentioned works used ad-hoc simulation tools for macro-models and did not profit from available simulation platforms, there are other works that use an existing RTL platform to extract statistical power models for system-level power estimation. For instance, authors in [98] proposed a statistical power estimation method embedded in a SystemC code translator. For that, they use a VHDL to SystemC translation tool to

rapidly convert the RTL code into higher level of abstraction models. By adding power analysis capability to the translation tool, it is possible to obtain switching activity information (and power results) from simulation with a higher level of abstraction. Some parameters are expected to be defined in the target code in order to have reliable results considering different technologies.

Ahuja et al. [34] have recently proposed a methodology to create abstract statistical power models from cycle-accurate Finite State Machine with Datapath (FSMD) hardware co-processors and its use at system-level for power estimation [34]. Another example consists in the Chip Vision's Orinoco tool [5] enabling the analysis of the power consumption based on a compiler which extracts the control flow and the execution of the binary to collect profiling data.

ChipVision has recently developed PowerOpt a low-power system synthesis tool that analyzes power consumption at system level. It automatically optimizes for low power, while synthesizing ANSI C and SystemC code into Verilog RTL designs, producing a low-power RTL architecture. This tool exemplifies a trend in power estimation and optimization at the functional level using high level synthesis methods and tools.

2.2.1.2 Cycle-Accurate Level

In order to get a better trade-off between power estimation time and accuracy, several studies and tools have relied on Cycle-Accurate (CA) simulation techniques for evaluating system power consumption. A common method for power estimation at this level of abstraction is to integrate a power consumption pattern corresponding to each component into the architectural simulators. Then, the overall system power consumption is computed during the simulation at each cycle based on the occurrence of relevant components activities. Wattch [53], SimplePower [149] and Skyeye [58] are examples of Cycle-Accurate power estimation tools. These tools use micro-architectural simulators to evaluate the performance of each component in a system with the help of analytic power models.

Giving a system architecture mainly composed of a superscalar processor and a memory hierarchy, these tools aim at optimizing the processor micro-architecture for a given application as well as the memory hierarchy in order to find the best configuration for performance and consumption. In [39] and [38], authors propose a dynamic power model

selection scheme for Cycle-Accurate IP models. Computation effort among different SoC components is allocated at run-time for the best estimation time and accuracy trade-off.

Another Cycle-Accurate component-based simulation framework for energy consumption estimation and optimization has been proposed by Abril et al. [31]. Their approach relies on extending behavioral models of a SoC components with energy models that take into account operations executed per transition into the components state machines. In [105], Lee et al. have developed the *Power ViP* framework which is also built on a component-based approach to provide Cycle-Accurate power estimation for a SoC composed of Cycle-Accurate IP models. The characterization phase of a peripheral device power consumption values is based on the identification of the device relevant activities and is done at gate level. However, their method is not generic enough since different ad-hoc techniques have to be used to model power in each component. Concerning Cycle-Accurate MPSoC systems, the MPARM platform developed at the University of Bologna [43] presents an example of simulation environments dedicated for this kind of models. This platform integrates a power consumption model for each component, enabling hence accurate power estimation.

Although these Cycle-Accurate methods fairly give accurate power analysis results, they are criticized for their significant simulation and evaluation time required. In addition, hardware system level models are often designed for functional verification or co-simulation. These models are most of the time not Cycle-Accurate but functional models, precisely TLM models.

2.2.1.3 Transaction-Level

The first estimation performed with an Approximately Timed (or PVT) model within SystemC framework has been achieved in [70] and [71]. In this work, authors have presented a method of building transaction-based power models. Their approach is based on a hierarchical tree structure which resumes all the types and granularities of transfers between the different blocks as well as the possible containment relationships between these transactions. The power value of each transaction in the tree can be fixed or parameterized. An important part of their work deals with the characterization methodology that helps to deduce power consumption of coarse-grain activities at higher level and hence power values per transaction using the gate level simulation. Then, they have shown how

SystemC TLM-based simulation environment can be augmented with transaction-based power functions for power estimations.

In [41], a hybrid power modeling methodology has been applied to the main components of a MPSoC architecture for accurate power estimations at the PVT level. This methodology supposes PVT and Cycle-Accurate level models of the different components are available and relies on the identification of the pertinent activities that consume power for each component at both fine grain level for PVT models and coarse grain level for Cycle-Accurate Bit-Accurate (CABA) models. Power costs of coarse grain activities at the PVT level are deduced from those defined at the CABA level, and were characterized at gate level or with analytical models. However, Cycle-Accurate models for all the SoC components do not always exist and is considered as a major drawback of this approach.

Authors in [129] aim at finding a better trade-off between these two correlated aspects, the power model granularity and the system abstraction level. For that, they develop an accurate and fast power estimation virtual platform by combining Functional Level Power Analysis (FLPA) for hardware power modeling and a system-level simulation technique for rapid prototyping. The functional power estimation part is coupled with a OVPSim 2 simulator [21] in order to obtain the needed functional-unit activities for the power models.

Contrary to the mentioned works which mainly target exploration of efficient hardware architecture exploration, Lebreton et al. [104] propose a state-based power profiling for each component in an Approximately-Timed platform which is tailored for advanced DPM architectures. To the best of our knowledge, this is the only research work that deals with power gating and DVFS architecture management and exploration at the transaction-level. By considering advanced DPM and DVFS power architectures of each individual IP core, authors split the state of each core into a functional phase and a DPM mode. A functional phase is characterized by its energy measured at lower levels than TL and time duration (e.g. wait, read, compute). A mode is a particular DPM mode (e.g. on, sleep, off). With DVFS, voltage and frequency are likely to change, independently of the phase.

The proposed generic power modeling framework serves to instrument an existing functional SystemC/TLM platform with timing to perform power estimations and to derive power management policies in globally-asynchronous locally-synchronous (GALS) Network-on-Chip (NoC) architectures. In order to facilitate the instrumentation phase,

a library, named *tlm_power* has been developed. It is composed of a generic set of C++ classes to model the different functional phases combined with the DPM modes and to monitor power within a SystemC/TLM framework.

In addition to *tlm_power* library, a set of tools has emerged in order to ease power estimation and analysis at Transaction-Level. In [82], a SystemC class library is proposed to calculate the energy consumption of hardware described with SystemC TLM, and the power model was based on experimental results.

The PKtool [147] [61] [28] is a free open source class library, built as an extension of the SystemC language. The estimation approach on which it relies considers that transaction handling determines the dynamic behavior (operations) of a module. Thus, the main entities considered to compute power estimations are the TLM2.0 functions that realize transactions transport and their characteristic data (e.g. phase and generic payload input parameters or return status value). It provides C++ macros that monitor calls to these functions in order to update during simulation the dissipation contributions of each SystemC/TLM module. It allows associating to each module a set of power models that are linked to each function. It considers that transaction-related energy costs are derived by a macro-modeling approach based on low-level measures (e.g. gate-level).

The Aceplorer tool [3], a commercial tool developed by the Docea Power company, represents a post-processing analysis-specific tool. It requires creating functional scenarios according to the power model description specified in this tool. These scenarios are usually provided by the simulation of the functional SystemC/TLM model.

Some EDA vendors were conducted to enabling power estimation and analysis into their existing virtual platform (VP) tools mainly using annotation-based techniques. Mentor Graphics provides for example an additional timing and power analysis toolset to the Vista platform [150]. This toolset enables power models to be annotated into transaction-level models. Mentor Graphics power models are component-based and a power policy table is associated to each IP and resumes its characteristic power parameters.

Similarly, Synopsys has focused on the instrumentation approach of a SystemC/TLM VP. For that, it proposed a power estimation API used with the Component Creator tool [109], a feature of Synopsys's Innovator VP tool. This API accelerates creating transaction-level models and allows new IPs creation with clock, voltage, power state and power estimation interfaces. IP power parameters (clock frequency, voltage value,

and power states) are entered by the user before simulation using the Innovator graphical interface. Power equations used for power estimation at run-time are internally inserted in the created IP code. Power parameters are gathered data from lower level tools like Power Compiler, providing more accurate data once the implementation has progressed beyond the system-level. Contrary to a generic power dashboard IP which is used for total system power estimation and is integrated in the DesignWare System Level Library (DWSLL), a power manager IP that is required to control the different IPs' power interfaces at runtime has to be totally defined by the user.

In their new virtual prototyping tool Virtualizer [26], launched in 2012 and replacing their previous Innovator tool, Synopsys uses Tcl scripts to automatically annotate the functional SystemC/TLM platform with power consumption values and component-based state models read from Excel sheets. The power control interface between the power management unit and the different blocks has been let trivial so far. It simply consists in clock and voltage signals as well as an additional SystemC signal dedicated to synchronize the power management unit (PMU) activity and the functional block undergoing a state transition.

In addition to integrating power-aware methods and analysis capabilities into existing commercial TL virtual prototyping tools, coupling multiple simulators is another solution used by industrials and EDA tool vendors to cope with non-functional properties (power/temperature) analysis at Transaction-Level. For instance, a collaboration between STMicroelectronics, Docea Power and the Verimag Laboratory in the French Help project ⁴ has led to a coupled simulation of a SystemC/TLM model with the ATMI and ACEplorer power and temperature solvers [63] [52]. Power and temperature analysis is done during the SystemC/TLM functional simulation based on the stimuli sent by the SystemC/TLM platform, which in turn can take decisions based on the non-functional simulation. However, this kind of solutions do not provide any real effective help on the way to verify and check a power intent and its corresponding power strategy according to the expected functional and temporal behaviors of the system.

⁴the ANR Arpege HELP (High Level Models for Low Power Systems) Project bearing reference ANR-09-SEGI-006, <http://www-verimag.imag.fr/PROJECTS/SYNCHRONE/HELP/>

2.2.1.4 Using Model Driven Engineering Approaches

Several works have used MDE to cover different power related issues in embedded systems. Among these issues, early power estimation was widely addressed. Conversely, power management and optimization issues are still on the rise. They have recently emerged as a challenging MDE-centric research field.

First, MARTE [25] and SysML [17] UML profiles, have already defined power modeling semantics for high level power consumption characterization. In order to enable specifying power features and analyzing a system power consumption, MARTE proposes in its Hardware Resource Modeling and Non-Functional Properties packages, a power package (*HW_Power*) that enables specifying power consumption and heat dissipation for each hardware component. Moreover, it allows defining power supply components (*HW_PowerSupply* and *HW_Battery*), heat dissipation reduction components (*HW_CoolingSupply*), as well as extra-functional properties (*Non-Functional Properties (NFPs)*), such as power, current and voltage.

However, a major limit of this standard is that it only provides power consumption values that remain fixed all along the component use. This standard still misses semantics for the specification of dynamic power reduction and management techniques useful to perform power management solutions exploration. SysML proposes similar semantics for extra-functional properties (called *Type Values*) definition but also suffers from the same limitations as MARTE.

In [123] and [122], UML diagrams and profile for Schedulability, Performance and Time (named *UML/SPT profile*) [18] have been used to describe an embedded system. An UML-based tool called *SPEU* (System Properties Estimation with UML) was used only before the transformation step to perform analytical power estimations. Aiming at selecting the most adequate application and architecture modeling solution that fulfills the best energy, cycles and memory requirements, a Design Space Exploration (DSE) tool was used to automatically explore different solutions.

Authors in [132] and [73] do not explicitly look at MDE-based exploration, but rather focus on early and accurate power estimation. They have used the Architecture Analysis and Design Language (AADL) [126] to describe embedded application and operating systems. Populated with power models including operating system services overhead, the

Consumption Analysis Toolbox (*CAT*) was used to obtain power estimates. In [42], authors present a MDE-based methodology to automatically generate MPSoC system model descriptions at different simulation levels. The Gaspard [27] MDE environment based on the MARTE standard [25] was used in this work. The same MDE-based methodology was adopted in [143] to design and integrate in a non-intrusive way power estimators between hardware components models. Hence, required simulation code is automatically generated and used to estimate the system power consumption during simulation.

Some recent works have extended the MARTE profile with additional useful concepts for early power estimation and analysis in order to overcome its limitations. Arpinen et al. [36] have aimed at modeling Dynamic Power Management (DPM) aspects in embedded systems by proposing an extended DPM MARTE profile. Unlike [42] and [143], the MARTE allocation profile is used in [36] to associate application functionalities with system power modes and not hardware components. They define a power state machine (PSM) for each system component. Based on this PSM knowledge, different power system configurations are defined. A power system configuration (defined as a MARTE Configuration extension) groups the active power states of the system components when an application use case is running. Indeed, the application is modeled as a set of use cases. Each use case is allocated to a power system configuration. Whenever a specific use case occurs, the associated system power configuration is activated and the components power states associated with this configuration hold.

In the same way, Hanger et al. [89] have proposed a power consumption analysis view profile based on MARTE. They have defined stereotypes to specify a power model of the system components and the executed application tasks. Each element contains specific power features that are used to evaluate the system power consumption and to explore the optimal power solution.

The common drawback of [36] and [89] work is that they are still at a conceptual level and need a connection to simulation level for energy dissipation analysis. Alternatively, authors in [85] have recently proposed a multi-view modeling approach based on UML MARTE and SysML extensions. Their approach relies on adding specific separate views to specify power management techniques (namely a power view, a clock view, an equational view and a control view). By using transformations, analysis/tool specific models are then built in order to extract properties from the different views and enable the use of

specific analysis tools. In their work, a transformation of their multi-view model into a model for Docea Power's Aceplorer tool [3] for TLM post-processing power analysis has been elaborated.

Actually, in order to validate the different views and relationships between them as well as the impact of the power structure and behavior on system functionality and energy efficiency, a system functional view that models the hardware architecture behavior under the different application loads is mandatory. Such view has been indeed defined in Gomez et al. [85] for this specific purpose. In general, once the interaction between the different views is validated, a transformation into a power-aware SystemC/TLM platform source code is then possible. The additional modeling and verification effort required on the meta-models on the one side, and on the generated code on the other side, represents a well-known drawback of this kind of approaches.

Mostly, a pre-verified functional SystemC/TLM model already exists. In this case, directly adding power aware behavior to this source code and validating it would be faster and less tedious than the UML-based approaches. Furthermore, separation of concerns applied at the meta-model level is guaranteed unless it is translated through specific transformation rules into separation of concerns methods to be applied on the SystemC/TLM code. This task is not trivial and separation of concerns is often required to be validated again on the generated SystemC/TLM source code.

2.2.2 State-of-The-Art on Low Power Design Standards Use

Using the low power design standards (UPF and CPF) requires simulators as well as debug and analysis environments that understand the power architecture specification, infer, simulate and evaluate supply networks and power-aware behavior such as power shutoff, non-retained registers states and non-shifted logic voltage values.

Recognizing the importance of such a concern, some EDA tool vendors have come up with automation solutions for low power design, verification and exploration built on support for existing low-power design standards (UPF or CPF). For instance, Mentor Graphics provides the Questa Power Aware Simulator [12] that focuses on automated verification of a UPF-defined power intent starting from RTL. The Synopsys Design Compiler Synthesis tool has also enabled UPF synthesis. In [146], Archana develops an example of

a low power design using, on the one hand a complete Synopsys top-down generic flow and on the other hand, a Synopsys UPF synthesis flow for the same design. Then, the author compares area, power and timing monitoring results for both methods reported by the Prime power tool having as input the gate level netlist. It has been observed that the UPF synthesis provides better power savings and timing than the generic flow (either with multi-threshold [96] implemented power technique or with no low power techniques employed) against an increased area in the UPF synthesis due to the additional logic added to control the power domains.

Moreover, different industrial researches have addressed not only simple RTL power-aware simulation but mostly automation approaches and tools for power-aware verification ranging from static analysis techniques to simulation-based ones [64] [78] [130]. For instance, [65] and [40] have used UPF to describe the power design and perform simulation-based functional verification at RTL. Precisely, Bembaron et al. power-aware simulator relies on special Verilog behavioral models that are manually written and contain power aware functionality [40]. They trigger special events that the simulator recognizes and then performs specific actions on the target to reflect the corruption, save, and restore behavior. In [37], Bailey strongly emphasizes on the importance of checking state retention bugs since retention is the most power gating intrusive mechanism that may alter the system functionality if not well-chosen and well-defined. In order to analyse and verify for functional, electrical and structural correctness and completeness of a power architecture specification, Cadence has provided CPF-based Conformal Low Power (CLP) tools throughout the low power flow starting from RTL.

Conversely to manual traditional approach used to specify power-aware requirements and test benches, Trummer et al. have proposed in [144] a methodology for automated simulation-based verification of the power-aware design at system-level. In this work, automation focuses on automated parsing and analysis of semi-formal use case documents in order to automatically create a verification environment. The semi-formal use case documents are used to describe the power-aware design requirements in terms of functionality and power state. The generated verification environment launches test cases derived from these use cases invoking an equivalent behavior in the system during simulation.

While all these mentioned works address structural low-level power architecture properties verification, there are other high-level architectural power architecture properties

focusing rather on inter-power domain state properties as well as transitions in the power control signals. In [90], Hazra et al. have recently proposed an architectural power intent property generation methodology using UPF-extracted assertions. In their work, architectural UPF-based power intent properties are formally expressed using several pre-defined predicates related to abstract interpretations of the architectural power domains states. These architectural properties are automatically translated into assertions using the low-level signals. Their approach leverages the per-domain properties extracted from UPF specifications.

Note that these strongly correlated ad-hoc approaches addressing power-aware verification of properties extracted from low power industry standards lack a unified and well-defined low power verification flow. For that, the Verification Methodology Manual for Low Power (VMM-LP) published in 2009 by Synopsys [93] has provided detailed classifications and examples of different low power verification properties and bugs. It has also recommended specific methods to check and correct them.

From this state-of-the-art section, one can conclude that low power design standards, either UPF or CPF, have been only used at low-level design flows starting from RTL for early power optimization, verification and design exploration. In [69], the author outlines the importance of leveraging these standards to TLM and study the potential interaction between power-aware TL and digital RTL/Gate models.

Chapter 3

Overview of the USLPAF Framework

3.1 The Need for the USLPAF Framework	88
3.1.1 Capturing Power Intent at Transaction-Level	88
3.1.2 Power-Aware Modeling Issues at Transaction-Level	99
3.2 The USLPAF Structure and Features	113

CONTRIBUTIONS of this thesis can be resumed in the development of a common and unified framework for power-aware modeling and verification at Transaction-Level called USLPAF. Contributions including the methods and tools proposed in this thesis help achieving specific goals and dealing with some modeling issues at this level of abstraction. This chapter explains the problems coming with power-aware Transaction-Level modeling and the objectives of our proposed framework. It also outlines the general structure of the proposed framework as well as the key features of each of its base components.

3.1 The Need for the USLPAF Framework

3.1.1 Capturing Power Intent at Transaction-Level

3.1.1.1 What if the low power flow is extended to TLM?

In a power-aware design, the power intent is combined with the functional intent in order to manage power consumption. When compared with its traditional version, a power-aware

design exhibits two major differences: the first one is the added power intent specification overlaying the functional description. The second one is the incorporated power-aware behavior used to mimick the power-down/wakeup behavior and reflecting the impact of the specified retention and isolation strategies. Tools involved in the different stages of a SoC design flow have to correctly interpret power intent information. They also have to virtually create and infer the low-power structures specified by the power intent in order to enable power-aware simulation, verification and equivalency checking. Indeed, power-aware simulation modifies the behavior of a design to reflect low power design intent in power down and power up situations. Power-aware verification is needed to check the operation of the design under active power management, including the low-power structure, state retention and restoration on power-down and the interactions of subsystems in various power states. Power-aware equivalence checking is necessary to verify that each tool used throughout the design flow has interpreted the power and functional intents in the same way.

With the use of a power format file, either CPF or IEEE 1801 (UPF), for power intent specification at RTL, the functional intent becomes the combination of the RTL description and the power intent file. Then, throughout the overall design stages, the user defines low-power intent in one place instead of many tool-dependent places. In other words, simulation tools and other downstream tools for synthesis, verification, equivalency checking, and place and route have the same power format file as a starting point as depicts Figure 3.1. Nevertheless, all these tools must be power-aware, that is to support interpretation of CPF or UPF commands and translate them into the native tool commands. For instance, in order to model the retention behavior, the RTL code must be modified such that each register state is saved in an extra inferred retention state variable for the save operation on power-down and reinitialized from this variable for the restore operation on power-on. One can merely do this by writing a UPF code when a power-aware simulator supporting UPF is used. To show the impact of UPF commands on such a simulator behavior, consider the RTL and UPF codes shown respectively in Figure 3.2 and Figure 3.3. The *set_retention* command in Figure 3.3 represents the UPF command used to specify to which power domain the retention strategy will be applied and the always on (i.e. never switched-off) power net for the retention registers. By default, this command applies a full retention strategy to the specified power domain by converting all its registers to retention registers. The *set_power_control* command indicates the save and

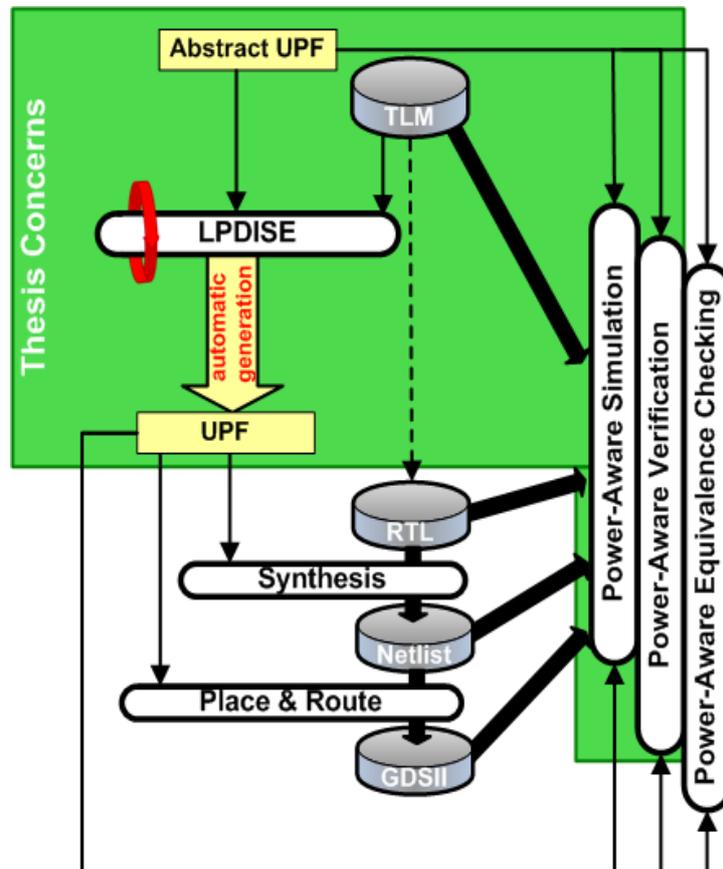


Figure 3.1: Extending the Low Power Flow to TLM

```

always @ (posedge clk or negedge nrst) begin
    if (!nrst)
        current_state <= 4'b0101;
    else
        current_state <= next_state;
end

```

Figure 3.2: RTL Functional Code Example

restore control signals.

Having both of these codes as inputs, the power-aware simulator will behave as if we had added to the RTL code of Figure 3.2 the two processes depicted by Figure 3.4.

However, when a simulator that does not support UPF is used, a script that makes these modifications to the RTL code must be added. The Figure 3.5 shows an example of

```

set_retention my_retention_strategy
    -domain my_power_domain
    -retention_power_net VDD_SOC

set_power_control my_retention_strategy
    -domain my_power_domain
    -save_signal {SAVE posedge}
    -restore_signal {NRESTORE negedge}

```

Figure 3.3: UPF Code Example for Retention Strategy Specification

script code added to the RTL for retention behavior simulation using an "ifdef" statement to control simulation. It is vital to point out that RTL power controller code which is responsible for changing save and restore control signals of a power domain is also added to the initial RTL whatever the method used for sticking power intent (ie. either using UPF commands or RTL scripts).

```

reg [3:0] save_current_state;
always @ (posedge SAVE) begin
    save_current_state <= current_state;
end
always @ (negedge NRESTORE) begin
    current_state <= save_current_state;
end

```

Figure 3.4: Code Added by The Power-Aware Simulator as Interpretation of UPF Commands

```

`ifdef RTL_PG_EMULATE
    reg [3:0] save_current_state;
    always @ (posedge SAVE) begin
        save_current_state <= current_state;
    end
    always @ (negedge NRESTORE) begin
        current_state <= save_current_state;
    end
`endif

```

Figure 3.5: Script Code Added in Case of a Non Power-Aware Simulator for Retention Behavior Simulation

As stated in previous chapters, by adding and simulating power intent starting from Transaction Level of Modeling (TLM), more significant power savings can be achieved since any virtual low-power management structure can be added and tested rapidly and easily. Hence, a challenging task of this thesis consists in enabling **Low Power Design Intent Space Exploration (LPDISE)** at this level of abstraction. LPDISE consists in exploring different power intent specifications of a SoC according to specific requirements of low power techniques. The aim is to early identify the most energy-efficient power management structure, including low-power structures as well as power management policies and architecture, while respecting functionalities required by the embedded application.

Performing LPDISE requires:

- Integrating Transaction-Level power models and estimation capabilities.
- Modeling units in charge of efficiently managing the low-power structure states.
- Modeling and specifying the power control network in charge of power control commands and information transmission between Transaction-Level blocks of a virtual platform.

Nevertheless, LPDISE requires first of all capturing power intent at Transaction-Level. To do so, the ideal way is to write a new UPF file with new low-power requirements at each LPDISE iteration while maintaining the same Transaction-Level functional code. But, some issues are encountered at this point: actually, Transaction Level simulation and verification as well as equivalence checking between TLM and RTL models must be power-aware. Unfortunately, according to the state-of-the-art works, none of the existing TL simulators is power-aware nor support power format files (neither UPF nor CPF). In addition, none of the previous works have dealt with such power-aware issues at Transaction-Level as stated in the previous chapter. Possible solutions to deal with this issue are either to add scripts to the TL functional code for power-aware behavior simulation (Figure 3.5) or to add power-aware interpretation capabilities to the standard SystemC simulator. The first solution corresponds to having many copies of the same functional code to simulate different power intent alternatives. This would slow-down LPDISE and would not be optimal. The second solution requires abstraction of some UPF semantics to fit the transaction level semantics and preserve a high simulation speed at this level. For instance, we believe that level shifters specified in UPF are not relevant at a Transaction-Level since they do not affect the functionality of the design. From a logical perspective they are just buffers. Then, their placement and properties as specified in UPF can be easily and statically deduced from the power domains partitioning and

features. Details and other examples on this point are given in further chapters.

To overcome these limitations, we propose to abstract the UPF standard semantics to specify power intent separately from the functional TL-model using a UPF-like methodology as illustrated by Figure 3.1. We focus as well on defining methods that enable power-aware simulation and verification of the power intent in conjunction with the functional intent. An abstract UPF specification will hence represent an input of the power-aware simulation and verification stages as well as an input of the LPDISE exploration phase. As shown in Figure 3.1, the output of this exploration phase is a standard UPF file automatically generated from the abstract UPF description of the most energy efficient and correct power intent alternative deduced at the Transaction-Level. Such a generated file eases the connection to RTL tools and represents an input of the classic low-power UPF design flow. It can be used as a low-power reference specification for RTL design teams.

3.1.1.2 What if a power domain-based reasoning is applied?

In order to rapidly and easily deduce RTL-based UPF semantics from abstract Transaction-Level UPF ones requires the adoption of the separation of functional and low-power concerns used by power formats at Transaction-Level. This methodology represents the backbone of the **Component-Based Development (CBD)** approach.

In CBD, software systems are built by assembling components already developed and prepared for integration. CBD has many advantages including more effective management of complexity, reduced time to market, increased productivity, improved quality, modularity and reusability. [140] gives a general definition of a component: *"a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."*

By defining and composing component interfaces, separation of concerns principle in CBD is achievable at the component-level as well as at the system-level. At the component-level only, many aspects either functional (such as computation and communication) or non-functional (such as timing and power) can be placed into separate components which are then composed and coordinated. According to system-wide coordination, the composed components communicate with each other via interfaces. When a component offers services to the rest of the system, it adopts an interface that specifies

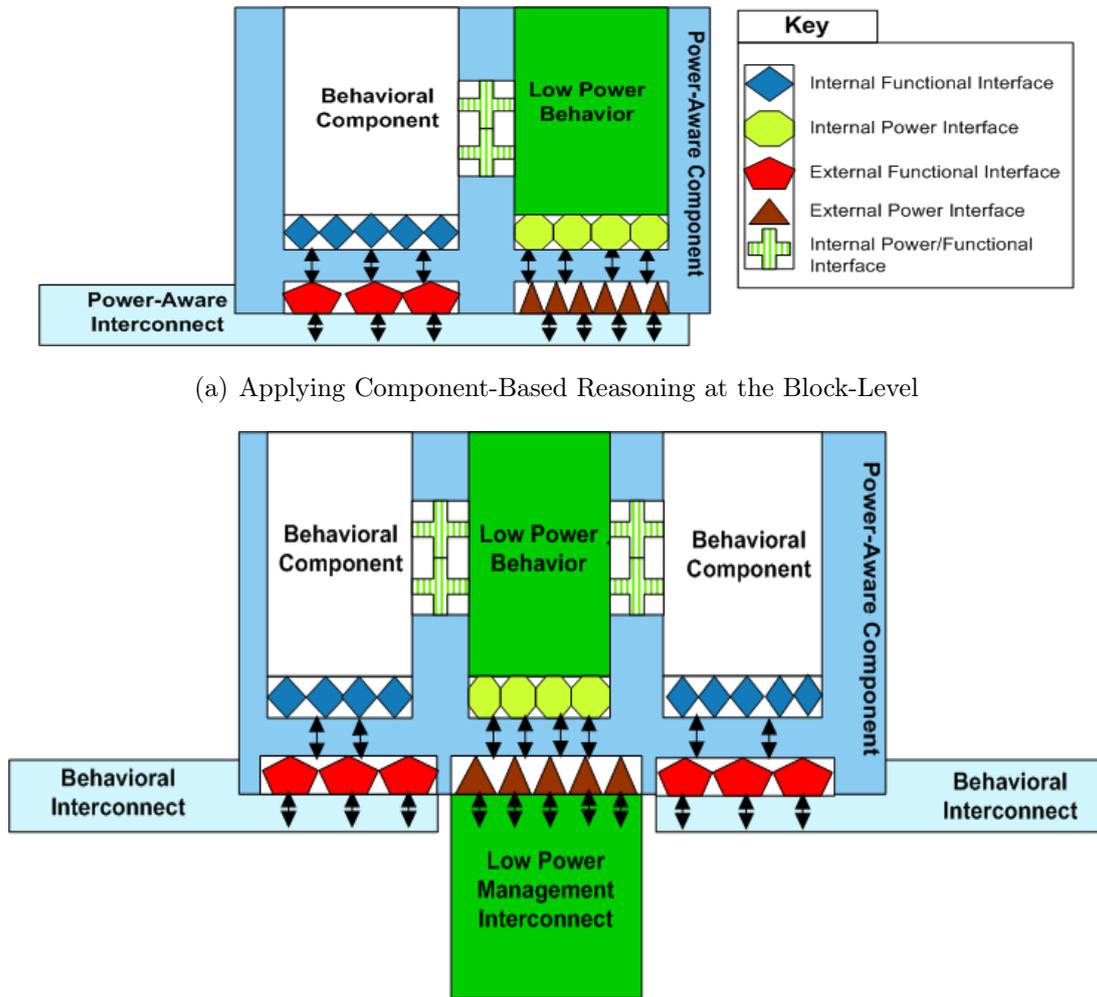
the services potentially used by other components, and how they can do so.

Component-based models have been widely used in the software industry such as **CCM** (CORBA Component Model) models [80]. But they have been also adopted in the hardware industry for the design of embedded systems. Intellectual Properties (IPs) are off-the-shelf hardware components proposed either as physical blocks (to be plugged directly in the hardware platform), or as software specifications (to be integrated during the design phase).

Actually, a close relationship exists between platform-based Transaction-Level Virtual Prototyping (TLVP) approaches and Component-Based Design (CBD) approaches as illustrated in Figure 3.7. Indeed, building Transaction-Level virtual platforms commonly relies on assembling pre-modeled and pre-verified software IP cores described in SystemC TLM. The various models (functionality and timing) of each IP are commonly related and combined as separate sources in order to control the simulation speed (that depends on simulation purposes). For instance, the OSCI standard TLM 2.0 [124] offers the possibility to switch between Loosely Timed (LT) and Approximately Timed (AT) modes of operation depending on the required accuracy degree. Furthermore, the TLM modeling approach highlights the concept of separating communication from computation within a system. Each hardware block in a TL platform consists in a SystemC TLM module. The behavior of such a module is internally modeled by a collection of concurrent processes and threads which determine the component internal state. Through a specific TLM communication structure, namely channel or interconnect, communications are established between SystemC modules according to a well-defined communication protocol. The TLM 2.0 OSCI standard [124] defines the base protocol for functional communication between behavioral TLM components through a memory-mapped interconnect. According to the component-based modeling approach, each SystemC TLM module can be seen as a behavioral component with two different interfaces as depicted in the Figure 3.6(a).

- **An external functional interface:** this interface is appropriate to the execution environment. It concerns functional sockets including ports, calls to methods of transport core interfaces for accessing a module as well as the functional communication protocol over the interconnect.

- **An internal functional interface:** this interface is appropriate to each component. It represents the set of registered callback methods associated with each functional socket. These kinds of methods are called whenever an incoming transport interface method call



(a) Applying Component-Based Reasoning at the Block-Level

(b) Applying Component-Based Reasoning at the Power Domain Level

Figure 3.6: Interfaces of a Power-Aware Transaction-Level Component

arrives on a component functional socket. They determine the current operational status of a component module.

Separation of functional and power concerns as defined by the CBD approach and the power format standards requires adding component-wise power-aware capabilities. To do so, each behavioral component is extended with a power-aware part that exposes three different interfaces as illustrates Figure 3.6(a).

- **An external power interface:** this interface determines how the components relate to each others in the complete system-level platform for both functional and power aspects. This interface mainly ensures power control information and commands transmission be-

tween components while maintaining a correct interplay between these communications and the existing functional ones. It consists in specialized TLM power sockets including a power-aware communication protocol, ports and methods call for transport core interfaces.

- **An internal power interface:** this interface consists in callback methods registered with each power socket. These methods are called whenever an incoming transport interface method call arrives on a component power socket. They define the power intent of a component model, determine its local power state and appropriately control the internal low-power behavior.

In order to support UPF standard semantics, the local power states of a component do correspond to the states of this component's power domain (usually expressed in terms of this domain supply nets). Transitions between these states occur upon a change in the external power interface and according to callbacks of the internal power interface.

- **An internal power/functional interface:** this interface is required in order to describe how the functional and power parts are assembled within each power-aware component model. In other words, a power-aware component represents the assembly of the behavioral component and the power-aware part as illustrated by Figure 3.6(a). This kind of interface is required to simulate the impact of low-power behavior on the functional one and to handle interaction and synchronization between them. This interface consists in either physical ports or method calls or both.

However, according to the UPF standard semantics, low-power elements and related low-power control signals have to be systematically specified in the context of a power domain. For instance, Figure 3.3 depicts a UPF code example for retention specification in which retention control signals are defined as inputs of the *my_power_domain* domain. According to changes in these signals states, the register states of these power domain's components are either retained or reset. Actually, behavioral components (i.e. SystemC TLM modules) within a power domain share the same low-power features. They also have the same power state since this latter is controlled in the same way. Therefore behavioral components within the same power domain require the same internal and external power interfaces.

This power domain reasoning imposed by the UPF standard has to be preserved when abstracting UPF semantics to TLM. Thus, it is rather wise and optimal to model a power

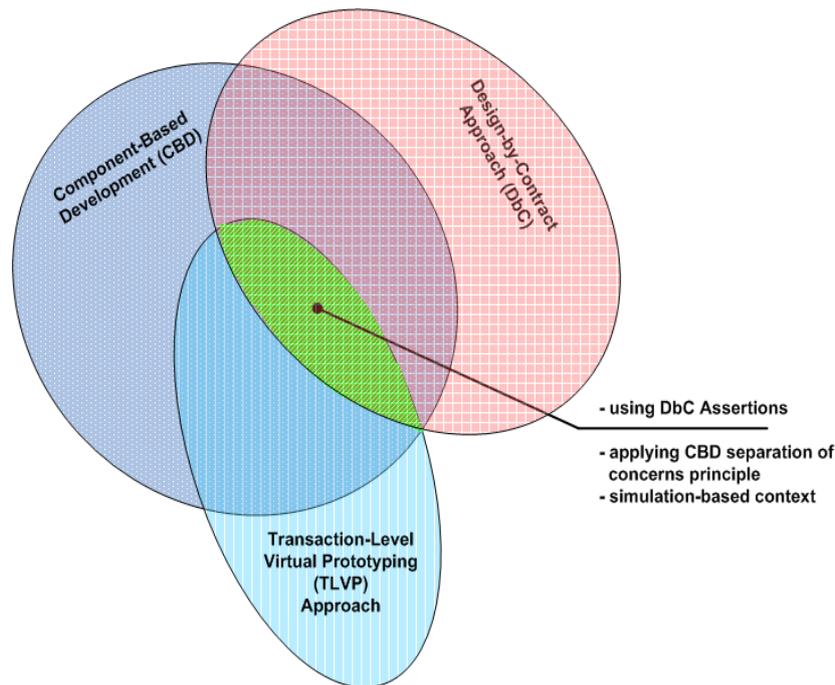


Figure 3.7: Relationship between DbC, CBD, and TLVP Approaches

domain as a power-aware component that gathers the behavioral TLM modules belonging to this power domain as illustrated by Figure 3.6(b). This modeling approach avoids the duplication of power interfaces for each SystemC module inside a same power domain. However, as shown in Figure 3.6(b), internal power/functional interfaces still have to be considered. These interfaces assemble functional and power behavioral aspects in the same power domain component.

In order to stickly apply the principle of power and functional separation of concerns, power communications have to be separated from functional communications as well. This can be achieved by adding a specialized power interconnect component. As shown in Figure 3.6(b), this component can be composed with power-aware components of a platform through external power interfaces and is in charge of handling power communications between power-aware components.

As illustrates Figure 3.7, Component-Based Design (CBD) approaches are usually used in combination with Design-by-Contract (DbC) approaches that were first proposed by Meyer [118] for the object-oriented language Eiffel [119]. Let go back to the general definition of a component given by Szyperski: *"a component is a unit of composition*

with contractually specified interfaces and explicit context dependencies only" [140]. This definition highlights the notion of contract as part of the notion of a component interface: to be able to compose components into systems, each component must provide one or more interfaces. An interface defines operations provided by a component and forms a contract between the component and its environment. A contract specifies the behavioral aspects of a component and is used to ensure some true conditions during execution of a component with its environment. Such conditions may represent functional or non-functional (QoS) properties of the different components' interfaces in a system. To support the composability of components whatever the execution context, dependencies between components must also be explicitly specified as contracts.

So, applying the Design-by-Contract (DbC) principle to the low-power behavior interfaces shown in Figure 3.6(b) would ensure behavioral coherence between functional and power-aware components. More precisely, a safe reuse and validation of both individual components (behavioral components) and a power-aware component (i.e. power domain) whatever the power-aware execution context would be guaranteed.

This dissertation actually studies the intersection between Component-Based Design (CBD), Design-by-Contract (DbC) and Transaction-Level Virtual Prototypes (TLVP) principles as illustrated by Figure 3.7. Identifying the relationships between the three approaches should come up with efficient and compatible modeling solutions for power-aware behavior simulation and verification issues at Transaction-Level. For example, TLVP approaches focus on simulation techniques a developer can observe the possible behaviors of the system hardware components when playing embedded software scenarios. For that, contracts of power-aware interfaces added to the initial TL simulation model must be specified directly into the TLVP code in the form of assertions. As a consequence, these assertions would be dynamically checked during simulation and contracts violation would be corrected through handling exceptions. The Chapter 4 goes into more details on the application of Design-by-Contract (DbC) principle to specify dynamic power-aware interfaces contracts into TL simulation models.

3.1.2 Power-Aware Modeling Issues at Transaction-Level

3.1.2.1 The accuracy problem

The accuracy of a Transaction-Level model is related to the levels of details captured by this model. This has a direct impact on power estimation accuracy and power management application as explained in the following.

Energy consumption information is usually available under the form of "so much consumption per unit of time". The total consumption is computed by an integration over the time. Therefore, the model must absolutely be timed so that power consumption estimation and analysis can be performed. As stated in the previous chapter, estimating power consumption with accuracy has been a major concern of ESL power-related researches as stated in the Chapter 2. Power consumption estimation bottleneck at the ESL is still achieving a good tradeoff between estimation accuracy and simulation speed-up. More accurate energy values are naturally obtained at the Cycle-Accurate Bit-Accurate (CABA) level than at an Approximately Timed (AT) level (i.e. PVT models). Indeed, at the CABA level, data transfer along a request or a response needs several cycles and power estimation is analyzed cycle by cycle. At the PVT level, this transfer is rather considered as an undivided operation and a power cost is attributed to the entire request or response packet. As a consequence, the specification of components' power models plays a primary role for reducing power estimation error between these two levels.

In this dissertation, the proposed approaches have naturally targeted timed TL models (the AT level). However, these models remain valid when they are refined to the CABA level.

Independently of the timing information level in a TL model, other performance factors may affect power estimation accuracy and have several impacts on power management opportunities. In particular, the details of communication bus protocol represent a significant factor. For instance, this includes the size of the data transported by a transaction. Single-word transactions can be accurately transferred using the real bus data width. However, when using the multiple data burst transfers capability instead, a block of transactions can be rapidly transferred. In this case, the burst transfer features must be indicated as transaction attributes.

The number of protocol phases needed to perform a transaction and their sequencing

rules represent another major factor. A communication can be modeled by a one-phase transactional protocol (see the Section 2.1.4 of the Chapter 2) [124]. In this case, a transaction transferred between an initiator and a target is blocking. When using this kind of transactions, the master will be blocked as long as the destination slave has finished processing the communication. To enable the bus pipelining feature, a multi-phase transactional communication using non-blocking transactions is required instead. In this case, an initiator can issue multiple transactions without waiting for the first transaction to be completed. In the case of a two-phase transactional protocol, a transaction is composed of a request phase and a response phase: one for sending the request to the target and another separate one for the target to send back the response. Actually, more phases can be used to model the bus more accurately. This feature impacts not only the power estimation accuracy but also power reduction opportunities and may consequently change the power management profile. Different energy savings can be obtained for different communication models of the same platform.

The use of transaction pipelines or/and large transaction bursts increases the bus throughput (or bandwidth). The throughput is defined as the amount of data in bytes transferred over the bus model in one second. In a timed Transaction-Level model, the bus throughput is correlated with the number of transmitted transactions. It can be measured by the number of bytes for each transaction, summing all related transactions and divided by the elapsed time for the related set. The bus throughput is a relevant feature to consider when applying power management on the bus component model. Indeed, a high-bandwidth traffic compared to the theoretical bus bandwidth indicates a frequent use of the bus component. This implies a high energy consumption of the bus and complicates its energy management.

Capturing the micro-architecture of a component in a model has also a non-trivial influence on the power estimation accuracy and the power manager functionality. The bus arbitration is one of the most critical features covered by micro-architecture. It defines a way to handle multiple requests and determine which master is allowed to access to a bus and when. It usually increases latencies of the transferred transactions resulting in more accurate power consumption values. Transaction latency is defined as the delay between the start and the end of a transaction. When multi-phase transactions are used along with bus arbitration, higher transactions latencies are likely obtained. Compared to a TL model including a Bus Functional Model (BFM) (i.e. without arbitration), the activity

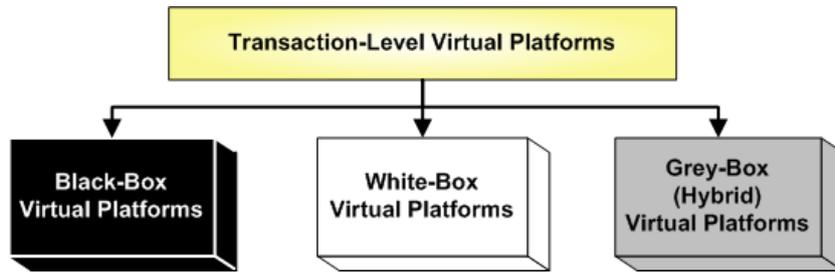


Figure 3.8: Different Types of Transaction-Level Virtual Platforms

profile of the simulation TL model would be changed and power reduction opportunities identified by the power manager would be different.

Another pertinent feature covered by the microarchitecture is the internal memory used by each component (internal FIFOs, depth of the pipeline, temporary memory storage). This feature is needed for low power specification and management at Transaction-Level, precisely for the specification of retention registers. In the Chapter 5, we highlight the importance of such information for applying state retention on power-down and achieving higher energy savings. Nevertheless, only modeling internal memories of a TL component does not allow the component state retention simulation in some cases. The considered virtual platform type strongly constraints this task. Figure 3.8 shows the different types of virtual platforms:

- **White-box virtual platforms:** a white-box virtual platform is composed of white-box components. A white-box component offers direct and complete access to its source code.
- **Black-box virtual platforms:** a black-box virtual platform is composed of pre-assembled commercial Intellectual Properties (IPs) distributed in binary forms for protection of intellectual property and trade secrets. This kind of IPs represents a black-box component with limited internal structure and status changes observability.
- **Grey-box virtual platforms:** a grey-box virtual platform is composed of both white-box and black-box components. The term hybrid is also used to designate this kind of platforms.

In the Chapters 5 and 6 of this thesis, we point out differences to add low power management features between white-box and black-box components. This is especially true for power estimation and management accuracy and flexibility. We also present

solutions to handle power management for both cases.

3.1.2.2 The power/latency trade-off problem

Power management is mainly intended to reduce the overall system power consumption. However, local minimization at each particular instant does not necessarily yield the best global result. This is due to time and energy penalties incurred upon power mode change of a component. For instance, to save power in lower power modes of a device or subsystem, an idle period of this device has to be long enough to compensate for the overhead of the power state change. The minimum idle time for which power can be saved is called the break-even time (T_{be}) [125] and depends on individual devices. Let us consider a device whose power state transition delay is T_0 (including high to low power mode transition $T_{h \rightarrow l}$ and the reverse transition $T_{l \rightarrow h}$) and the transition energy overhead is E_0 ($E_0 = E_{h \rightarrow l} + E_{l \rightarrow h}$). We suppose that its power in the high and low power modes states is P_h and P_l respectively. On the left of Figure 3.9, the device is kept in the high power mode; on the right side, the device is in the low power mode. The break-even time makes energy consumption in both cases equal. Namely,

$$P_h * T_{be} = E_0 + P_l * (T_{be} - T_0) \text{ or } T_{be} = \frac{(E_0 - P_l * T_0)}{(P_h - P_l)}.$$

The break-even time has to be larger than the transition delay. Therefore,

$$T_{be} = \max\left[\frac{(E_0 - P_l * T_0)}{(P_h - P_l)}, T_0\right].$$

As a consequence, the break-even time of a power domain, noted (T_{be_pd}), is a function of the break-even times, transition energy overheads, high and low power modes states of all this power domain's components.

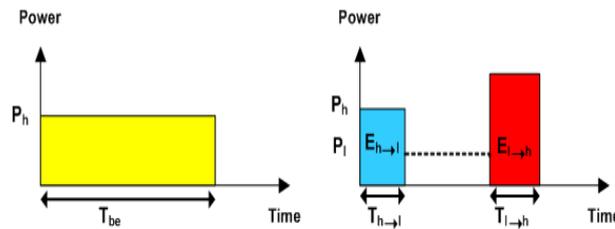


Figure 3.9: T_{be} Makes the Energy Consumption Equal [45]

Thus, when managing power domains states, if the idle time of a power domain is less than T_{be_pd} , changing its power state to a lower power mode will increase the power consumption of the system. Otherwise, it will reduce the power consumption. In other words, if the power domains of a system have a high T_{be_pd} , putting power domains in lower power modes during their idleness period may not be effective.

In spite of domain-level minimization, time and energy overheads due to dependencies between some power domains (i.e. case of functional dependencies between two components belonging to different power domains) may not lead to system-level power reduction. In order to save power while ensuring a correct behavior, a Power Management Unit (PMU) must carefully manage communications between power domains and implement an efficient power management strategy.

At Transaction-Level of Modeling, data communications between two components consist either in read or write transactions or in interrupt signals. If two components belong to different power domain, these communications would cross power domains boundaries. So, specific power modes of the communicating power domains are required before and potentially after such interactions. In this case, some additional power management events are required to notify the power management unit of an intended power domain communication. For instance, when a domain A communicates with a domain B, both domains must be in active power modes. Otherwise, the communication will fail leading to erroneous functional behavior of the system. To avoid such situations, they must be explicit in the model and, an adequate power management strategy is required. In order to avoid some inter-power domain communications (either transactions or interrupts), a power management strategy must carefully handle the synchronization between functional activity and power management one.

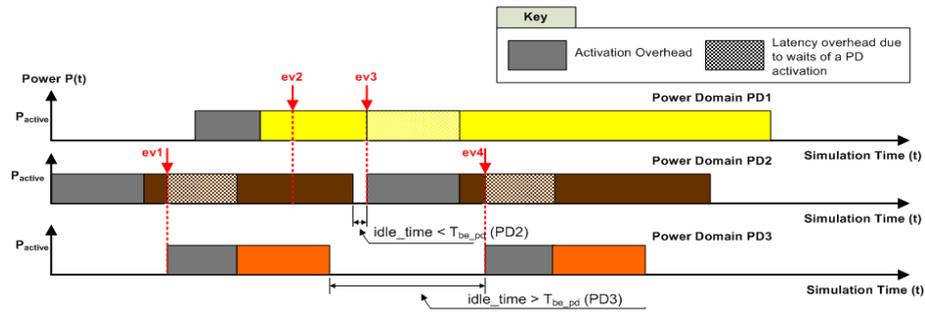
Ideally, a Power Management Unit (PMU) will block functionality of the communicating power domains as long as their required power modes are set. Although this strategy guarantees power management correctness, significant delays may be added. As a consequence, it may result to in short idle times less than the power domain break-even time T_{be_pd} leading instead to wasted power. Figure 3.10 illustrates the effects of this type of power management policy applied to three dependent power domains, PD1, PD2 and PD3, such that PD1 depends on PD2 and PD2 depends on PD3. As shown in Figure 3.10, PD1 communicates with PD2 by issuing the ev2 and ev3 power management events,

whereas PD2 communicates with PD3 by issuing the ev1 and ev4 events. Figure 3.10 also illustrates possible improvements to achieve more energy savings by reducing power and time latencies and highlights the complexity for adding power management features to the functional ones.

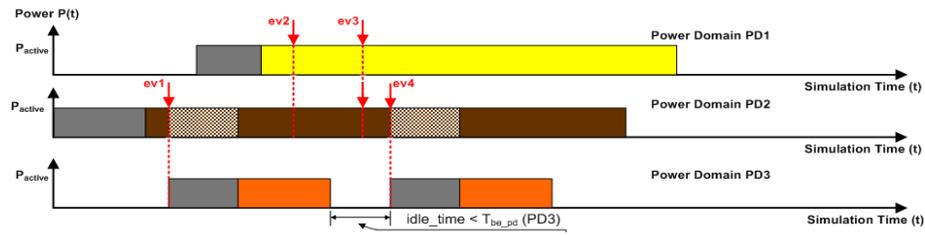
Figure 3.10(a) shows power profiles of the three power domains activated according to a reactive power management strategy. This strategy simply disables each power domain which is not in use. A power domain is only activated upon the notification of a power management event requiring a higher power mode for this domain. As depicted in Figure 3.10(a), when an active power domain needs to communicate with an inactive one, it has to wait for its activation. A non-trivial activation time would hence cause the waiting domain to remain in the high power mode longer, so resulting in wasting power. The added latencies can completely break the existing synchronization of components and requires to correctly synchronizing the new global model as it will be explained in the following section.

The worst situation occurs when such added latencies make some constraints (such as a particular QoS requirement or a real-time constraint) impossible to guarantee. Let us consider for instance a component C1 in a PD1 power domain that needs to perform some processing a given number of times within a fixed period of time. The decoding of the audio parts of a digital recording are usually synchronized with the video decoding by this means. If during this processing, C1 requests periodically some data from another component C2 of another power domain PD2 and needs for that to wait for the activation of PD2, the occurrence period of the processing in C1 will not be preserved. Another drawback can occur when using such power management strategies: a power domain may commonly be deactivated for less than T_{be_pd} , resulting in larger power consumption. This is the case of PD2 in Figure 3.10(a) for instance. PD2 is deactivated when it is idle, and then immediately activated as it is required by the PD1 (upon the occurrence of ev3).

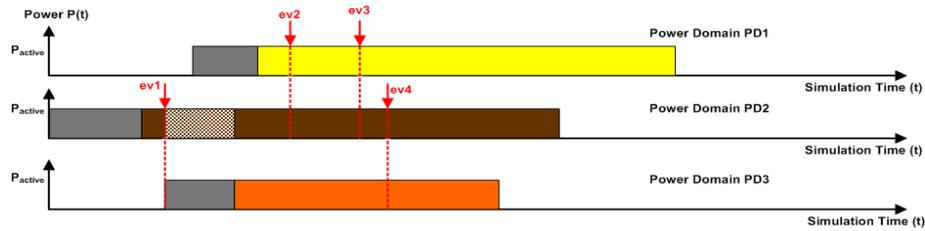
An obvious improvement of efficiency for such a power management strategy is to prevent deactivation of power domains if their idleness time is less than the corresponding T_{be_pd} . This can be achieved by exploiting the data flow in a given system and associating specific power modes for some power domains to a specific sequence of events. This method supposes that events usually occur in known patterns. Figure 3.10(b) shows the impact of such an improvement on the overall power consumption and time latencies. In



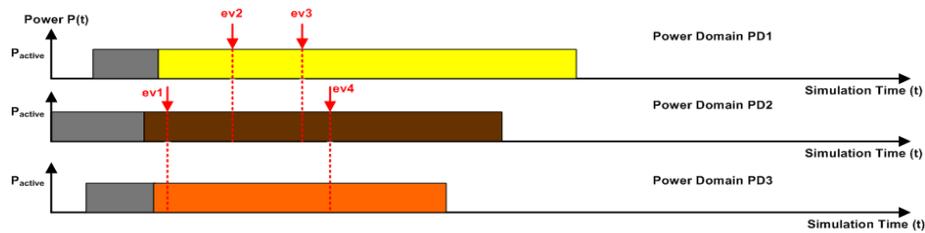
(a) Applying a reactive power domain management strategy



(b) Power domain management improvements to prevent quick power domain mode switching



(c) Iterative power domain management improvements are required to prevent quick power domain mode switching



(d) Power domain management improvements using prediction to prevent delays for resume time latencies

Figure 3.10: Impact of the power domain management strategy on handling time and energy overheads in case of depending power domains

this example, being notified of ev2 occurrence, if the PMU knows that occurrence of ev3 will arrive soon, it could keep the PD2 power domain in the high power mode. As it can be seen in Figure 3.10(b), this eliminates the PD2 short idle time observed in Figure 3.10(a). As PD3 does not wait anymore for the PD1 to resume its activity, the processing in the three power domains can finish earlier. However, this solution may lead to new short idle times appearing in other power domains activity profiles. As illustrated in Figure 3.10(b), eliminating the startup overhead of PD2 reduces the PD3 idle time to a level less than the $T_{be_pd}(PD3)$.

Actually, this kind of improvements must be done in an iterative way to prevent short idle times. For that, more event sequences are needed to be associated with specific power domains modes. This kind of iterative improvements is shown in Figure 3.10(c). Here, when ev2 occurs, if the PMU knows that ev3 and ev4 will soon occur, it ensures that the three power domains are in active states ready to receive this events sequence. This is how the short idle time of PD3 in Figure 3.10(b) is eliminated in Figure 3.10(c). As a consequence, the time latency due to PD3 activation is removed and the time spent in an active power mode by PD2 and PD3 is reduced by the activation time of PD3.

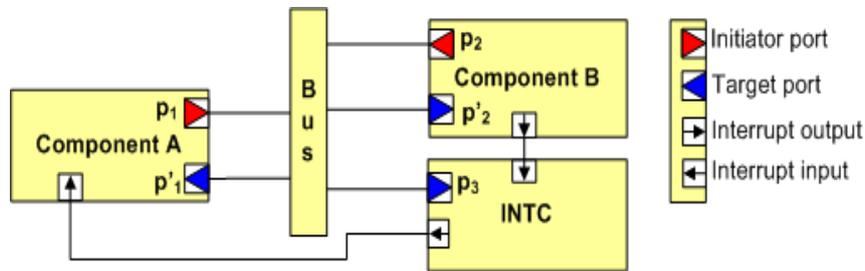
If the PMU predictively activates domains, a more optimal and efficient power domains management can be obtained by eliminating the switching time overhead as shown in Figure 3.10(d). As it can be observed on this figure, anticipating the PD3 activation before the occurrence of ev1 eliminates the wake-up time and energy overheads and shortens the required activity time of both PD2 and PD3 power domains. In addition, beginning the PD2 processing activity implies starting the PD1 activity after a certain amount of time. Instead of waiting first for its activation time, the PD1 can be powered-on before this time elapses saving hence time and power. This kind of prediction is employed by stochastic power management strategies [45][107]. In this kind of strategies, the PMU keeps statistics on the probability and amount of time to wait for each power domain's activation so that it can correctly anticipate necessary activations.

3.1.2.3 The synchronization problem

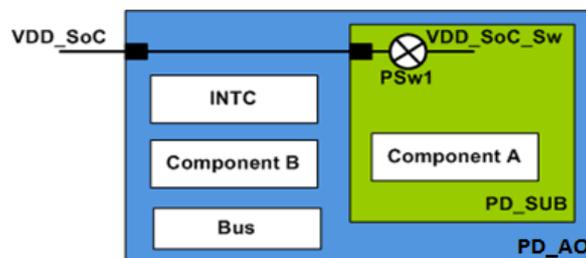
As previously explained, power management adds delays that can break the functional model synchronization. So, synchronizing the power management behavior with the existing functional one must be carefully performed in order to still respect the initial sys-

tem functionality as well as the performance constraints (e.g. QoS, real-time, energy-efficiency). The synchronization problem is specific to SystemC because of its cooperative simulation kernel. Indeed, when a simulation process runs, it is expected to execute a small segment of code and then return control back to the simulation kernel to allow other processes to run. Thus, SystemC simulation processes can be only synchronized through returning control periodically to the SystemC simulation kernel. This is done using the **SystemC wait()** function specifying either a time-out (using **wait(TIME_DELAY)** statements) or an event (using **wait(EVENT)** statements).

Depending on the wait statement, a functional synchronization inside a TL model can have two forms. The first form is timing-dependent and uses the standard **wait(TIME_DELAY)** method calls to constrain the execution order of SystemC processes. The second form is timing-independent and rather uses **wait(EVENT)** statements. To illustrate the potential impact of power management delays on these two functional synchronization forms, a basic example is proposed. We will use the functional TL model depicted by Figure 3.11(a) with two master/slave modules (components A and B), a behavioral bus and an interrupt controller. We associate to this system the power architecture illustrated by



(a) Functional Transaction-Level platform



(b) Power architecture alternative

Figure 3.11: Example TLM platform and corresponding power architecture

Figure 3.11(b). In this example, components A and B are respectively mapped in `addr2` and `addr1` addresses. Let us also consider that all components are put in an always-on power domain (i.e. which can never be switched-off) except for the component A which rather belongs to the power-gated `PD_SUB` power domain.

Figure 3.12(a) is an example of a timing-dependent synchronization form used in the functional TL model of Figure 3.11. Here, using `wait(20, SC_NS)` for component A and `wait(100, SC_NS)` for component B will make component B be executed systematically after component A. However, this execution sequencing may not be maintained when power management features are added. Let us go back to our example. The component B power domain (`PD_AO` in Figure 3.11(b)) is still always-on. However, the component A power domain (`PD_SUB` in Figure 3.11(b)) is power-gated and requires activation before any operation can be performed by the component A. However, the `PD_SUB` power domain can have high activation latency that possibly impact energy savings of a power management solution. Indeed, activation latency can have several sources such as initializing the component or restoring the values of its registers. It depends on the number and the types of switches around or within the power gated component as well as on the amount of data to be restored and the size of storage elements. The intuitive way to take into account such latencies in a power-aware SystemC simulation is to always block activities of all components belonging to a power domain with undergoing power mode change. This can be merely achieved using a SystemC wait statement on a fixed activation time delay

<pre> 1. // Process – Component A wait(20, SC_NS); 2. do_some_computation1(); </pre>	<pre> // Process – Component B 1. wait(100, SC_NS); 2. do_some_computation2(); </pre>
---	---

(a) Example of timing-dependent functional synchronization

<pre> // Process – Component A //wait component A activation 1. wait(200, SC_NS); //begin processing 2. wait(20, SC_NS); 3. do_some_computation1(); </pre>

(b) Adding power management latencies

Figure 3.12: Impact of added power management latencies on timing-dependent functional synchronization

as shown in Figure 3.12(b), line 1. Unfortunately, this is a poor and risky method. For example, adding the `wait(200, SC_NS)` in Figure 3.12(b) is dangerous because it would modify the execution order so that the component B can be executed before component A.

Actually, using timing-dependent synchronization may potentially make the TL models less robust and not faithful to the real chip if some timings are differing from the model in reality. Reuse and portability of the TL model would also be limited when the parameters or the embedded application are changed. Similarly, embedding constant power management delays in the TL code ties the TL to the timing of a particular power architecture implementation. As a consequence, the power-aware component is less portable or reusable. Even migrating an existing platform onto a next generation technology, where the power gating timing would be different, would require changes to the power-aware TL model (i.e. to its functional synchronization). This issue can be overcome using a request-acknowledge handshake to control a power domain state. Thus, `wait(EVENT)` statements will be used instead to model wait for power management delays. Unfortunately, even by doing so, conservation of the initial functional synchronization is still dependent on the amount of time elapsed during the power mode transition. For instance, the functional synchronization in Figure 3.11 would be conserved only if activation latency of the PD_SUB power domain in Figure 3.12(b) is less than 20 nanoseconds .

Although timing-independent functional synchronization is a more interesting synchronization mechanism, similar issues can be encountered when adding power management delays either using `wait(TIME_DELAY)` or `wait(EVENT)` statements. Figure 3.13(a) illustrates an example of timing-independent synchronization between components A and B of Figure 3.11. With this code, we have the following execution sequence:

- `do_some_computation2()`
- `do_some_computation1()`
- `do_some_other_computation1()`
- `do_some_other_computation2()`

In fact, the component A process yields back the control to the SystemC scheduler and waits for the `ev1` event to be notified. Therefore, the component B process is executed instead. Then, this process would wait for the `ev2` SystemC event once a write transaction

is issued to the *addr2* address. As soon as this transaction is received by the component A, the *ev1* SystemC event would be notified and the Component A process would be resumed. Issuing a write transaction to the *addr1* address by the component A causes the notification of the *ev2* event. As a consequence, component B process would resume as soon as the component A yields. In this example, components A and B are dependent since the execution of a component drives or requires the execution of the other one.

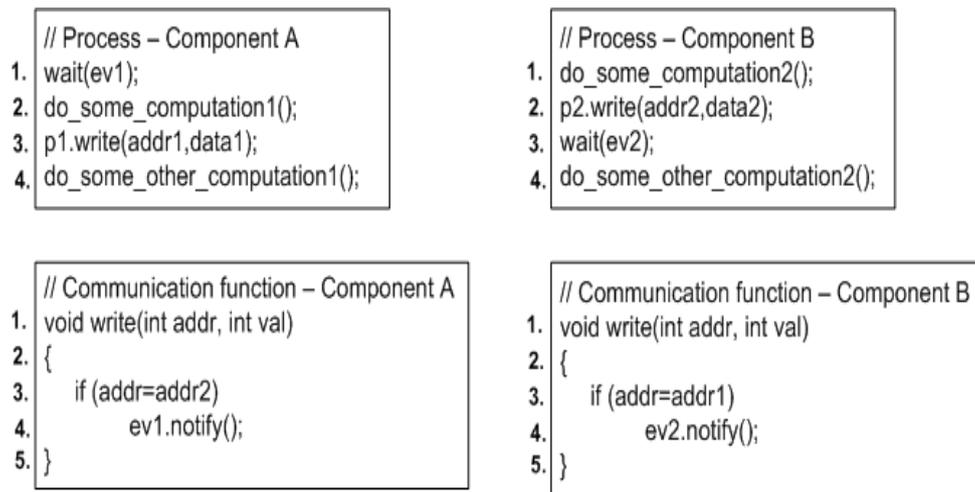
The typical error of such a functional synchronization is that when an event occurs, there is no trace of its occurrence than the side effects that may be observed as a result of processes that were waiting for the event. Thus, if no process is waiting to catch a triggered event, this event can go unnoticed. This kind of error can alter the intended system functionality and can even cause the SystemC simulation to starve and exit.

As adding power management delays has effects on execution order and time of processes, the following rule must be carefully respected in order to preserve the initial system functionality: *"to see an event, a process must be waiting for it"*. The simplest way to fulfill this requirement is to guarantee that events in the power-aware TL model are triggered in the same order as the functional version. Therefore, the processes must also follow the same execution sequence. However, this condition is not sufficient to guarantee the functional synchronization conservation. For instance, what would happen if a `wait(200, SC_NS)` statement precedes `wait(ev1)` in order to simulate the component A activation latency as illustrated in Figure 3.13(b)? In that case, if the `do_some_computation2()` function and the write to *addr2* method call inside the component B code are executed in less than 200 nanoseconds, the *ev1* event would be notified before the component A process would be waiting on it. As a consequence, while the component A process would be waiting for the *ev1* event to be triggered again, the component B process would be waiting for the *ev2* event that never occurs. In conclusion, both components A and B would end up in a deadlock.

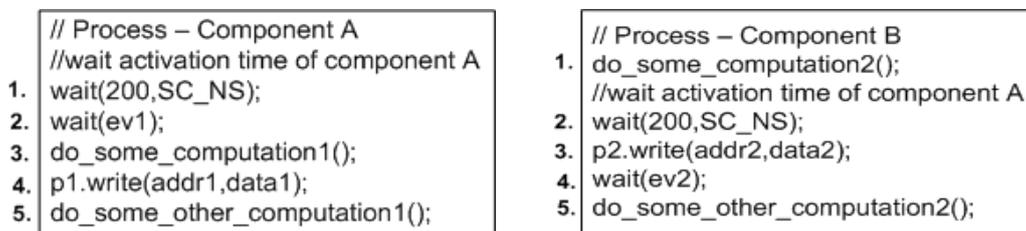
A possible solution to this problem is to block operation of dependent components whenever one of them is undergoing a power domain mode change and that is whatever their power domains membership. Again, this is intuitively done by yielding the control to the scheduler through adding `wait()` statements. The critical question is: what is the most suitable locations to place these yielding instructions? Let us go back to the example of Figure 3.13(b) illustrating a faulty power-aware synchronization. A possible

solution to this situation is to add a `wait(200, SC_NS)` statement into the component B code as shown in Figure 3.13(c). As the write transaction to `addr2` would incur the `ev1` notification, the added `wait` statement must precede this method call. Nevertheless, the added `wait` statement for the power mode change delay in Figure 3.13(b) is not necessary anymore and can be omitted.

This solution is definitely an application of the TLM general synchronization principle proposed in [62]. The author of [62] has defined two fundamental concepts for functional synchronization of components at Transaction-Level. The first concept is **System Synchronization Points(SSP)** defined as *"logical instant of the simulation which corresponds to the synchronization of two or more components"*. The second concept consists in **System Synchronization Mechanisms (SSM)** defined as *"Concrete, sequential piece*



(a) Example of timing-independent functional synchronization



(b) First alternative for adding power management latencies

(c) Second alternative for adding power management latencies

Figure 3.13: Impact of added power management latencies on timing-independent functional synchronization

of SystemC code in a component model, that corresponds to a SSP". SSMs of the models are used as locations where to place yielding instructions. Based on these two concepts, the line 2 of the Component B process code in Figure 3.13(a) is considered for example as the SSM that corresponds to the SSP "resume_componentA". In fact, the write transaction in the component A's register, having *addr2* as address, is actually meant to make the component A resume its activity, so this transaction accomplishes a synchronization. As shown in Figure 3.13(a), the yielding instruction `wait(ev2)` is placed after the SSM in order to let the other component perform its task. As illustrated in [62], this principle can be used to locate points in the simulation time where switching to the timed version of a pure TL functional model is performed. The objective was to enrich PV models with additional timing information (T model) to get a detailed timed model (so-called PV+T). The author of [62] argues that the switch between PV and T models can be correctly managed through intercepting SSMs. The synchronization problem encountered when building PV+T models is similar when timed TL models (i.e. PVT models) are enriched with Energy models (E models), so building PVT+E models, as being studied in this thesis. Therefore, the synchronization principle of [62] can be adopted in our case: the synchronization problem can be efficiently solved through intercepting SSMs to place

```

// Process – Component A
1. do_some_computation();
2. do{
3.   p1.read(addr3,data);
4. } while(data != 0x01);
5. do_some_other_computation();

```

(a) Using the polling synchronization mechanism

```

// Process – Component A
1. do_some_computation();
2. while (! interrupt)
3. {
4.   wait (ev3);
5. };
6. do_some_other_computation();

//communication function for
//receiving interrupt
7. void handling_interrupt (bool irq)
8. {
9.   interrupt = irq;
10.  if (irq) {
11.    ev3.notify();
12.  }
13. }

```

(b) Using the interrupt synchronization mechanism

Figure 3.14: Impact of functional synchronization mechanisms on power management opportunities

appropriate `wait()` statements needed for power management latencies simulation without breaking the existing functional synchronization. The solution shown in Figure 3.13(c) For instance consists in placing a yield instruction in line 2 before the SSM in line 3 as it has been earlier proposed.

Another issue of power-aware synchronization is related to the communication functional scheme that is modeled. Some functional synchronization mechanisms such as the polling may constrain the power architecture specification as well as the employed power domain management strategy. For example, in Figure 3.14(a), the component B's status register having `addr3` address is tested by the component A as long as its value is changed to `0x01`. Unquestionably, both components need to be in active modes during the polling period. Otherwise, functional and power management features will not be coherent. This synchronization mechanism is more expensive in energy consumption compared to interrupts mechanism. In Figure 3.14(b), the component A is now waiting for an interrupt. Hence, it would yield and wait for the `ev3` SystemC event (notified in the function handling interrupt reception) to resume. Meanwhile, the component A can be put in a lower power mode in order to save energy.

3.2 The USLPAF Structure and Features

In the previous section, we have listed challenges for capturing power intent at Transaction-Level. We have also argued some choices and directives to be taken when adding power management capabilities at Transaction-Level such as the application of a power domain-based reasoning as well as CBD and DbC approaches. We have shown that issues to consider when building power-aware TL models are various and complementary. The typical example is the impact of a power management strategy choice on handling power management latencies, thus on preserving the functional synchronization as well as on the obtained energy savings.

In this section, we introduce a complete collaborative framework to fulfil the challenges (ranging from modeling to verification) previously mentioned. Moreover, this framework comes up with reliable solutions for the different mentioned issues. As it covers and unifies in a single environment the power-aware modeling, simulation and verification aspects, this framework is called the Unified System Level Power Aware Framework (USLPAF).

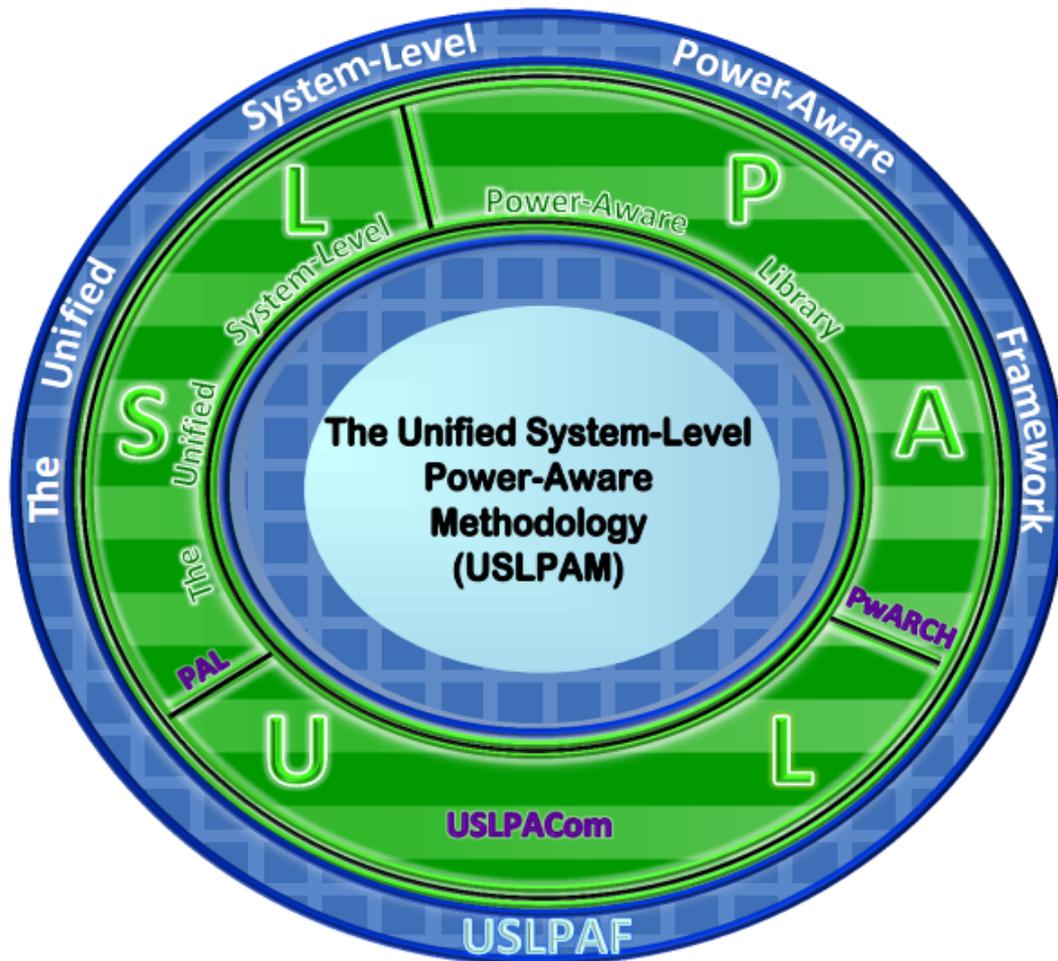


Figure 3.15: The Unified System-Level Power-Aware Framework (USLPAF)

In the following, we highlight the USLPAF structure and its key characteristics. Figure 3.15 shows the global structure of the USLPAF. Its two compliant parts are:

1. **The Unified System Level Power Aware Methodology (USLPAM)** is the heart of the USLPAF framework. It defines a well-structured design flow for:
 - Enriching functional TL models with power intent information and power management behavior.
 - Applying a power-domain based reasoning through modeling internal and external power interfaces, and power/functional interface of a power-aware component and correctly synchronizing power and functional behaviors.

- Extending the low power flow to TLM through the abstraction of the UPF standard semantics, the definition of methods for the LPDISE exploration stage, and the automatic generation of RTL-based UPF files from abstract TL descriptions of power intent alternatives.
 - Checking contracts for power-aware interfaces using assertions.
2. **The Unified System Level Power Aware Library (USLPAL)** is a set of software utilities for applying the USLPAM and is provided in the form of static C++ library. As depicted in Figure 3.15, this library includes:
- (a) **PwARCH** stands to **Power Architecture**. It is an Application Programming Interface (API) that abstracts UPF standard concepts as well as related power-aware behavior. PwARCH is used to apply the USLPAM on white-box types of virtual platforms.
 - (b) **PAL** is referring to **Power Aware Layer**. It is a set of classes facilitating the design of power-aware wrappers on top of functional TL modules. This utility is used for the USLPM application on black-box types of virtual platforms.
 - (c) **USLPACom** is referring to **Unified System Level Power Aware Communication**. It consists in a C++ class library extending the TLM 2.0 standard library to define a TL power domain management protocol interface called PDMgIF (standing for **Power Domain Management Interface**).

In the following chapters, features of each component of the USLPAF will be described in further detail.

Chapter 4

USLPAM: A Unified Methodology for System-Level Power-Aware Modeling and Verification

4.1	An Overview of the USLPAM Flow	118
4.1.1	The Software Flow Analysis Stage	118
4.1.2	The Power Management Points (PMPs) Identification Stage	119
4.1.3	The Power Intent Specification Stage	122
4.1.4	The PMU Modeling Stage	133
4.1.5	The Full Power-Aware Simulation Stage	149
4.1.6	The Power-Aware and Simulation-Based Verification Stage	149
4.2	The USLPAM Requirements	154

THIS chapter presents the general flow of the USLPAM methodology to add low power design, management and verification features to Transaction-Level Systems-on-Chip (SoCs) models. The fundamental principles on which this methodology is based are presented in the form of essential requirements. These requirements must be satisfied by each implementation of this methodology to adequately apply it.

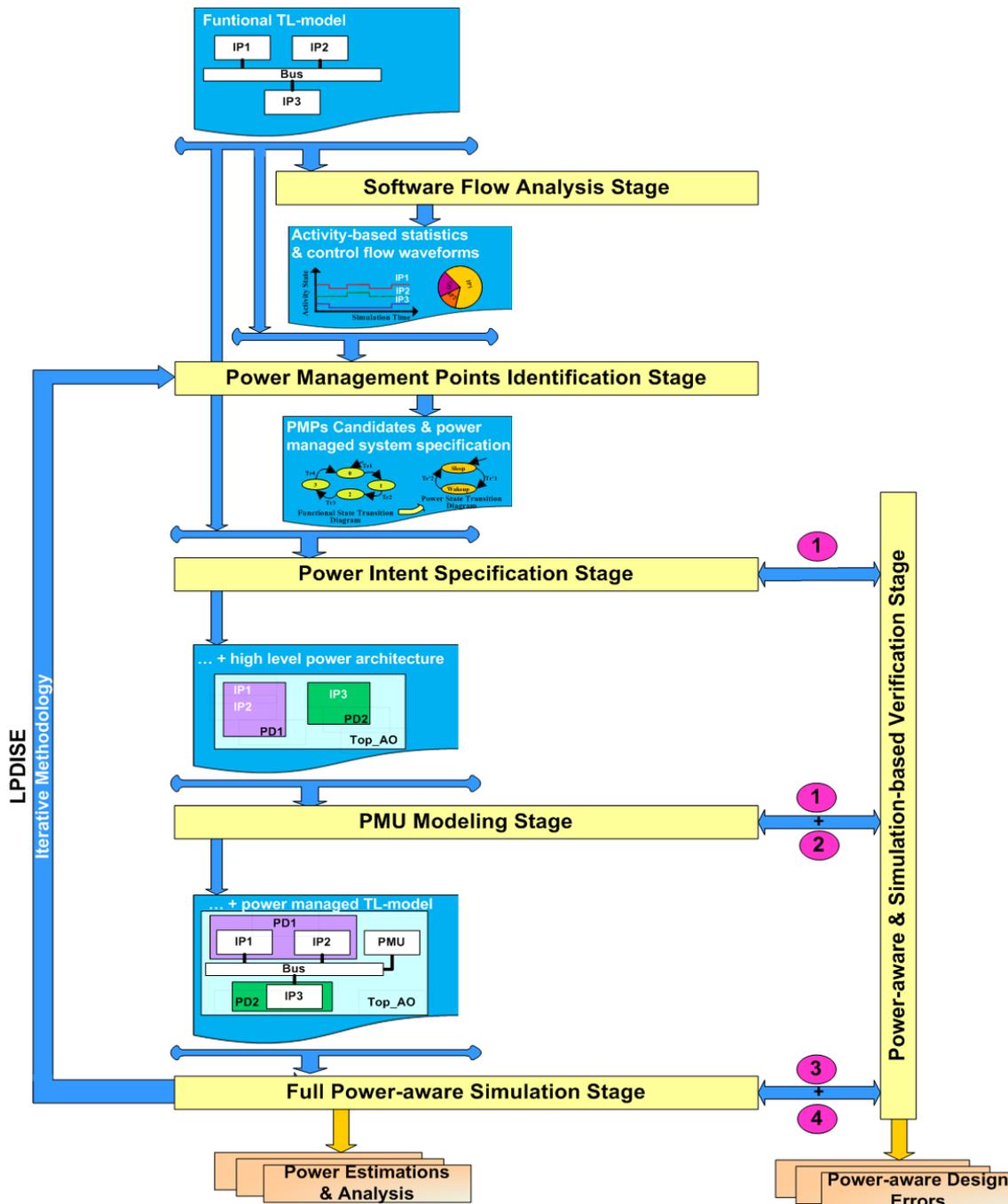


Figure 4.1: The General USLPAM Flow

4.1 An Overview of the USLPAM Flow

Figure 4.1 depicts the overall six-stage flow of the USLPAM methodology. As it can be seen, this methodology is composed of five successive stages and an orthogonal one that is dedicated for power-aware verification and is processed after each of the three last sequential stages. The main features and purposes of each stage are described in the following.

4.1.1 The Software Flow Analysis Stage

Given a functional TL model with no power features, the first stage in our methodology consists in analyzing how data are exchanged between the components of this model. A practical way to achieve this analysis is to proceed by simulation on representative test benches. However, if the application behavior could be described through a formal model (e.g. SDF graph), formal approaches could be applied to analyze data exchanges between components [49]. In this work, we consider a simulation-based approach to be able to cover all cases. This analysis stage allows the designer to understand when and how often each component is activated under different application scenarios such as watching a video, reading email or taking pictures in case of a smartphone TL model.

Capturing transaction traces also helps understanding the sequence of events, control flow and process scheduling. Thus, the designer can rapidly deduce dependencies between components activities and get an idea about possible correlations between hardware blocks under the embedded software execution. A typical example consists in putting strongly correlated components during simulation in a single power domain. Another example consists in putting a component that is frequently inactive for a period higher than its break-even time (T_{be}) in a power-gated domain.

Moreover, locating synchronization transactions at this stage is useful because it helps explaining components functional dependencies and determining potential candidate locations of added power control transactions. We define a synchronization transaction as a communication function call that causes a change in the activity profile of its target or source component when it occurs. It may consist either in a read or write transaction to a component internal register or simply in an interrupt signal. Indeed, synchronization transactions are part of the System Synchronization Mechanisms (SSMs) introduced in

[62].

Most of TL virtual prototyping tools (e.g. the Synopsys Platform Architect toolset and the Mentor Vista Architect toolset) provide various debug and analysis capabilities from which the developer can benefit to perform this first USLPAM stage. Alternatively, the synchronization principle for TLM proposed by Cornet [62] can be used to appropriately instrument the TL model code for waveform tracing of activity profiles per component. At the output of this stage, different alternatives of power domain partitioning and their related supply networks are determined.

For each power domain partition and supply network couple, memory storage elements that have to be saved during specific power domains power-down must be carefully identified. Legal code locations for power domains state change and power management behavior synchronization with the initial functional one must also be specified for each of these defined couples. The second USLPAM stage addresses these critical points.

4.1.2 The Power Management Points (PMPs) Identification Stage

The second stage of the USLPAM consists in defining a set of power management points (PMPs) based on the functional TL model description and the software flow analysis performed in the previous stage. We define a power management point as *a point in time where a power domain state is changed*. Given a power intent alternative (including power domains partitions and related supply network), a set of PMPs is assigned to each power domain.

This specification step prepares and eases the remaining USLPAM stages. On the one side, identification of PMPs helps in the design of the Power Management Unit (PMU) and the implementation of a power management strategy. Indeed, one can define a PMP as a possible location in the TL code where the power domain state is stationary and its state can henceforth be changed by the PMU. Obviously, PMPs are located between computations inside a power domain's components code. PMPs are specified by the designer based on an analysis of communication and computational patterns of each component in the underlying power domain. In some cases, their specification can be based on technical datasheets or specific workload requirements (i.e. QoS). Typically, consider the case of a power domain with different voltage levels. According to its technical datasheet, one of

this power domain's components may require to be supplied by the highest voltage power net in order to guarantee a high computing speed during a specific simulation period.

On the other side, PMPs definition is also useful to ensure coherence between existing model functionality and the added power behavior in terms of requirements for the model state maintenance between PMPs. Indeed, a PMP specifies potential power domain storage elements whose state must be retained before switching off this domain so that the TL model still operates correctly after this PMP is reached.

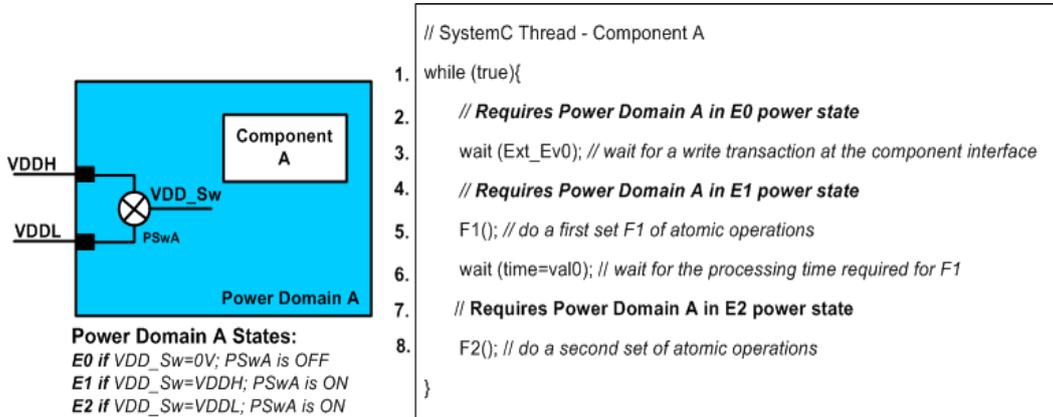
Actually, a power domain PMP, denoted $PMP(PD_i)$ where i denotes a particular power domain, is defined as a triplet:

$$PMP(PD_i) = \langle PwC_{candidate}(PD_i), Sleep_{candidate}(PD_i), Ret_{candidate}(PD_i) \rangle$$

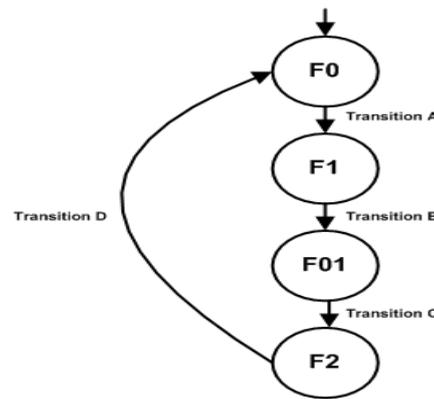
Where:

- $PwC_{candidate}(PD_i)$ is the **Power candidates set**. It is defined as the set of transitions from one system functional state to another on which the power domain state changes.
- $Sleep_{candidate}(PD_i)$ is the **Sleep candidates set**. It is defined as the set of transitions from one system functional state to another where the power domain can enter a sleep power state.
- $Ret_{candidate}(PD_i)$ is the **Retention candidates set**. It is defined as the set of couples (c, L) where c is a sleep candidate ($c \in Sleep_{candidate}$) and L is a list of retention storage elements. Here, the lifetime of each power domain state element is analysed with respect to each $Sleep_{candidate}$. Among different state retention approaches [96], replacing a standard register with a retention register is the approach used in this thesis. A retention register state will be locally saved during power-down and restored upon power-on as stated in the Chapter 2.

Let us consider the simplest case of a power domain A including a single component A shown in Figure 4.2(a). According to its supply network, the power domain A can be put in three different power states: E0 corresponding to a switched off power domain state, E1 corresponding to a power domain supplied with VDDH supply net and E2 corresponding to a power domain supplied with VDDL supply net. The SystemC thread pseudo-code in Figure 4.2(b) gives an idea about the component A functionality. Figure 4.2(c) gives the sequence of execution between the functional states of the Component A.



(a) Example of a Power Domain A Supply Network (b) Example of the Component A Functionality and PMPs Requirements



(c) Sequence of the Component A Functional States

Figure 4.2: Example of PMPs Specification

As it can be seen, the component has four different functional states. The first functional state corresponds to a wait state F0 (Figure 4.2(c)) in which the component A thread is blocked on the $wait(Ext_Ev0)$ statement (line 3 in Figure 4.2(b)). As soon as it receives a write synchronization transaction at its interface (leading to notifying the Ext_Ev0 event), the component A moves to the next functional state F1 (transition A in Figure 4.2(c)). This state corresponds to the execution of the $F1()$ atomic set of operations (line 5 in Figure 4.2(b)). It is followed by a second wait state F01 (transition B in Figure 4.2(c)) in which the component A thread is blocked in order to advance the simulation time by the required processing time of $F1()$ (line 6 in Figure 4.2(b)). As soon as this time elapses, the component A moves to the F2 functional state (transition C in

Figure 4.2(c)) in which it executes the $F2()$ set of atomic operations (line 8 in Figure 4.2(b)) and then blocks again in the wait state F0 (transition D Figure 4.2(c)).

According to this sequence of functional states, the PMU may put the power Domain A in the E0 power state (i.e. to power-down this power domain) before entering the F0 functional state (line 2 of Figure 4.2(b)). In this case, the PMU must activate the power domain A before moving from the functional state F0 to the functional state F1. In the example, we suppose that a high computing speed is required to achieve the $F1()$ set of operations (a requirement extracted from the Component A datasheet for instance). Therefore, the power domain A is required to be in the E1 power state before entering the F1 functional state (line 4 in Figure 4.2(b)). Conversely to the wait state F0 where a transition to a power-down state can be performed, the power domain A must remain active during the wait state F01. Then, as no power performance was required in the Component A datasheet for the F2 functional state, the PMU may put the power domain A in the E2 power state before moving to F2 (line 7 in Figure 4.2(b)). To resume, power management points of the power domain A include the transitions A, C and D as power candidates and the transition D as a sleep candidate.

As it will be discussed in the Chapter 5, another use case of power management points consists in validating some power-aware properties against PMPs specifications. An example of these properties is that any component in a power-gated power domain can resume activity after a PMP if only registers in this PMP's $Ret_{candidate}$ set have the same values as those stored before reaching the PMP.

4.1.3 The Power Intent Specification Stage

The previous stages could lead to the off-line specification of different power domain partitioning alternatives and the main features of each one. However, behavior related to state change of power elements in each alternative as well as its impact on the existing functional model behavior and on energy savings of the overall system have not been really specified or analyzed at these previous stages. This is rather done in simulation throughout the downstream stages of the USLPAM flow. In particular, the power intent specification stage starts by concretely adding to the existing TL model code, the appropriate power architecture elements that correspond to one of the alternatives specified at the previous stage. Furthermore, at this stage, behaviors of these added elements have

to be correlated with the existing functional ones. To do so, abstract UPF specification and simulation semantics that fit a transaction level of modeling are used. It is worth noticing that the focus of this thesis is on capturing abstracted UPF-based power intent at Transaction-Level. The primary goal behind this is to early perform LPDISE and generate a register transfer level UPF specification of the most energy-efficient power management architecture evaluated at Transaction-Level (Figure 3.1).

4.1.3.1 The Main Abstracted UPF Concepts at Transaction-Level

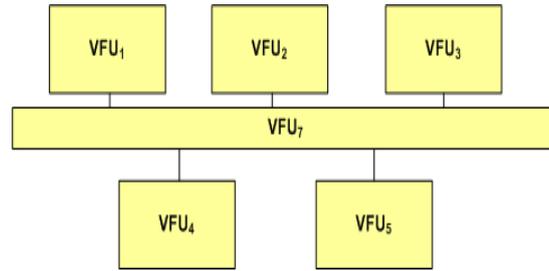
Figure 4.3 depicts the main abstracted UPF elements potentially involved in a TL power intent specification. It includes power domains, power switches, primary supply nets, retention supply nets, isolation supply nets, retention registers and isolation outputs. Each SystemC module in the TL model (the so-called Virtual Functional Units (VFU) in Figure 4.3(a)) belongs to a specific power domain. In addition, a hierarchical organization of power domains must be enabled as specified by the UPF standard semantics. As a consequence, two types of power domains can be modeled:

- A power domain of type "**container**" is composed of at least another power domain such as the PD_2 power domain in Figure 4.3(b).
- A power domain of type "**nested**" is included in a power domain of type container. For instance, PD_{21} is nested in the PD_2 container power domain. Nevertheless, a nested power domain can also be a power domain of type container such as PD_2 which is nested in PD_Top power domain and represents at the same time a container for PD_{21} and PD_{22} power domains.

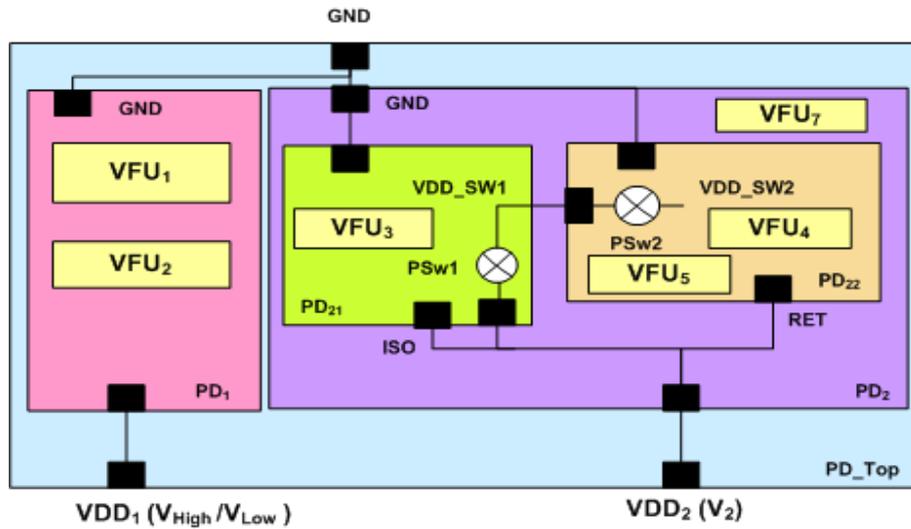
Similarly to the UPF standard semantics, concepts of voltage domain and power domain can be merged. By doing so, a power domain can be either power-gated or voltage-scaled or non-scaled according to its attached supply network (i.e. supply nets and switches).

- A **power-gated domain** has a power switch that provides the primary power supply. A power-gated domain can be powered down when all its functional modules as well as its nested domains' functional modules are unused. During a shutdown period, if retention supply nets have been specified for this type of domain, then they provide power to its retention registers to enable a fast state store and restore.

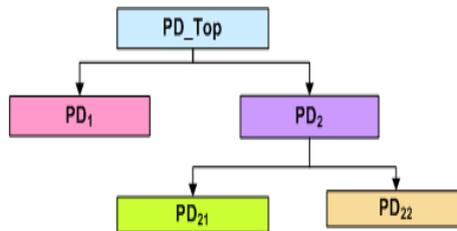
PD_{22} is a power-gated domain because its primary power net is the output supply net



(a) A Functional TL Model Example



(b) Adding Power Architecture Specification



(c) Power Domains Hierarchy

Figure 4.3: Abstract UPF Semantics For Power Intent Specification at Transaction-Level

of PSw_2 power switch (Figure 4.3(b)). Therefore, it can be completely switched off when the functional modules VFU_4 and VFU_5 are unused. Then, the RET supply net will be used to provide power to PD_{22} retention registers instead.

- A **voltage-scaled power domain** is either supplied by different primary supply nets of type power net, where each one has a different voltage value, or is supplied by a single

power net having different scalable voltage values. This type of power domains cannot be completely powered down but can be set in different low-power states according to the voltage value of its supply net(s).

PD_1 is an example of a voltage-scaled power domain (Figure 4.3(b)). As it has no power switch at its boundary, it cannot be completely switched off. However, its primary supply net VDD_1 has two possible voltage values V_{High} and V_{Low} . V_{Low} is used to set PD_1 in a low-power mode.

- A **non-scaled power domain** has a single primary power net with a unique voltage value. Once powered on, such power domains can neither be entirely switched off nor set in low-power modes. As an example, PD_2 is a non-scaled power domain as it has only one primary supply net VDD_2 with one possible voltage value (V_2) (Figure 4.3(b)). PD_2 is therefore called an "always-on" power domain (AON).

Usually, a top-level power domain, that does not contain any logic elements other than the root of the design, is defined. In Figure 4.3(b), the PD_Top is an example of a top-level power domain. The purpose of this type of power domains is to define the interface to the off-chip power sources and provide the top-level supply network.

Given this brief introduction to the basic abstracted UPF concepts in the USLPAF framework, it is not immediately clear how these abstract UPF elements would behave upon a power domain state request coming from the power management unit.

4.1.3.2 Inferring the Abstracted UPF Concepts Behavior to TLM

Figure 4.4 shows an example of the required power connections for the power switch PSw_1 in Figure 4.3. As it can be seen, a power switch component has naturally an input supply net (VDD_2 in Figure 4.4(a)) and an output supply net (VDD_Sw1 in Figure 4.4(a)). Its output supply net is considered as the primary power net of the power switch's power domain ($PD1$ in Figure 4.4(a)). Nevertheless, power control signals must also be defined for each power switch component. These signals connect the power switch to the power management unit and are used to control power to all the logic in the power domain functional components (e.g. to $VPU3_module()$ in Figure 4.4(a)). Each combination of these signals states defines a state of the power switch. In this way, upon a specific state change of its control signals, the power switch behavior can be specified including other power components behaviors such as the isolation cells and the retention registers.

For instance, in Figure 4.4(a), the power management unit de-asserts *sleep_in* to power down the *PD1* power domain and asserts *sleep_in* to power this power domain up. The signal *sleep_out* is the acknowledge signal that indicates that the switch has completed its power up/power down. According to the UPF-based definition of the *PSw₁* power switch shown in Figure 4.4(b), the *create_power_switch* UPF command can be used with specific options in order to specify the power switch supply nets (by using the *output_supply_port* and *input_supply_port* options), control ports (by using the *control_port* and *ack_port* options) and states (by using the *on_state* and *off_state* options). Note that this power switch *PSw₁* can be put in two different states: *ON* and *OFF* by respectively asserting and de-asserting the *sleep_in* control signal.

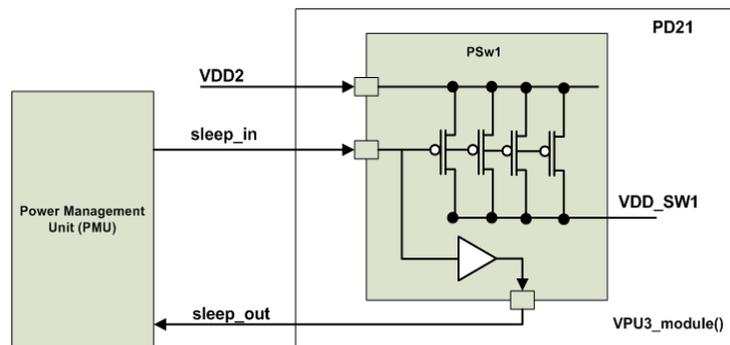
Similarly to these UPF-based semantics, behavioral aspects of the different abstracted UPF concepts must be defined and inferred to TLM. By referring to section 3.1.1.2 and Figure 3.6, the power intent specification stage focuses on defining the internal power interface as well as the internal power/functional interface for each power domain component, the so-called *power – aware component*. An example is given in Figure 4.5 to illustrate what these interfaces roles might look like. The example considers two SystemC TLM modules of the functional TL model (i.e. components A and B) gathered in a same power domain. Inside the power domain component, two interfaces and a low power behavior part are added. Each plays a specific role to make the components A and B aware of the specified power intent. Note here that the concepts of the power-aware component and its interfaces are somewhat abstract and their modeling techniques and mechanisms will be further described in the Chapter 6.

The example of Figure 4.5 only gives a brief sketch how internal power-aware interfaces can operate during power-down. As it can be seen, power intent specification of a power domain is defined by its internal power interface. In case of a power-gated domain, such a specification includes supply nets and power switches as well as the retention registers and isolated interfaces of the power domain functional modules. Upon the reception of a power management command on the external power interface, this command is first processed inside the internal power interface and potentially routed to the low power behavior part. This part uses information provided within the power intent specification to appropriately modify some functional components settings over the internal power/functional interface. In the power gating case illustrated by Figure 4.5, before changing the power domain state (e.g. by changing its power switch state), the internal power interface converts first the

sleep command into a series of function calls transmitted to the low power behavior part in order to handle isolation and retention. In the next section, we will further discuss the control sequencer responsible for such a specific sleep/wake-up function calls sequence. This step is needed to effectively determine what impact has the defined power intent on the initial behavior of this power domain components.

Power elements specified in the internal power interface includes information about the components' registers that need to be retained and the components' outputs that need to be isolated. This information is used by the low power behavior part to appropriately set required changes inside the power domain's components source code. As depicts Figure 4.5, all non-retained registers values are reset and randomized values are assigned to the outputs specified as isolated.

A key question any designer might ask is *"How to identify the right set of registers that must be retained or reset and the right set of components outputs*



(a) Power Connections of The PSw_1 Power Switch Component

```

0: create_power_switch PSw1
1:     -domain PD21
2:     -output_supply_port {VDDO VDD_SW1}
3:     -input_supply_port {VDDI VDD2}
4:     -control_port {SLEEP, sleep_in}
5:     -on_state {ON VDDI {SLEEP}}
6:     -off_state {OFF {!SLEEP}}
7:     -ack_port {SLEEPOUT sleep_out {SLEEP}}
8:     -ack_delay {SLEEPOUT 1}

9: map_power_switch PSw1
10:    -domain PD21
11:    -lib_cells /PMK/HEAD16
    
```

(b) The UPF Specification of The Power Switch PSw_1 Component

Figure 4.4: Inferring Power Gating Behavior to RTL Using UPF Semantics

that must be isolated?"

Actually, the power intent specification stage is strongly tied with the PMPs identification stage. On the one side, registers inside the PMPs $Ret_{candidate}$ sets have to be specified as retention registers at the power intent specification stage. The designer can even associate to each specified retention register a set of PMP identifiers. This is useful when applying a more refined power management scheme as it will be discussed in detail in the next chapter. On the other side, isolated interfaces could be automatically deduced from the power domain partitioning features. The Chapter 6, dealing with utilities provided in the USLPAF to create the power intent and coordinate its behavior with the functional model, will discuss automatic generation of isolated interfaces specification.

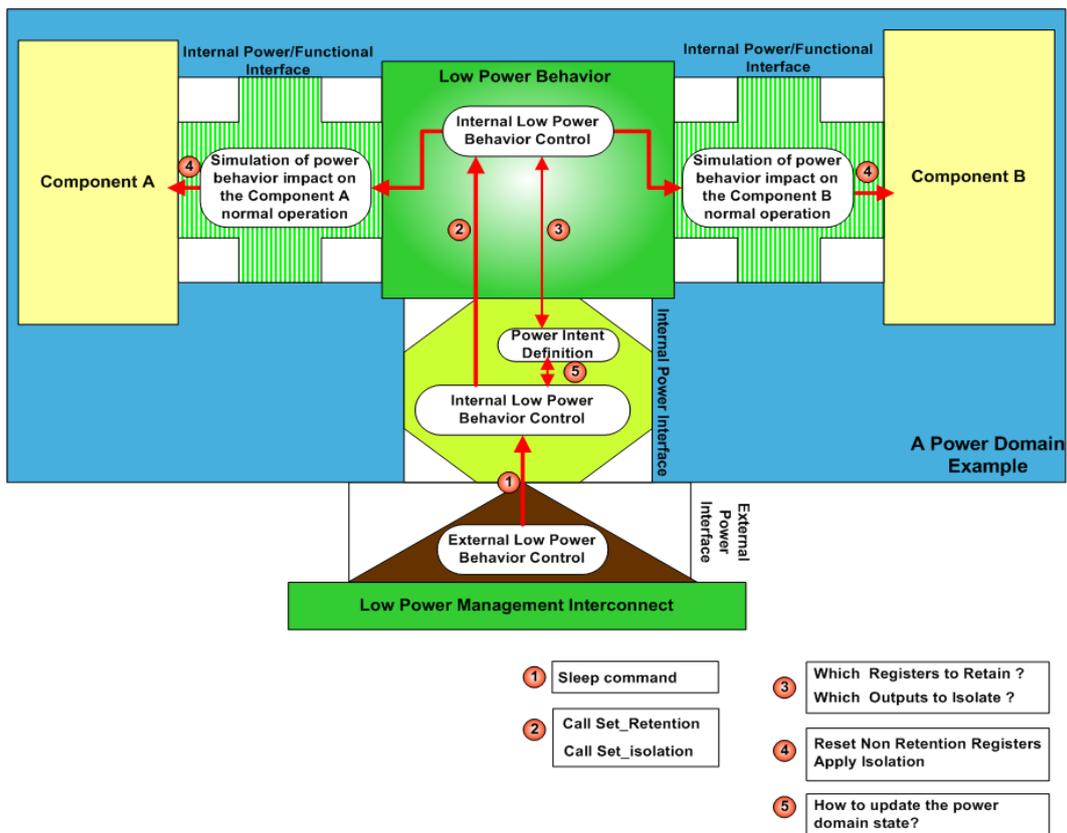


Figure 4.5: Example of the power-aware internal interfaces use during power-gating

4.1.3.3 Power Estimation Models

Among the mandatory modeling steps to be additionally performed in this stage is to couple the power-aware TL-model with a generic power estimation model in order to evaluate at runtime (i.e. in simulation) the power intent efficiency and its management alternative specified at each LPDISE iteration. The power domain reasoning adopted in power-aware interfaces design has also been used to achieve this power estimation goal. The idea is that components from a same power domain share that power domain characteristics. Thus, their power states correspond to their power domain's state. They are controlled in the same way and are changed simultaneously. So, automatically evaluating and updating power consumption values of each power domain when a *power event* is received would be a good and modular power estimation technique. A *power event*, shortly named PwE, is defined as an event that provokes a change in the power architecture state (typically in a power switch state or a supply net voltage) upon the reception of a power control command from the power management unit (PMU). Then, a power monitor can be defined to capture power events (PwEs) and automatically update appropriate power equations. More details on monitor modeling will be given in the Chapter 6. Let us now focus on the power models and equations used to estimate power consumption while guaranteeing power domain reasoning.

We consider that each SystemC module in the TL-model is characterized by an instantaneous dynamic power consumption $P_{DE_dynamic}$ parameter, given by Equation 4.1, and an instantaneous static power consumption parameter P_{DE_static} given by Equation 4.2.

$$P_{DE_dynamic}(t) = C' \cdot V^2(t) \cdot f_{clock}(inWatt) \quad (4.1)$$

$$P_{DE_static}(t) = V(t) \cdot I_{leakage}(inWatt) \quad (4.2)$$

Actually, Equation 4.1 and Equation 4.2 correspond respectively to Equation 2.1 and Equation 2.2 in the Chapter 2, the Section 2.1.5.1. Recall that, C' , f_{clock} and $I_{leakage}$ are constant and technology-dependent parameters that characterize a functional block implementation. These parameters may come from a datasheet or extracted from low level simulations (typically at Register Transfer Level or Gate Level). Therefore, they are specified during the power intent specification stage as constants. They are attached

to functional modules when superimposing the power intent design with the functional design and coupling their behaviors. Then, they are kept static during simulation and are used to re-evaluate Equation 4.1 and Equation 4.1 as soon as a related PwE occurs.

At the power domain level, each power domain has P_{PD_total} , $P_{PD_dynamic}$, P_{PD_static} and E_{PD_total} parameters referring respectively to its instantaneous total power consumption, instantaneous dynamic power consumption, instantaneous static power consumption and its total consumed energy. These parameters are updated when a PwE resulting in a PD power state change occurs.

Nevertheless, the UPF-like hierarchical construction of power domains complicates the implementation of this power estimation method. In this case, updating power consumption values during simulation must be performed carefully. Gathering a set of functional blocks and/or other nested power domains into a single power domain implies that all these elements share the same power characteristics and are all influenced in the same way by a change of state in their PD container. So, when a PwE occurs resulting in one or more power domains state change, a recursive update of power and energy parameters is needed. More concretely, let us consider the power architecture example in Figure 4.3. A power state change of PD_{22} occurred at a PwE instant t_1 induces an update of this domain power consumption first, then its container PD_2 power consumption followed by an update of PD_Top power consumption (container of PD_2). On the other hand, computing the overall energy consumption until a PwE occurs requires an update of PD_{22} , PD_{21} , PD_2 , PD_1 and PD_Top energy values in this order.

To have power consumption per power domain, P_{PD_total} , $P_{PD_dynamic}$, P_{PD_static} and E_{PD_total} parameters will be calculated as indicated in Equation 4.3, Equation 4.4, Equation 4.5 and Equation 4.6.

$$\forall j/0 \leq j \leq NbPD$$

$$P_{PD_total}^j(t) = P_{PD_dynamic}^j(t) + P_{PD_static}^j(t) \quad (4.3)$$

$$P_{PD_dynamic}^j(t) = \sum_{i=0}^{NbFM(j) \neq i} P_{DE_dynamic}^i(t) + \sum_{k=0}^{NbNES(j) \neq k} P_{DE_dynamic}^k(t) \quad (4.4)$$

$$P_{PD_static}^j(t) = \sum_{i=0}^{NbFM(j) \neq i} P_{DE_static}^i(t) + \sum_{k=0}^{NbNES(j) \neq k} P_{DE_static}^k(t) \quad (4.5)$$

$$E_{PD_total}^j = E_{PD_total}^j + [P_{PD_total}^j(lastT^j) * (CurT - LastT^j)] \quad (4.6)$$

Where:

NbPD: total number of power domains in a system

NbFM(j): total number of functional modules of the PD number j

NbNES(j): total number of nested power domains of the PD number j

CurT: simulation time when a PwE occurred

LastT^j: last update time of the PD number j (*PD_j*) power values

Given the power/latency tradeoff problem exposed in the previous chapter, energy transition penalties must be considered when updating the total energy consumption values. Figure 4.6 compares the power consumption behavior for the same device without power gating, with power gating but without retention application, and finally with power gating and retention application. When operating without power gating, the device has a constant leakage current in sleep power mode (top of the Figure 4.6). Using power gating reduces the leakage during the inactive state to zero. However, additional dynamic power consumption corresponding to the transition overhead must be considered (middle of the Figure 4.6). This overhead is due to the time and energy penalties induced by the switching fabric to power-on or off the power domain.

Typically, each switching fabric has hundreds (or more) switches acting in parallel as shown in Figure 4.4(a). Thus, the control signal from the Power Management Unit (PMU) to the switches is daisy-chained. This means that the control signal from the PMU (e.g. the *sleep_in* signal in Figure 4.4(a)) is connected to the first switch and it buffers (with an appropriate delay) the signal and sends it on to the next switch. As a consequence, it takes some time from the assertion of a "power up" control signal (e.g. the *sleep_in* signal in Figure 4.4(a)) until the power domain is effectively powered up. That is, all the registers resume their normal operation and all the continuous assignment and combinational processes resume. At this time, an acknowledge control signal is set high (e.g. the *sleep_out* signal in Figure 4.4(a)), informing the power controller that the power domain state is completely set. This acknowledgement time delay depends on the technology-specific switching fabric used. Indeed, specifying explicitly which power switch cell is to be used for the corresponding switch component (by using the *map_power_switch* UPF command shown in Figure 4.4(b)) justifies the fact that this cell delay can be recognized

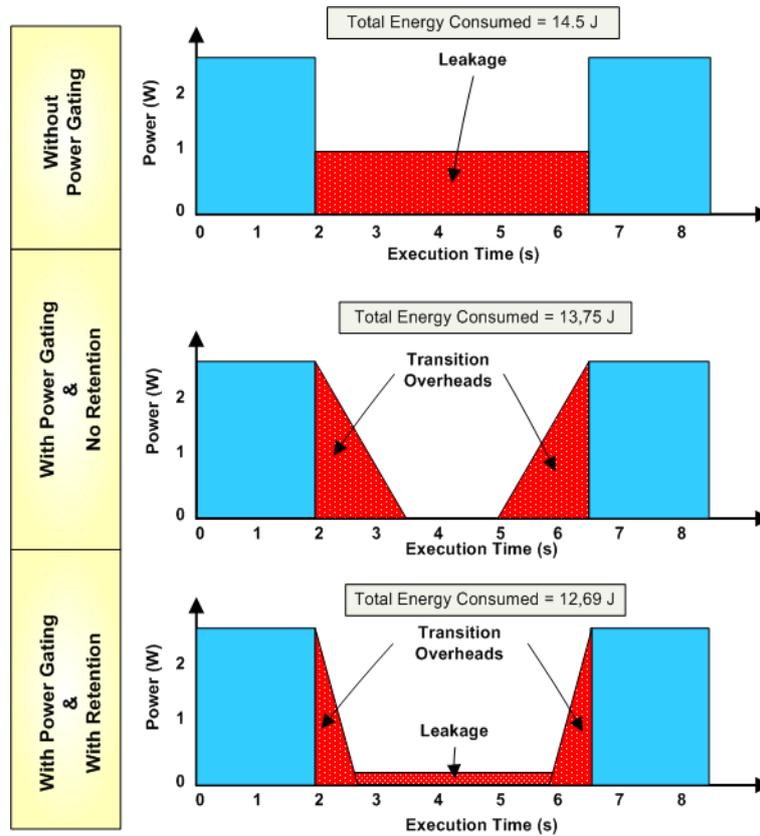


Figure 4.6: Comparison of Energy Consumed With/Without Power Gating

in advance.

Another primary contributor in power gating transition overheads is the storing of the information in an external storage memory before entering a low-power inactive state and restoring it after a wake-up event. Alternatively, when each standard register, whose state requires to be saved during power-down, is replaced with a retention register, this register state will be locally saved and restored instead. Hence, transition overheads are widely reduced and can even be neglected (bottom of the Figure 4.6). However, using this register-based retention approach results in a non-null power consumption during the sleep power mode. This leakage is due to the fact that shadow registers in retention registers must be powered by an "always on" supply rail during power gating. Nevertheless, this retention approach still saves significant amounts of time and power during power-up and power-down as shows Figure 4.6.

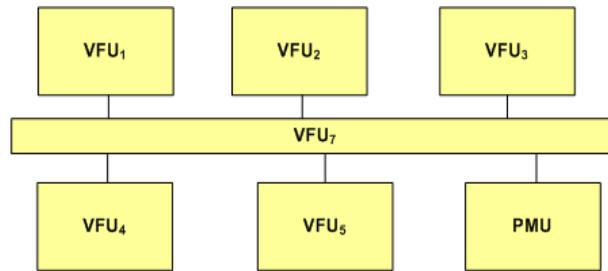
In our power-domain-based power equations, we consider that a constant duration

t_{switch} is required for a power domain to switch from an active mode to an inactive one and vice-versa. This duration corresponds to the time required by a power switch component to change its state between wake-up and sleep power modes. This consideration is conformed to the power switch simulation semantics defined in the UPF standard. Indeed, UPF supports assigning a delay for the acknowledge signal using the *ack_delay* option with the *create_power_switch* UPF command as shown by Figure 4.4(b). The transition overhead in terms of energy value either in a power-up or a power-down is computed as the product of the power consumption value just before the power mode transition and the t_{switch} divided by two as shows Figure 4.6.

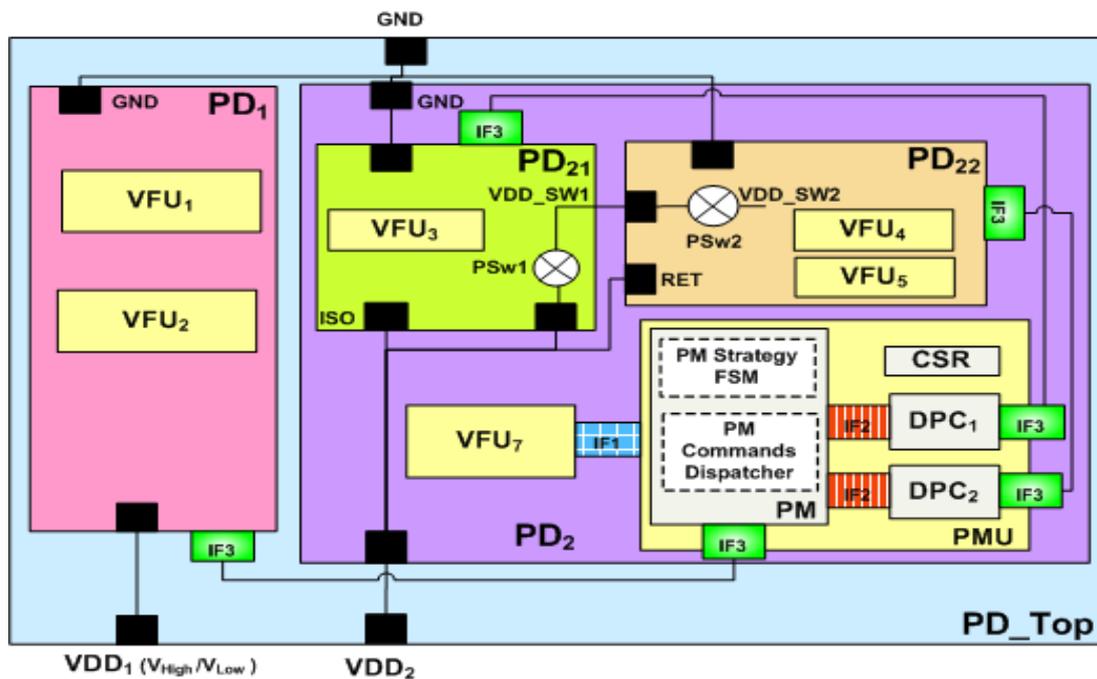
The leakage dissipation due to retention during a sleep power mode is computed based on a *Ret_Factor* parameter assigned by the designer to each power domain and based on lower level simulations. Multiplying the power consumption in the previous active mode by this *Ret_Factor* gives the power consumed while in a sleep state. *Ret_Factor* is a value comprised between zero and one and depends on the number of retained registers inside a power domain and the voltage of the always on retention net.

4.1.4 The PMU Modeling Stage

After the power intent specification stage, comes the Power Management Unit (PMU) modeling stage (see Figure 4.1). This unit consists in a functional TL SystemC module that is responsible to adjust the power domains states according to the system power management requests. Rather than using classical component-based power management strategies that simply change components power states by appropriately setting some attributes inside the underlying component module, this unit employs power domain management strategies. Each strategy has to control the whole state of a power domain by only adjusting its power infrastructure state (including its power switch state, its supply nets states, contents of its retention and non-retention registers and values of its isolated outputs). Power management requests must be added at this USLPAM stage as TLM transactions for power domain management control. We denote such a transaction by PwCTr. For each specified PMP, a PwCTr transaction can be added for transmitting a specific power domain state setting request to the PMU. Depending on the power management strategy, a PwCTr can be added at the embedded application level or inside specific hardware modules.



(a) Adding the PMU functional module to the TL-model



(b) The PMU Structure and Required Interfaces

Figure 4.7: The PMU Features

Figure 4.7 depicts the main features to be considered when modeling a PMU and integrating it into the TL-model. As it can be seen in Figure 4.7(b), a PMU generally belongs to an "always-on" (AO) power domain (PD2 in Figure 4.7(b)). It needs to stay powered up in order to capture and respond all incoming PwCTr transactions. The general structure of the PMU is also shown in Figure 4.7(b). A PMU is mainly composed of a Power Manager (PM) SystemC TLM sub-module and a central set of Domain Power Controllers (DPCs) SystemC TLM sub-modules. The PM implements a specific power domain management strategy (the *PM Strategy FSM* part in Figure 4.7(b)). It also

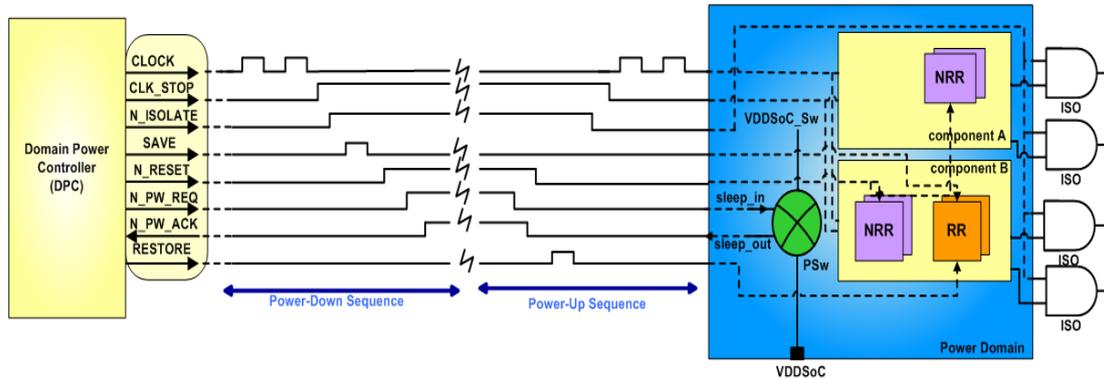


Figure 4.8: Hookup and Power-up/Power-down Sequencing of a Domain Power Controller

adjusts the voltage-scaled power domains states and requests adequate DPCs to change their power domains' states (the *PM Commands Dispatcher* part in Figure 4.7(b)).

Indeed, a DPC has to be associated to each power-gated domain in order to change its state between sleep and wake-up under the request of the PM. This kind of transitions must be done through changing the power components' states of the underlying power domain according to a well-defined sequence. Figure 4.8 depicts how control signals of a domain power controller can be bound to the different power components in this DPC related power domain as it can be specified using the UPF language. This figure shows as well an example of a power-up/power-down sequence that must be strictly followed by a DPC in order to correctly and safely set a gated power domain state.

For instance, to power-down a power domain with retention, a DPC has to:

- Stop the clocks (by asserting the CLK_STOP signal in Figure 4.8), in the appropriate phase to minimize leakage into the power-gated domain.
- Assert the isolation control signal (the N_ISOLATE signal in Figure 4.8) to put all the domain outputs in a safe state with respect to inputs of connected power domains which remain in power-on state.
- Assert the state retention save condition (the SAVE signal in Figure 4.8) for retention registers in the domain (denoted RR in Figure 4.8).
- Assert reset (the N_RESET signal in Figure 4.8) to the non-retained registers (denoted NRR in Figure 4.8) in the domain, so that they are powered-up in the reset condition.
- Assert the power gating control signal (the N_PW_REQ signal in Figure 4.8) to power down the domain (i.e. switching off its power switch).

Here, it is the responsibility of the power switch (actually the switching fabric) to assert the `N_PW_ACK` signal when power is completely switched off. The reverse sequence is practically implemented on power-up as shown in Figure 4.8. A Transaction-Level DPC model has to implement a similar sequence to change power components states on a power domain power-down or power-up. Such a sequence has to take into account the abstract semantics used in the previous USLPAM stage to specify the behavior of each power component. An abstraction of the communication between a DPC and a power domain at TLM should also be strongly considered in order to preserve a high simulation speed and adopt a similar communication as in TLM. Actually, when modeling a TL PMU model, a special care must be taken when modeling its communication interfaces. In general, there are three different TL interfaces to be modeled as illustrated by Figure 4.7(b).

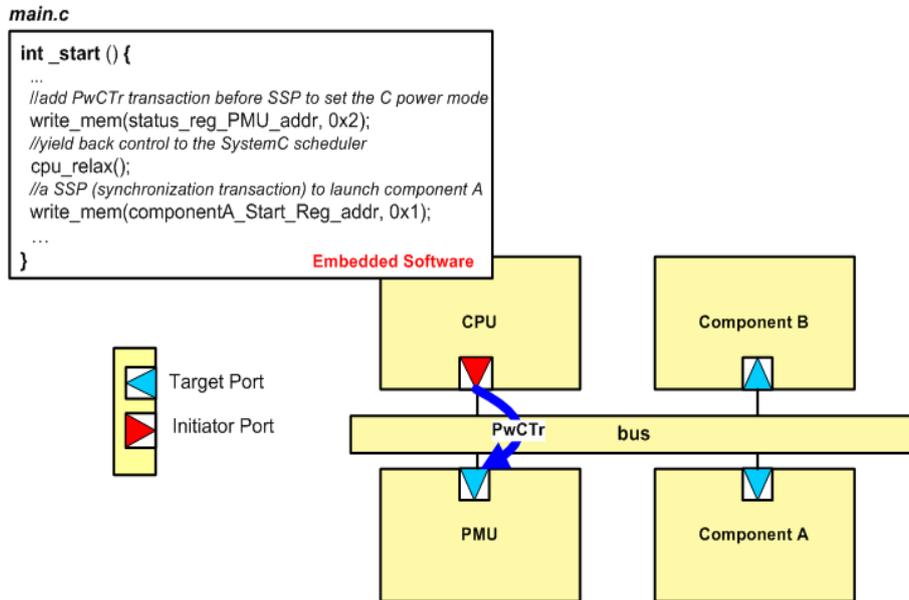
- **A functional interface**, denoted **IF1** in Figure 4.7(b), lies between the PMU sub-modules and the other functional modules of the TL model. Over this interface, functional bus transactions (i.e. transactions transmitted over the functional TL bus) are mainly transmitted to initialize the PMU or configure its registers (denoted CSR in Figure 4.7(b)) in order to transmit power management requests (PwCTr) to the PMU. These transactions may also be used to read a PMU register in order to get information about some power domains states or the current PMU activity.
- **An internal interface**, denoted **IF2** in Figure 4.7(b) lies between the PM sub-module and DPCs sub-modules. This interface employs a request-acknowledge handshake so that the PM controls a DPC activity. By simply using SystemC signals, the PM requests a DPC to change the state of its corresponding power domain from wake-up to sleep and vice-versa. Then, it waits for acknowledgement from each DPC once this latter finishes setting the power domain state.
- **A power domain management interface**, denoted **IF3** in Figure 4.7(b) lies between the PMU sub-modules and power domains in order to change power domains states according to the power management strategy. As shown in Figure 4.7(b), a communication over this type of interface holds between a DPC and a power domain and includes the control sequencing performed by the DPC upon the reception of the PM request over IF2 (Figure 4.8). A communication over this type of interface can also occur between a PM and a voltage-scaled domain to change its voltage supply net value. In the Chapter 6, we explain in detail how such a control interface can be implemented to be appropriately compatible with the UPF-based abstract power-aware simulation semantics used at the

previous USLPAM stage. we also present a more generic power domain management interface, denoted the PDMgIF interface, that separates functional and power management communications while applying a power domain reasoning. In this case, if we refer to Figure 3.6(b), the PDMgIF interface corresponds to the external power interface concept. The most interesting aspect in this interface is that it is reusable whatever the evaluated power management strategy or power architecture.

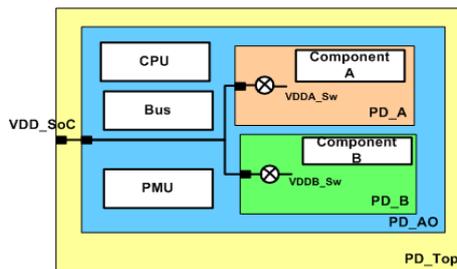
In order to control a system's power domains states, different power domain management strategies can be used. Each strategy may require specific power domain control semantics to use the three power domain management interfaces (Figure 4.7(b)) for local power domains states control. Such power management strategies may range from the static ones (such as that employed by the TI's PRCM based on a power state table matching each system use case with a specific combination of power domains states), to more complex dynamic ones (such as those based on predictive techniques). In the following, we present three examples of power domain management strategies: scenario-based, reactive and scenario-tracking strategies. Power domain control semantics in these strategies were inspired from state-of-the-art power management interfaces (e.g. PRCM, PCI, PCIe, ACPI presented in Chapter 2 Section 2.1.5.3) and adapted to a power domain context use. Each strategy requires specific power domain control semantics and implements differently the PMU interfaces. We will see how the different PMU sub-modules operate and how the PMU's different interfaces are used under each power domain management strategy. Note that our choice of these power management strategies in this thesis does not exclude the possible use of other power domain management strategies along with the PMU model and the different modeling approaches proposed in this thesis.

4.1.4.1 The Scenario-Based Power Management Strategy

This strategy relies on the specification of a static power state table (PST) which summarizes all the possible system power modes. Each system power mode represents a combination of power domains states and corresponds to power requirements of a specific software scenario. This PST-based power management strategy is originally adopted by the UPF standard [30]. In fact, the `create_pst` and `add_pst_state` UPF commands allow to create a power state table that can be used to specify the relationships between different power domains states. It has to be mentioned that, according to UPF semantics, potential



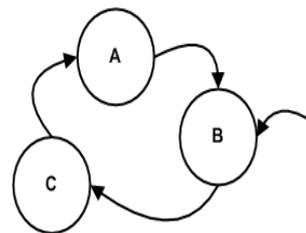
(a) Example of Power Control Transactions (PwCTr) in a Scenario-Based Power Management Strategy



(b) Example of Power Domain Partitioning

	PD_Top	PD_Ao	PD_A	PD_B
A	ON	ON	ON	ON
B	ON	ON	OFF	OFF
C	ON	ON	ON	OFF

(c) Example of a PST



(d) Example of Legal Power State Transitions (PSTrans)

Figure 4.9: Example of a Scenario-Based Power Management Strategy Use

states of a power domain correspond to states of its primary power nets. An example of a UPF-specified PST is shown in Figure 4.9(c) according to the functional platform in Figure 4.9(a) and the power supply network in Figure 4.9(b). In this kind of power management strategy, legal and illegal transitions between system power modes must also be communicated to the PMU. The UPF 2.0 command *describe_state_transition* specifies the legality of a transition from one object's named power state to another. An example of legal power state transitions set is shown in Figure 4.9(d) according to the PST of Figure 4.9(c). As it can be seen on this figure, the transition from the system power mode B to the system power mode A has been prohibited for instance.

In order to implement this kind of power management strategies, a specification of a static power state table and a related set of legal transitions among system power modes (i.e. the lines of this table) has to be provided at the previous USLPAM stage using abstract UPF semantics. While possible software scenarios can be deduced from the software flow analysis, power domains states combination in each system power mode can be recognized by identifying possible assembly between power domains PMPs. This step can be achieved at the second USLPAM methodology (the PMPs identification stage).

In a scenario-based power management strategy, the PST and the transitions set represent input parameters to the PM module and are used to build the PM power management finite state machine as it is depicted by the PM module constructor in the pseudo code of Figure 4.10. The FSM states rely on the different system power modes of the PST. Transitions between states correspond to the specified legal transitions set. Each transition corresponds to a specific configuration of a PMU register performed through a PwCTr transaction over the IF1 interface (Figure 4.7(b)). In this kind of power management strategies, a PwCTr transaction is generally added at the embedded application level. To each system power mode in the considered PST is associated a functional transaction to enable this power mode when it occurs and it generally corresponds to a system synchronization point (SSP) [62]. Such a functional transaction must be preceded by the adequate PwCTr and a wait statement for the PMU response to indicate the ending of the system power mode setting and to allow resuming the normal system operation.

As illustrated by Figure 4.9(a), a PwCTr transaction is added at the embedded software in order to request the PMU to set the power mode C. In fact, this is done by writing to its status register (the *Status_reg_PMU_addr* register) the specific value *0x2*. As

it can be seen, this PwCTr transaction precedes the functional write transaction to the *componentA_Start_Reg_addr* register. Indeed, this functional transaction is a SSP since it will trigger the component A activity. Thus, the PD_A power domain (in Figure 4.9(b)) must be activated before receiving and handling this functional transaction. The `cpu_relax()` code line added to the embedded software code allows yielding back control to the SystemC scheduler, hence allowing the PMU module to handle the PwCTr transaction.

Although, by using this wait statement, control is given to the PMU processes to be executed, synchronization is still required when a DPC is changing a power domain state. Let us take a look at the PMU internal operation: upon the reception of a PwCTr transaction, the *PM Strategy FSM* process locates the power domains states configuration corresponding to the requested system power mode. Depending on the current system power mode and the requested one, this process updates the local power states of voltage-scaled power domains as well as the content of an *update_PD* vector (for instance by using the *set_supply_states* function in the Figure 4.10). The *update_PD* vector is mainly used to identify the new required local power states of power-gated domains. Then, the control is given-up to the *PM Commands Dispatcher* process of the PM sub-module. By comparing previous states of the power-gated domains with the *update_PD* vector value, this process determines which power gated domains need an update in their states. Once identified, it simultaneously requests adequate DPCs modules to update their related power domains states and blocks waiting for DPCs acknowledgement signals and yielding hence back control to the SystemC scheduler. Figure 4.11 gives an example of the *PM Commands Dispatcher* process of the PM sub-module and explains how the *update_PD* vector is used for the DPCs request procedure.

If activities of the functional units inside power domains that will undergo a power state change are not blocked, the SystemC scheduler can give control to one of these units' ready processes (especially those that were waiting for a time to elapse) to execute before power domains DPCs processes. This could badly affect the coherence between the power architecture state and the functional behavior: when executed before DPCs processes, a functional unit's process may be obliged to access and use functional units that belong to powered-down domains before it yields on another wait statement. The accessed domains would be rather active if their DPCs were able to execute before this process, otherwise the global system state is not coherent.

```

//PM Module Constructor
void PM (sc_module modulename, PST* modes, PSTrans* transitions)
{
    ...
    //a SystemC thread to set the requested global system power mode
    SC_THREAD (PMStrategyFSM);
    // a SystemC thread to request adequate DPCs to change their power domains states
    SC_THREAD (PMComanndsDispatcher);
    // a SystemC method sensitive to acknowledge signals of the different DPCs
    SC_METHOD(HandleDCsOutputs);
    ...
    //initializations
    status_register:=0x0; //this register stores the state of the requested system power mode in the PST
    prec_register:=0x0; //this register stores the state of the current system power mode in the PST
    started:=false; //this boolean value is true only if a PwCtr transaction is received by the PMU
    need_update:=false; //this boolean is used to check if the PM commands dispatcher has to be activated or not
    ...
}

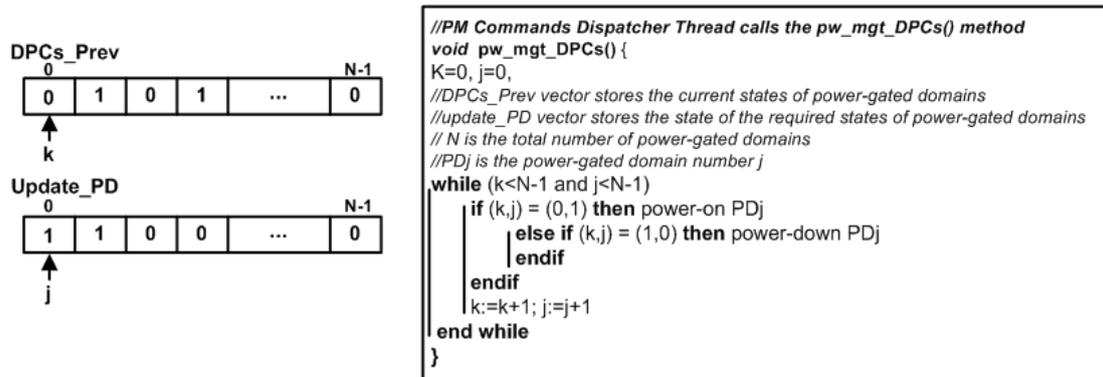
//PM Strategy FSM process
void PMStrategyFSM() {
    while (!started and status_register != prec_register) {
        switch(status_register) {
            //val i corresponds to the system power mode (line i) in the PST
            case (val 1): //checks that the transition is defined in the PSTran
                is_legal_transition (prec_register, current_register);
                set_supply_states (1); //set the power state mode
                break;
            ...
        }
        need_update:=true; //activate the PMComanndsDispatcher thread
        update_event.notify();
        started:=false;
    }
}

// the set_supply_state method
void set_supply_states (int i) {
    for j:=0 to number columns of PST do
        string type:=get_type_SN(PST[0,j]);
        if (type is « switched ») fill the update_PD vector;
        else set supply net volatge to PST [i,j];
        end if
    end for
}

//PM Commands dispatcher Process
void PMComanndsDispatcher () {
    while (!need_update) {
        wait(update_event);
        pw_mgt_DPCs(); //request adequate DPCs to set their power domains states using the update_PD vector
        need_update:=false;
    }
}
....

```

Figure 4.10: Pseudo-code of the Power Manager Module

Figure 4.11: Pseudo-code of the *PM Commands Dispatcher* Process

Although adding some appropriate verification mechanisms would merely detect this kind of errors as it will be seen in the next section, adjustments of the system functional synchronization are required to maintain correct operation. **The first principle** is that activity of functional blocks whose power domains would undergo a power state change must be blocked. **The second principle** emphasizes on blocking any activity that might occur inside a power domain which is functionally or structurally dependent of a power domain undergoing a state transition. The blocking is done using a wait statement for an event that would be notified by the PM when this latter receives all the acknowledgement signals from DPCs (i.e. when all required power domain state changes are over and the global system power mode is completely set). Identification of these wait statements locations is not obvious and can be identified based on power domains PMPs specification.

4.1.4.2 The Reactive Power Management Strategy

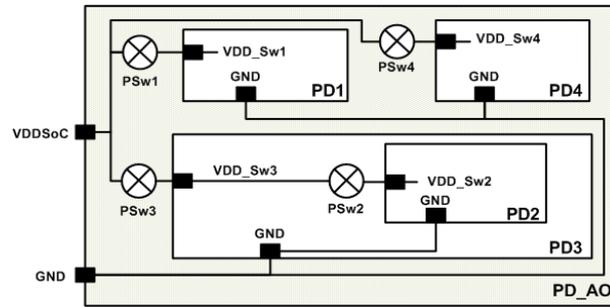
Unlike the scenario-based power management strategy that uses the IF1 interface (Figure 4.7(b)) to transmit PwCTr transactions, the reactive power management strategy uses IF3 interface, so that power domains can transmit PwCTr transactions to the PMU in order to request a power domain state change. Note that this interface (IF3) is additionally used by the PMU in both strategies in order to appropriately set the requested power domain state by changing specific power components states (power switch state, supply nets voltage, retention and non-retained registers contents, ...). In particular, traditional functional transaction semantics which are used for TLM communication between blocks of a TL model are also used to add PwCTr transactions in case of a scenario-based power

management strategy. However, additional semantics and fields are required for PwCTr transactions in case of a reactive power management strategy to allow a power domain to request a power state change. In the Chapter 6, we explain how TLM 2.0 extension semantics could be used to model the required power control semantics in a reactive power management strategy.

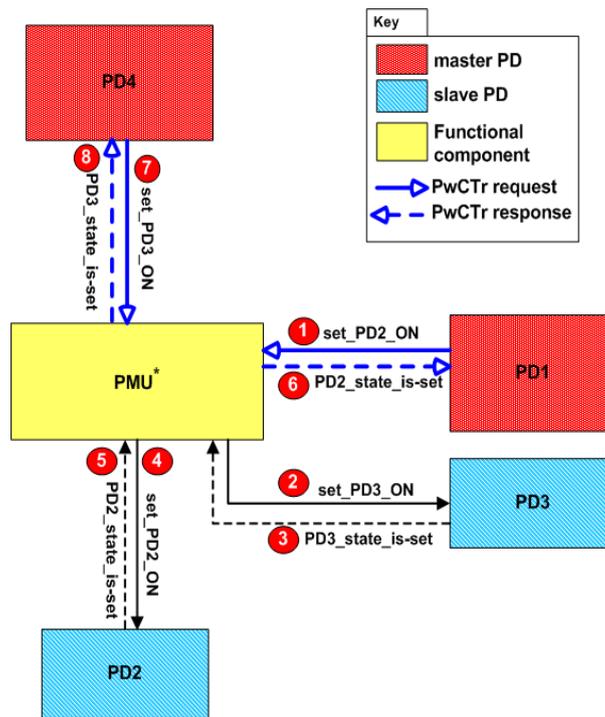
In the following, the main features and rules of a reactive power management strategy are presented. First, two types of power domains can be distinguished: master and slave power domains. Only a master power domain can initiate a PwCTr transaction to the PMU in order to change its local power state or another slave power domain state. A master power domain includes at least one master functional component and uses its PMPs to fill PwCTr transactions' fields. A slave power domain does not have any influence over its local power mode. The PMU changes a slave power domain state upon an explicit request of another master power domain or to handle a required dependency between power domains states.

This point exposes a fundamental difference between scenario-based and reactive power management strategies. While in the case of a scenario-based power management strategy, dependencies between power domains are already taken into account in a system power mode specification, they may not be considered when transmitting a PwCTr transaction in the case of a reactive power management strategy. They are rather managed by the PMU in this case. For that, the PMU uses a list of dependencies between power domains states, and based on this list it can either handle or override or put on standby a request.

Figure 4.12(b) illustrates the reactive PMU activity upon the reception of power control transactions considering the power architecture in Figure 4.12(a). Here, the PD1 master power domain requests the PMU to activate the PD2 slave power domain by issuing a PwCTr transaction throughout the IF3 interface. The PMU first checks the possible dependency combinations between the PD2 power domain and remaining power domains. According to the PMU's dependencies list, a wake-up dependency exists between PD2 and PD3 as PD2 cannot be activated unless PD3 is already activated. Indeed, as illustrated by Figure 4.12(a), the input supply net of the PD2 power switch represents the output supply net of the PD3 power switch. Therefore, the PMU first activates PD3 followed by PD2. Then, the PMU acknowledges the PD1 master domain throughout the IF3 interface,



(a) Example of Power Architecture



* : this module belongs to the PD_AO power domain

(b) Example of Power Control Transactions (PwCTr) Flow in a Reactive Power Management Strategy

Figure 4.12: Handling Dependencies in a Relative Power Management Strategy

and also indicates that the requested PD2 power domain state has been successfully set. The PMU has to keep track of power domains actually used and the master domain that has effectively changed a power domain state. This information will be useful to the PMU for determining its adequate behavior when a request to change a power domain state is received. To illustrate the need and role of such information, let us go back to the example

in Figure 4.12(b). If the PD4 power domain issues a request to the PMU to activate the PD3 power domain, the PMU immediately responds with an acknowledge indicating that the PD3 is already activated. In the meantime, if the PD4 power domain requests to deactivate the PD3 power domain, the PMU will put this request on standby. Such a request is taken into account only if the PD1 power domain requests to deactivate PD2 as well. Actually, due to the existing dependency between PD2 and PD3, deactivating PD3 requires that the PMU first deactivates PD2 as shown in Figure 4.12(a). In this case, the PMU will respond to the pending PD4 request indicating that PD3 has been successfully deactivated.

Note that the functional operation of a power domain requesting a state change must be blocked as long as the PMU responds to this request. Therefore, arbitrary wait for response latencies may appear at the master power domains level due to the PMU behavior implemented to sequence the response to requests and manage dependencies. These latencies may sometimes lead to a real time constraint violation or a miss of a QoS requirement. In order to avoid this situation, a high or a low priority to each issued PwCTr request is assigned. A request with a high priority is considered by the PMU to be handled as soon as possible. However, a low priority request can be lodged in the PMU and

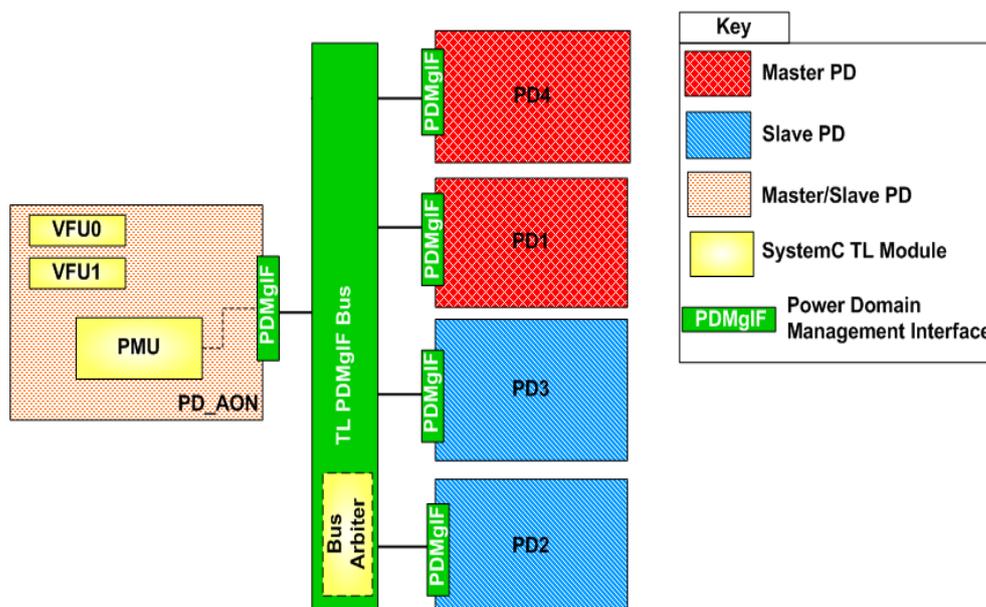


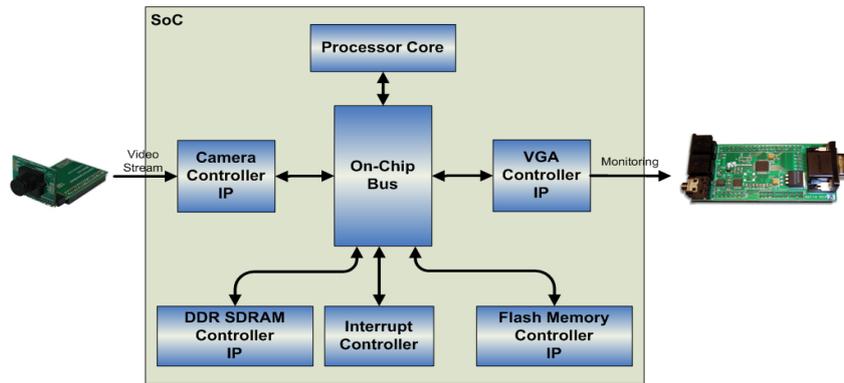
Figure 4.13: A PDMgIF Bus Interface for Inter-Power Domain Communication

handled later. For instance, if PD4 (Figure 4.12(b)) uses a high priority PwCTr request to deactivate PD3 power domain, the PMU must immediately inform PD1 about this change since PD2, a power domain that depends on PD3 power state, is being used by PD1. This may cause PD1 to block or not its functional activity. Then, the PMU effectively changes the PD3 state, as well as the PD2 state (due to existing dependency between these two power domains).

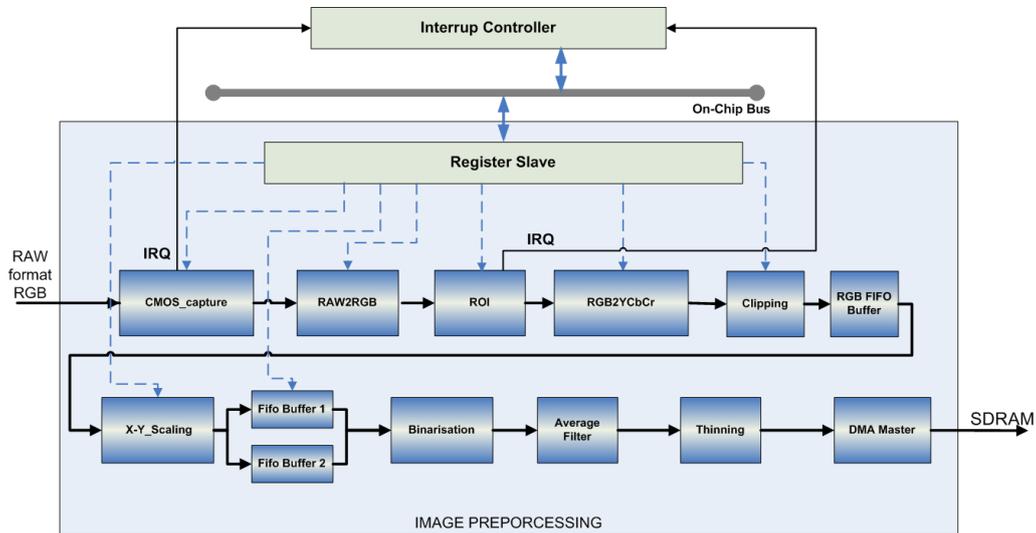
Due to the multiple PwCTr requests transmitted to the PMU and their different priorities levels, a specialized power domain management interface that stands between power domains and replaces the IF3 interface is strongly needed. Such an interface, called PDMgIF, is responsible for conveying power control transactions transmitted from master power domains to the PMU, as well as other forms of power domain management transactions such as the transactions transmitted from the PMU to power domains. PDMgIF is also designed to transfer PwCTr transactions with low priority. Figure 4.13 illustrates the proposed common PDMgIF interface used for the transfer of power domain management transactions. As it can be seen, PwCTr transactions are issued from a master power domain over a PDMgIF TLM port and over a PDMgIF bus to the PMU module encapsulated into the always-on (AO) power domain. In particular, the AO power domain represents a master power domain since it can initiate PwCTr transactions as well as a slave power domain since it can receive PwCTr transactions from master power domains to be processed by the PMU. Note that this PDMgIF bus is granted to one of the requesting master power domains based on an arbitration mechanism involved in this bus model. More details on the transaction-level model of the PDMgIF protocol interface will be given in the Chapter 6.

4.1.4.3 The Scenario-Tracking Power Management Strategy

This strategy is similar to the scenario-based one: the PMU still uses a PST to set the requested system power mode and PwCTr transactions are added at the embedded software level as usual functional transactions that configure the PMU with the required system power mode. However, one of the most notable features compared to other power management strategies is that master power domains are allowed to issue power management events to the PMU in the form of transactions over the PDMgIF bus interfaces and that is in order to inform the PMU about a system functional state. This information would



(a) Data Acquisition and Display System



(b) The Camera Controller IP: the complete acquisition chain

Figure 4.14: A Functional SoC Example

help the PMU to determine the right PST’s system power mode to set. Similarly to the reactive power management strategy, power management events sent by a master power domain may inform about its proper functional state or the functional state of a slave power domain. To identify power management events, a selection of PMPs per power domain must be performed.

Power management events are not frequently issued. They are issued in order to indicate to the PMU a power management requirement triggered by an external or internal event interrupting the CPU (such as a touch screen causing a new executed software scenario, or a digital temperature sensor providing an alert signal to the PMU when the

temperature exceeds a limit). Figure 5.7 depicts a functional example of a data acquisition and display system-on-chip. Here, the camera controller IP includes an intelligent motion estimator hardware block (called the *CMOS_capture* block in Figure 5.7) that identifies the lines and pixels valid at the camera output and prepare them for treatment provided by the next block. For instance, if two consecutive identical images are captured by the camera, this block would indicate to the CPU the intent of non-use of the rest of the chain blocks by sending an interrupt to the CPU. In its turn, the CPU would correctly configure the chain register slave (by disabling the start bits of the rest of blocks inside the register slave for instance).

Functionally, the remaining blocks of the camera IP controller would be inactive but they would still consume power if a PMU does not effectively deactivate their underlying power domains. For that, when the *CMOS_capture* block issues an interrupt to the processor, it also issue to the PMU a power management event over the PDMgIF bus interface in order to inform it about the possibility to power-down specific domains. By affecting a high priority to this request, the PMU will then use its functional interface to poll the slave register until it detects the end of the CPU handling of the *CMOS_capture* block interrupt. To do so, the PMU may use the TLM read command field of the PDMgIF TL interface. Once captured, the PMU changes the system power mode to the adequate one, so that the rest of the acquisition chain blocks (except for the *CMOS_capture* block) belong to deactivated power domains.

A similar use case of power management events can be noted when the Region of Interest block (*ROI* in Figure 4.14(b)) selects a region of the scene where motion will be ignored. In this case, this block will interrupt the CPU and transmit a power management event to the PMU. Functionally, if the ROI algorithm succeeds to identify the region of interest, the CPU will configure the remaining blocks to process a smaller image size. It disables for instance the FIFO buffer 2 block in Figure 4.14(b). In addition, since there is no longer a need to detect all the image contours nor extract thinner details, the filter block and the thinning block will be disabled as well. So, once it receives the ROI power management event, the PMU will select a system power mode in the PST for which power domains including the average filter block, the thinning block and the FIFO buffer 2 block are powered-off.

Note here the importance of synchronization between functional effects of such a rel-

event interrupt and its effects on the system power mode. The occurrence of functional effects may functionally impact blocks that will be changed power state as soon as the PMU reacts to the received power management event. In this case, functional effects must take place before power effects. Moreover, power management events transmitted over the PDMgIF TL bus represent a good solution to the impossible use of a wait statement inside the interrupt handling code section (at the embedded software level). Recall that wait statements must succeed a PwCTr transaction in order to block the system functionality and the control will be hence transferred to the PMU to execute power mode transitions. This is why power management events best replace PwCTr transactions transmitted either over the functional bus in the scenario-based power management strategy, or over the PDMgIF in the reactive power management strategy. More details on how power management events are modeled and transmitted over the PDMgIF transaction-level interface model will be given in the Chapter 6.

4.1.5 The Full Power-Aware Simulation Stage

At this stage, the resulting TL power-managed behavior is simulated. This stage is processed in parallel with the verification one. During simulation, functional coherence between the augmented TL-model and the power design needs to be verified. The system power-aware behavior is proved coherent if no verification properties are violated during simulation. Furthermore, mechanisms that update at runtime power and energy consumption equations while maintaining a power-domain-based reasoning are added. Examples of such mechanisms are presented in further chapters. Log files for power analysis should be automatically generated at the end of the simulation. These files would be helpful to analyze and compare different power-management solutions, as well as selecting the most energy-efficient power management solution.

4.1.6 The Power-Aware and Simulation-Based Verification Stage

Although the functional behavior of the TL-platform is supposed to be correct before applying our methodology, checking bugs that may appear due to the power-management features added to the initial TL model is absolutely mandatory. So, in order to ensure that each stage has been correctly performed, a contract-based and dynamic verification process

is added to the USLPAM flow. As shown in Figure 4.1, the verification stage is processed orthogonally to the previous ones. The proposed verification process is contract-based since it applies the "Design-by-Contract" (DbC) principle [118] to check power-aware properties. This principle considers that a contract is a specification put in the form of an implication between, on the one hand, a set composed of an assumption clause (also called pre-condition) and a potential satisfy clause (also called invariant), and on the other hand, a guarantee clause (also called post-condition).

In the Chapter 3, we have discussed the complementarity between Component-Based Development (CBD), Transaction-Level Virtual Prototyping (TLVP) and Design-by-Contract (DbC) (Figure 3.7). We have come up with the idea of specifying contracts for the different power-aware interfaces added to the initial TL model and illustrated by Figure 3.6(b). This can be achieved by specifying contracts for each relevant component involved in the final TL power-domain-managed model. In such a model, three types of components can be distinguished:

- **Power components:** represent abstract UPF-like power concepts used to specify power intent of a TL-design (e.g. power switches and supply nets).
- **Functional components:** represent Intellectual Properties (IPs) of the considered TL-model.
- **Mixed components:** represent PMU modules and their sub-modules (i.e. DPC and PM modules). They are responsible to set power states of functional components according to a power management architecture and strategy.

Thus, contracts specify potential interactions between at least two components. Actually, a component may simply require using another component to perform a specific functionality. It can also modify some of the other component's characteristics as a part of its functionality. To be done in a safe and correct way, these kinds of interactions must be characterized by a set of assume/guarantee/invariant properties that form the contracts of a component.

As far as simulation is concerned in our USLPAF framework, the verification process in the USLPAM methodology is dynamic in the sense that assume-guarantee contracts are incrementally added and validated in simulation during the methodology application. Actually, we have added contracts as executable specifications that are monitored at runtime and expressed by writing assertions that trigger exceptions whenever a contract

Contract	Stage (s)	Interacting components	Verification Aim	Property Example
Class 1	Power intent specification PMU modeling	Pw Component / Pw Component	Power architecture is correctly structured Structural dependencies are respected	Behavior induced by a particular placement of power switches must not result in a functional bug
Class 2	PMU modeling	Pw Component / Mixed Component Mixed Component / Mixed Component	Correct functionality of the PMU	During simulation, a transition to a specific system power mode is required whereas such a transition has been specified as illegal
Class 3	Full power-managed simulation behavior	Functional Component / Pw Component	A TL model enters correctly a system power mode like specified in the considered power state table	Noting an activity in a power domain which is powered down
Class 4		Functional Component / Mixed Component	Added power-aware features do not alter intended functionality	Issuing a transaction to a module which is still undergoing a power state transition

Pw Components = Power objects used to specify power intent
Mixed Components = PMU module and submodules
Functional components = Hw blocks of the considered TL-platform

Table 4.1: An Overview on The Different Classes of Contracts Involved in The Power-Aware Verification Process

part (assume/guarantee/satisfy) is violated.

As it can be seen in Figure 4.1, the USLPAM verification stage can be performed after each sequential stage to check for a specific category of properties. Depending on the required interactions of components at each stage of the methodology, contracts have been classified into four different classes. Each class gathers all possible properties between two specific types of components. Table 4.1 summarizes the different classes of contracts. They are described in the following:

- **Contracts of class 1**

This class of contracts specifies interactions between power components of a design. The objective is to verify the correctness of a power architecture structure including the hierarchy and composition relationships between its power elements. A typical error is to forget to attach at least one functional TL block to a power domain. In this case, the

power domain is not necessary. Furthermore, each system power mode specification (a line of a power state table) must respect structural dependencies between power domain partitions. As illustrated by Figure 4.1, this kind of errors can be detected when simulating the system after the power intent specification stage.

Additionally, this class of contracts is used to ensure that the power domain ordering rules are not violated during simulation. These rules define the order that must be respected to turn some power domains on or off. They are imposed by a specific hierarchical composition of power domains and a particular placement of power switches. Let us take Figure 4.12(a) as an example. Here, PD3 is a container power domain and PD2 is a power domain nested in PD3. As a consequence, PD3 and PD2 can be individually switched using respectively their PSw3 and PSw2 power switches. By considering the hierarchical relationship between PD3 and PD2, the output supply net of PSw3 represents an input supply net for PSw2. Therefore, PD2 must be already switched off before turning off the PD3 power domain. This property can be considered as an assume part of a contract. It must also be checked that all power domains which are powered by the PSw3 output supply net (whether nested in PD3 or not) are powered down when PD3 is switched off. This property represents the guarantee part of the same contract. Note that such a contract specifies the behavior of power switches according to supply nets.

Errors related to power domain ordering rules are examples of violated contracts of class 1 which can only be checked after the PMU modeling stage as shown in Figure 4.1. In fact, the power management behavior is only defined at this stage through the addition of power control transactions and the integration of the PMU into the TL platform.

- **Contracts of class 2**

Class 2 contracts target the specification of interactions between mixed components, i.e. between the power manager (PM) and domain power controller (DPC) components inside each power management unit (PMU). In addition, this class of contracts specifies interactions between mixed components and power components. This class of contracts can only be verified after the PMU modeling stage and aims at checking the correct functionality of the PMU modules as well as their integration into TL-models. For instance, a power state transition can be required during simulation whereas it is missing in the graph of legal power transitions. This error can be corrected in two different ways: either, a new legal transition is specified, or the PMU performs legal intermediate transitions

until reaching the required system power mode.

Another example of class 2 contracts consists in checking that each Domain Power Controller (DPC) correctly performs wake-up or sleep transition sequences. During such transitions, it must be verified that states of specific power components (power switches and retention supply nets) in a switched PD have been changed in a specific order by the corresponding DPC.

Specifications of interactions between a power manager (PM) and domain power controllers (DPCs) belong to contracts of class 2 as well. A DPC which has switched off a power domain while the PM has requested to power it on represents a serious error. This kind of issue is caused by an erroneous functionality of the PM or the DPC module. Furthermore, the PMU functionality must identify and respect specific dependencies between power domains. For instance, let us consider again the example of PD3 and PD2 shown in Figure 4.12(a). Here, the PM is not allowed to request a PD3 switch-off as long as transition of PD2 to OFF is not over. More generally, simultaneous transition requests (to DPCs) to switch off or on a power domain can be error-prone. Thus, contracts of class 2 can be used to specify an order of transitions between specific states of power domains.

- **Contracts of class 3**

These contracts specify relations between functional and power components. They are checked at the final stage (i.e. when simulating the power-managed behavior of a TL-model). A functional hardware block can perform different activities. Each of these activities can be launched just after either a specific configuration of this block's internal register, or a specific exchanged transaction at this block's interface, or an internally triggered event. To be performed, a block activity may require specific power properties to be satisfied such as a specific state of the block's power domain or a specific value of one of its internal registers. A typical example is when a functional block receives or transmits a transaction. In that case, its power domain must not be switched off.

Let us zoom in more details on the use examples of the class 3 contracts. A TL block becomes most of the time functionally idle when a wait statement in its source code is reached. In this case, power requirements just before and after wait statements may be different. For instance, when a wait on an event statement is reached in a block, the power domain of this block can be downright powered down. But, it must be verified that this power domain has been already woken up just before the expected event is triggered and

before the block resumes its normal activity. Adding this kind of class 3 contracts is done in general by surrounding the SystemC wait statement with assume properties.

The mechanism used to check this kind of properties is based on observing the internal state of the functional modules and integrating monitors into the execution model. These monitors trigger an error whenever power characteristics or functional behavior of a block does not match a class 3 property. In the next chapter, our monitoring framework will be explained in detail.

- **Contracts of class 4**

Contracts of class 4 specify relations between functional and mixed components at the final USLPAM sequential stage (Figure 4.1). Verification focuses here on the compatibility between the PMU functionality and the activity of hardware modules extended with power control transactions. Indeed, the PMU activity must not alter hardware modules activity during execution. For example, to set up a system power mode, a PMU performs specific power domain state transitions according to a power management strategy. However, performing a power domain state transition requires that all hardware modules of that power domain are functionally idle (i.e. waiting for an event, time duration or a signal) during this transition. This contract represents an invariant property that is checked before and after a power domain transition performed by a PMU sub-component. Similarly, when an activity is detected in a hardware block, it must be verified that the power domain of this block is not currently in a power state transition. A violation of this contract proves a wrong synchronization between this hardware block and the PMU.

Ideally, contracts of each class should be checked after a specific sequential stage. This facilitates identifying sources of errors. However, our verification process is flexible since each class of contracts can also be verified after a specific stage. For instance, if contracts of class 2 have not been verified after the PMU modeling stage, they can be checked during the full power-aware simulation stage.

4.2 The USLPAM Requirements

In order to summarize our proposed methodology and modeling choices, we propose a set of fundamental principles on which the USLPAM methodology is based. These principles

are presented in this section in the form of essential requirements that must be satisfied by each implementation solution of this methodology in order to adequately apply it. These requirements have been used as guidelines to the different modeling approaches proposed in the following chapters.

Requirement #1: *The methodology implementation should allow enabling and disabling power features according to the simulation aim.*

Either timed or untimed, Transaction-Level models are meant to be functional models dedicated for early and rapid functional verification and co-simulation. They do not necessarily contain power features to exploit early power analysis and estimation. The added power features throughout our methodology flow including power estimation and management capabilities may slow down the simulation speed. For that, they should be enabled only for power analysis purposes, otherwise disabled. Requirement #1 emphasizes on this kind of separation between functional and power concerns within Transaction-Level models.

Requirement #2: *Power-aware features including power network specification as well as power estimation and control are added based on a power domain based reasoning.*

Requirement #3: *The UPF (IEEE-1801) industry standard semantics are used as the reference to add power intent at Transaction-Level.*

Requirement #2 supports the power domain based reasoning employed throughout our methodology flow. Such reasoning must be involved in the power estimation mechanisms that rely on updating power consumption values per power domain whenever a power domain state is changed. This requires that the power domain reasoning should also be applied when managing power consumption by using strategies that control the power states of power domains rather than states of individual TL components. For its part, this power management principle needs applying a power domain reasoning to specify the power characteristics of a SoC model. Information about the SoC power domain partitioning and the power features of each power domain must be provided. Such a specification must almost be corresponding to semantics and composition relationships of the Unified Power Format (IEEE-1801) standard as imposes requirement #3. In this way, the designer easily imports or exports a UPF standard specification based on the Transaction-Level specification in order to use it as a golden power reference for RTL

design teams.

Requirement #4: *All blocks involved in a power domain state change should be blocked as long as the PMU ends setting the requested system power mode.*

Requirement #5: *Each power-gated domain needs a separate power controller which automatically controls the power down and power up sequencing.*

Requirement #6: *The power management strategy as well as the PMU model should be designed to use the three different power management interfaces.*

Synchronization between the PMU and the blocks undergoing a power state change further to a power control explicit request is of prime importance. Requirement #4 emphasizes on the fact that these blocks must be blocked until all required power domain state changes are over and the requested power mode is completely set. Still in the context of a power domain reasoning adoption, requirement #5 captures a fundamental design principle when modeling the PMU: inside the PMU module, central DPC sub-modules are gathered. A separate DPC has to be associated to each power-gated domain in order to change its state between sleep and wake-up. As a DPC structure and behavior are identical for each gated power domain, only a generic DPC model can be modeled and then reused to instantiate the required number of DPCs modules.

While being complementary to requirement #5, requirement #6 imposes that the PMU activity and structure must involve the three different power management interfaces depicted in Figure 4.7(b). This requirement has to be satisfied by each PMU implementation whatever the models of these three interfaces and their used communication mechanisms.

Requirement #7: *The verification process should be power-aware and contract-based and should dynamically check all the defined classes of contracts.*

Requirement #8: *The contracts should be inserted or removed without editing the source code and a possible selective enabling of the different categories of checks (e.g. preconditions, postconditions, or invariants) should be allowed.*

Since the functional simulation platform already exists, our verification process focuses on the dynamic verification (during simulation). The main objective through this verification process is to detect violated power-aware contracts among the four defined classes

of contracts (requirement #7).

Despite of being necessary, adding contract checking features induces a simulation performance penalty. For that, possible enabling/disabling of contract checking independently of remaining stages of the methodology is a required strategy to control simulation time. Such a strategy also allows that all assertion checks are only viewed during development and testing and omitted at the commercial release of the code.

Requirement #8 supports this strategy. However, it also defines fine-grained control over the different types of assertions (assume, guarantee and satisfy) for more flexibility of the verification process. So, the developer can select which categories of checks are inserted into a code and enabled during simulation.

Chapter 5

PMPs Specification and Simulation-Based Power-Aware Verification

5.1	Identification of Power Management Points Candidates	158
5.1.1	Methodology for PMPs Specification	158
5.1.2	Power-Aware State Modeling of Black-Box and White-Box IPs	165
5.2	Dynamic Contracts for Verification of Power-Aware Properties	175
5.2.1	Design Verification Techniques	175
5.2.2	Verification of Power-Aware Designs	180
5.2.3	A Modular Power-Aware Verification Flow	187
5.3	Conclusion and Discussion	191

5.1 Identification of Power Management Points Candidates

5.1.1 Methodology for PMPs Specification

As introduced in the previous chapter, a power management point (PMP) of a power domain is defined as a point in time where this power domain can be changed power

state by the PMU. In other words, it is defined as a possible power domain state when the functional components of this power domain are stationary, i.e. they wait for a logical time to occur in order to change their operational state. Here, logical times refer to times where an external or an internal event is triggered. These events are usually in the form of exchanged synchronization transactions and result in novel atomic (i.e. non-interruptible) activities of the component. Thus, power management points of a power domain can be identified through the specification of the different power management points of this domain's components. PMPs components are determined based on an analysis of their computation and communication patterns upon running the embedded application. They are sometimes imposed by the designer or the component datasheet.

In order to specify the potential power domain state changes during the functioning of this domain's components, we classified PMPs of a component into power candidates (denoted $PwC_{candidate}$), sleep candidates (denoted $Sleep_{candidate}$) and retention candidates (denoted $Ret_{candidate}$). Thus, a component's PMPs, denoted $PMP(C_i)$ where C_i denotes a particular component of a power domain, is defined as the triplet

$$PMP(C_i) = \langle PwC_{candidate}(C_i), Sleep_{candidate}(C_i), Ret_{candidate}(C_i) \rangle$$

Where:

- $PwC_{candidate}(C_i)$ is the power candidates set. It is defined as the set of transitions from a stationary functional state of the component to another on which the power domain state changes. In SystemC/TLM, a stationary functional state of a component corresponds to set of atomic operations of this IP between two logical times.
- $Sleep_{candidate}(C_i)$ is the sleep candidates set. It is defined as the set of transitions from one stationary functional state of the component to another where the power domain can enter a sleep power state. In SystemC/TLM, a functionally idle state of a component corresponds to wait SystemC statements. In such states, the component operation is blocked and waiting for a logical event to occur in order to process the next atomic operations set. These idle states represent hence the most effective places to put sleep candidates, that is to power-down the power domain. Nevertheless, not all wait statements are functionally idle states. For instance, a wait statement on a time (e.g. `wait(time=val0)`) may be used to advance the simulation time by a time equal to the computation time required to perform a previously executed set of operations. This kind of wait statements is specific

to a SystemC/TLM modeling and simulation and cannot be considered as an idle time to place a sleep candidate.

- $Ret_{candidate}(C_i)$ is the retention candidates set. It is defined as the set of couples (c, L) where c is a sleep candidate ($Sleep_{candidate}$) and L is a list of retention storage elements. The purpose of this kind of candidates' specification is to identify which storage elements states need to be retained when powering-down the power domain at a specific time. By doing so, the power domain does not randomly enter a powered-down power mode and registers contents needed for resuming computation when the power domain will be powered up again are not reset on power-down. Retention candidates are precisely useful at the power intent specification stage when applying a partial retention strategy [96] to a power domain. Recall that at this stage and in this case, the set of non-retained registers must be explicitly specified in order to reset their state on power-down. This set is easily deduced from the retention candidates set identified at the power management points' USLPAM stage. Any register that has not been specified in the retention candidates set across all sleep candidates is a non-retention register and its state must be reset on power-down.

Classification of registers states into temporary and persistent states aids in the identification of retention candidates of a component. This classification is done based on an analysis of use patterns of the component registers between the stationary functional states of a component. Recall that in a stationary state, a component is waiting for a logical time that changes its operational state to occur. When this logical time occurs, a set of registers states may be read, written or simply internally modified. Among these registers states, those that are not required for correct operation after the occurrence of this logical time (i.e. the component functional state changes to perform another set of atomic operations) correspond to temporary values and can be discarded. However, those affecting the component behavior or that are simply required to perform the next component activities are persistent and must be retained on a $Sleep_{candidate}$ PMP.

PMPs of components belonging to a single power domain define the set of power management points of a power domain, denoted $\mathbf{PMP}(PD_i)$, where PD_i denotes a particular power domain. As already stated in the previous chapter, $\mathbf{PMP}(PD_i)$ is defined by the triplet:

$$\text{PMP}(PD_i) = \langle PwC_{candidate}(PD_i), Sleep_{candidate}(PD_i), Ret_{candidate}(PD_i) \rangle$$

Where:

- $PwC_{candidate}(PD_i)$ is the power domain's power candidates set. It represents the union of power candidates of this power domain's components.
- $Sleep_{candidate}(PD_i)$ is the power domain's sleep candidates set. It is defined as the union of sleep candidates of this power domain's components.
- $Ret_{candidate}(PD_i)$ is the power domain's retention candidates set. It is defined as the union of retention candidates of this power domain's components.

In this definition, power domain's PMPs are viewed as specific times during the system execution where events can be sent to the PMU to export the power requirements of this power domain as well as the functional status of its components (active or inactive). When receiving these events, the PMU first updates its database of requested power domains states and their current components' functional status and reached PMPs. Based on this database, it decides to accept, reject or delay the handling of these events according to the power management strategy that it implements.

Our method to locate the different components' PMPs is to:

1. First model each SystemC/TLM component behavior as an Extended Finite State Machine (EFSM)
2. Add power modes specifications to each EFSM model and extract power management points candidates. Such specifications consist in predicates on an E variable value where E designates the component's power domain mode (e.g. sleep, running, active_high, active_low ...).

As far as we know, there is no modeling approach that encodes SystemC/TLM behavior into an EFSM model. Therefore, we give in the following, a set of general rules to build an EFSM model capturing the behavior of a SystemC/TLM block that is relevant to a power-aware modeling.

An EFSM is given by a 6-tuple $\langle X, Y, S, S_0, V, T \rangle$, where:

X, Y, S, S_0, V and T are finite sets of inputs, outputs, states, starting (reset), variables, and transitions, respectively. Each transition $t \in T$ is a 6-tuple: $t = (s_t, q_t, x_t, y_t, P_t, A_t)$ where s_t, q_t, x_t and y_t are the start(current) state, end (next) state, input, and output,

respectively. $P_t(V)$ is a predicate on the current variable values and $A_t(V)$ is an action on variable values. We assume that predicates are exclusive for transitions outgoing from each state. The transition is followed at state s_t if the input condition is satisfied and the predicate $P_t(V)$ evaluates to *TRUE*. If the transition is followed, the machine outputs y , changes the current variable values according to $A_t(V)$, and moves to state q_t .

In order to encode a TL IP behavior into an EFSM, let us first summarize the main and general features of its SystemC/TLM code previously explained in the Chapter 2: first, wait statements locations inside this code widely contribute in defining this IP behavior. Wait statements are generally put on synchronization events, on a processing or a functional time delay. Synchronization events can be internally or externally notified. Internally notified events are triggered inside the IP (e.g. such an event can be triggered by an IP sub-module to synchronize with another sub-module of the same IP). However, externally notified events are triggered upon the reception of a transaction from other IP-cores or a signal value change. Second, a part of an IP behavior can be captured through observing and analyzing specific exchanged (transmitted or received) transactions at its interface.

So, according to our approach, encoding a Transaction-Level IP behavior into an EFSM consists in considering that:

- Each state $s_i \in S$ is an operational state of the IP gathering a set of its atomic operations (i.e. lying between two wait statements in general).
- Each input $x_i \in X$ corresponds to an internal or external event.
- A variable $v_i \in V$ corresponds either to a register value in the underlying IP, or a time value, or a power mode or a number of transmitted or received transactions.
- An action A_i corresponds to transmitting read or write transactions via TLM ports, or to incrementing a counter, or to setting a condition on a specific IP's register. It can also correspond to a null action ϵ .
- Outputs are not drawn for simplicity.

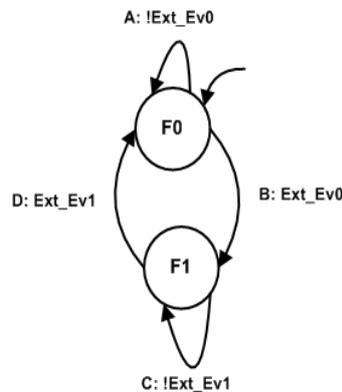
Let us consider S_i a successor operational state of S_j . S_i is dependent of the operational state S_j , if S_i uses a specific register state which has been modified by S_j . It is required to model such a form of dependency since it will be useful for determining candidates for retained registers in the next step. In our EFSM-based IP functional specification, this kind of dependencies is modeled as predicates on registers values.

Given an EFSM-based model of the functional IP behavior, the next step is to add power-aware specifications to this model. The EFSM modeling the power manageable behavior of the IP is then obtained by adding, to the functional specification, predicates on the E variable value. This variable indicates the desired power mode of the IP's power domain. For a switched-off power domain, we assume that the E variable is particularly set to the E0 value (i.e. sleep power mode). A good question is hence: how to identify sleep candidates of an IP belonging to a power domain of type power-gated (i.e. that can be powered down) based on this IP's behavioral EFSM model?

```

// Example of a Slave Component's Process
1: While (true) {
2:   do_some_computation_0();
3:   wait(Ext_Ev0);
4:   do_some_computation_1();
5:   wait(Ext_Ev1);
6: }
    
```

(a) Example of a Component Process



(b) Example of an EFSM-Based Functional Model

Figure 5.1: Building an EFSM-Based Behavioral Model of a TL Component

According to our EFSM formulation, a self-loop in the EFSM-based behavioral model of an IP indicates that either the inputs do not change the IP's current functional state or the IP is functionally idle and is waiting for a specific input combination to change its functional state (i.e. do another set of atomic operations). Thus, these idle conditions

in which the IP remains stationary until the required input combination occurs represent obvious sleep candidates to put the IP in the E0 power mode (i.e. to power-down this IP's power domain). Figure 5.1(b) depicts an example of an EFSM modeling the functional behavior of the component's process source code shown in Figure 5.1(a). This process begins by performing a set of operations corresponding to the execution of the function `do_some_computation_0()` (Line 2 of Figure 5.1(a)), then blocks on a wait for an external event statement (Line 3 in Figure 5.1(a)). As it can be seen in Figure 5.1(b), the `do_some_computation_0()` function corresponds to the EFSM state F0, while the wait state for the `Ext_Ev0` event is modeled as the self-loop transition A on state F0. Once the `Ext_Ev0` event is received, the component functional state moves (the transition B in Figure 5.1(b)) to the EFSM state F1. Actually, the F1 state corresponds to the execution of Line 4 in Figure 5.1(a). The second self-loop corresponding to the transition C in Figure 5.1(b) makes the component blocked in the F1 state waiting for the `Ext_Ev1` event to occur.

In more details, according to our EFSM formulation, not all self-loops represent sleep candidates. Only self-loops with a null action represent real sleep candidates. Otherwise, transitions that depend on a current variable cannot be considered as sleep candidates because they indicate that either the IP is in middle of a computation (e.g. receiving or transmitting transactions) or is blocked on a functional wait statement (e.g. wait statement aiming at only advancing simulation time).

$$Sleep_{candidate} = \bigcup \{t \in T \setminus s_t = q_t \wedge P_t(E = E0) \equiv \text{TRUE} \wedge A_t = \epsilon\}$$

For each specified candidate $C \in Sleep_{candidate}$, registers whose states need to be retained represent all the variables, corresponding to registers' values, and having predicates or undergoing actions on their current values during one or more transitions occurring between this sleep candidate and the next sleep candidate.

Let a branch $s_t \xrightarrow{t \in T} q_t$ indicate that, if the IP is in the state s_t , then taking the transition t , it ends at the state q_t . Let a transition sequence ω be composed of zero or more successive transitions of the same EFSM. A sequential successor state q of state s , reached by a transition sequence ω is then denoted by $s \xrightarrow{\omega} q$. Using these encodings, the $Ret_{candidate}$ set of an IP is then defined as follows:

$$Ret_{candidate} = \bigcup \{(v_i, t_i) \in (V, Sleep_{candidate}) : \exists t \in \omega, \exists t' \in Sleep_{candidate} \setminus s_{t_i} \xrightarrow{\omega} q_{t'} \xrightarrow{t' \in T} q_{t'} \wedge P_t(v_i) \equiv \text{TRUE} \vee A_t(v_i) \neq \epsilon\}$$

In case of a power-gated power domain, we have given a standard method to easily detect sleep candidates and retention candidates from functional EFSM models of this power domain's components. Nevertheless, in case of a multi-voltage scaled power domain, choosing the adequate active power mode of the IP required to perform an atomic set of operations (i.e. entering a functional state in the EFSM model) remains either up to the designer or according to requirements in the IP datasheet. Thus, according to our semantics for a power-aware EFSM-based model of an IP, each transition having the predicate on the E variable different from that in its predecessor transition represents a candidate $C \in PwC_{candidate}$. Let F be a function that defines the variable E labeling a predicate on a transition $t \in T$ (i.e. the function $F_t(E)$ returns the value of the variable E on the transition t). The $PwC_{candidate}$ set is then defined as follows:

$$PwC_{candidate} = \bigcup \{t' \in T : \exists t \in T \setminus s \xrightarrow{t} q \xrightarrow{t'} p \wedge F_t(E) \neq F_{t'}(E)\}$$

5.1.2 Power-Aware State Modeling of Black-Box and White-Box IPs

The goal of this section is to show how the PMPs specification methodology described above can be applied in a white-box TL description and a black-box TL description. This section explores as well the differences between the two IP cases when adding them power intent and management specifications. In each case, an IP-based functional behavior analysis under the software application is first performed. As previously specified, this behavior is then encoded into an EFSM model that is used to identify PMPs for each IP version. We conclude the section with an observation on the differences between white-box and black-box power intent and management specifications that must be considered through comparing the obtained power manageable EFSMs and the different set of PMPs of the same IP in its two versions (white-box and black-box).

In order to generalize the study of the two power-aware IP versions, let us consider a generic and simplified example of a slave/master TL IP block. Figure 5.2 depicts the interface and internal processes of the white-box version of this IP.

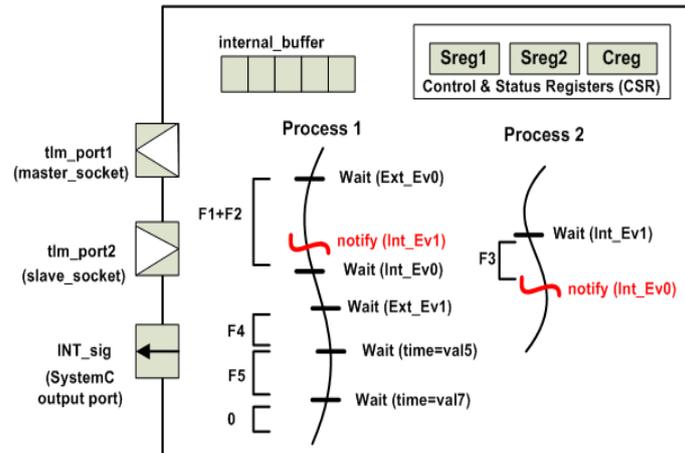


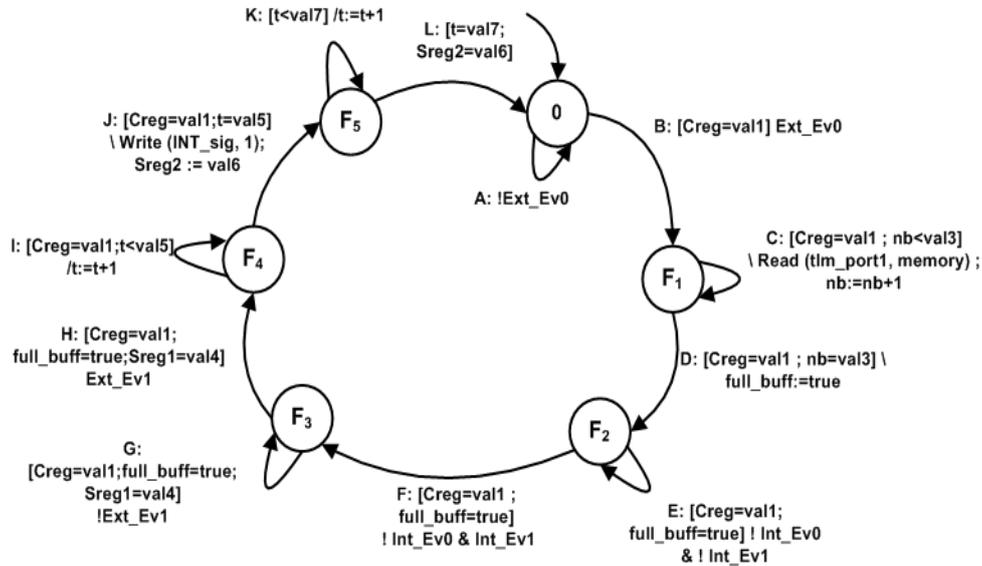
Figure 5.2: An Example of a Slave/Master SystemC-TLM White-Box IP block: Interface and Structure

5.1.2.1 Description of the IP Structure and Behavior:

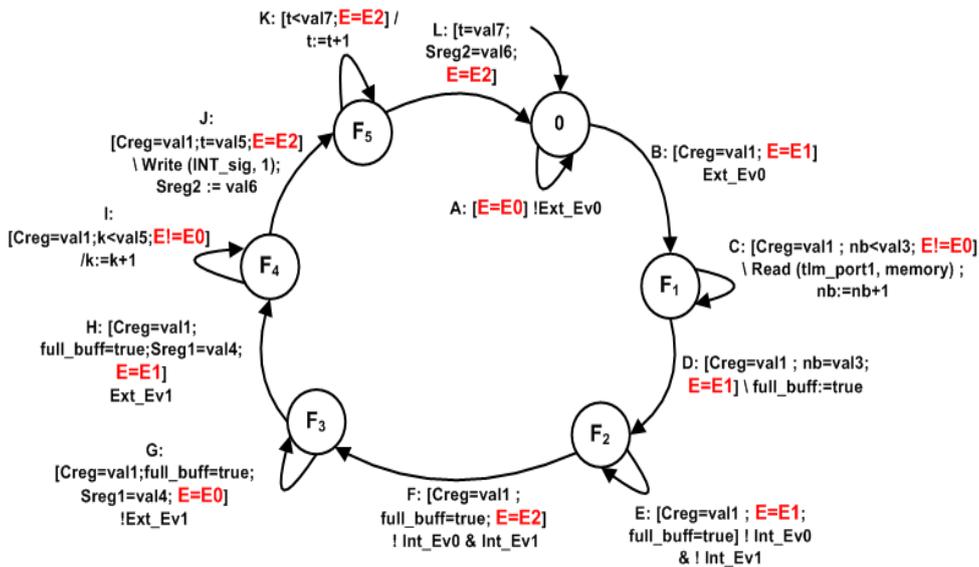
As illustrated by Figure 5.2, the register structure of this IP is composed of memory-mapped registers: a control register, Creg, and two status registers, Sreg1 and Sreg2. This set of registers can be accessed from outside this block via read or write bus transactions sent over its tlm_port2 interface. Moreover, there is an internal register, called internal_buffer, which cannot be accessed from outside the IP and is only used to load the data read from a memory block before processing it.

The behavior of this IP is given by two processes which yield on a wait statement (either on a time or on an event). We distinguish between internal events and external ones. Internal events, denoted by Int_Evi, are defined as events used to synchronize processes within an IP. External events, denoted by Ext_Evi, are defined as events used to synchronize interactions between IP blocks (*i* is the number of the event). This kind of events is notified further to a communication from outside the IP (i.e. upon receiving a transaction or an interrupt). Between each two wait statements, there is a set of non-interruptible (atomic) IP operations, denoted by Fi.

Figure 5.3(a) gives an EFSM-based modeling of the White-Box IP TL behavior depicted in Figure 5.2. This EFSM model results from the product of the individual EFSM models associated with each process in the IP. As it can be seen on this EFSM model, the IP leaves the functionally idle state (state 0) as soon as the Ext_Ev0 external event is



(a) State Transition Diagram of an EFSM Modeling the Functional Behavior of a White-Box IP



(b) State Transition Diagram of an EFSM Modeling the Power Managed Functional Behavior of a White-Box IP

Figure 5.3: Example of the EFSM-Based Methodology Application on the White-Box IP Version

notified upon receiving a transaction that writes *val1* into the IP's CReg1 register. From the EFSM model, this causal relationship between this external event and this write trans-

action can be deduced from the Creg variable value in the transition B predicates. Then, the IP performs a first set of atomic transactions F1 during which it accesses the memory through transmitting to it a *val3* number of read transactions. It fills its internal buffer with this data copied from memory (transition C).

The IP moves from performing the F1 set of operations (functional state F1 in the EFSM) to performing the F2 set of operations (functional state F2 in the EFSM) only if a *val3* read transactions from the memory have been received and its internal buffer is completely filled (equivalent to the boolean condition *full_buff* evaluated to *true* in transition D). Transition from the functional state F2 to the functional state F3 in the EFSM is conditioned by the internal notification of the Int_Ev1 event. Here, the control is transferred to Process 2 of the IP which executes the F3 set of operations. An obvious consequence of the execution of the F3 set of operations is the setting of the Sreg1 status register to *val4* as it can be deduced from the predicates of the transition G in the EFSM model. Note that the control goes back to the Process1 of the IP to execute the F4 set of operations (functional state F4 in Figure 5.3(a)) only if the Ext_Ev1 event is notified by Process 2.

After the F4 execution, the IP functionality blocks on the *wait (time=val5)* statement (used to advance the simulation time by a *val4* time units). Once *val5* time value is elapsed, moving from the functional IP state F4 to the functional IP state F5 is done upon the reception of an INT_sig interrupt signal value (transition J) which internally modifies the memory-mapped Sreg2 status register to the *val6* value. Blocking on the functional state F5 is due to a wait statement for a functional time value equal to *val7* simulation time units. When this functional time is elapsed, the IP returns to the idle functional state waiting for Ext_Ev0 to occur again.

As it can be seen in Figure 5.3(a), states of specific IP registers required to perform a set of operations at a functional state have been expressed as predicates of the transition to this functional state. For instance, note that the predicate *[Creg=val1]* is present on almost all the EFSM transitions. This particular Creg register state indicates that the IP has been already initiated by the application and is hence capable to operate. Note also, that the F2, F3 and F4 set of operations use data in the *internal_buffer* which explains the *[fill_buff=true]* predicate on transitions E, F, G and H of the EFSM in Figure 5.3(a).

5.1.2.2 Building Power-Aware EFSM Models

Let us consider three possible power modes for the IP of Figure 5.2: E0 corresponds to a sleep state (null voltage value), E1 corresponds to a state requiring higher data processing speed (high voltage value), and E2 corresponds to a state requiring a slower data processing speed (low voltage value). These IP power modes correspond to those of its power domain.

Figure 5.3(b) represents the power-manageable EFSM of the white-box IP version obtained by adding power modes predicates on the behavioral EFSM transitions. Note here that, semantically, not all self-loops correspond to a functionally idle state enabling a power-down of the underlying power domain, thus setting the E0 power mode before entering the next functional state. For instance, as long as the IP is in state F1 (in Figure 5.3(b)), it is required to remain in an active power mode during the transition C until it transmits a *val3* number of read transactions to the memory. In the self-loop E from state F2 in Figure 5.3(b), although the system is functionally idle waiting for the internal event *Int_Ev1* to occur, it cannot be put in a power-down mode. Indeed, this internal event is notified by an internal process of the IP which means that the IP is still functionally active as long as the *Int_Ev1* is not notified.

The predicate $[E!=E0]$ on the I transition of state F4 in Figure 5.3(b) requires that the IP is put in an active power mode but never powered-down while blocked in the *wait(time=val5)* statement. Indeed, this kind of wait statements semantically used to advance the simulation time by a processing time value, do not really correspond to an idle functionality. Similarly to this case, a requirement to set the IP power mode to E2 has been specified for as predicate for the k self-loop which refers to a *wait(time=val7)* statement. Actually, this wait statement corresponds to a wait for a functional time that is required to the correct functionality of the IP and is usually given in the IP datasheet. Note that the E2 power mode setting requirement first appears on the J power mode transition as an assumption to enter the F5 operational state. Here, as operations performed in the F5 state do not necessarily require a high processing speed, the designer chooses to impose that the IP is put in lower power mode E2 before entering F5. Otherwise, the IP power mode remains E1.

Note that only self-loops that have a null action and external events as inputs represent sleep candidate transitions (e.g. transitions A and G in Figure 5.3(b)).

5.1.2.3 White-Box Vs. Black-Box

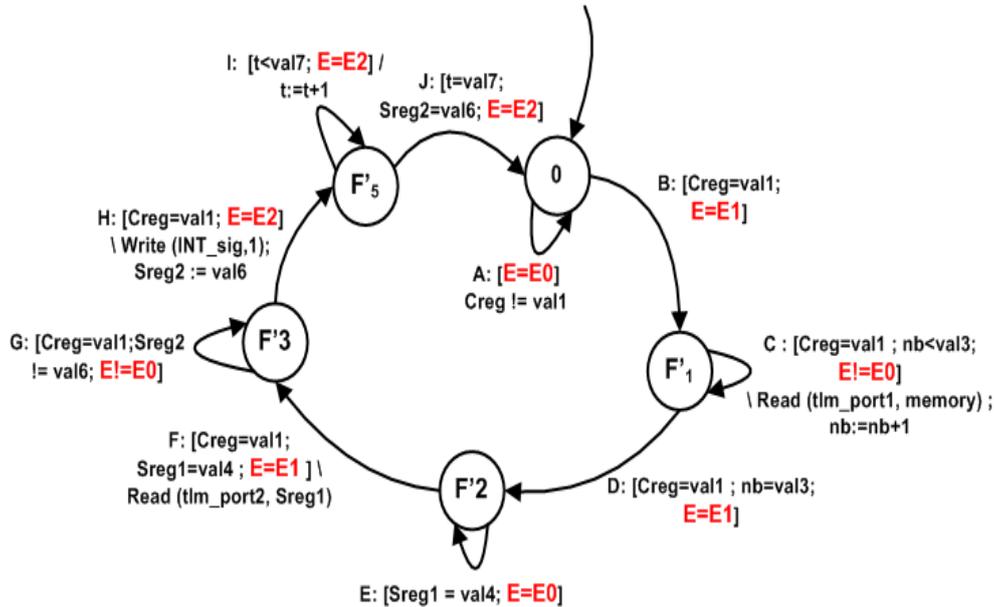


Figure 5.4: State Transition Diagram of an EFSM Modeling The Power Managed Functional Behavior of a Black-Box IP

The EFSM model depicted in Figure 5.4 represents the power-manageable behavior of the black-box version of the same IP of Figure 5.2. Conversely to the white-box case where the functional EFSM model can be drawn based on a full knowledge of the IP source code, the operational states of the black-box functional EFSM model can only be determined through capturing and understanding exchanged transactions at the IP interface.

Actually, most black-box IP cores are software configurable and black-box IPs' vendors are required to offer minimum information, not only about the IP interface signals, but also about each memory-mapped register of the IP. This information is mandatory for the embedded software developer to correctly configure and use the black-box IP. Hence, based on this information and through analysing transactions at the black-box IP interface and monitoring changes in states of the IP's memory mapped (i.e. accessible from outside the IP) control and status registers, the designer can deduce the current functional state of the IP.

For instance, as it can be seen on Figure 5.4, the designer knows that the IP will enter the operational state F'2 if a number *val3* of outgoing read transactions to the memory

block has been reached. Ending of operations in F'2 can be recognized when the memory-mapped SReg1 changes to *val4* value. It can also be recognized that F'3 will be entered upon the reception of a read transaction in the SReg1 memory-mapped register. This transaction will trigger the Ext_Ev1. Note that Ext_Ev1 in Figure 5.4 is the input that fires the H transition on Figure 5.3(a).

Compared to the white-box power-aware EFSM model in Figure 5.3(b), there is less functional states in the black-box case than in the white-box case. Indeed, some functional states in the black-box case correspond to the fusion of several states in the white-box case. For instance, states F2 and F3 in the white-box EFSM (Figure 5.3(b)) are grouped into a unique state F'2 in the black-box EFSM (Figure 5.4) because internal events cannot be observed in black-box IP blocks.

	White-Box	Black-Box
Sleep _{candidate}	A, G	A, E
Ret _{candidate}	(G, {internal_buffer, Sreg1, Creg})	(E, {Sreg1, Creg})
PwC _{candidate}	B, F, J, G, A	B, A, E, H

Table 5.1: White-Box Vs. Black-Box

As a consequence, fewer power domain modes transitions may be specified. For instance, transition to the E2 power mode during the F transition in Figure 5.4 cannot be done in the case of the black-box IP since the transition from F2 to F3 functional states in the white-box EFSM cannot be captured in the black-box case. Alternatively, in the black-box EFSM, only a power mode transition to E1 has been specified before entering the F'2 functional state. Thus, the white-box version has more power candidates (*PwC_{candidate}*) than the black-box one as it can be seen in Table 5.1. This table resumes all the differences between the power-aware white-box and black-box EFSMs on Figures 5.3(b) and 5.4 in terms of *PwC_{candidate}*, *Sleep_{candidate}* and *Ret_{candidate}* sets.

Among the PMPs identification stage benefits is that registers in the *Ret_{candidate}* set become the set of retention registers during the power intent specification stage when opting to a partial retention of the IP state on power-down. Table 5.1 shows that, although the white-box sleep candidate G is the same as the black-box sleep candidate E, the set of retained registers is not the same. Indeed, due to the limited observability of internal state

changes of the black-box IP, dependencies on internal registers values between operational states cannot be detected. Therefore, not retaining states of some internal registers before powering down the IP can lead to an erroneous IP functionality. For instance, to not retain `internal_buffer` before entering F'3 would block some operations performed in F'3 state. Therefore, a verification process that captures this kind of specification failures is mandatory. Although retaining the full state of the IP block avoids this kind of failures, this conservative approach still has great area penalty [96].

5.1.2.4 Using PMPs to Locate Power-Aware Checks in the SystemC/TLM IP Code

The power-aware EFSM model of a SystemC/TLM IP represents a good support to build a reliable power-aware dynamic verification in coherence with the functional IP behavior. Before entering specific functional states, predicates on values of specific IP's registers and power modes represent examples of power-aware properties to be checked before performing the functional state transition. In the white-box case, the IP's power management points (PMPs) in the power-aware EFSM model facilitate the identification of locations in the SystemC/TLM IP code where assumption clauses must be added to guarantee a power management IP behavior in accordance with the EFSM-based specification.

Figure 5.5(a) depicts examples of locations in a simplified source code of the IP in Figure 5.2 where power-aware checking code must be added according to predicates in the power-aware EFSM model. Note that requirements on register values and power modes in the power-aware EFSM predicates are translated into preconditions (i.e. assumptions) of some methods and wait statements in the IP source code. These preconditions correspond to comments in the code of Figure 5.5(a) starting with `//requires`. For instance, as it can be seen in this Figure, before executing the `F4()` set of operations, the IP source code is instrumented with an assumption clause checking that the IP power mode has been just put into the `E1` power mode and that the `internal_buffer`, `Creg` and `Sreg` registers have maintained the adequate state required for the correct execution of `F4()`. In particular, as it is required to check that a slave IP is already put in an active power mode before receiving and processing an incoming transaction, the transport implementation methods in this IP source code (corresponding to read and write methods in Figure 5.5(a)) must be preceded by preconditions that check the IP's power domain mode.

```

1: ...
2: //requires (IP_State != E0)
3: //Read Interface Implementation
4: tlm::tlm_response_status read (data& d,
5:   addr a){
6:   ...
7: }
8: //requires (IP_State != E0)
9: //Write interface implementation
10: tlm::tlm_response_status write (data d, addr
11:   a){
12:   ....
13: }
14: //Process 1 Implementation
15: void process1()
16: {
17:   //requires (IP_State = E0)
18:   wait (Ext_Ev0);
19:
20:   //requires (IP_State = E1)
21:   //requires (Creg = val1)
22:   F1();
23:   //requires (IP_State = E2)
24:   //requires (full_buff = true)
25:   F2();
26:
27:   //requires (IP_State = E1)
28:   //requires (full_buff = true)
29:   wait (Int_Ev0);
30:
31:   wait (Ext_Ev1);
32:
33:   //requires (IP_State = E1)
34:   //requires (full_buff = true and Sreg1
35:     = val4 and Creg = val1)
36:   F4();
37:   //requires (IP_State != E0)
38:   wait (time=val5);
39:   //requires (IP_State = E2)
40:   //requires (Sreg2 = val6 and Creg =
41:     val1)
42:   F5();
43:   //requires (IP_State != E2)
44:   wait (time=val7);
45: }
46: //Process 2 implementation
47: void process2()
48: {
49:   //requires (IP_State = E1)
50:   wait (Int_Ev1);
51:   //requires (IP_State = E2)
52:   //requires (full_buff = true and Creg =
53:     val1)
54:   F3();
55: }
56: ...

```

(a) Example of PMPs placement into the White-Box IP source code in the form of power-aware assumptions

```

1: //A block capturing transactions and checking
2: //power-aware properties at the IP interface
3:
4: ...
5: //requires (IP_State != E0)
6: //Read Interface Implementation that captures
7: //input read transactions at the IP interface
8: tlm::tlm_response_status read (data& d, addr a){
9:
10: //Conveying first the input read transaction to
11: // the IP to fonctionnally handle it
12: tlm::tlm_response_status response=IP.read (d,a);
13: switch (a){
14: case IP_Creg_OFFSET:
15:   break;
16: case IP_Sreg1_OFFSET:
17:   //requires (IP_State = E1) if d=val4
18:   break;
19: case IP_Sreg2_OFFSET:
20:   break;
21: default:
22:   ...;
23: }
24: return response;
25: }
26: //requires (IP_State != E0)
27: //Write interface implementation
28: tlm::tlm_response_status write (data d, addr a){
29:   switch (a){
30:     case IP_Creg_OFFSET:
31:       //requires (IP_State = E1) if d=val1
32:       ... = d;
33:       break;
34:
35:     case IP_Sreg1_OFFSET:
36:       ... = d;
37:       break;
38:     case IP_Sreg2_OFFSET:
39:       ... = d;
40:       break;
41:     default:
42:       ...;
43:   }
44: //Conveying then the input write transaction to
45: // the IP to fonctionnally handle it
46:   return IP.write (d,a);
47: }
48: ...
49:
50: //a process handling output transactions at the
51: // IP interface
52: void process(){
53:
54:   initiator_port.read (address_mem, d);
55:   compt=compt+1;//counts the number of
56:   // transmitted read transactions
57:   //requires (IP_State = E1) if compt=val3
58:   ...
59:   //requires (IP_State = E2)
60:   INT_Sig.write(1);
61: }

```

(b) Example of PMPs placement into a block wrapping the Black-Box IP in the form of power-aware assumptions

Figure 5.5: Using PMPs for Checking Power-Aware Specifications in a SystemC/TLM IP Code

Conversely to the white-box case, the technique of adding power-aware checking code

according to the specifications in the black-box power-aware EFSM model cannot rely on a source code instrumentation method. A good alternative solution to add power-aware checking code in case of a black-box IP is to add a separate block that wraps the black-box IP core, captures and checks exchanged transactions at the IP interface before conveying them to their destination. As a black-box IP behavior can be deduced from specific states of read or written registers or specific features of exchanged transactions at the IP interface (such as the number of transmitted transactions to a specific destination or the transmission of an interrupt signal), power-aware checking codes has to be embedded in the wrapping block before or after the transmission (or the reception) of such relevant transactions.

Figure 5.5(b) depicts examples of locations to add power-aware checking code in a block wrapping the black-box IP of Figure 5.2 according to the power-aware specifications in Figure 5.4. As it can be seen, the wrapping block code duplicates the transport interface implementation methods (read and write methods) of the black-box IP which are classically used to handle read or write transactions received at the slave IP interface. However, their implementation code in the wrapping block is quite different from the black-box IP one. It only aims at checking the conformity of specific power-aware properties of the IP with the EFSM-based power-aware specifications upon the reception of these transactions which are then conveyed to the IP black-box to handle them normally.

As shown in Figure 5.5(b), received read transactions at the IP interface, which are relevant for power-aware checking, are first conveyed to the black-box IP (their callee) in order to be normally handled. In their return path to the caller, power-aware checks are applied to the read data value at the wrapping block level. Conversely, as received write transactions are intrusive in the sense that they may trigger a specific IP behavior, relevant write transactions need first to be checked against power-aware specifications in the wrapping block and then conveyed to the black-box IP to be normally handled. For instance, a write transaction to the Creg register must be first captured at the wrapping block level. If the value to be written in this register is `val1`, then it must be checked that the IP has been already put in the E1 power mode in accordance with the power-aware specification in transition B of Figure 5.4. By doing so, when this transaction is afterwards transmitted to the black-box IP, it is ensured that the operational state F'1 (Figure 5.4) will be entered in safe power-aware conditions.

Transactions transmitted by the black-box IP involving specific power-aware requirements must also be captured and checked in the wrapping block. For instance, in Figure 5.4, the transmission of the interrupt signal INT_sig by the IP of Figure 5.2 drives the IP's F'5 operational state that requires the E2 power mode. In Figure 5.5(b), this requirement has been added in the wrapping IP code process just before the transmission of the interrupt signal.

When comparing Figures 5.5(a) and 5.5(b), it can be concluded that a more detailed and flexible power-aware checking code localization can be done in the white-box IP case than in the black-box one. Moreover, two different methods are required to add power-aware behavior and verification to a functional description of a black-box and a white-box IP behavior. The Chapter 6 goes into more details and proposes a modular modeling approach and a reusable utility to handle each case while taking into account these basic differences.

5.2 Dynamic Contracts for Verification of Power-Aware Properties

In this section we present our simulation-based verification framework used to check power-aware properties throughout the USLPAM verification stage. We begin with an overview on existing design verification techniques followed by a state of the art of power intent verification methods. We then outline the main techniques and mechanisms used in our verification framework.

5.2.1 Design Verification Techniques

5.2.1.1 Static Verification

Static verification, also called formal verification, is used to analyze a formal model of the system without executing it. If the design is not written in a formalism with formal semantics, it needs to be translated into a formal representation (mathematical model system), and its associated formal specification needs to be written as well. Precisely, formal verification is defined as the cooperation between the mathematical model of a

system, a specification language both concise and unambiguous, and a proof method for verifying compliance properties [116].

The methods used for static verification differ in the way to perform abstractions. There are mainly two major types of formal approaches: deductive approaches via automated systems demonstration or algorithmic approaches using model checkers [60] that perform exhaustive exploration of the possible states set of the abstract model in order to prove that a condition is satisfied (or not) to all of the system inputs.

The great benefit of static verification is that it provides a strong and accurate proof since it examines all the possible scenarios in the design. However, in case of complex design systems, practical application of this kind of verification is limited to a part of the design, even to only small blocks that contain mostly control logic such as state machines. In fact, state explosion is the commonly faced problem when extracting the formal model of such a complete system.

Among formal specification languages, one can mention VDM++, Astral, Scade, Lustre, Esterel, Syncharts, and Signal. In the special meta-modeling field, one can mention the Object Constraint Language (OCL) which provides constraint and object query expressions on any Unified Modeling Language (UML) model or meta-model.

5.2.1.2 **Dynamic Verification**

While static verification checks if the system conforms to the specification without executing it, dynamic verification checks if a particular execution of the model conforms to the specification. In dynamic verification, the specification consists in a set of properties expressed in terms of logical properties or temporal logics. When the Model Under Verification (MUV) executes, a set of checkers run in parallel. They monitor inputs to the MUV and extract relevant execution traces such as a sequence of relevant events or function calls, and that is in order to check if the desired properties are indeed satisfied or not.

Two major dynamic verification approaches can be distinguished: white-box and black-box approaches. In a white-box dynamic verification approach, the verification framework is given full access to the MUV implementation (source code). However, in a black-box dynamic verification approach, dynamic verification is done only on components interfaces

that are given public access due to the limited observability of the components source code. Thus, the verification effort does not depend on the specific implementation.

In general, dynamic verification reduces the debug time since it speeds the time to locating difficult bugs by identifying where in a design the bug first appears. However, as simulation does not consist in an exhaustive representation of the system functionality, dynamic verification provides no guarantees that the system can never violate the specification.

5.2.1.3 Assertion Based Verification

An assertion is a quite simple design check embedded into the MUV to verify the assumptions about how such a MUV should operate, both by itself and in relation to the rest of modules with which it communicates. It consists in a conditional statement about a specific behavior or property that is expected to hold. Whenever the design does not behave the way it was intended or a property is broken, the assertion flags the exact time and location of the problem.

Checking with assertions presents several advantages. On the one hand, limitations of formal verification makes checking assertions during simulation more practical as it offers an early indication of a potential problem and reduce the overall debugging time. In fact, when an assertion fires, the problem and its source are immediately identified and debugged. On the other hand, in an assertion-based framework, assertions usually incorporate monitors (checkers) which are modeled according to an effective assertion-based methodology tightly integrated with a larger design methodology. The emergence of assertion-based standards makes the development of monitors fast, modular and methodical. For instance, OVM (Open Verification Methodology) [20] is an assertion-based methodology with a supporting building-block class library for modular verification environment construction. Verification components can communicate with MUV components through transactional interfaces. UVM (Universal Verification Methodology) [2] is a recent Accelera's assertion-based methodology standard which is derived from OVM. Based on a base-class library, this methodology provides TLM-driven built-in automation and testbench capabilities. The OVL (Open Verification Library) is also a library example of predefined assertions that lets the designer use the same assertion specification with different flavors (VHDL, Verilog ...).

Static assertions can be embedded into the code (written in VHDL, Verilog or SystemC) as simple `assert (bool)` statements to check for some properties. However, a more appropriate assertion language such as the Property Specification Language (PSL) [8] or System Verilog Assertions (SVA) [9] is needed to capture complex intended behaviors of the design in a formal way (such as temporal properties that specify sequential behaviors). Methods using such languages are called semi-formal methods because they still rely on mathematical basis but do not provide exhaustive checks. They combine simulation and formal verification approaches and are rather used to overcome the drawbacks of both approaches.

5.2.1.4 **Enabling Design-By-Contract in an Assertion Based Verification Process**

As initially introduced by Bertrand Meyer, the Design-by-Contract (DbC) approach lays out a clear division of responsibilities between a component implementation and client code that uses it. Strongly tied with component-based development principles, DbC enables a modular and safe systems construction by assembling its components [118] [117] [75] [97]. A contract delineates what each component may assume and what each component is obligated to ensure. It is violated when one component does not respect this contract. Such a violation is ideally detected until runtime when components' cooperative behaviors are executed.

Reasoning on contracts can be either performed formally or by using assertion-based mechanisms. On the one side, formal methods rely on validating components composition based on an abstract (formal) specification of components behaviors [59]. On the other side, assertion-based contracts express program invariants, pre- and post-conditions, as Boolean type expressions that have to be true for the contract being validated. This type of contracts has been made available for different languages using either dedicated language for contracts such as Java Modeling Language (JML) for Java or using a separate set of macros to instrument the initial code with contracts specifications such as the `iContract` and `jContractor Library` for Java and `Nana library` for C++.

While formal contracts require a formal specification of components, assertion-based contracts offer the ability to detect errors close to source easing analysis and correction. Nevertheless, defensive programming remains a major and common drawback of assertion-

based contracts to be avoided. Defensive programming refers to the practice of writing additional code to check whether the contract is violated. This practice decreases the code performance during execution and damages the fundamental aim of DbC that is the clean separation between the specification of a component and its implemented behavior. In particular, it is hardly unavoidable when expressing, through contracts, temporal properties that require saving some execution traces and sequences. The notion of time and trace has been rather defined in several state-of-the-art approaches enabling formal reasoning on contracts [84] [151] [108].

In the context of object-oriented programming, the notion of component is applied to a class. The services provided by a class correspond to its public methods (i.e. methods that can be called by another component). Invariants and pre/post-conditions are associated with the class methods and are considered as a contract between the class and its caller.

Two major paradigms have emerged in this context: white-box and black-box testing. In the white-box one, assertion-checking contracts can be included in the component source code surrounding the class methods. This paradigm is useful for component developers who have direct access to the component source code. By contrast to white-box testing, contracts in the black-box testing paradigm check only interfaces violations of a component. This paradigm is rather used when a component is distributed in compiled form only. In this case, contracts should not be embedded inside the component source code and the principle of separation between a component functional behavior and its verification should be absolutely applied in order to ensure reuse and modification flexibility of contracts specifications once a component is packaged for distribution.

SystemC/TLM models can be considered as object-oriented programs since written in C++. Assertion-based contracts principles in object-oriented programming can thus be applied to such models. To the best of our knowledge, the only state of the art specifying contracts for SystemC/TLM models is [51] [50]. In this work, a notion of control contracts has been defined as formal contracts applied to a formal component model for embedded systems called 42. To each SystemC/TLM model, a corresponding 42 formal model can be defined. Then, by assembling 42 components each modeling a SystemC/TLM component behavior, an execution mode for the full system made of components' control contracts can be used in order to easily debug and check the correct behavior of the assembled components.

5.2.2 Verification of Power-Aware Designs

Multi-power domain partitions and management make low power verification a necessity but also an arduous task. The verification complexity of low-power managed designs is mainly due to the high degree of system integration, to the large number of operation power modes caused by the increasing use scenarios in software and to excessively fine power gating granularity requiring multiple and complex power domain levels each involving nested and hierarchical power domains. In general, low power verification aims at ensuring that power and functional components work together reliably at all times once assembled in a single power-domain managed final system. Actually, bugs in a low power managed design can be caused by a variety of reasons: either a faulty low power structure or a faulty control or a faulty architecture. We explain and exemplify each of these reasons in the following. We also highlight the importance of checking some bugs which are relevant at Transaction-Level.

5.2.2.1 Structural Bugs

Checking these errors aim at proving that the power intent is complete and consistent. The most common structural bugs are related to power domain spatial crossings specified in a power intent. Examples are missing isolation cells or level shifters, incorrect isolation polarity, incorrect isolation gate type, redundant isolation, incorrect domain or type of level shifter [137]. Some structural errors can be easily verified at RTL such as isolation polarity. However, there are other errors that can only be checked at the gate level such as wrong isolation gate type. Other structural errors can be rather checked statically whatever the abstraction level since verification relies on an abstract concept such as power state table. Indeed, missing isolation cells and level shifters can be detected statically from the power state table. The common point between structural errors is that it often takes only a static check to detect them. Different industrial tools and simulators have been conceived for that purpose. Ranging from the register transfer level to the gate level, we namely mention the Mentors GraphicsTMs Questa tool [12] for power aware verification and SynopsysTMs MVRC [11] for static multi-voltage rule checking.

The fundamental questions that we have asked in this thesis are: among this kind of errors, which are the relevant and important errors to be checked at the Transaction-

Level? Which are the most suitable verification mechanisms to be used at this level to check such errors? Recall that none of the state-of-the-art works have treated low power design verification at Transaction-Level which makes responses to these research-centric questions original and innovative.

Even by abstracting power intent at TLM, missing protection interfaces may always be statically checked from the PST. Other structural bugs can be checked statically such as the belonging of at least one SystemC module to a power domain and the declaration of each power element in the context of a valid power domain. One can also statically verify that there is no contradiction between power switches placement and power domain states combination in each PSTTMs power mode. Such verifications are essential in order to rigorously respect power intent specification rules imposed by the UPF standard [30] and hence prepare and facilitate the automatic generation of a UPF code from the TL abstract power intent specification.

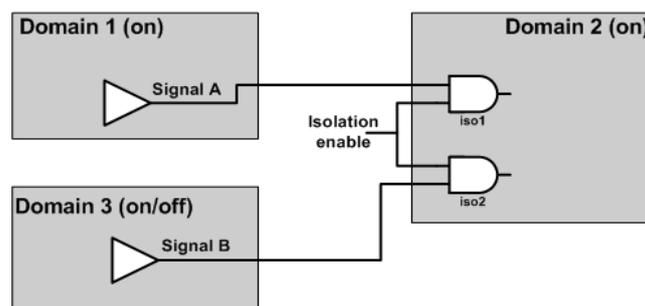


Figure 5.6: Redundant Isolation [137]

However, some structural errors which are relevant at Transaction-Level are rather more efficiently checked during simulation. Figure 5.6 illustrates an example of the redundant isolation error. Here, when domain 3 is switched off, the iso2 isolation cell is enabled at the domain 2 interface. Consequently, the iso1 isolation cell is also enabled alternating hence communications between domain 1 and domain 2 which are still on. Note that the problem is ideally resolved by removing the useless iso1 isolation cell. At Transaction-Level, such a problem can be detected by embedding appropriate assertions inside domain 2 functional blocks. In general, a wrong placement of an isolation cell can be detected dynamically when a random value, generated at a power domain interface in order to mimic this domain interface isolation, propagates and alternates the initial

functionality of some blocks and in some cases even leading to deadlock. In addition, disrespected or missing functional dependencies between power domains by the power management strategy can only be efficiently detected during simulation.

Nevertheless, there are errors like isolation gate type which are impossible to be detected at Transaction-Level. Other errors are more rigorously detected at downstream stages rather than TLM such as the domain or type of a level shifter. Therefore, a refinement of a TL abstract power intent specification and its re-verification at downstream stages is still strongly recommended.

5.2.2.2 Control/Sequence Bugs

A first part of these errors concern the wrong sequencing of power controls for a specific power domain. Checking this kind of errors aims first at ensuring that the PMU functions as intended. For instance, as depicted by Figure 4.8, on power-down, it is required that the retention save signal is triggered just after triggering isolation signal and before the supply of the domain is switched off.

Although both CPF and UPF formats offer a few number of commands and arguments for power-aware verification (e.g. the UPF command `bind_checker` and `-assert_r_mutex`, `-assert_s_mutex` or `-assert_rs_mutex` arguments of the `set_retention_control` UPF command [30] or `assert_illegal_domain_configuration` and `create_assertion_control` CPF commands [29]), they are still needing a RTL power-aware simulator that properly interpret these specific commands. Moreover, current UPF and CPF versions are missing many other semantics to enable expressing other possible power-aware assertions.

Alternatively, PSL has been widely used with the RTL-based power managed models to check correct low-power behavior. Let us go back to Figure 4.8. The following PSL code snippet shows an example of PSL-based assertion `PwAssert_SAVE_before_NPwREQ` that may be inserted into RTL code in order to check that *"On power-down, it is required that the retention SAVE signal is triggered before the domain primary supply net is switched off."*

```
//retention must occur before power-off
PwAssert_SAVE_before_NPwREQ:
assert always (!SAVE;SAVE /-> (!N_PW_REQ before !SAVE));
```

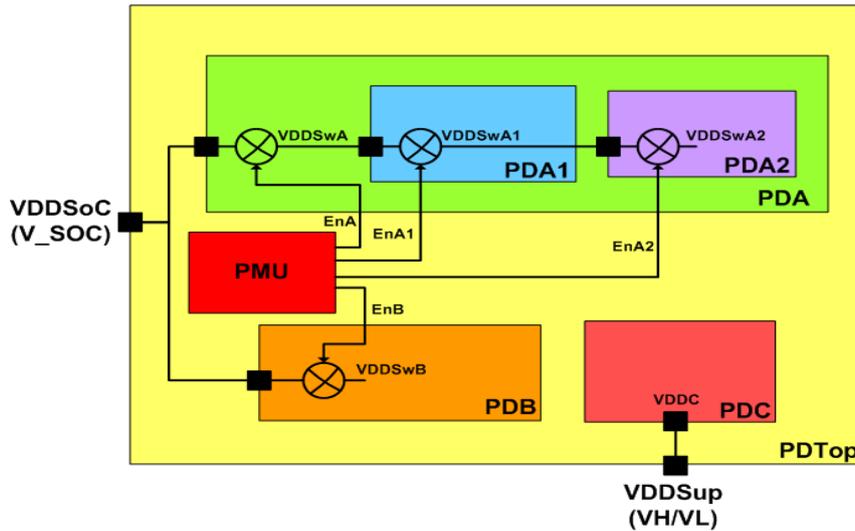
So, in order to enable the TL power control behavior checking, a verification method that allows analog PSL-based checks while being compatible with the abstracted TL power-aware control is needed.

Another part of control/sequence bugs concern transitions sequencing either between system power modes or power domains states. For instance, such sequencing errors may be manifested in the form of rush currents when power domain transitions are performed at the same time. They must also be detected when violating a functional or structural dependency between power domains during a system power mode transition. Indeed, in order to ensure respecting dependencies among power domains states, it is usually recommended to explicitly specify and check transition sequences between power domains states.

Concerning system power modes transitions, numerous transitions sequences may be specified as legal even for a small design with few system power modes. However, specifying legal intermediate transition sequences that help to reach and set a specific system power mode represent a practical solution to limit the number of allowed power modes transitions.

Figure 5.7 depicts examples of specified transitions sequences. It must be checked that power state transitions that occurred during simulation have respected such sequences. In Figure 5.7(c), there are for instance only 6 legal system power modes transitions out of 16 possible ones. Note also that a direct state transition from state 1 to state 3 has been banned. To perform it, an intermediate transition from state 1 to state 2 followed by another one from state 2 to state 3 are imposed. This particular choice may be justified by the absence of a software scenario requiring a direct power transition from state 1 to state 3. It may also represent a personal designer choice in order to limit the legal transitions number.

Contrary to system power modes which are specified according to the embedded software executable use cases, combination between power domains states in each system power mode should be coherent and respectful to functional and structural dependencies between power domains. As it can be seen on Figure 6.8(a), a PST specification misses a specification of transitions sequencing between power domains states for each system power mode. Alternatively, Figure 5.7(c) depicts the sequences of transitions between



(a) Example of Low Power Design

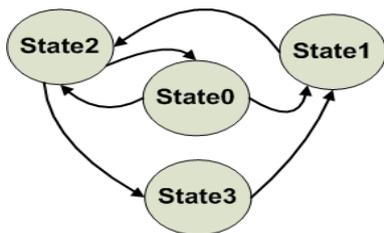
Power Domains States

	PDA	PDA1	PDA2	PDB	PDC
State 0	OFF	OFF	OFF	OFF	OFF
State 1	ON	ON	ON	ON	VH
State 2	ON	ON	OFF	OFF	VH
State 3	ON	OFF	OFF	ON	VL

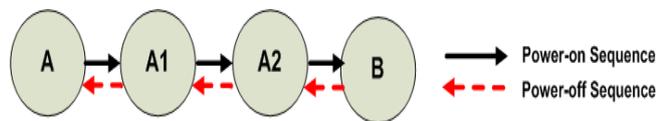
System Power Modes

 Respect of Structural Dependencies
 Respect of Functional Dependencies

(b) State Table for The Example Design System Power Modes



(c) Allowed System Power Modes Transitions



(d) Allowed Sequencing Between Power Domains States

Figure 5.7: A Specification Example of Allowed Power States Sequences

specific power domains states that must be followed during system execution. Here, the power-up sequence A->A1->A2 as well as the power-down sequence A2->A1->A reflect both structural dependencies between these three power domains. These dependencies

are due to a particular placement of power switches as illustrated in Figure 5.7(a). The A2->B and B->A2 sequences impose rather the respect of the functional dependency between the A2 and the B power domains.

The fact that some of the A2 functional blocks need some other functionalities of the B power domain to operate justifies these latter sequences. Nevertheless, according to state 3 in the PST of Figure 6.8(a), powering down the A2 power domain does not necessitate powering down the B power domain. The execution trace must conform to the two specified sequences (Figure 5.7(c) and Figure 5.7(d)) and not cause a deadlock.

A more serious problem with power states transitions can occur during simulation and must be captured. In fact, during execution, transitions can occur for multiple reasons and perhaps conflict with each other. For instance, an incoming phone call may direct the CPU to operate at 1.2v whereas a camera click in progress may be operating the CPU at 1.4v. Therefore, power mode transitions have to be checked also for conflicting transitions. Such errors can be resolved either by specifying an additional power mode that groups the both conflicting power modes' power requirements or by assigning priorities to conflicting transitions.

Checking how many times a power domain has been changed state is also an example of property that helps detecting an error in the power domain partitioning or even in the power control such as needless power events emission.

In general, checking control/sequence properties mainly focusing on transitions between power states require monitoring the transactional traffic passed through modules interfaces and storing the traffic information relevant to check such properties. Temporal relationships ranging from precedence to sequence ones have to be used to express such properties. Therefore, the PSL language represents a good candidate to do so. Its Verilog or VHDL flavor has been widely used to inject such properties into the RTL-based power managed system model. For instance, the following PSL code snippet shows examples of properties to check the sequence between A, A1, A2 and B power domain states transitions on Figure 5.7(d).

```
PwoffA1_unless_A2isPoweredoff: assert always (!EnA2 |-> !$rose(EnA1));
PwoffA_unless_A2isPoweredoff: assert always (!EnA2 |-> !$rose(EnA));
PwoffA_unless_A1isPoweredoff: assert always (!EnA1 |-> !$rose(EnA));
```

PwoffA2_unless_BisPoweredoff: assert always (!EnB /-> !\$rose(EnA2));

Although the SytemC flavor of PSL can be used to express properties for a TL power managed system, using it to check this kind of power-aware properties depends on how the power domains control is modeled at this level. In all cases, this also requires the definition of dedicated monitors.

5.2.2.3 Architectural/Coherence Bugs

This type of errors is related to the interaction between functional features and added power features inside each functional component of the TL model. These errors may also occur if some power and functional features of the whole TL model do not match when two functional components communicate with each other.

State retention is among primary sources of serious errors of this type. Indeed, this particular power feature requires capturing additional functional behavior according to save, restore, power-up and power-down control signals. This added behavior is naturally intrusive and may eventually alter the system functionality or even generate deadlock situations. In order to detect this problem, the designer must check the conformity of the new code-execution to the specification of specific registers state requirements placed into appropriate source code locations. Recall that such a specification has been performed at the identification of PMPs candidates (i.e. the third) USLPAM stage (see for instance the C++ code snippet in Figure 5.5(b)). Detected errors can occur either due to inappropriate moments for retention and non-retention registers state control or due to a missing or even faulty specification of some retention and non-retention registers performed at the power intent specification USLPAM stage.

So, a question that resumes verification issues regarding the state retention problem is: How to check for changes to saved registers and dependencies on unsaved ones?

First of all, contents of registers after save and restore must simply be checked to ensure that functionality has been indeed changed according to the power-aware functionality that has been added. To do so, the power domain registers states must be locally stored just before changing this power domain state in order to compare them with their new state after this power domain control. Secondly, dependencies of some components operations

on unrestored (i.e. non-retained) registers must be carefully checked.

5.2.3 A Modular Power-Aware Verification Flow

Given the panorama of design verification techniques presented above, our USLPAM verification approach implements assertion-based and dynamic power-aware checks. To take advantage of an existing SystemC/TLM simulation platform, the platform modules user code is instrumented with assertions that check coherence between functional and power behaviors. In order to elaborate a modular power-aware verification framework, a design by contract approach has been applied to the different interacting components in the final power-managed system model. In the previous chapter, we have shown how power-aware checking properties are classified into four different classes of contracts (see Section 4.1.6 of the Chapter 4). Each class is dedicated to check interfaces of two communicating types of components. When referring to the three types of bugs in a low power managed system listed earlier, each of our four classes of contracts is used to capture a specific type of these bugs. The class 1 contracts aim at detecting structural bugs while class 2 contracts aim at detecting control/sequence bugs. The classes 3 and 4 contracts serve to locate architectural/coherence bugs.

The contract-based reasoning in our verification approach imposes the use of different types of assertions: mainly, assume assertions to check pre-conditions and guarantee assertions to check post-conditions. Satisfy type statements are also employed either to check invariant conditions or to write additional code required for an assume or a guarantee condition checking. Here, we apply assertion-based contracts principles in object-oriented programming considering the fact that SystemC/TLM models are basically object-oriented programs.

In our approach, assume and guarantee types of assertions are injected in some classTMs methods before or after specific statements execution. Figure 5.8 depicts an example of class 1 contracts that checks interfacing properties between the two power components: supply nets and power switches. By interfaces we mean here the use of the services (i.e. methods) and attributes of one component by another component. The properties being checked in Figure 5.8 are:

(Property 1) *Switching off a power domain requires that all its nested power domains are already powered-down.*

```

void Power_Switch::Set_OFF_State() {
    //Switching off a power domain requires that all its nested power domains (if they exist) are already
    //powered down
    Assume ( Check_Nested_Domains( ), "Invalid state transition of the power switch" + this->GetSwitchName( ) +
    "due to an invalid state of a nested power domain", _LINE, _FILE );

    //Functional code: setting the power switch OFF state
    ...

    //All power switches having an input supply net connected to the primary power net (output of a power /switch
    // of the powered down domain must be switched off as well.
    Guarantee ( Check_Output_Dep( ), "a primary input supply net connected to the output of the power switch"+
    this->GetSwitchName( ) + "is not in a valid state", _LINE, _FILE );
}

```

Figure 5.8: Example of Class 1 Contract-Based Assertions Inserted in A PowerSwitch Class[137]

(Property 2) *All power switches having an input supply net connected to the primary power net of the powered-down power domain must be switched off as well.*

As it can be seen in this figure, property 1 is an assume condition required to be satisfied before switching off a power switch while property 2 is a guarantee condition that must be satisfied before exiting the Set_OFF_State() method of the PowerSwitch class. The set of these two properties form a contract between a power switch component and supply nets components and checking them would ensure a safe power domain state change. If one of these properties is violated, an exception is thrown and a message indicating the origin of the error appears as it can be seen in the Assume and Guarantee clauses in Figure 5.8. Note also that a checking method is called in each of these two clauses (e.g. Check_Nested_Domains() and Check_Output_Dep()). Adding such methods to help checking the property refers to as the defensive programming practice inevitable when using assertion-based contracts. Note that this example checks structural bugs that can be more simply checked statically as explained in the previous section. In the Chapter 6, we explain how this kind of bugs can be checked with the Object Constraint Language (OCL) constraints at the meta-modeling level.

Let us go back to Figures 5.5(a) and 5.5(b) representing examples of PMPs locations added to a component's user code. In these figures, the //requires lines are examples of Assume clauses locations that check for class 3 contracts (i.e. interfaces between functional components and power components (Table 4.1)). In this case, the added checking code ensures that functional components communicate in a safe way and their operational state

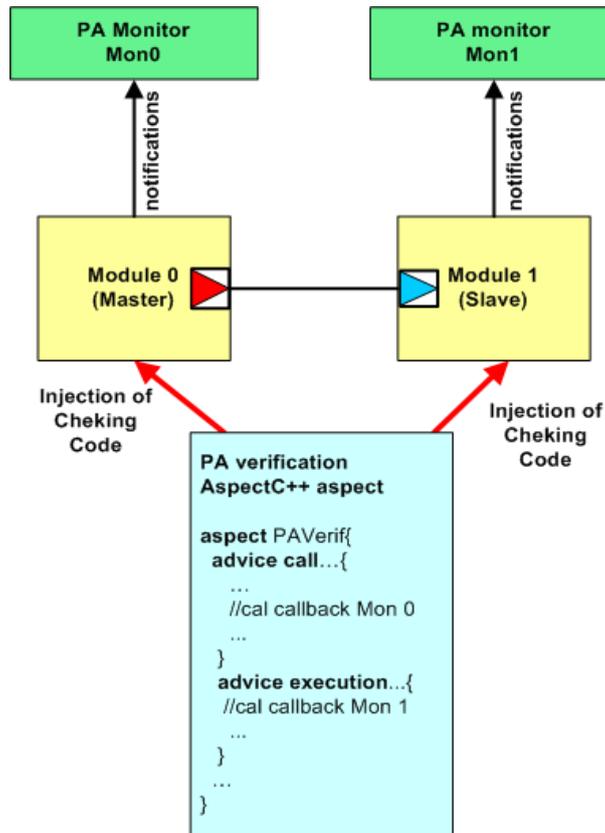
is changed in coherence with their power architecture state.

Note that each functional component of a TL platform must be instrumented with analog assertions at specific locations to check class 3 contracts. For instance, assume clauses must absolutely be added either before a call to a read or a write transport interface method on the master component side, or on entering the implementation of these interfaces (i.e. `read()` and `write()` methods) in the slave component side as shown in Figure 5.5(b). These clauses allow to check that a component's power domain has been already put in the adequate power domain state before appropriately handling the received transaction and its potential effects on the receiver component's operational state.

Checks on the component's power domain state or on specific registers state depending on the arguments passed to or returned from the `read()` or `write()` method is also a required instrumentation-based practice according to our verification approach. For instance, in Figure 5.5(a), the two requirements `//require(IP_State=E1)` and `//require(Creg=val1)` (lines 19 and 20) after the `wait(Ext_Ev0)` statement (line 17), can be alternatively checked before the method `write(val1, Creg1_addr)` returns, since the `Ext_Ev0` event is triggered upon the reception of this transaction.

Note that such instrumentation approach is tedious and disrespects the principle of separation of concerns. In order to check power-aware properties in a modular fashion, each SystemC/TLM module should be attached to a power-aware monitor which observes the functional behavior of the component and executes appropriate power-aware checks when PMPs in the components are reached. Here, each monitor is required to be instantiated when the monitored component is instantiated and the module user code must be instrumented in PMPs locations in order to notify the power-aware monitor. For that, location of the currently reached PMP must be exposed to the power-aware monitor to execute appropriate checks when notified.

A modular solution to expose PMPs locations to a well-defined monitor is to use the Aspect Oriented Programming (AOP) paradigm. When referring to AOP concepts, a power-aware verification (named *PAVerif*) aspect should be defined as a collection of advices implementing crosscutting power-aware verification concerns in a modular way, hence ensuring separation between the functional behavior and the checking features. In our case, PMPs locations would represent AOP joinpoints which are defined as the locations in a user code where aspects are inserted. Advices declared in the *PAVerif* aspect



(a) Example of AOP-Based Power-Aware Monitored TL Example

```

advice execution ("tlm::tlm_response_status Module1:: write (...)") : before {
//Inner Scope
{
//get the Module 1 power-aware monitor
Monitor * Mon1= Monitor->get_monitor_by_index(1);
//Obtain data and address values to send to the monitor
uint32_t data_to_send = (uint32_t) *(unit32_t *) tjp->arg(1);
uint32_t address_to_send = (uint32_t) *(unit32_t *) tjp->arg(2);
Mon1 -> callback_vals (data_to_send, address_to_send);
}
}
    
```

(b) AOP Advice to Expose Arguments of the Write Interface Implementation Method on the Slave Side

Figure 5.9: Using AOP and Callbacks of Monitors for a Modular Power-Aware Verification Framework

code would then specify the code that should run when a joinpoint is reached. They are woven into the user code (i.e. the SystemC/TLM platform source code) automatically using AspectC++ [136].

Figure 5.9 depicts an example of use of AOP advices conjunctly with power-aware monitors. For instance, in the example of advice shown in Figure 5.9(b), the joinpoint (*execution("tlm::tlm_response_status Module1:: write(...)")*:before) specifies that this advice code is executed at the beginning of the write() method in the Module1 source code. The advice code instruments automatically the Module1 code at this location with a callback function *callback_vals()* of the *Mon1* power-aware monitor that is attached to Module 1. This callback implementation inside the *Mon1* code checks the Module 1's power domain state and some of its registers values before handling the received write transaction. As depending on the written data and register address, different power-aware checks are possible (see for instance Figure 5.5(b)). This information must be communicated to the *Mon1* power-aware monitor to be used by its *callback_vals()* callback function. For that, the advice in Figure 5.9(b) uses the built-in AOP call *tjp->arg(n)* which exposes the n-th parameter of the Module 1's write() function.

5.3 Conclusion and Discussion

Unlike the recent works carried out to formalize functional SystemC TL models [62] [121] [91] [51], the EFSM-based method for PMPs identification, presented in this chapter, does not investigate a rigorous manner to formalize the TLM approach. It is rather a way to ease the specification of power intent and power domain management requirements based on a functional description of a TL model.

We have also shown in this chapter how this PMPs identification method facilitates the placement of the power-aware assertion-based contracts in a SystemC/TLM user code. Although the fact that our checking method does not ensure maximum coverage of power-aware errors and that some power-aware properties are more efficiently checked statically (e.g. power manager internal functionality), the AOP-based monitoring approach implemented in the USLPAF framework allows modular checks of relevant power-aware properties and ensures a minimum level of trust in coherence between the initial functional behavior and added power management features.

In the next chapter, we present the different utilities involved in the USLPAF framework to ease the simulation-based USLPAM methodology stages (i.e. power intent specification, PMU modeling, full power-aware simulation stage and power-aware and simulation-based verification). Two of these utilities (the PAL and PwARCH utilities) involve a different power-aware contract-based checking method. Each of these methods is appropriate for a TL component type (black-box or white-box). In the next chapter, we will show how each of them define and use power-aware monitors. An interesting feature of the PAL and PwARCH utilities is that they can be used standalone (as presented in the Chapter 6) or along with the AOP-based verification framework, presented in this section, for more modularity and automation purposes.

Chapter 6

The USLPAL Base Utilities

6.1	Source Code Instrumentation For the USLPAM Application: A White-Box Based Approach	194
6.1.1	Overview of the White-Box Approach	194
6.1.2	Enhancing the USLPAM Using a Model driven Engineering (MDE) Approach	210
6.1.3	Concluding Remarks	214
6.2	Power-Aware Wrappers For The USLPAM Application: A Black-Box Based Approach	215
6.2.1	Overview of the Black-Box Approach	215
6.2.2	Application on Case-Studies	223
6.2.3	Concluding Remarks	232
6.3	The USLPAL Base Utilities for the USLPACom	233
6.3.1	Motivations	233
6.3.2	Power Domain Based Modeling Approach	235
6.3.3	PDMgIF: a Transaction-Level Interface Protocol for Power Do- main Management	240
6.3.4	Application on a Case-Study	249
6.3.5	Locality and Scalability	253
6.3.6	Concluding Remarks	257

THIS chapter presents the different software utilities included in the USLPAL library of our USLPAF framework. First, the main features of the PwARCH utility are described

with particular emphasis on the source code instrumentation approach that this utility enables in order to easily apply the USLPAM methodology on White-Box types of IPs in virtual platforms while meeting all this methodology requirements.

The PAL utility is then presented as an alternative implementation solution built around a power-aware wrapper-based approach in order to facilitate the USLPAM methodology application on Black-Box types of IPs in virtual platforms with support to all the requirements.

Next, the main features of the USLPACom library for a TLM2.0 model of a power domain management protocol interface are presented. At the origins of this utility, we propose a new approach to model this specialized Transaction-Level interface while keeping complementarity to the previous modeling and implementation solutions and guaranteeing compatibility of the USLPACom utility with the remaining USLPAL ones.

6.1 Source Code Instrumentation For the USLPAM Application: A White-Box Based Approach

6.1.1 Overview of the White-Box Approach

The approach presented in this section is part of the articles published in [111] and in [113].

As stated in the Chapter 2, instrumentation and annotation based approaches have been quite used in the state-of-the-art works on adding power information and analysis capabilities to TL virtual prototypes. They all take advantage from an open source code of a SystemC TLM model and a full visibility of its internal structure including objects and attributes. This ability to look inside the TL components and have a detailed knowledge about its internal workings enables an accurate and flexible non-functional information addition, and hence more detailed and complete system performance analysis. Actually, although considered as ad-hoc, instrumentation-based methods for TL analysis purposes are suitable for a transaction level of abstraction since the functional validation of the embedded software is the only primary concern at this level of modeling and other timing,

performance or power analysis concerns remain optional.

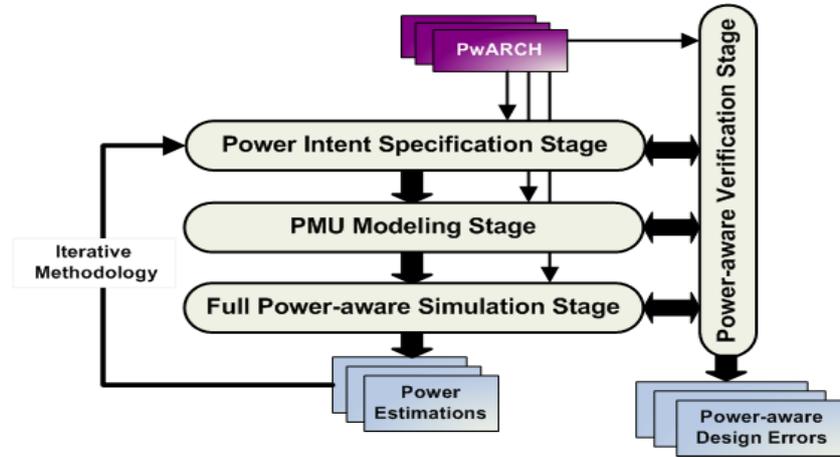


Figure 6.1: Using the PwARCH Utility within the USLPAM Simulation-Based Flow

In the same context, this section presents a source code instrumentation-based approach to apply the USLPAM methodology on white-box types of IPs in virtual prototypes. As depicts Figure 6.1, this approach relies on the use of the PwARCH utility of the USLPAF framework to ease performing each simulation-based stage of the USLPAM methodology flow starting from the power intent specification stage. In the following, we present the main features of the PwARCH utility and we explain how each of them can be used to instrument a TL model source code with required power-aware information at each USLPAM stage.

6.1.1.1 The PwARCH Utility Features

PwARCH is a set of C++ classes easing the instrumentation of an open SystemC TLM source code with the required power-aware features at each USLPAM stage. Figure 6.2 depicts the general class structure of PwARCH while Figure 6.3 shows the composition relationships between the different PwARCH classes and the purpose of each one.

As it can be seen in these figures, a first group of PwARCH classes allows the creation of power objects with abstract UPF specification and simulation semantics. Secondly, PwARCH includes the "DCHFSM" (Domain Controller Hierarchical Finite State Machine) generic class dedicated for power domain internal state control. The third group

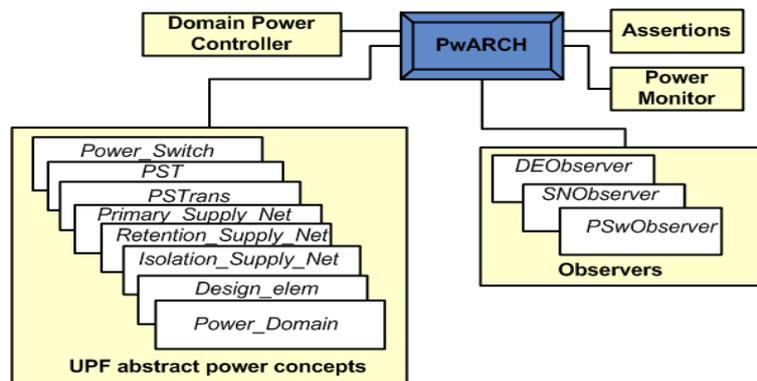


Figure 6.2: PwARCH General Class Structure

of PwARCH classes is dedicated to simulation-based power consumption monitoring and computing as well as specific power-aware properties checking. In the following, the main features of each group of classes are explained. We use example of Figure 6.4 illustrating the overall instrumentation-based approach as a support to show how PwARCH classes contribute to the USLPAM application while meeting all its requirements.

- Abstracting UPF Concepts:** Power domain, primary, retention and isolation supply nets, power state table (PST) and power state transitions (PSTrans) consist in the set of UPF concepts relevant to TLM that have been adopted and whose semantics have been abstracted. While UPF commands are transformed into classes' constructors, options of a UPF command correspond either to classes' attributes exposing features of power components or to classes' methods exposing rather their behavior. A typical example of UPF TL abstraction in PwARCH is that a specification of a power switch control signals required for the power switch behavior simulation is replaced by abstract function calls that set the power switch state to ON or OFF, and that is by simply adjusting the voltage value of its output supply net.

As a support to the USLPAM **Requirement #3** (see the Section reqs), the hierarchy of composition between these abstracted power components imposed by the UPF standard semantics has been maintained. This would allow easily managing these components instantiation and control and help an easy generation of the RTL-based UPF file that reflects the TL abstract power architecture. The UPF abstract concepts part in the UML class diagram (figure 6.3) depicts the different composition, dependency and inheritance relationships between the different PwARCH power components. For instance, power

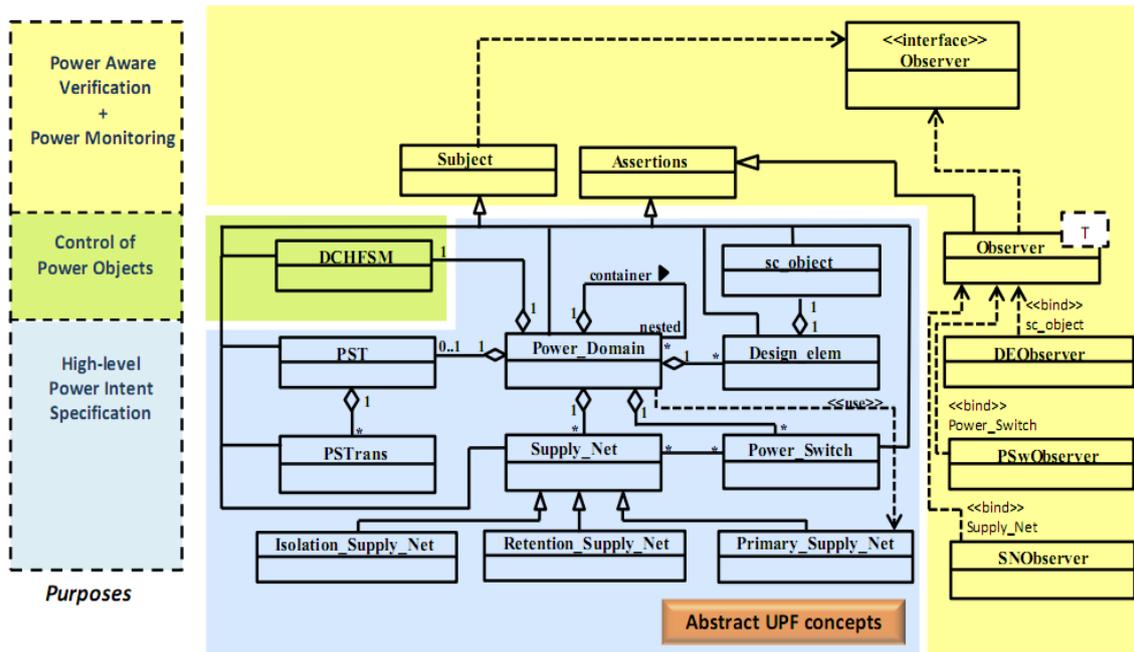


Figure 6.3: Partial Class Diagram for Concepts in PwARCH: Purposes and Relationships

components contributing to define a power domain state must be attached to this power domain when instantiated. This is the case for example for supply net components that must be instantiated in the context of a valid (i.e. already instantiated) power domain as depicts the composition relationship between the power domain class and the supply net class in Figure 6.3. Note also that power domains can be composed and instantiated hierarchically using PwARCH. This facilitates the control of their states and their attached power components (power switches, supply nets ...). The type of a power domain is automatically set during its instantiation by checking if it has been attached to another power domain or not. On the one side, a power domain specified as a container holds a list of all its nested power domains. This list is automatically filled and facilitates the identification of power domains types and the management of their hierarchy and states control. On the other side, a power-domain is automatically typed power-gated, voltage-scaled or non-scaled as introduced in the Section 4.1.3 of chapter 4 depending on the primary supply nets attached to this power domain at their instantiation time. In PwARCH, a primary supply net is by default typed "power" supply net when it is instantiated. Its type attribute is changed to "switched" if it has been attached to a

power switch object as its output supply net. The power domain to which this power switch is attached will be then of type power-gated.

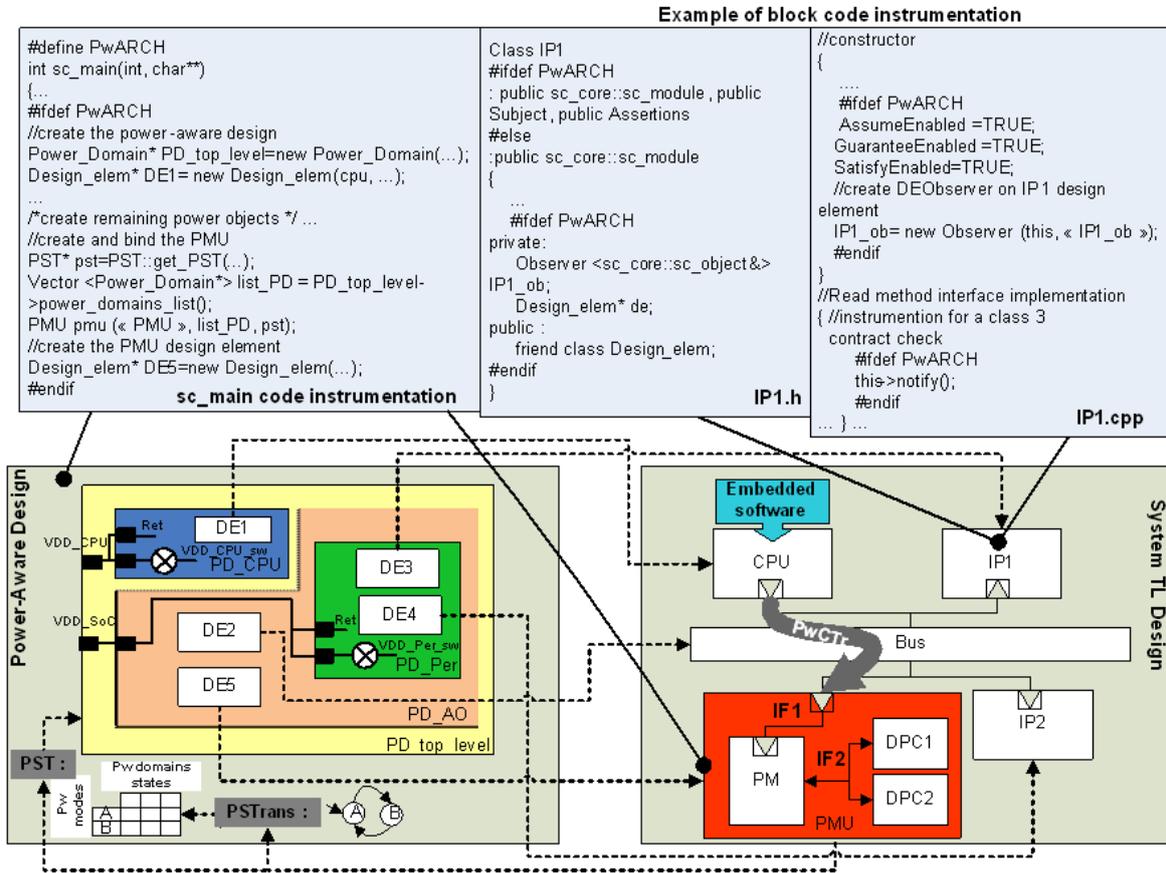


Figure 6.4: The Instrumentation-Based Approach

Other power concepts and features that are not defined by UPF have been added in PwARCH in order to ease power-aware simulation through the internal power/functional interface (Figure 4.5). For instance, In order to reduce the complexity of power management, especially in a design organized hierarchically in power domains, we impose through PwARCH that a PST object is attached to a power domain of type container when it is instantiated. Semantically, a PST resumes the power management strategy applied only to the related container PD.

Among the fundamental added concepts is the design element concept. In a system model described with SystemC/TLM, a design element is semantically defined as a func-

tional SystemC module or sub-module and each power domain is then composed of a set of design elements. Therefore, a design element object must be simultaneously attached to a SystemC object of type `sc_module` in the functional TL model and to a specific power domain when instantiated from PwARCH. As a consequence, a design element object would play the role of bridge between the power design constructed using PwARCH and the functional TL design.

Figure 6.4 illustrates interactions between a power-aware design fully built using PwARCH and an instrumented system TL functional design. Note that a power design is built by augmenting the main class of the TL-design with an additional "PowerMain" code section that uses the PwARCH library. This added code section is preceded by a `#ifdef PwARCH` statement as shown in Figure 6.4. For that, it is compiled only when PwARCH is defined (`#define PwARCH`) in the main code of the platform supporting hence the USLPAM **Requirement #1** that enforces separation between power and functional concerns (see section 4.2). As it can be seen, abstract UPF power objects from PwARCH are instantiated in this code section and some of their attributes are set. The instantiation is done in a specific order to meet the composition dependency rules among the different power objects. Dashed arrows in Figure 6.4 represent pointers to the destination objects in the functional design that tie the two designs.

In order to make a design element properly play its role as bridge between both designs, some instrumentation of functional components is required, and even mandatory when partially retaining a power-gated domain state. Actually, when partially retaining a power domain state on its power down, states of the internal registers of this domain functional components that were not specified as retention registers will be reset to their default value. Recall that this power-aware behavior is particularly intrusive since it alters the internal state of a functional component and potentially its working in case of inappropriate chosen retention strategy. Therefore, mechanisms to infer such behavior into the components functional one while keeping a separation of power and functional concerns methodology are required.

Our source code instrumentation approach based on PwARCH use enables such mechanisms. First, a type (either "full" for full retention or "partial" for partial retention) is assigned to each retention supply net instantiated from PwARCH at the power intent specification stage. Couples consisting of a design element and their non-retention registers

are attached to each partial retention supply net. As non-retention registers correspond to either public access (memory-mapped registers) or even protected or private access (non-memory mapped registers) internal data members of the functional component module, a design element pointing to this functional component must be in anyway allowed to access these internal members and change their values. For white-box virtual platforms, this can be simply achieved by adding a friend declaration to the PwARCH `Design_elem` class inside each functional component header file such as in the IP1 header file pseudo code in Figure 6.4. In order to support the USLPAM **Requirement #1**, this declaration must still be preceded by a `#ifndef PwARCH` statement as depicts Figure 6.4.

- **Power Estimation and Analysis:**

PwARCH allows adding power models and computing power and energy total consumption during simulation according to a power domain based reasoning as specified in the Section 4.1.3.3 of the Chapter 4 fulfilling hence the USLPAM **Requirement #2**. When instantiated from PwARCH, a `Design_elem` object is assigned technology-dependent power information (such as leakage current, capacitance load and clock frequency) related to its referenced functional component. Retention supply nets are also assigned a `RET_FACTOR` value to compute dissipated static power of its corresponding power domain during its power-down period (see Section 4.1.3.3). A switching time delay, used to take into account time and power penalties of a power domain state transition in the total power consumption, is assigned to each power switch object when instantiated from PwARCH.

In addition, a `Power_Monitor` object is automatically instantiated when instantiating the top level container power domain in the "PowerMain" code section and is put in this power domain context. As it was explained in detail in Section 4.1.3.3, this PwARCH `Power_Monitor` object will be alerted during simulation as soon as the power architecture state changes so that it automatically and recursively updates power domains equations according to equations(3), (4), (5) and (6) in Section 4.1.3.3. During its operation, it logs states, voltage and power consumption values changes of the system power domains. At the end of simulation, these logfiles can be plotted in diagrams or viewed in a waveform viewer to analyze the power behavior of the obtained TL power managed system.

In order to trigger the power monitor to perform these power values computations and updates, PwARCH implements a scalable and easy to use mechanism based on the use of a

C++ observer design pattern. As depicts Figure 6.3, PwARCH includes a PSwObserver class which is derived from the generic abstract Observer class and templated on the PwARCH Power_Switch class. PwARCH also includes a SNObsServer class which is derived from the generic Observer class as well and rather templated on the PwARCH Supply_Net class. Each of these two observer classes implements a callback method that calls the *update_total_pw(Power_Domain PD, float Voltage, Boolean Transition)* method of the Power_Monitor object whenever the underlying observer is notified.

Note here that via this method call, the power domain undergoing state transition, as well as its new primary supply net's voltage value are required to be communicated to the power monitor. Through the boolean *Transition* argument, the power monitor is also told whether a transition from sleep to wakeup state (or vice versa) is taking place so that the power monitor adds energy penalties in the updated power consumption values. Moreover, it is worth mentioning that PwARCH implementation provides the power monitor with a database required for its functioning. This database concerns the power domains partitioning and hierarchy as well as on power domain membership and features of each power element in the power architecture.

To enable the PwARCH power estimation capability, the only white-box platform user code that has to be instrumented is the main hardware platform source code. More precisely, in the "PowerMain" code section, each Power_Switch object must be attached to a specific PSwObserver object and each Primary_Supply_Net object of type "power" must be attached to a specific SNObsServer object. The *set_On_state()* and *set_OFF_state()* methods of the PwARCH Power_Switch class already involve a notification to the corresponding PSwObserver object. So, whenever a power switch object in the power architecture is changed state during simulation, its related PSwObserver will be automatically notified to execute the power monitor at the current simulation time. Similarly, the *set_net_state()* method of the PwARCH Supply_Net class already involves a notification to the corresponding SNObsServer object. So, whenever a primary and voltage-scaled supply net object in the power architecture is changing state during simulation, its related SNObsServer will be notified to execute the power monitor at the current simulation time.

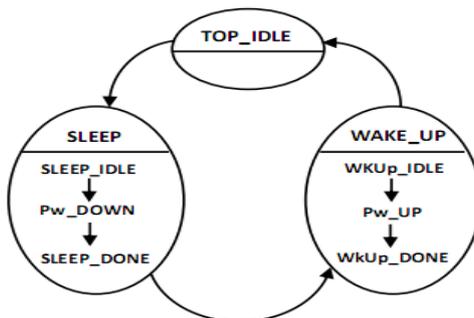
- **Control of Power Domains States:**

The PwARCH utility provides built-in features and mechanisms to facilitate and accelerate the PMU modeling stage of the USLPAM methodology while fulfilling this stage

modeling requirements (section 4.2). The generic DCHFSM class, being part of the PwARCH class structure as depicts Figure 6.2, represents the most important built-in feature of PwARCH. This class implements the generic Domain Power Controller (DPC) behavior since such a controller handles the same state machine and the same power-up and power-down sequences for each power-gated domain. Hence, a DPC object has just to be instantiated from PwARCH and attached to one power domain of type power-gated on which this DPC will act. This can be understood from the composition relation between the DCHFSM class and the Power_Domain class in Figure 6.3. Each instantiated DPC object has just to be bounded to the adequate PM module and will automatically change the state of the power domain to which it has been attached upon the requests received by the PM module. Such PMU modeling techniques imposed by PwARCH enforce the full support of the USLPAM **Requirement #5**.

Recall that the PMU component is modeled according to the general guidelines given in the Section 4.1.4 and to the requirements listed in section 4.2 of the Chapter 4. Recall also that the functional interface (IF1 in Figure 4.7) still represents TLM ports in which power control transactions (PCTr) are transmitted to the PM sub-module. The internal interface (IF2 in Figure 4.7) still consists in a pair of request and acknowledge signals between the PM and each DPC. Conversely to the standard implementation method of IF1 and IF2, PwARCH proposes an implementation method of the power domain management interface (IF3 in Figure 4.7) that specifically characterizes the white-box source code instrumentation approach and fits the UPF-like abstract power-aware simulation semantics provided in PwARCH. As PwARCH uses as a support the UPF standard, it enables declaring a Power State Table, a fundamental concept of UPF, using the PwARCH PST class (Figure 6.3). Therefore, it is noteworthy to mention that PwARCH eases implementing in particular a scenario-based power domain management strategy and designing a PMU model endowed with its three necessary power management interfaces meeting hence the USLPAM **Requirement #6** and **Requirement #2**. As illustrated by Figure 6.4, the dashed arrow starting from the PMU model in the functional system TL model and pointing to the PST object in the power design model attests the role a PST plays to bridge the two designs in our white-box PwARCH-based approach.

Let us now detail specificities of the IF3 modeling approach enabled by the PwARCH utility. Actually, IF3 represents function calls to methods of some power components (e.g. `set_on_state()` and `set_off_state()`) of the PwARCH `Power_Switch` class to respectively



(a) Hierarchical Finite State Machine (HFSM) of the PwARCH DPC model

```

//_linked_PD: power gated domain to which the DCHFSM has been attached;
//sleep_state: current sleep state (SLEEP_DONE/SLEEP_IDLE);
//enable_PwG: signal value sent from the PM to the DCHFSM
//top_state: a state indicating the current power state of the _linked_PD
CASE top_state OF
....
SLEEP :
    IF sleep_state = SLEEP_DONE AND enable_PwG =TRUE THEN
        Sleep_state:= SLEEP_IDLE;
        top_state:=WAKEUP;
    ELSE
        //power and energy values before entering a power mode transition
        CALL Power_Monitor.update_total_pw (_linked_PD, 0, FALSE);
        //wait energy transition time that is the ack_delay data member of _linked_PD power switch
        wait (ack_delay);
        //enter sub-states of the Sleep state
        CALL DO_NEXT_STATE (Pw_DOWN);
        Sleep_state:= SLEEP_DONE;
        Return;
    ENDIF
WAKE_UP :
....
ENDCASE

```

(b) Pseudo-code of a DPC HFSM (SLEEP State)

Figure 6.5: The Power Domain Management Interface in PwARCH

switch on and off a power switch). These methods are called from the DCHFSM process that implements a hierarchical finite state machine (HFSM) in charge of automatically changing the local power state of a power domain under a received request coming from the PM. Figure 6.5(a) illustrates our HFSM model for power-gated domain state control where each state of the HFSM is decomposed in sub-states each executes sequentially.

Hereinafter, Figure 6.5(b) gives a pseudo-code of the HFSM implemented by one DPC process. Particularly, the pseudo-code shows how a transition to SLEEP state, which is one HFSM top level state, can be performed by a DPC module. Here, entering to Pw_DOWN State (DO_NEXT_STATE (Pw_DOWN) function in Figure 6.5(b)), means setting essentially the power switch of the _linked_PD to OFF state. But before that,

a call to the `update_total_pw(Power_Domain PD, float Voltage, Boolean Transition)` method of the power monitor takes place with the boolean transition argument set to `TRUE`. This method call aims at adding the energy cost, induced by the wake-up to sleep state transition, to the total energy consumption. Afterwards, a first check for the presence of any isolation supply nets in `_linked_PD` is done. If an isolation supply net was attached to this power domain, interfaces of the design elements of this power domain are randomly modified. Then, a second check for the presence of any supply nets in `_linked_PD` is done. If a supply net of "partial" type was found, couples of design elements and their non-retention registers attached to this supply net are used to reset these registers values inside the corresponding SystemC modules. Recall that declaring the `Design_elem` class as a friend class within each SystemC module helps the design element to access to all data members of its referenced functional block. The final step after handling retention is to set the power switch of the `_linked_PD` to the `OFF` state. The observer on that power switch will then automatically update power consumption values by calling the `update_total_pw(Power_Domain PD, float Voltage, Boolean Transition)` method of the power monitor while taking into account state power consumption dissipated due to a potential state retention; note that the boolean transition argument is set to `FALSE` this time.

As underlined in Section 4.1.4.1, mechanisms for synchronization between a PMU module and the other functional modules when a power domain state is changing must be considered. In other words, the execution of a master module which sends a power control transaction to a PMU must be blocked until this PMU finishes the transition to the required global power state. To establish such synchronization, we consider that each design element detains a particular event. According to our source code PwARCH-based instrumentation approach, such synchronization is established by the use of that particular event in each design element object declared as an attribute of the PwARCH `Design_elem` class. So, whenever a TL-module sends a `PwCTr`, its corresponding design element object remains waiting (through a SystemC `wait (event)` statement) for the notification of its own event attribute. In turn, when a PMU finishes a transition to the requested global power state in the PST, it notifies by default events of all design elements included in the context power domain of its PST. By this mechanism, the USLPAM **Requirement #4** is fulfilled.

- **Power-Aware Verification:**

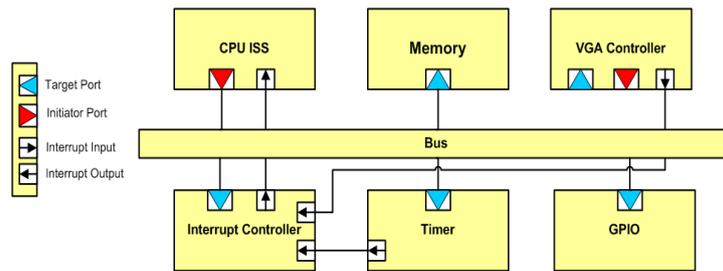
As depicts Figure 6.3, the PwARCH utility provides a generic C++ "Assertions" class that enables the implementation of different types of contracts in an assertion-based manner meeting the USLPAM **Requirement #7**. This class includes *Assume* and *Guarantee* methods used to check respectively assume and guarantee power-aware properties. These two methods raise an exception when their boolean arguments are false. In some cases, to check that an assume or a guarantee property is not violated, a set of conditions have to be satisfied. Hence, another method named *Satisfy* is used to check a condition in such a context. In other words, *Assume* and *Guarantee* methods can sometimes call a set of *Satisfy* methods to check that the specified property is correct. A message reporting the source of the error and generated by the exception can be appended as an argument to each of these three methods.

To apply assertion-based contract checking inside a class, the class being checked must inherit from the Assertions class and used types of checks inside the class must be enabled as it can be seen in the IP1 code in Figure 6.4 supporting hence the USLPAM **Requirement #8**. Moreover, note that in Figure 6.3, the Assertions class is inherited by all the PwARCH classes which are involved in the power objects specification or their control. These added assertions, do neither affect the functional behavior of the system, nor cause side effects on individual objects. Nevertheless, defensive programming implies writing additional code used in particular to verify arguments of *Assume*, *Guarantee* and *Satisfy* methods. Such additional codes have been implemented in "PwARCH" and so, are hidden to the user in order to facilitate and speed the verification process.

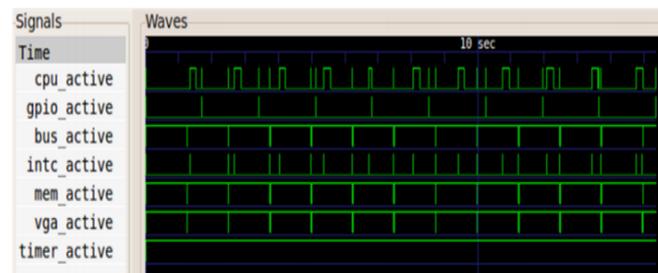
Actually, contracts of class 1 are already implemented inside PwARCH classes. Thus, they are transparent to the user of the PwARCH utility. However, contracts of class 2 must be manually inserted into the PMU code. Similarly, contracts of class 3 and 4 are inserted inside the source code of the other TL-hardware components using DEObserver objects. Indeed, each DEObserver object is attached to a design element object when instantiated from the PwARCH DEObserver class as depicts Figure 6.3. Then, as shown in the IP1 implementation file in Figure 6.4, the source code of hardware functional components must be instrumented with lines of code to notify the related DEObserver object during simulation where a contract checking is relevant. When notified, the DEObserver checks the validity of assume and guarantee properties of a specific type of contracts through calling adequate methods in Assertions class. More details on the USLPAM simulation-based power-aware verification approach have been given in the Chapter 5. In addition, source

code instrumentation with DEObserver objects notifications as imposed by PwARCH can be automated using the aspect-oriented approach presented in the Chapter 5 as well.

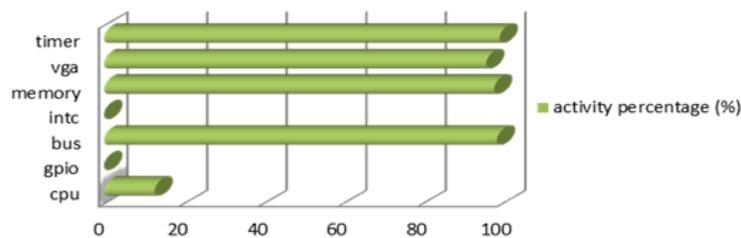
6.1.1.2 Application on a Case-Study



(a) The Case Study Platform



(b) Activity Waveforms of Hardware Components



(c) Activity Percentage per Component

Figure 6.6: The Case-Study: Architecture and Transaction Flow Analysis

To demonstrate the white-box approach, we consider an existing Approximately-Timed (AT) [124] TL-platform (Figure 6.6(a)) with no power management features. The embedded application implements Conway’s game of life. The CPU computes a first image by reading and writing from/to the Memory. Then, peripherals are initialized. The VGA

Controller uses a double-buffer to avoid visual glitches when the image changes. First, it reads the image from the Memory (first buffer) and displays it. Games of life iterations are cadenced by the Timer. Hence, an interrupt which is raised by the Timer, is driven to the Interrupt Controller which drives it to the CPU. Then, the CPU handles this interrupt by computing a new image in a second buffer while communicating again with the Memory. Henceforth, the VGA Controller is informed by the CPU about the new image address, and will display this new image after the display reaches the end of the screen. A button mapped as a GPIO is checked periodically. This SW flow is then periodically repeated.

First, a software flow analysis is performed in order to determine possible system scenarios (i.e. use cases). This task was automated by attaching observers on input and output ports of each component. By detecting these ports state changes, these observers trace the activity of the corresponding component during simulation. As a result, the waveform shown in Figure 6.6(b) was obtained and statistics about the total percentage activity of each component was reported as depicted in Figure 6.6(c). Contrary to the VGA Controller, Memory, and bus components which were active most of the simulation time, the CPU component was functionally idle for successive time durations.

A viable power architecture solution must hence allow energy savings of the CPU during its periods of idleness. This is achieved by powering the CPU down (so by placing a power switch in its power domain) or by supplying it by a lower primary supply voltage (as considered in our power intent solutions). Furthermore, note that such activity traces facilitate defining a power state table (PST) according to a power architecture specification. For instance, the VGA Controller and the Memory activities are strongly correlated when displaying an image (Figure 6.6(b)). Therefore, in a "display" system power mode of a PST, the power domains of these components must be both powered-up.

As shown in Figure 6.7, different power architecture alternatives have been elaborated and evaluated while taking into account this SW flow. Figure 6.9 depicts as well the hierarchy and characteristics of the different power domains according to alternatives (b), (c), and (d) of Figure 6.7. Note that the power domains partitioning and hierarchy, as well as the membership of hardware blocks (design elements) per power domain are different in each of these alternative. In particular, the alternative (a) corresponds to a unique always-on (i.e. never switched off) power domain that groups all the platform HW blocks (Figure 6.7, alternative (a)). Figure 6.8 shows the power state table (PST) and legal

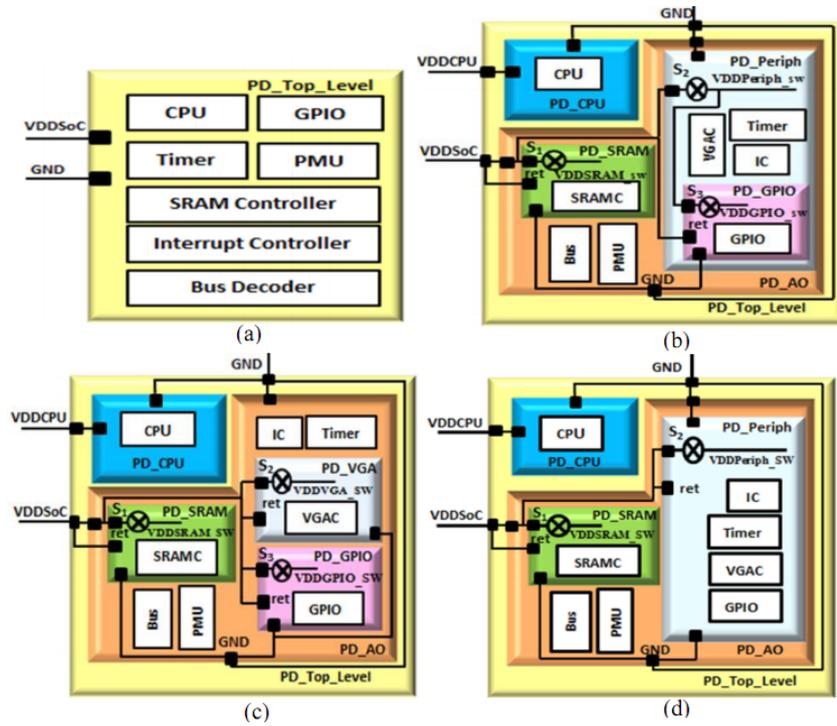


Figure 6.7: Power-Aware Architecture Alternatives

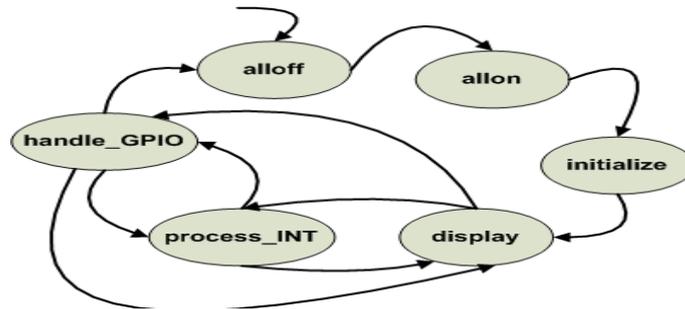
power state transitions (PSTrans) corresponding to the alternative (b).

HW components of this TL-model have been implemented on a Virtex-4 FPGA device. The Xilinx Power Estimator tool has been then used to get technology-dependent power characteristics (such as leakage current and load capacitance) which are used to feed power models of each DE. Results show that (b), (c) and (d) alternatives in Figure 6.7 provide at least 90% of energy savings compared to a unique power domain design ((a) alternative). The (b) alternative represents the most energy-efficient power domains partitioning since about 58% of energy savings is observed compared to (d) alternative and 7.3% compared to (c). Furthermore, the obtained power-aware simulation speed remains similar compared to the non-instrumented version. For instance, simulation time for alternative (b) is only 0.03% slower than alternative (a).

Table 6.1 shows a set of violated contracts further to errors done when elaborating the (b) alternative (Figure 6.7, alternative (b)) using our methodology. Here, simulation is only run after the implementation of all stages. Violated assume and guarantee properties were reported during the simulation period (16 seconds) in a log file. Note that because of

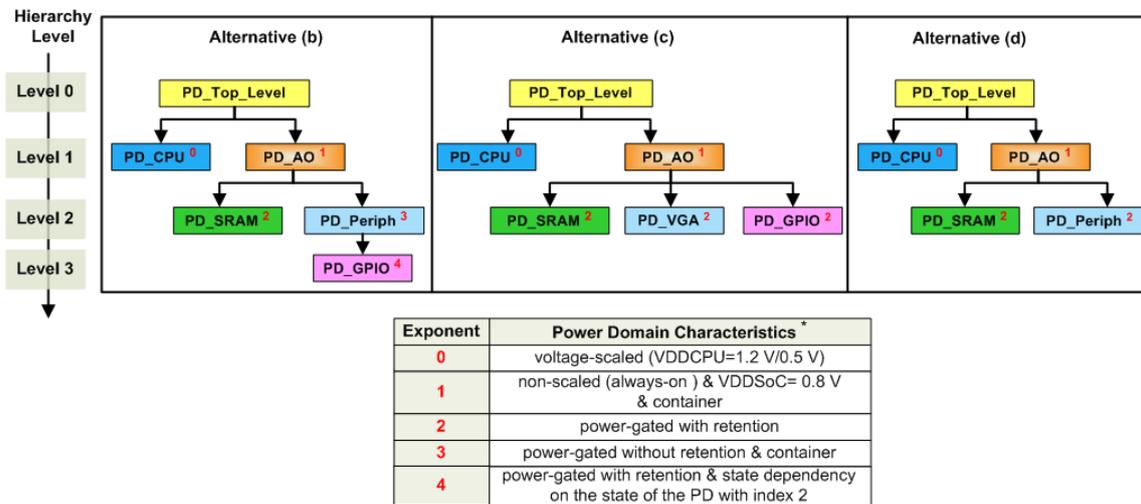
GPS \ LPS	VDDSoC	VDDCPU	VDDSRAM_SW	VDDPeriph_SW	VDDGPIO_SW
alloff	OFF	OFF	RAM_OFF	PER_OFF	GPIO_OFF
allon	ON	ON_H	RAM_ON	PER_ON	GPIO_ON
initialize	ON	ON_H	RAM_ON	PER_OFF	GPIO_OFF
display	ON	ON_L	RAM_ON	PER_ON	GPIO_OFF
process_INT	ON	ON_H	RAM_ON	PER_ON	GPIO_OFF
handle_GPIO	ON	ON_H	-	PER_ON	GPIO_ON

(a) Power State Table (PST)



(b) Set of PSTrans

Figure 6.8: Application of the Power Intent Specification Stage



* Except the PD_Top_Level, each power domain is nested in at least one other power domain (see the power domains hierarchy relationship above). In case of a power domain of type container, we particularly mention this in this list.

Figure 6.9: Power Domains Hierarchy and Characteristics in Each Power Domain Partitioning Alternative

Inserted Fault	Total Errors Number	Violated assertions	The Component throwing the assertion error	Contract class
The transition from “display” to “handle_GPIO” PST power modes is not authorized	155	The transition from “display” to “handle_GPIO” PST power modes is not authorized	PMU (Power Management Unit)	2
		An activity is noticed in the “gpio” block while PD_GPIO is inactive	“gpio” hardware block	3
In “all_on” PST power mode, PD_Periph is powered down while the PD_GPIO is powered on	6	Invalid state transition of the power switch S ₃ due to an invalid state of its input supply net	Power switch S ₃	1
		An activity is noticed in design element “vga” while PD_Periph is inactive	“vga” hardware block	3
		An activity is noticed in design element “timer” while PD_Periph is inactive	“timer” hardware block	3
Only the “vga” unit activity is blocked just after sending a power control transaction to the PMU in order to set the “display” power mode	44	The PM cannot perform a power state transition of the PD_CPU because “cpu” design element is still functionally active	PM (Power Manager)	4
		To display an image, the system power mode does not match the “display” power mode configuration as specified in the PST	PM (Power Manager)	3

Table 6.1: Excerpts of Power-Aware Verification Results

a single inserted fault multiple violated contracts of different classes were detected. This demonstrates the strong complementarity and coherence between all classes of contracts implemented by our methodology.

6.1.2 Enhancing the USLPAM Using a Model driven Engineering (MDE) Approach

The MDE approach presented in this section is part of the article published in [110].

According to our source code instrumentation approach based on the use of PwARCH utility, a power intent alternative specification is performed through the manual writing

of a "PowerMain" code section at each USLPAM iteration. Recall that, at the USLPAM power intent specification stage, the designer instantiates within the "PowerMain" code section the required power objects from PwARCH in a specific order so that the different UPF-like composition relationships between the PwARCH power concepts are respected. So, when designing complex power intent alternatives with a large number of hierarchically structured power domains, this manual instantiation task would represent a real burden for the designer since too much modeling and debug time would be needed to correctly structure the power design. Thereby, the aim of rapidly exploring different power intent alternatives at TLM and early deciding about the most energy-efficient one would be strongly constrained.

In addition, we have mentioned in section 3.1.1.1, that connecting our Transaction-Level power-aware design flow with the classic RTL low-power UPF design flow can be done through automatically generating the UPF file description from the abstract specification of the most energy-efficient power intent alternative deduced at the Transaction-Level using our USLPAM flow. Here, the relevant question any reader might ask is: *How to generate a complete RTL-based UPF specification from a "Power-Main" code section that uses only the abstract UPF-like semantics of PwARCH and misses a set of UPF concepts and semantics not relevant at TLM ?*

To overcome these two major bottlenecks, we propose in the following a Model-Driven Engineering (MDE) approach to generate on the one hand correct Transaction-Level power intent specifications in the form of "PowerMain" code sections, and on the other hand, a UPF standard file describing an energy-efficient power architecture of a SoC. This MDE approach enhances our proposed USLPAM methodology flow since it accelerates the low power design intent space exploration (LPDISE) by fully automating the specification of power intent alternatives while verifying in parallel related structural properties. Using this approach, an automatic generation of an efficient power intent specification fully described with UPF commands is also enabled. As a main consequence, the development and debug time required to manually write correct abstract UPF-like specifications at TLM and correct UPF specifications at RTL are hence significantly reduced.

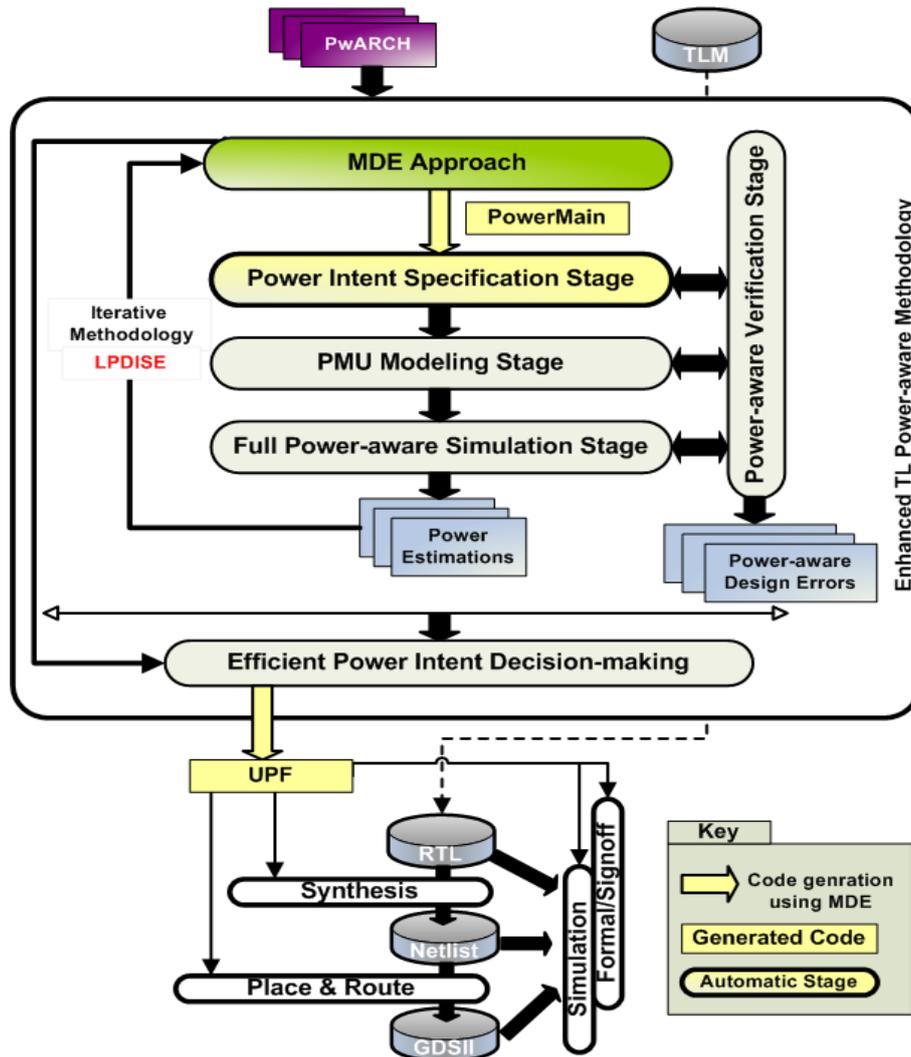


Figure 6.10: MDE Approach Integration in the USLPAM Simulation-Based Flow

6.1.2.1 The Proposed MDE Approach

Following a Model Driven Engineering (MDE) approach is a well-suited solution for our automation purposes. Indeed, as explained earlier in section 2.1.2, using the Model Transformation (MT) key aspect of any MDE development process allows producing executable models (or codes) from high level models. Each MT is performed using a transformation engine based on a source model and transformation specification rules to generate a target model. Among the main MDE features introduced in section 2.1.2 in the Chapter 2, the specified transformation rules can be modified or extended allowing definition of a new

MT targeting a different model. Thereby, several MTs can be defined based on the same high-level abstraction model but generating different target models.

According to our automation purposes, Model transformations that should be used in our proposed MDE-approach are only Model-to-Text (M2T) transformations: only executable models (codes) are generated from a specific high-level model. As shown in Figure 6.10, the USLPAM simulation-based flow has been extended with a MDE initial stage. This stage automates the power intent specification by automatically generating the "PowerMain" code section based on a Power Intent (PI) metamodel use. The MDE approach is applied at each iteration.

According to the USLPAM methodology, after specifying a system power intent alternative, the augmented design model is simulated to check for class 1 contracts. These contracts specify structural properties as well as relationship between different power objects in a power intent specification. As an example, a contract is used for checking the validity of primary power nets' states when defining power modes of a power state table. These contracts figure in the PwARCH library as preconditions and postconditions on some methods. They are implemented as assertions allowing hence simulation-based verification. However, contrary to the rest of contracts classes, static verification of class 1 contracts would certainly be enough. For that reason, this step in the power-aware verification stage has been automated through applying such kind of contracts to the high-level source model. In order to produce a structurally correct "PowerMain" code, this MDE-based verification step is henceforth done during the MDE-based power intent specification. That is why such a step has been totally migrated to the power intent specification stage of the USLPAM methodology and joined with it as shown in Figure 6.10.

Furthermore, the USLPAM methodology allows exploring different power management solutions for a SoC described at Transaction-Level. Each solution includes a power architecture and a PMU model controlling this architecture. It is fully simulated at TL. By comparing the different solutions, the most energy-efficient power architecture can be identified with its valid functional PMU. As the selected power architecture uses an abstract specification of the UPF standard, it can be fully transformed into a UPF source code. This code can hence represent a reference standard file for the Register Transfer Level (RTL) design team. Indeed, RTL designers can attach the generated UPF file to a

synthesis tool which is able to capture UPF power intent. Later, the UPF file specification can be refined and verified in an incremental way throughout the RTL to GSII design flow. The corresponding PMU TL-model can also be used as an executable specification to write its corresponding RTL code.

The most important benefit of automating UPF code generation using our MDE approach consists in the high degree of confidence the designer can have in the correctness of the generated UPF file. Indeed, due to implicit and explicit properties added to the PI metamodel, defining a UPF-file is no more error-prone: the generated UPF file is correct regarding to rules and semantics defined by the UPF language and standard [30]. As a consequence, this reduces significantly the verification and validation cost of a UPF power specification at levels of simulation lower than Transaction-Level.

The way of automating the "PowerMain" code generation and the UPF code generation is described in A as well as results illustrating the efficiency of this automatic code generation step.

6.1.3 Concluding Remarks

Taking advantage of the source code accessibility and instrumenting it with additional power-aware features is a fast, flexible and subtle method especially if the source code locations requiring instrumentation can be easily identified. The PwARCH utility use makes the instrumentation task easier across the different modeling techniques it presents, while ensuring full compliance with the requirements imposed by the USLPAM methodology. The MDE approach use eases much more this instrumentation task at the first simulation-based stages of the methodology. Although few additional refinements of the UPF file generated from abstract power intent specifications are still required to fit absolutely a RTL use, the MDE approach widely contributes to accelerate LPDISE and save time and modeling effort of the RTL design teams.

More generally, the implementation white-box approach of the USLPAM methodology mainly aims at rapidly simulating the effects of power gating and multi-voltage architecture alternatives on the functionality of the entire system as well as on the obtained energy savings. This rapidity is, first, due to the use of a particular power domain management ineterface (IF3 in Figure 4.7) instead of power gating control signals (e.g. power switch

control signal, retention save/restore signals ...). Second, this is due to the introduction of the design element concept which mirrors the impact of the power behavior inside a power domain on functional blocks of this same domain. Indeed, directly adding such an intrusive behavior inside the hardware blocks would instead require rethinking functionality and synchronization. Conversely, the white-box approach succeeds at separating power and functional concerns despite the sophisticated modeling of the power managed behavior and the power connections it implements.

Like most approaches based on source code instrumentation, our proposed white-box approach can absolutely not be applied to black-box TL-platforms due to the different constraints exposed by black-box IP cores. The following section, lists these different constraints and explains how overcoming them using an alternative black-box approach.

6.2 Power-Aware Wrappers For The USLPAM Application: A Black-Box Based Approach

6.2.1 Overview of the Black-Box Approach

6.2.1.1 Constraints of the USLPAM application on Black-Box Virtual Platforms

Let us first detail how specific features of a TL black-box IP constraint the application of our methodology on a TL platform with black-box IPs.

- **TL Black-Box IP Main Features**

Recall that the black-box basic feature is the limited observability of internal state changes of IP cores. However, without looking inside a black-box IP, the developer can in most cases determine its behavior. Actually, most black-box IP cores are software-configurable and their operational status can be determined through capturing and analyzing exchanged transactions at their interfaces. For instance, read or write transactions to memory-mapped control and status registers (CSRs) may give information about current operations of this IP. In addition, IP vendors offer minimum information concerning mainly the IP interface signals and memory-mapped registers of each IP (e.g. description of their offset and bit fields' access). This kind of information is mandatory for the

embedded software developer to correctly configure and use a black-box IP. In particular, only memory-mapped registers of a black-box IP are usually public in virtual prototyping tools so as to facilitate the debug of a packaged and distributed IP.

- **Constraints on Power Intent Specification & Simulation**

As it was depicted in the white-box approach, power-aware behavior may be intrusive and alter the IP initial functionality. For that, care must be given when simulating behavior of an IP enriched with power intent. In particular, we believe that the specified state retention strategy may alter the IP functionality if not well chosen. Recall that in our work, the retention-register approach based on replacing a standard register with a retention register is used. Recall also that in a retention register, state is locally preserved during power-down and restored at power-up (see Chapter 2, Section 2.1.5) [96]. So, all non-retained registers must be initialized on power-down, so that they power up in the reset condition. A block state can be fully retained (i.e. all its registers are replaced with retention ones). However, this can incur an area penalty in some designs [96]. Therefore, application of a partial IP state retention is almost efficient.

At Transaction-Level of modeling, simulation of partial state retention requires only resetting non-retained registers of an IP during its power-down while states of retention registers remain untouched. Using the USLPAM methodology, retention and non-retained registers of each block are specified at the power intent specification stage according to the power domains' $Ret_{candidate}$ sets identified in the Power Management Points (PMPs) identification Stage. The initialisation of non-retained registers state is performed at the PMU modeling stage.

In partial retention, only memory-mapped registers represent possible candidates of non-retained registers in a black-box IP. This is due to the public access given only for this type of registers. So, resetting these registers from outside the black-box IP while powering down is possible. Remaining registers such as internal memories and buffers are usually made private with no access from outside the black-box IP. So their state cannot be changed. Unlike the white-box IP case, such registers are still considered as retention registers in the black-box case. This constraint limits possible power intent alternatives that preserve the correct initial behavior

- **Constraints on Power-Aware Contract-Checking**

The major constraint related to the power-aware verification stage is that power-aware

assertions cannot be embedded into a black-box IP source code. In the TL white-box IP case, atomic operations (i.e. non-interruptible) can be surrounded by class 3 and 4 checks included in the IP source code. As an example of class 3 precondition properties, an IP operation can only be performed if a specific register state has been retained during the last power-down (P1).

In the black-box version, two constraints can be faced for (P1) checking. First, this check is only possible if this operation can be accessed from outside the IP. Otherwise, if the beginning of this operation can be identified through capturing transactions to a specific memory-mapped register at the IP interface, (P1) can be placed before receiving such transactions. Second, states of only memory-mapped registers can be checked from outside the black-box IP. These constraints limit the number of power-aware properties that can be verified in case of a black-box IP and impose a particular checking method.

In the following, two critical questions are addressed: *How power-aware simulation and verification can be achieved without accessing the IP internal structure or requiring source code changes? How the USLPAM methodology simulation-based stages can be applied on TL platforms including black-box IPs while taking these constraints into account?*

6.2.1.2 Power-Aware Wrapper Features

The proposed black-box approach consists in encasing each black-box IP of a platform in a power-aware wrapper. By using this approach required power-aware features are not hardcoded into the IP component but are rather layered on top of it. Hence, power and functional concerns of an IP are separated.

This specialized power-aware layer has two main features: the first is to specify power intent for the wrapped IP. The second consists in checking the relevant power contracts properties. Figure 6.11 depicts the general structure and features of a power-aware wrapper.

- **Power Intent Specification and Simulation**

A power-aware wrapper includes power intent, as well as mechanisms for simulating power-aware behavior of the black-box IP. It provides a power interface that connects the wrapped IP to the power management unit (PMU) as illustrates Figure 6.11. It also

allows modifying the internal state of the wrapped IP as soon as changes occur on the power interface.

A power interface contains at least a voltage signal (e.g. VDD_Sw in Figure 6.11) representing the IP primary power net. It can also include a retention voltage signal (e.g. VDD_Ret in Figure 6.11) which supplies retention registers of the wrapped IP during power-down. This interface gathers also control signals handled by the power controller of the wrapped IP fulfilling hence the **Requirement #5** of the USLPAM methodology. These signals are mainly used to save or restore retention registers content on power-down or power-on and to reset (partial reset) not retained registers on power down. For instance, in case of applying a partial retention strategy, retention registers are saved on power-down (e.g. when the save control signal in Figure 6.11 is asserted) and restored on power-on. However, not-retained registers are initialized on power-down.

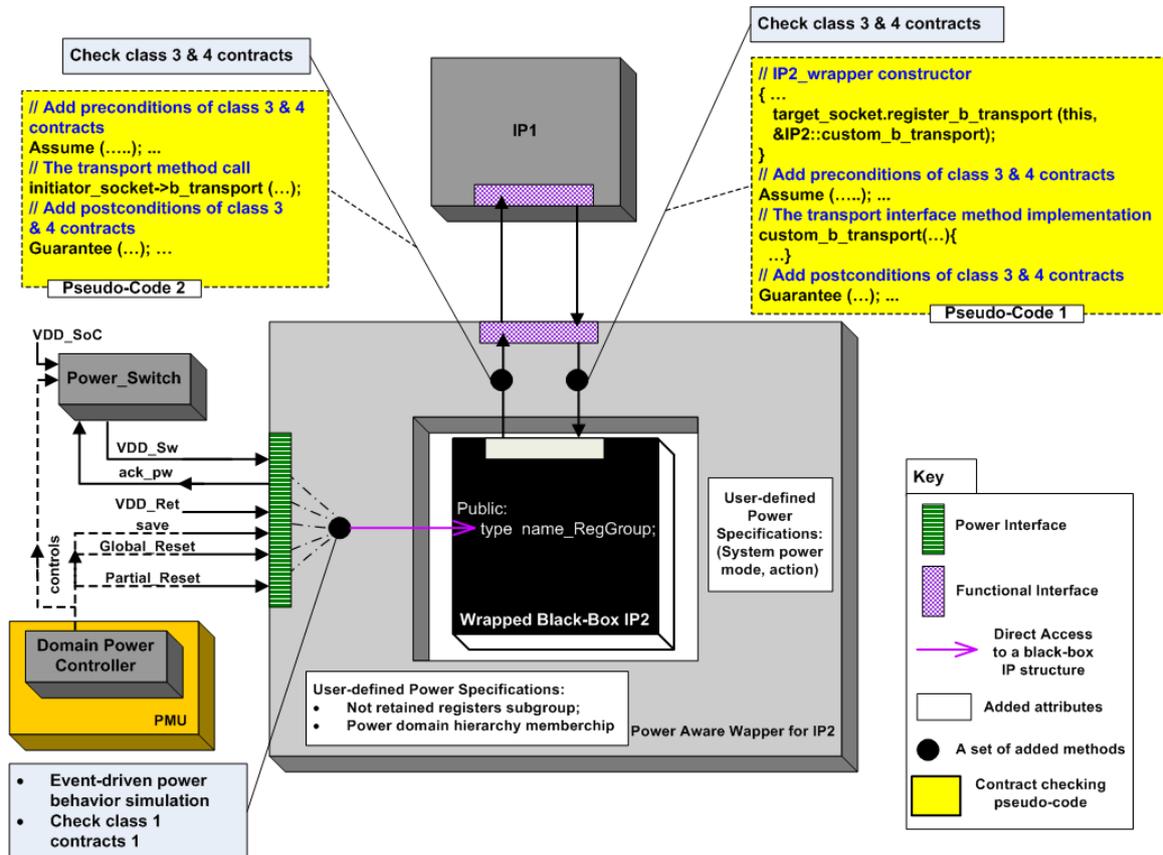


Figure 6.11: Structure and Behavior of a slave/master IP's Power-Aware Wrapper

As depicts Figure 6.11, simulation of this behavior is done by only resetting the non-retained registers once a partial reset signal is received. Remaining registers that must be retained are not touched. For that, definition of the not-retained registers and their characteristics such as their default value, offset and bit fields access inside the wrapper code is required. As we merely suppose having direct access to memory-mapped registers of the black-box IP, each of these defined registers points to its corresponding register in the wrapped IP. In this way, they are effectively changed to their reset value inside the encapsulated IP code on power-down.

A power-aware wrapper also ensures event-driven power-aware behavior simulation and estimation. For that, we have added methods into the wrapper's code as depicts Figure 6.11. To support the **Requirement #2** of the USLPAM methodology, these methods are called when changes occur on the power interface. Each method is in charge of handling input signals (e.g. VDD_Sw) as well as power-down and power-on sequencing by notifying specific events to which the underlying wrapper listens. For instance, asserting VDD_Sw in Figure 6.11 would notify a pw_off_req event to initiate power-down. On power-down completion, a pw_update event is notified so that an update of power values is performed.

Note that such a power-aware wrapper allows modeling the essential UPF power intent concepts in support of the **Requirements #2** and **#3** of the USLPAM methodology: whereas, power switches are modeled as separate modules, supply nets are modeled as signals. Information on power domain partitions and hierarchy can be deduced from connection of voltage signals.

According to this proposed black-box approach, the PMU module is implemented as a new unwrapped block according to the general guidelines given in section 4.1.4 of the Chapter 4 and while supporting the **Requirements #4**, **#5** and **#6** of the USLPAM methodology. Recall also that the functional interface (IF1 in Figure 4.7) still represents TLM ports in which power control transactions (PCTr) are transmitted to the PM submodule. The internal interface (IF2 in Figure 4.7) still consists in a pair of request and acknowledge signals between the PM and each DPC. Conversely to the standard implementation method of IF1 and IF2, interface IF3 implementation is different. Indeed, it is the signal-based power interface of the power-aware wrapper which enables interactions between a domain controller and related power components.

- **Power-Aware Contract Checking**

Our proposed power-aware wrapper plays the role of a "checking" wrapper. Checking concerns only the four-class power-aware contracts. Given the constraints on power-aware contracts checking explained earlier, we have duplicated the functional interface of a black-box IP within the wrapper. The goal is to capture the beginning and the end of some IP operations and surround them with assume and guarantee assertions. Recall that assume assertions are used to check the precondition part of a contract, whereas guarantee assertions are used to verify the post-condition part of a contract.

As depicts Figure 6.11, a power-aware wrapper provides a functional interface which is similar to the one used by the black-box IP. Semantically, they differ on how they behave when either a precondition or a postcondition of an invoked operation is violated. A two-way checking wrapper has been modeled: it reports both its client and wrapped IP interface violations. Clients represent IP blocks communicating with the black-box IP through invoking its public operations.

According to Transaction-Level of Modeling Key Concepts stated in Chapter 2 Section 2.1.3, the wrapper functional interface mainly consists of TLM ports and interrupts which allow the wrapped IP blocks to communicate with other blocks of the platform [124]. So, before conveying relevant transactions to their destination, the wrapper is designed to intercept them at its functional interface and check appropriate power properties. For that, it implements contracted interface method calls inside the wrapper.

Recall that, in the TLM context, communication can only be established through calls to the TLM transport interface methods (`b_transport()` or `nb_(fw/bw)_transport()` methods) (Chapter 2 Section 2.1.3) [124]. Clients may hence represent slaves for the layered black-box IP. In this case, contract-checking code must be placed around the call to transport interface methods inside the wrapper as illustrated by the pseudo-code 2 in Figure 6.11. However, clients may also represent masters for the layered IP. In this case, power contract-checking code must be placed around the transport interface methods implementation inside the wrapper as illustrated by the pseudo-code 1 in Figure 6.11.

It is worth mentioning that only class 3 and 4 contracts are checked at this level. For instance, when IP1 communicates with IP2 through a transport interface method call, the IP2 power domain must already be powered-on. This is an example of a class 3 precondition that must be checked using an assume assertion at the wrapper functional interface, before entering the transport method implementation in IP2. When the transaction re-

sponse is returned, the wrapper captures it and checks the validity of the register data transported to IP1. As this data will naturally be used by IP1, IP2 must guarantee that the read register has not been reset during the last power-down.

In support of the **Requirement #7** of the USLPAM methodology, class 2 contracts are still fully implemented inside the PMU module. Class 1 contracts are implemented inside power switches. Another part of them is implemented inside power wrappers and is checked on entry to or on exit from the power interface. Class 3 and 4 contracts are implemented inside the wrapper on entry to and exit from the functional interface methods.

6.2.1.3 The PAL Utility For Reuse and Modularity

```
class Pw_Prefs
{
public:
    enum Extra_Category {
        AssumeCondition = 0,
        GuaranteeCondition,
        EntryStaisfy,
        ExitSatisfy,
        Create_Pw_Wrappers,
        Install_Pw_Wrapper_Support,
        Retention,
        FullRetention
    };

    Pw_Prefs(char* class_name);
    bool is_enabled (Extra_Category c);
    void set_pref(Extra_Category c, bool new_val);
    void attach_assertion( Extra_Category c, char* method);
    void handle(Extra_Category c, char* method);

private:
    std::map<Extra_Category,bool> prefs;
    typedef std::vector<char*> checked_methods;
    std::map<Extra_Category,checked_methods> contract_prefs;
};
```

Figure 6.12: The Pw_Prefs Class of the PAL Library

The proposed wrapper-based approach can be applied whatever the IP functional behavior as it clearly separates functional and power concerns of each IP. The modularity of this approach is ensured through the use of the PAL utility included in the USLPAL library. This utility represents a set of C++ classes used as base classes that model a

generic structure and behavior of a power-aware wrapper. Defining an IP power-aware wrapper would hence be through extending these classes or redefining some of their methods. As a consequence, this utility makes the refinement and reuse of an IP power-aware wrapper to explore different power intent alternatives for a given virtual platform simpler.

When created, a power-aware wrapper points to the IP to wrap and a list of enabled preferences, denoted Pw_Prefs, is defined. This list indicates the basic options enabled inside a power-aware wrapper. Figure 6.12 depicts the Pw_Prefs class included in the PAL utility. As it can be seen in this Figure, examples of these options are the enabling of preconditions (AssumeCondition), postconditions (GuarateeCondition) and invariants (EntrySatisfy, ExitSatisfy) checking, the creation of power-aware wrappers (Install_Pw_Wrapper_Support and Create_Pw_Wrappers) and the use of partial or full retention strategies (Retention and FullRetention). The Pw_Prefs mechanism allows selective enabling of the wrapper's capabilities without editing its source code in support of the USLPAM's **Requirement #8**. To add wrapper support at link time, a power-aware wrapper of an IP should extend the Wrapper_Factory_Support class of the PAL utility whose code is shown in Figure 6.14 and override its add_wrapper() method. As it can be seen, each power-aware wrapper is created using the factory pattern [79] (see the Wrapper_Factory class of the PAL utility in Figure 6.13) which checks the Create_Pw_Wrappers option in Pw_Prefs to construct or not the wrapper object. This

```

class Wrapper_Factory{
public:
    static sc_core::sc_object* create();
protected:
    typedef sc_core::sc_object* (*Wrap_Fn) (sc_core::sc_object*);
    static Wrap_Fn wrap_fn;
};

sc_core::sc_object* Wrapper_Factory::create()
{
    sc_core::sc_object* result = new sc_core::sc_object();
    if (wrap_fn)
        result=wrap_fn (result);
    return result;
}
Wrapper_Factory::Wrap_Fn Wrapper_Factory::wrap_fn=0;

```

Figure 6.13: The Wrapper_Factory Class of the PAL Library

```
class Wrapper_Factory_Support : public Wrapper_Factory
{
public:
    Wrapper_Factory()
    {
        //When instantiated, install the wrapper creation function
        if(prefs.enabled(Extra_Pw_Prefs::Install_Pw_Wrapper_Support))
            wrap_fn=&add_wrapper;
    }
private:
    static Events_Notification* add_wrapper(sc_core::sc_object* kernel)
    {
        if(kernel && prefs.enabled(Pw_Prefs::Create_Pw_Wrappers))
            return new Wrapper (kernel, prefs);
        else
            return kernel;
    }
    static Pw_Prefs prefs;
};
```

Figure 6.14: The Wrapper_Factory_Support Class of the PAL Library

mechanism allows fulfilling in particular the USLPAM's **Requirement #1**.

6.2.2 Application on Case-Studies

6.2.2.1 Application on an Audio System Virtual Prototype

Experimental results of the black-box approach application on an audio system virtual prototype presented in this section have been published in [115].

In this section, we demonstrate how the USLPAM methodology can be easily integrated into an existing virtual prototyping tool. As the Synopsys's Innovator tool offers black-box types of virtual prototypes, we have used it to also validate our wrapper-based approach.

- **A Transaction-Level Virtual Platform for Audio Codec System**

An existing software virtual prototype in the Synopsys DesignWare System-Level Library (DWSLL), named "Timed_926" has been chosen as a starting point for building an audio application. As depicts Figure 6.15, the "Timed_926" platform is an approximately-timed (AT) [124] TL platform based on an Instruction-Set Simulator (ISS) for the ARM926EJ-S processor and incorporating black-box TL IP models from the

DWSLL. A detailed description of memory-mapped registers, as well as interfaces of each block is given. Each block can be configured through editing the ARM embedded software.

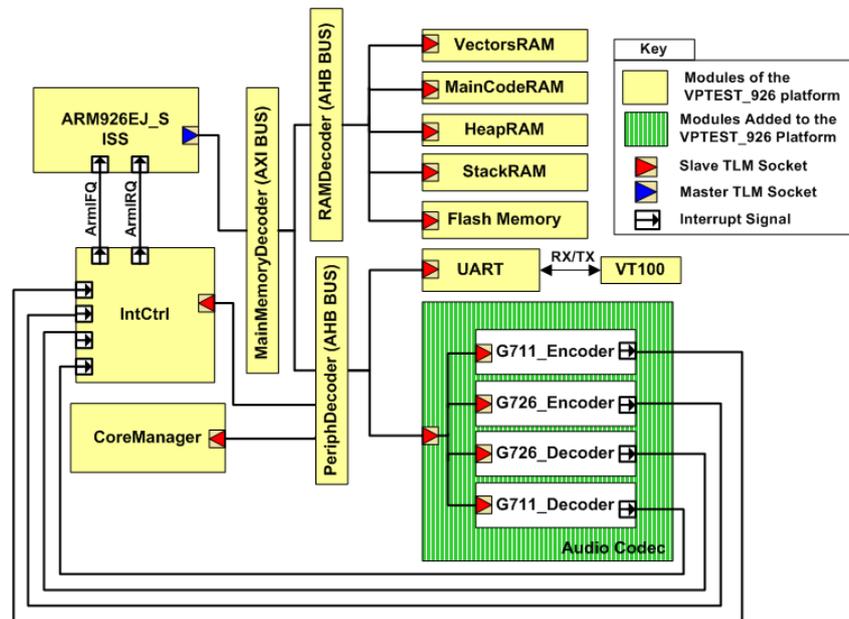


Figure 6.15: The Audio Virtual Platform Block Diagram

The audio virtual platform has been built on top of the "Timed_926". It models a voice messaging system which mimics for instance a phone answering machine. As illustrates Figure 6.15, an audio encoder/decoder hardware accelerator based on the G.711 (Pulse Code Modulation (PCM)) and the G.726 (Adaptive Differential Pulse Code Modulation (ADPCM)) speech codecs ITU-T standards [10] has been added. This accelerator is composed of four TLM sub-modules. On the one side, the G.711 encoder module creates a 64 kbit/s bitstream from an analog signal sampled at 8 khz. The G.711 decoder does the opposite. On the other side, the G.726 encoder encodes into 5, 4, 3 or 2 bits per sample the 64kbit/s bitsream. The G.726 decoder implements the reverse procedure. These modules, created with the Innovator's Component Creator tool, are included in the Synopsys DesignWare System Level Library (DWSLL) for easy reuse. The ARM embedded application has been enriched with different application scenarios: record a voice message, play a recorded message or an incoming one.

- **Power-Aware Wrappers Development**

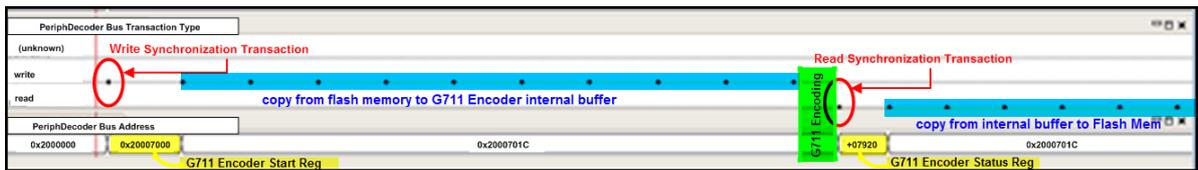


Figure 6.16: Excerpt of the Transaction Flow During the Record Scenario Using Platform Analyzer Tool

Using the Synopsys’s Platform Analyzer tool, activity profiles of each hardware component for each application scenario can be captured and analyzed in order to determine power intent alternatives. For instance, Figure 6.16 depicts the transaction flow observed on the PeriphDecoder bus at the beginning of the record scenario execution. The main phases of the G.711 encoding can be detected: once writing to the G711 encoder’s start register (the first synchronization transaction in Figure 6.16), the G.711 encoder performs compression on a 10-sample block. These samples have been already copied from the flash memory to the G711 encoder’s internal buffer. As illustrates Figure 6.16, the ten write transactions to the G.711 encoder over the PeriphDecoder bus represent this copying phase. Afterwards, the read transaction to the G.711 encoder’s status register (the second synchronization transaction in Figure 6.16) indicates the end of the G711 compression. The encoded samples are then transferred back to the flash memory. As shows Figure 6.16, ten read transactions follow the read synchronization one. The same flow is repeated until the end of linear samples. Then, the G.726 encoder uses a similar flow to encode samples in the flash memory.

Given this software flow analysis, a power intent alternative where the memory is powered-down during each G.711 and G.726 encoding and decoding is possible. For example, this is the case for alternative (a) in Figure 6.18 and Table 6.2. As indicated in the PST, the system is put in the transfer_record power mode when a 10-sample block is transferred between the flash memory and the internal buffer of the G.711 or G.726 encoders. Before encoding, the system is rather put in the record system mode. As indicates Table 6.2, this mode corresponds to a switched-off flash memory’s power domain.

Using the Innovator toolset, a power intent alternative is elaborated by (1) placing the power switch modules, (2) creating and (3) parameterizing IP power-aware wrappers

6.2 Power-Aware Wrappers For The USLPAM Application: A Black-Box Based Approach

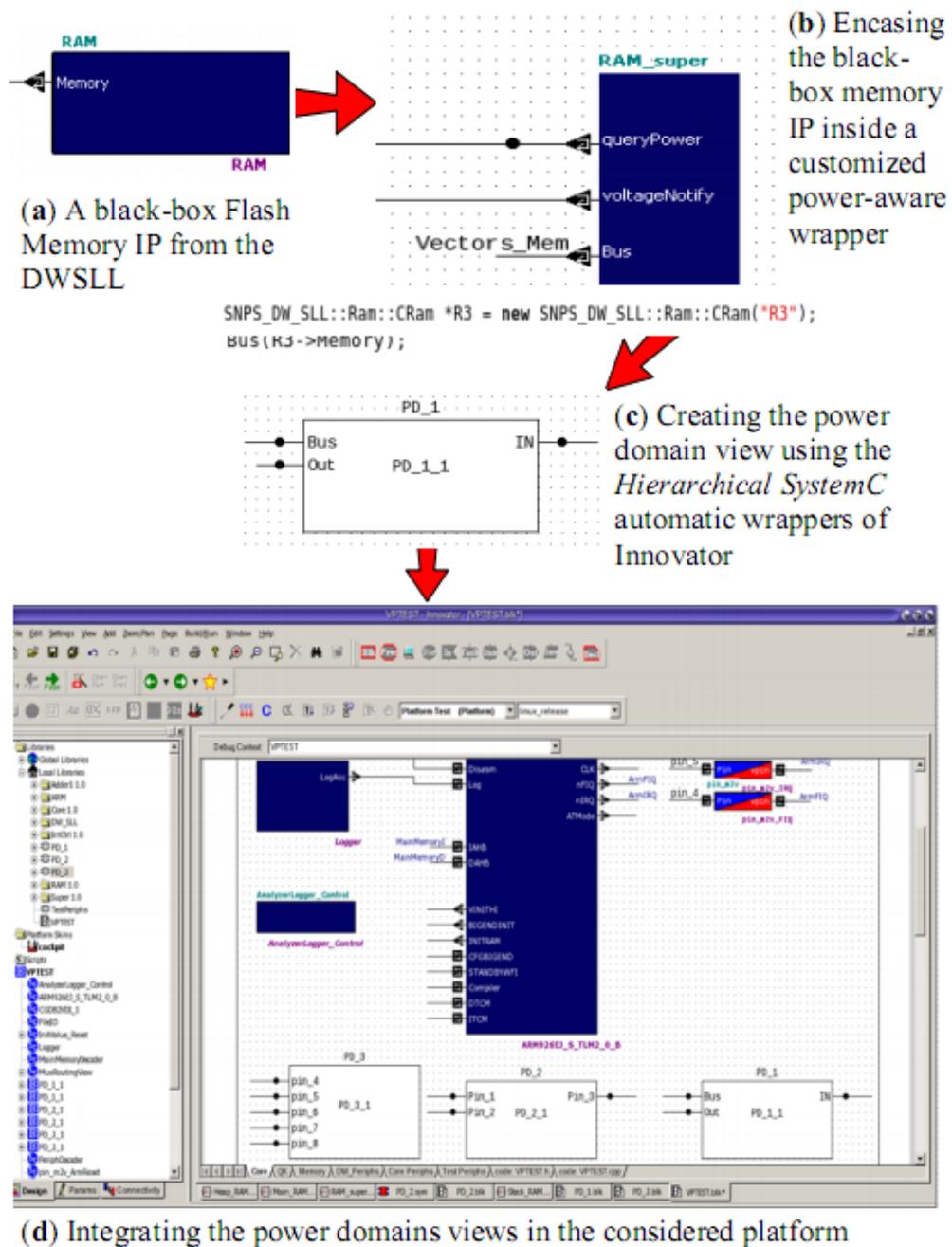


Figure 6.17: Developing Power Wrappers Using the Innovator Tool

using the PAL utility (4), implementing the power state table (PST) header file, (5) implementing the power management unit by adding to it (6) the required domain power controllers, (7) enriching the embedded application with power control transactions ac-

	VDD_CPU	VDD_SoC	VDDRAMSw	VDDEncSw	VDDDecSw
all_off	OFF	OFF	OFF	OFF	OFF
init	ON_H	ON	ON	OFF	OFF
config_ADPCM	ON_H	ON	OFF	ON	ON
record	ON_L	ON	OFF	ON	OFF
play	ON_L	ON	OFF	OFF	ON
transfer_record	ON_H	ON	ON	ON	OFF
transfer_play	ON_H	ON	ON	OFF	ON
idle	ON_L	ON	OFF	OFF	OFF

Table 6.2: Power State Table for Alternative (a)

According to the defined PST and finally, (8) creating the power domains view using the Hierarchical SystemC Innovator wrapping capability.

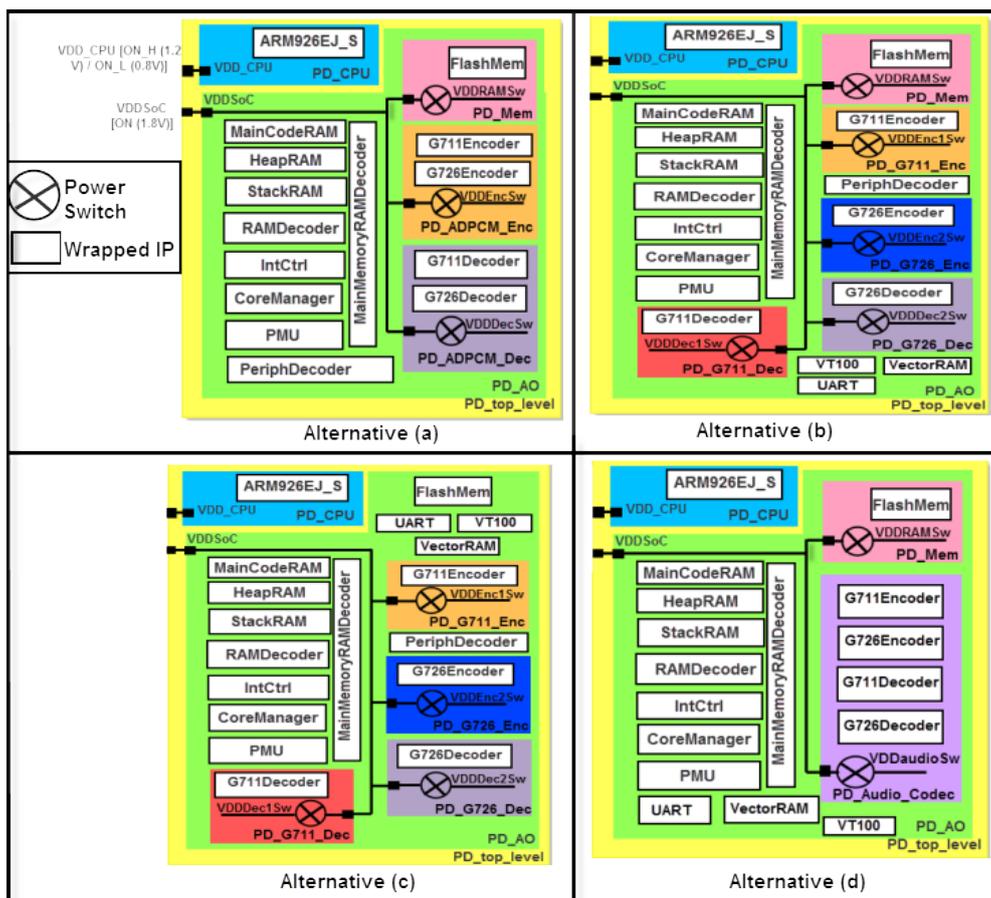


Figure 6.18: The Considered Power-Aware Architecture Alternatives

Power Intent (PI)	Number of Power Switch State Transitions					Time Switching Penalty = 200 ns	
	Flash Mem	G.711. Enc	G.726 Enc	G.711 Dec	G.726 Dec	Energy Value (J)	Energy Savings (%)
PI (a)	26373	6		6		7,23	20,81
PI (b)	26373	3	3	3	3	6,32	30,8
PI (c)	26373	2				9,03	1,1
PI (d)	0	3	3	3	3	4,3	53
Without Power Intent	Energy = 9,13 (J)						

Table 6.3: Energy Savings for the Different Power Intent Alternatives According to the Play & Record Software Scenario

For each IP in the virtual platform, its power-aware wrapped version (consisting in the IP itself encased in its power-aware wrapper) is created only once using the Component Creator tool and instantiated henceforth from the DWSLL whenever needed. The last step (step (8)) serves only to group wrapped IPs belonging to the same power domain in order to better structure the low power design. For example, Figure 6.17 shows steps required to build the flash memory power domain (PD_1) starting by adding a power-aware wrapper to the DWSLL IP and ending with integrating PD_1 into the initial virtual platform. As a consequence, evaluating a new power intent alternative requires redoing only steps (3), (4), (5), (7) and (8). It is also worth mentioning that a power switch IP, as well as a generic domain power controller IP are created only once using the Component Creator tool. They are afterwards instantiated from the DWSLL whenever needed.

- **Experimental Results**

Figure 6.18 depicts the different tested power intent alternatives for the audio system VP. Only the power state table of alternative (a) is given in Table 6.2. In alternative (d), the four audio sub-modules belong to the same power-gated domain. In alternative (a), similarly to decoder sub-modules, encoder sub-modules are gathered in a single power-gated domain. In alternative (b) and (c), each audio sub-module belongs to a separate power-gated domain. Unlike the other alternatives, the flash memory IP-block in alternative (c) belongs to an always-on power domain.

Table 6.3 shows results obtained for a Play & Record scenario. Note that alternative (d) is the most energy-efficient one with 53% of energy savings compared to the non-partitioned initial platform. Note also that power intent alternative (d) achieves up to 32% of energy savings compared to alternative (b). This is due to the considerable power penalties caused by the frequent transitions of the Flash IP in alternative (b) from power-off to power-on (up to 26373 transitions on Table 6.3). It is also worth mentioning that using power-aware wrappers adds a negligible amount of simulation run-time overhead. For instance, simulation speed for alternative (d) is only 0.02% less than the initial behavioral platform. Moreover, in order to evaluate a new power intent alternative, redesigning and rebuilding required power-aware wrappers and power management blocks is only a matter of hours.

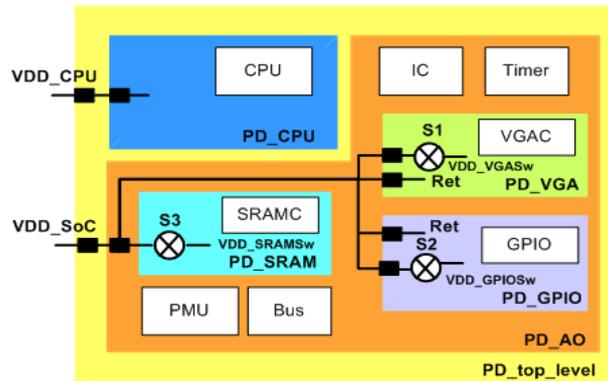
6.2.2.2 Black-Box Versus White-Box Comparison Results

An article on the comparison of our proposed white-box and black-box approaches has been published in [112].

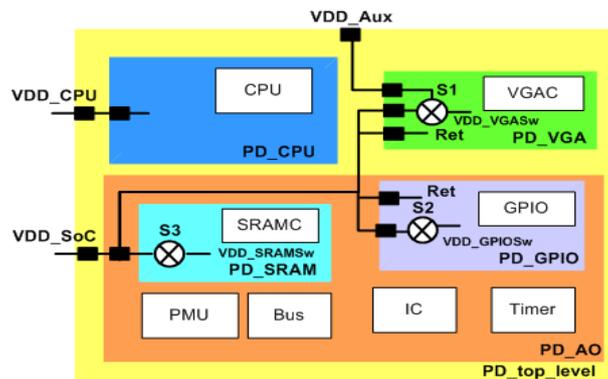
In order to compare the black-box approach with the white-box approach presented earlier, we have applied the power-aware wrapper approach to the black-box and white-box versions of the same AT platform of Figure 6.6(a) and compare performance results obtained in both cases.

Table 6.4 lists the considered comparison parameters. Each parameter has been measured in the black-box platform and compared to its value obtained within the white-box platform. Power domain partitions depicted by Figure 6.19 have been considered. Comparison results are given in Table 6.4 as increase or decrease percentages. Note that possible additional energy savings can be obtained when using the white-box platform rather than the black-box one. For instance, 48% of energy savings is observed using the black-box version. However, an increase by up to 10% in energy savings is noted when using the white-box version. Indeed, this is due to defining an additional power candidate in the white-box case. This power candidate ($PwC_{candidate}$) consists in providing a low voltage (VDD_Aux in Figure 6.19(b)) rather than a high voltage (VDD_SoC in Figure 6.19(b)) to supply the VGA controller's domain. This power mode transition is performed just before the VGA block begins drawing an image on the screen and can only be added

in case of open code access. Otherwise, the operation requiring this transition cannot be captured at the wrapper level. It is noteworthy that this added transition induces in the white-box case an increase in the number of power mode transitions and in the activity percentage of the power management block as shown in Table 6.4.



(a) Black-Box Platform



(b) White-Box Platform

Figure 6.19: A Power-Aware Architecture Alternative

As a consequence, the SystemC global simulation time required to display an image slightly increases in the white-box case compared to the black-box one due to the introduced transition time penalty. One can then deduce that a more accurate power intent specification and TL simulation leading to higher energy savings can be achieved in the white-box case than the black-box one.

On the other hand, Table 6.4 indicates that running the power-managed black-box platform takes more time than running the power-managed white-box version. Here, the running time metric in Table 6.4 means the execution speed during the display scenario

simulation. Naturally, this difference is due to the additional overhead imposed by the wrappers use. Indeed, the white-box implementation of the general methodology mainly aims at rapidly simulating power gating and multi-voltage architecture alternatives. First, this rapidity is due to the use of method calls to implement the IF3 power management interface. However, a power interface composed of different power gating control signals is instead added in each wrapper. Second, rapidity induced by the white-box approach is justified by the introduction of the PwARCH design element concept. This concept eases mirroring the domain-based power behavior on corresponding functional white-box blocks. However, in the black-box case, various alternative mechanisms are implemented inside each wrapper. In particular, redefining implementation methods of the functional TL interface inside each block’s wrapper is a largest contributor to this running time overhead.

Parameter		Comparison Results
Energy Savings		- 10%
SystemC Simulation Time		- 5%
PMU Activity		- 25%
Power Mode Transitions		- 35%
Running Time		+ 30%
Number of	Class 1 Checks	-28%
	Class 2 Checks	- 6%
	Class 3 Checks	- 8%
	Class 4 Checks	-10%

+ Increase - decrease

Table 6.4: Comparing the Black-Box Platform Performances With Those of the White-Box Platform

From checking results on Table 6.4, one can also observe that not all checks performed in the white-box platform case can be done in the black-box one. For instance, before entering the draw image phase, the VGA controller’s power domain is checked to be in a low voltage power mode and the VGA internal buffer storing the image is checked to be not empty. These checks are hard-coded into the VGA white-box block, but cannot be inserted in a power-aware wrapper. This is due to the absence of events at the VGA functional interface that help capturing, at the wrapper level, the beginning of the drawing operation. The added power candidate in the white-box platform implies not only adding

such class 3 contracts, but also adding some class 2 and class 4 contracts which are absent in the black-box case as illustrates Table 6.4.

In spite of the lack of flexibility and precision, the black-box approach remains more general than the white-box approach since it can be applied to both cases of platforms, even to a hybrid platform with mixed white-box and black-box IPs.

6.2.3 Concluding Remarks

We have presented a wrapper-based approach as a solution to apply the USLPAM methodology on black-box IPs of a virtual platform based on the use of the PAL utility. Modularity and reuse of this approach can be achieved using our guidelines for modeling structure and behavior of a Transaction-Level power-aware wrapper. By using the Synopsys's Innovator virtual prototyping toolset, we have proved that the simulation-based USLPAM flow can be easily and efficiently integrated into existing industrial ESL design flows based on virtual prototyping technology while meeting all the methodology requirements. The efficiency of the wrapper-based approach in terms of enabling fast exploration of different power intent alternatives with reduced modeling effort has also been proved.

We have also conducted a simulation-based proof of concept of the differences between white-box and black-box approaches. Although each approach follows the same power-aware USLPAM flow and meets its essential requirements, we have demonstrated the flexibility and moderate simulation speed cost of the white-box approach against the genericity of the black-box one. These differences are mainly due to the fact that, although inspired from the IEEE 1801 (UPF) standard semantics, managing retention of a black-box IP internal registers is almost impossible from outside the IP. This limitation restricts power intent alternatives potentially specified for a black-box IP. To address this issue, IP-XACT [7] standard could be extended to best provide constraints on an IP block power intent. Hence, automatic exploration of power intent alternatives and their IP-XACT based integration into TL virtual prototyping tools could be investigated in the future.

6.3 The USLPAL Base Utilities for the USLPACom

6.3.1 Motivations

Modeling techniques that we have proposed so far in this chapter use the UPF standard as support to build a structured high-level specification of multi-power domain architecture at TLM and to infer power-aware simulation features into functional designs. We have shown how to integrate such a specification into a Transaction-Level behavioral SoC model and evaluate different power architecture alternatives by only considering the scenario-based power domain management strategy. As explained in Section 4.1.4, this strategy is based on the use of the UPF-defined PST concept and in which power domain states control is done in a single direction, from the PMU components to power domains.

Nevertheless, an energy-efficient power domain management solution is defined by two fundamental and strongly correlated elements: an energy-efficient low power architecture composed of multiple power domains and an energy-efficient power management strategy for power domains states control. Thereby, an exploration of not only power intent alternatives but also of power domain management strategies is needed so as to decide about the energy-efficient overall power domain management solution.

In Section 4.1.4 of the Chapter 4, we have mentioned a set of power domain management strategies ranging from static ones, such as that used by the low power design standards, to dynamic ones, such as the scenario tracking strategy in which some components communicate to the PMU enough information about the system functional and power state. All these kinds of power domain management depict a strong design dependency between a power-domain-based architecture and the power management unit operation. However, while some strategies perform only unidirectional power control communications (mostly from the PMU to the system power domains), others implement bidirectional power control communications either between the PMU and power domains or even between the different power domains in a system.

In order to remove this dependency and enable a fast exploration of complete power management solutions implementing unidirectional or bidirectional power communications, a generic and common power domain management interface is required. Such an interface has to describe the protocol and data required for inter-power-domain communication while supporting a plug-and-play approach for power domains and PMU. By

referring to Figure 3.6(b) of the Chapter 3, such a common power domain management interface would represent the external power interface of a power-aware component that separates power and functional communications in order to stick to the power domain reasoning and separation of concerns methodology used by the low power design standards. Note that such an interface with such capabilities promotes the design of a power domain as a power-aware component that assembles the behavioral TLM modules belonging to a power domain as illustrated by Figure 3.6(b). Benefits and rationale of this structuring approach have been detailed in Section 3.1.1.2 of the Chapter 3.

Actually, plug-and-play approaches are very useful for fast exploratory studies performed during early stages of a SoC design flow, in particular at Transaction-Level of Modeling (TLM). Unfortunately, no TLM semantics for creating such a communication interface have yet been defined. Moreover, low power standards such as UPF and CPF define only semantics for a power-domain-based architecture. Although they define some semantics involved in the power domains states control (such as the PST concept and control semantics of the power switch concept) and useful for a PMU power management policy implementation, these low power design standards leave the definition of the PMU block structure and the power management strategy to control such an architecture to the designer.

In the Section 2.1.5.3 of the Chapter 2, we have listed the different power management levels and we have discussed relevant state-of-the-art power controller oriented and operating system oriented power management interfaces. None of these existing interfaces fits completely our common power domain management modeling requirements. Nevertheless, some of their relevant features can be adopted and even adapted to match our interface modeling purpose.

Considering this panorama of modeling needs and issues, we propose in the following a new PDMgIF interface dedicated for the control of power domains states. This interface allows the transfer of controls and events between power domains using well-defined concepts and according to specific protocol rules. By using the TLM-2.0 OSCI standard mechanisms to create protocol-specific TLM-2.0 interfaces, we present a TLM 2.0 model of the proposed PDMgIF bus protocol interface that separates functional and power management communications promoting hence a plug-and-play approach for power domains and PMU. The basic and generic features of this TLM 2.0 interface model represent the

USLPACom utilities part of the USLPAL library. We show how such an interface model can be efficiently added to a functional Transaction-Level model and used to construct a complete power-domain managed Transaction-Level model. We also demonstrate the PDMgIF flexibility and reuse with any power-domain-based power architecture and management strategy.

Recall that a quick read of the Sections 2.1.5.3 and 2.1.3 will help the reader in understanding the approach presented in the following.

We have published the concepts, the TL modeling approach and the performance evaluation results of the PDMgIF protocol interface in [114].

6.3.2 Power Domain Based Modeling Approach

In this section, our proposed power-managed system structure is presented and the main design requirements to be fulfilled by the PDMgIF protocol interface are extracted.

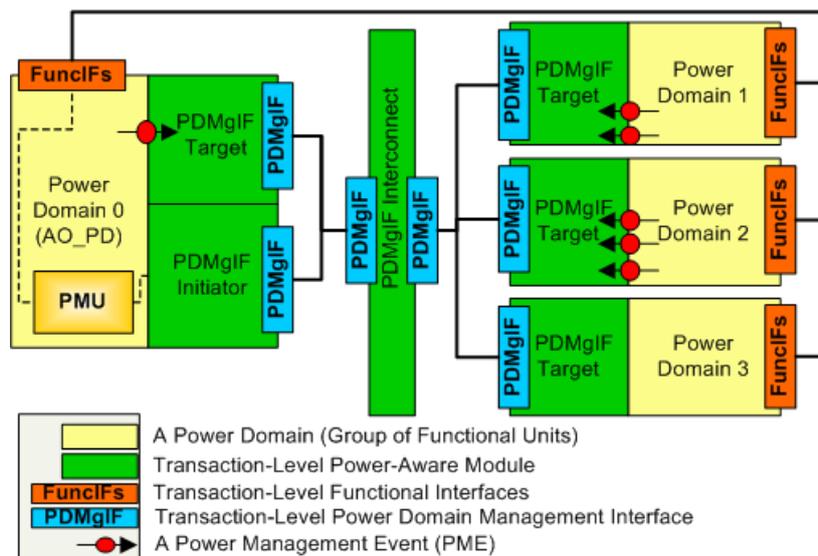


Figure 6.20: Layering the Power Domain Management TL Structure on Top of the Functional TL Model

6.3.2.1 Power Domains Layers

Considering a functional TL system model, we aim at constructing a power-managed system enabling power domains states control. Functional modules belonging to the same power domain share the state and the power control interface of their power domain. Therefore, we propose to layer a power domain management structure on top of the functional one. The generic view and components structure of this additional layer is depicted by Figure 6.20 and is explained in the following.

Each power domain part wraps the functional modules belonging to a same power domain. This part involves as well the power architecture specification and the different mechanisms required for the power-aware behavior simulation of the underlying power domain. Specific power domain management interfaces (PDMgIF) are required at the boundary of each power domain in order to ensure inter-power domain communications through a dedicated PDMgIF interconnect. As shown in Figure 6.20, PDMgIF target and initiator modules are also required in each power domain layer in order to manage state transitions and ordering of each received transaction at the PDMgIF interface.

6.3.2.2 Sourced Power-Aware Communications

Modeling a generic PDMgIF requires considering bidirectional communications useful for a power domain management decision. These communications may occur between power domains and the PMU on the one hand, and between the different power domains on the other hand. Thus, two types of transactions are considered in our modeling approach. First, an always-on (i.e. can never be switched off) power domain, denoted AO_PD, can transfer **power control transactions** through the PDMgIF interconnect to other power domains or to specific design elements in order to change their power states. A design element represents at least one functional module. Power control transactions are only issued by the AO_PD and may be pipelined depending on the considered power management strategy. The concept of power control transactions has been actually adopted from the control commands that can be transmitted over the MIPI's SPMI [13] power management bus and adapted to the power domain context needs.

The second type consists in **power management events (PME)**. They are defined as transactions transferred over the PDMgIF interconnect from a power domain to the PMU

module as illustrates Figure 6.20. A PME transaction is used either to inform the PMU about a design element functional state, or to request a specific power domain state. Thus, this kind of transactions is used to handle dependencies between the functional design and the power-aware one.

The concept of a PME that simply informs the PMU about a device state has been already used in the PCIe and PCI bus specifications [22] [23] as well as in the DPI interface [133] [134]. According to our power domain management modeling requirements, we have adapted this concept and added semantics to it. In particular, we have assigned to a PME transaction a high or low priority. Moreover, such a transaction can carry out an event among these three types: a *power PME* indicates a request to change an active power domain state to another active one. A *sleep PME represents a request to switch-off a power domain. It may occur for example upon a module task completion.* Finally, a *wakeup PME* represents a request to switch-on a power domain.

6.3.2.3 Identifier-Based Addressing and PDMgIF Compliant Components Classification

In order to address power domains on the PDMgIF interconnect, identifier numbers are used to identify power domains and design elements. Actually, this addressing method is similar to that of the MIPI's SPMI standard [13] but adapted to a power domain context use. An **Initiator Identifier (IID)** is given to the PMU unit. A **Target Identifier (TID)** is given to each design element (i.e. set of functional modules) in a power domain. Each power domain is given a unique **Power Domain Identifier (PDID)**. As a consequence, a design element of a power domain is identified by a (TID, PDID) pair. Two design elements of different power domains may have the same TID identifier.

The PDMgIF interface supports all power domains as PDMgIF targets, and only the AO_PD power domain as a PDMgIF initiator. PDMgIF targets that can arbitrate for the PDMgIF interconnect to initiate PME transactions are called **Request Capable Targets (RCT)**. Remaining targets are called **Non-Request Capable Targets (NRCT)**. Actually, the RCT and NRCT concepts have been inspired by respectively the Request Capable Slave (RCS) and the Non-Request Capable Slave (NRCS) concepts of the MIPI's SPMI standard [13] introduced in the Section 2.1.5.3 of the Chapter 2.

6.3.2.4 PDMgIF Initiator Requirements

Figure 6.21 details the structure of the AO_PD (power domain 0) layer of Figure 6.20. This power domain corresponds to the always-on SoC power domain and represents the PDMgIF initiator. As explained in Section 4.1.4 of the Chapter 4, our proposed PMU simulation model includes a Power Manager (PM) sub-module which coordinates functional blocks activities with their power domains states according to a power management strategy. The PMU module includes as well a Domain Power Controller (DPC) related to each power-gated domain and is responsible for its power-down and power-up by controlling a set of signals in a specific order. An example of this sequence is shown in Figures 4.8 and 6.21 and has been described in detail in Section 4.1.4 of the Chapter 4.

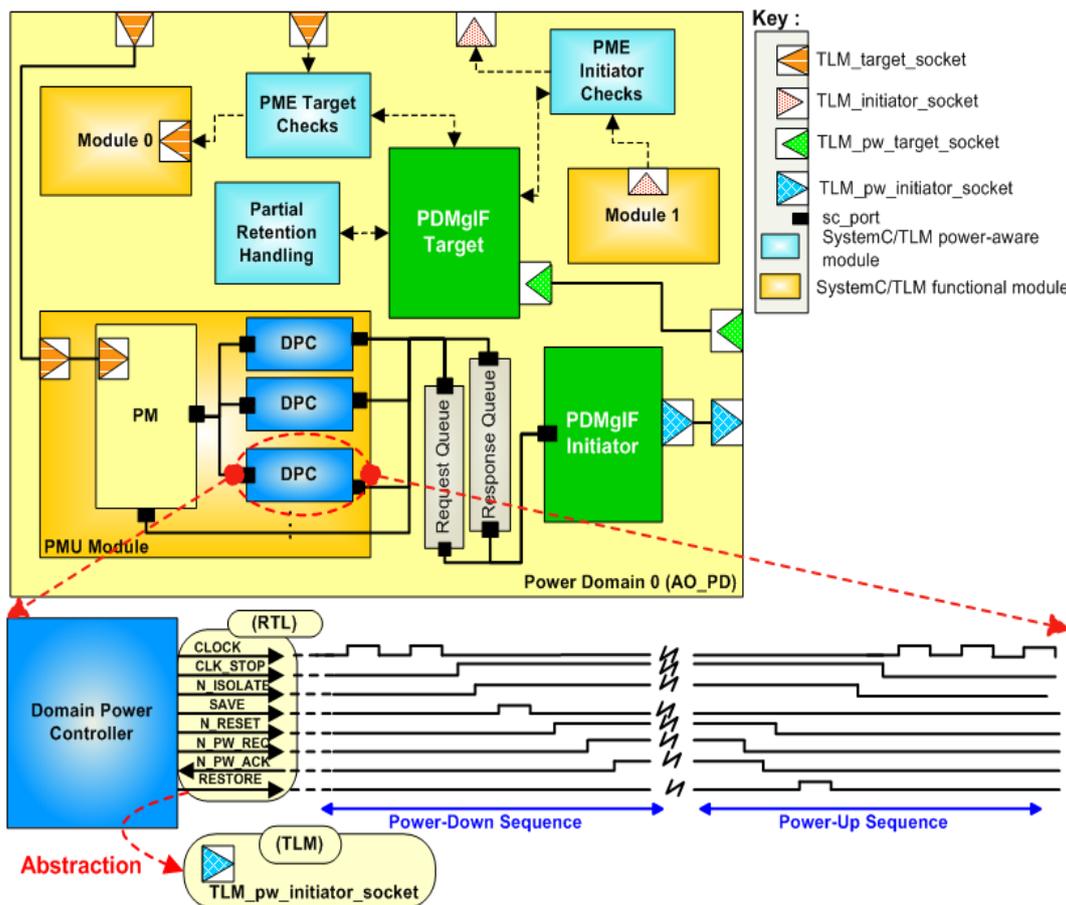


Figure 6.21: A Generic Example Showing the Internal Structure of the AO_PD Power Domain

At TLM, such a sequence of RTL control signals has to be converted to a single TLM function call and RTL signals will be replaced with a single specialized power socket (`tlm_pw_initiator_socket`) as depicts Figure 6.21. The Transaction-Level PMU model will then act as a generator of power control transactions and transmit only abstract data structures. Transactions transmitted through the `tlm_pw_initiator_socket` PDMgIF port are first received by a generic PDMgIF initiator module (i.e. the PDMgIF initiator module in Figure 6.21) that handles their transitions from one phase to another.

6.3.2.5 PDMgIF Target Requirements

Each AO_PD power domain represents at the same time a PDMgIF initiator and a PDMgIF target as shows Figure 6.21. In general, each PDMgIF target wraps a set of functional modules. Power states of these modules' power domain are controlled through power control transactions transmitted by the PMU module over the PDMgIF interconnect. Phase transitions of the received power control transactions at the PDMgIF target interface are handled by a PDMgIF target generic module (i.e. the PDMgIF target module in Figure 6.21). Once a power domain changes state, the PDMgIF target module triggers the Partial Retention Handling block shown in Figure 6.21. This block is in charge of simulating the impact of a power state change on the functional behavior of a power domain. In particular, when a partial retention strategy is applied to a power domain, this block resets the non-retained registers of this power domain's functional modules on power-down.

In case of a Request Capable Target (RCT), the PDMgIF target module is also in charge of collecting power management events (PMEs) from functional modules and transmitting them to the PDMgIF initiator in the form of PME transactions. In general, a PME precedes or succeeds a transaction issued or received at the functional interface module. Therefore, intercepting such relevant functional transactions and translating them to PME transactions are required in the power domain layer. This is the responsibility of the PME target checks and PME initiator checks blocks in Figure 6.21 at respectively the functional target and initiator interfaces.

In the next section, we propose a Transaction-Level model of the PDMgIF protocol interface which supports all these modeling requirements.

6.3.3 PDMgIF: a Transaction-Level Interface Protocol for Power Domain Management

6.3.3.1 Methodology for the PDMgIF Protocol Modeling in TLM 2.0

The methodology presented in this section proposes a structured way for creating the PDMgIF custom interface that enables power-domain-based communication for the PDMgIF protocol using the mechanisms provided by the TLM 2.0 standard [124]. The proposed methodology is composed of two distinct steps: protocol features definition and TLM 2.0 mapping. Each step is split further into one or more tasks, as shown in Figure 6.22.

The first step in the methodology consists in defining the relevant features of the PDMgIF protocol based on the analysis of the design and modeling requirements presented in the previous section. There are five main features to be extracted: protocol attributes, timing points, channels, state transitions and interconnect behavior. Protocol attributes represent the fields of data structure that can be transmitted during a communication between a PDMgIF initiator and a PDMgIF target. Timing points consist in synchronization points between a PDMgIF initiator and a PDMgIF target. We define a channel as a group of attributes and timing points. Defining channels helps designing the set of finite state machines (FSM) that capture the behavior of the PDMgIF protocol. Each FSM defines the state transitions between timing points of a specific channel. According to the PDMgIF protocol behavior, the structure and behavior of the PDMgIF interconnect are specified.

The second step of the methodology is the mapping of the previously defined features into TLM 2.0 structures. Attributes of each defined channel are mapped to its own separate custom generic payload (GP) extension based on the TLM 2.0 extension mechanism (see Chapter 2, Section 2.1.4). As the PDMgIF protocol is of pipelining capabilities, this will enable extensions to be processed and routed separately from other extensions. The different timing points are mapped into a custom phase object. The custom GP and the phase object form the new PDMgIF protocol traits class [124] which is used to parameterize the custom PDMgIF initiator and target TLM 2.0 sockets, as well as the TLM 2.0 transport interfaces (see Chapter 2, Section 2.1.4).

Furthermore, in order to implement the PDMgIF initiator and target generic base modules, each FSM channel is split into a target FSM and an initiator FSM. Then, the

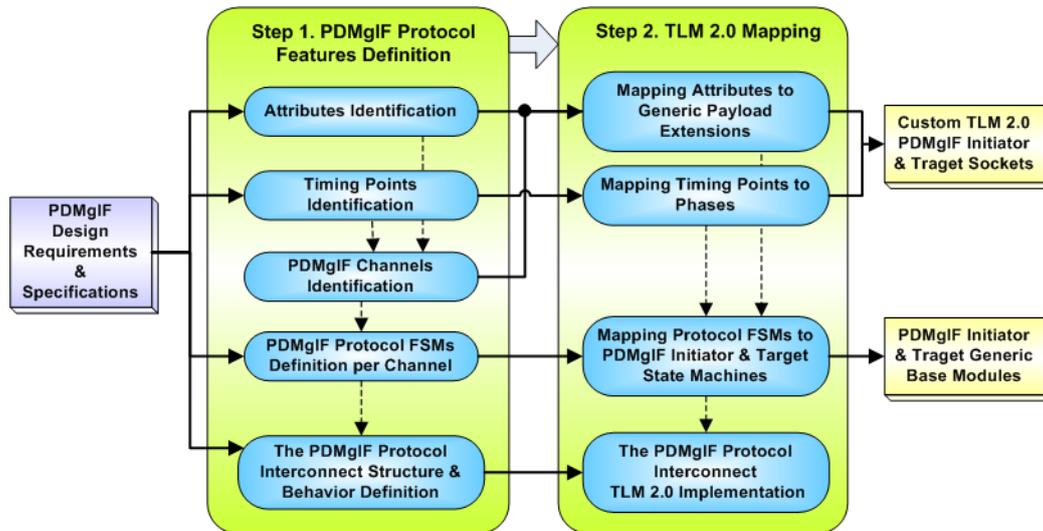


Figure 6.22: Overview of the General Modeling Methodology

PDMgIF initiator base module implements the initiator FSMs of both channels. Similarly, the PDMgIF target base module implements the target FSMs of both channels. Synchronization between the resulting FSMs is of prime importance and can be obtained through an inter-channel dependencies analysis.

6.3.3.2 Issues of Modeling the PDMgIF Interface Protocol in TLM 2.0

Two issues are encountered when modeling the PDMgIF protocol in TLM 2.0 [124]. The first is that the TLM 2.0 generic payload fields are inappropriate to model the data and controls transmitted over the PDMgIF interconnect. Nevertheless, TLM 2.0 has provided the TLM 2.0 extension mechanism to extend the generic payload with additional user-defined fields. So, we have chosen to model the data attributes involved in a power control transaction as a **tlm_pwctrl extension** and to model those involved in Power Management Event (PME) as another **tlm_pme_handling extension**. Although one could put all of the attributes of the two transaction types into a single TLM 2.0 extension, it is rather wise to use two separate generic payload extensions. Indeed, this enables PDMgIF bus pipelined capabilities and extensions can then be processed and routed separately.

The second issue is the modeling of the Request Capable Target (RCT) concept in

TLM 2.0. Indeed, modeling a target that initiates transactions would violate the request/response ordering rules of the TLM 2.0 basic protocol. In order to overcome this modeling constraint, a new protocol different from the TLM 2.0 base protocol has been defined. Fortunately allowed by the TLM 2.0 standard, the own request/response rules of this new protocol are also defined independently of the TLM 2.0 basic protocol rules. This new protocol is characterized by a generic payload extended with the two TLM 2.0 extensions (**tlm_pwctrl extension** and **tlm_pme_handling extensions**) and a new phase object (named **tlm_PDMgIF_phase**) gathering all the possible timing points of the two transaction types. The specialized TLM target socket at a PDMgIF target domain interface (named **tlm_pw_target_socket** in Figure 6.21) as well as the specialized TLM initiator socket at a PDMgIF initiator domain interface (named **tlm_pw_initiator_socket** in Figure 6.21) have been customized to this new protocol.

Let us consider a low power architecture of a SoC platform with n power domains and a maximum of p design elements in each power domain such as n and p are two generic

		Description	
tlm_pwctrl Channel	Attributes	CMD	Command Field Enumeration: RESET, SHUTDOWN, WKUP, SLEEP_RETAIN
		RESPONSE	Response status enumeration indicating the success or failure of the transaction
		PDID	7-bit Power Domain Identifier
		TYPE	Type of the power control transaction (FULL or PARTIAL)
		TID_MASK	32-bit mask to indicate a specific set of design elements in a power domain (only valid if TYPE is PARTIAL)
		PW_MODE	Used with the WKUP command and only in case of multi-voltage scaling technique to indicate the required wakeup voltage for a power domain
	Timing Points	TRANS_ID	Transaction identifier (needed for handling the transactions state transitions during simulation)
		BEGIN_PW_REQ	Initiator has set the power control transaction attributes and made the request
		END_PW_REQ	Target has accepted the power control transaction
		BEGIN_PW_RSP	Target acknowledges that it has correctly handled the power control transaction request
END_PW_RSP	Initiator has accepted the target acknowledge		
tlm_pme_handling Channel	Attributes	CMD	Command Field Enumeration: SLEEP_PME, WAKEUP_PME, PW_PME
		RESPONSE	Response status enumeration indicating the success or failure of the PME transaction
		TYPE	PME transaction enumeration type (PW_STATUS, PW_MODE_RQST, NO_INFO)
		RETAIN	Boolean attribute valid only with the SLEEP_PME command
		PDID	7-bit Power Domain Identifier of the PME transaction's power domain initiator
		TID	32-bit Target Identifier of the PME transaction's design element initiator
		SUB_PDID	7-bit Power Domain Identifier of the NRCT power domain, indicated when a RCT requests a power mode setting for another NRCT power domain
		PRIORITY	Bus arbitration level for request capable targets (RCTs) (HIGH or LOW)
	LOCK	Lock PDMgIF interconnect during a PME transaction	
	Timing Points	BEGIN_PME_REQ	A RCT has set a PME transaction attributes and initiates the Target Arbitration process
END_PME_REQ		The PDMgIF interconnect has accepted the Slave Arbitration request	
BEGIN_PME_TRANSFER		The PDMgIF interconnect transfers the PME transaction set by the RCT	
END_PME_TRANSFER		The target acknowledges that the PME transfer has been correctly done	

Table 6.5: Attributes and Timing Points of Each Channel

parameters (positive integers). In this case, the PDMgIF protocol allows defining up to n power domains in a SoC platform and up to p design elements per domain. So, each power domain must be assigned a unique n -bit PDID identifier and each design element in a power domain is assigned a p -bit identifier. n and p parameters choices depend respectively on the power domains number in a SoC low power architecture and on the maximum number of design elements included in this SoC's power domains. In a TL simulation, it is rather recommended to put these parameters generic so as the PDMgIF TL model can be easily reused and adapted to any low power architecture specification and any TL SoC model.

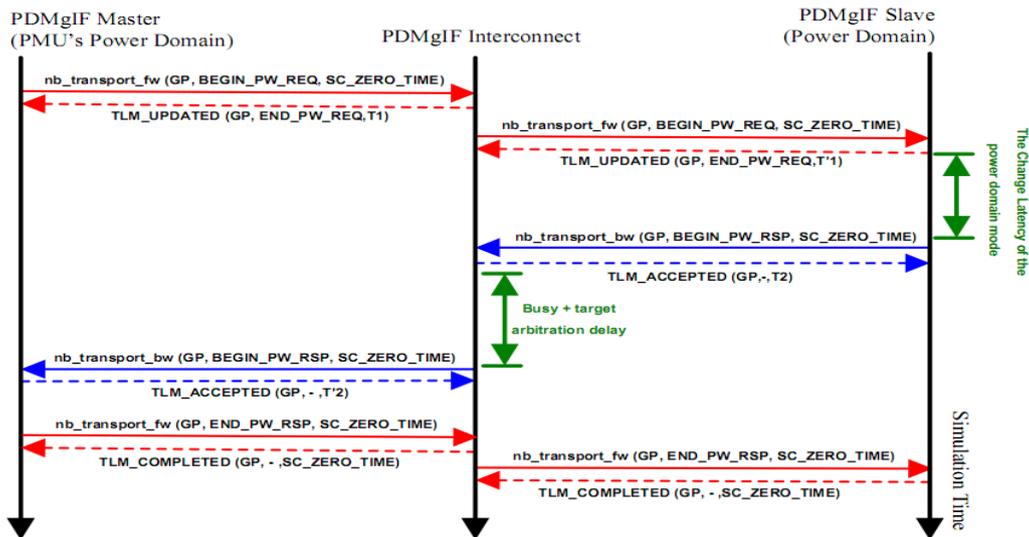
In the following sections, we set by default these parameters to 7-bit for a power domain identifier and 32-bit for a design element identifier. Table 6.5 shows the main features of our proposed PDMgIF TLM 2.0 model. They are detailed in the following.

6.3.3.3 The PDMgIF Channels and FSMs Definition

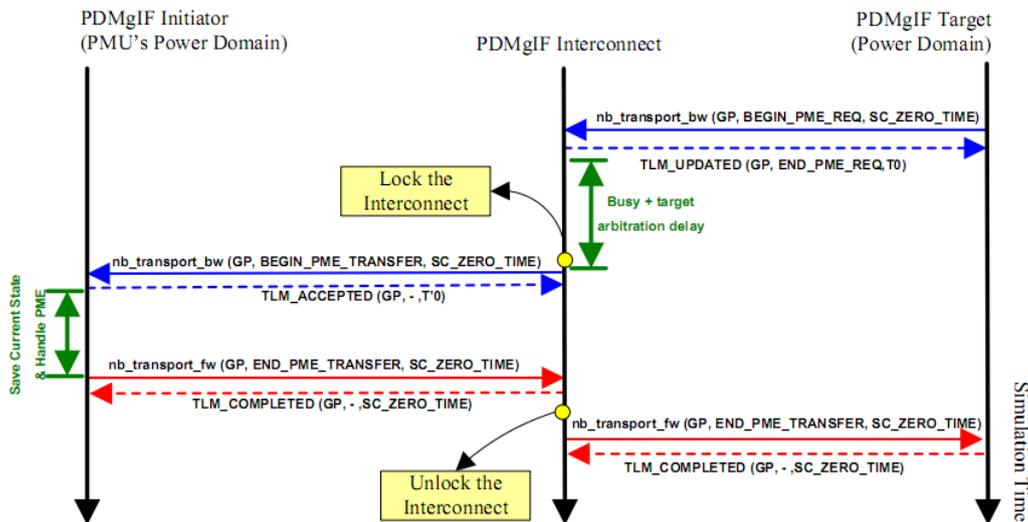
The `tlm_pwctrl` channel handles power control transactions initiated by the always-on power domain (i.e. the PMU's power domain). Each of these transactions carries either a RESET command to initialize a power domain state, or a SHUTDOWN command to switch-off a power domain without applying retention or a SLEEP_RETAIN command to switch-off a power domain while saving its retention registers and resetting the remaining ones. A WKUP command is used to switch to an active state.

When applying a multi-voltage scaling technique to a power domain, different active power modes are considered. Each corresponds to a voltage value. In this case, the PW_MODE attribute must be appropriately set. The TYPE attribute is set depending on the power control transaction destination. If the transaction intends to control the whole state of a power domain, this attribute is set to FULL. Otherwise, it is set to PARTIAL and the transaction serves to control only the power state of some design elements in a power domain. Such design elements are recognized through the 32-bit TID_MASK attribute of the transaction payload. The `tlm_pwctrl` channel attributes are mapped into a `tlm_pwctrl` extension of the TLM 2.0 generic payload. As illustrated by Table 6.5, a power control transaction can be split into four timing points that identify the beginning and the end of a power control request and response. Each timing point is mapped into a phase in the custom enumeration phase class, called `tlm_PDMgIF_phase`.

In order to allow pipelined transactions on the `tlm_pwctrl` channel, power control transactions are modeled using the non-blocking TLM 2.0 transport interface. Figure 6.23(a) depicts the permitted sequence of interactions between an initiator and a target



(a) Permitted Phase Transitions of The `tlm_pwctrl` Channel Using the TLM 2.0 Standard Transport Interfaces



(b) Permitted Phase Transitions of The `tlm_pme_handling` Channel Using the TLM 2.0 Standard Transport Interfaces

Figure 6.23: The PDMgIF Protocol Phase Sequences

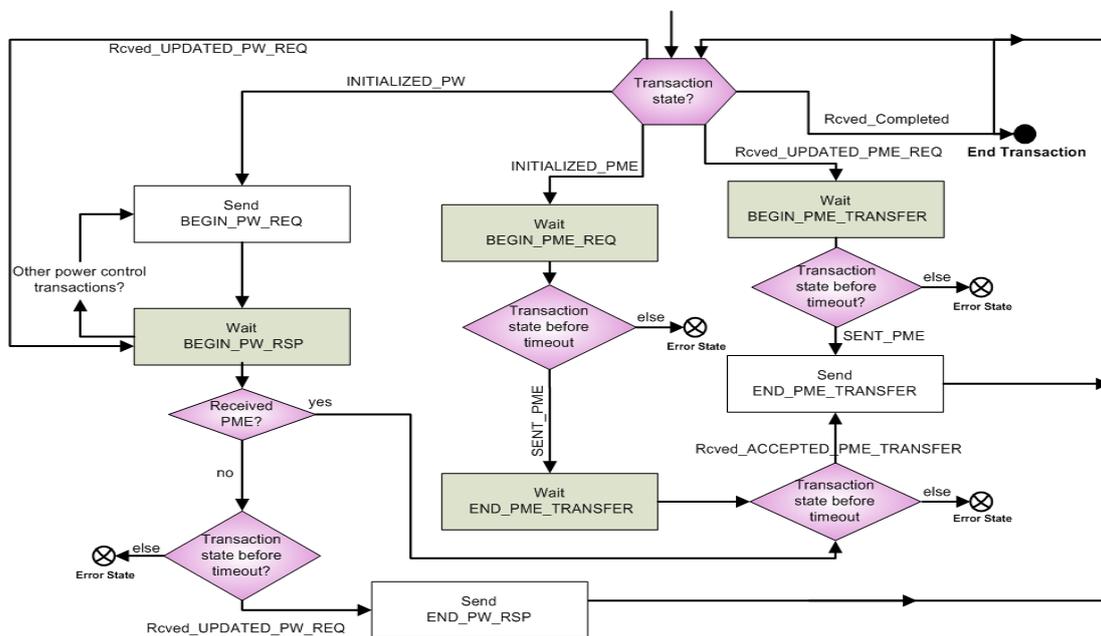
on the TLM 2.0 forward and backward paths [124] during a power control transaction course.

On the other side, the `tlm_pme_handling` channel transfers power management events. Attributes and timing points of this channel are listed in Table 6.5 and are explained in the following. Each PME transaction includes a specific command indicating the type of the PME event. Only Request Capable Targets (RCT) can issue this kind of transactions. Depending on the PME transaction goal, the TYPE attribute is set: PW_STATUS indicates that a transaction simply informs the PMU about a power domain functional status. However, PW_MODE indicates a request to set a specific power state. By setting the PRIORITY attribute, each PME transaction is assigned a high or a low priority value. This field is required for the target arbitration process. Although RCT power domains can issue a series of pipelined PME transactions, the PDMgIF interconnect must be locked once it is granted to a RCT by appropriately setting the LOCK attribute. This will force the PMU to save its current state and power management scheme status and only receive and handle the elected PME transaction. The PMU will then be prevented from exchanging any other data over the PDMgIF interconnect during this period. This is more useful in situations where the transmitted PME is timing-critical or have a direct impact on the power management decisions taken by the PMU.

The `tlm_pme_handling` channel attributes are mapped into a TLM 2.0 `tlm_pme_handling` payload extension. As illustrates Table 6.5, four timing points are supported within the lifetime of a PME transaction. Each timing point is mapped into a phase in the `tlm_PDMgIF_phase` class. According to the TLM 2.0 standard semantics, the same module can act as an initiator and a target when using the non-blocking blocking TLM 2.0 transport interface [124].

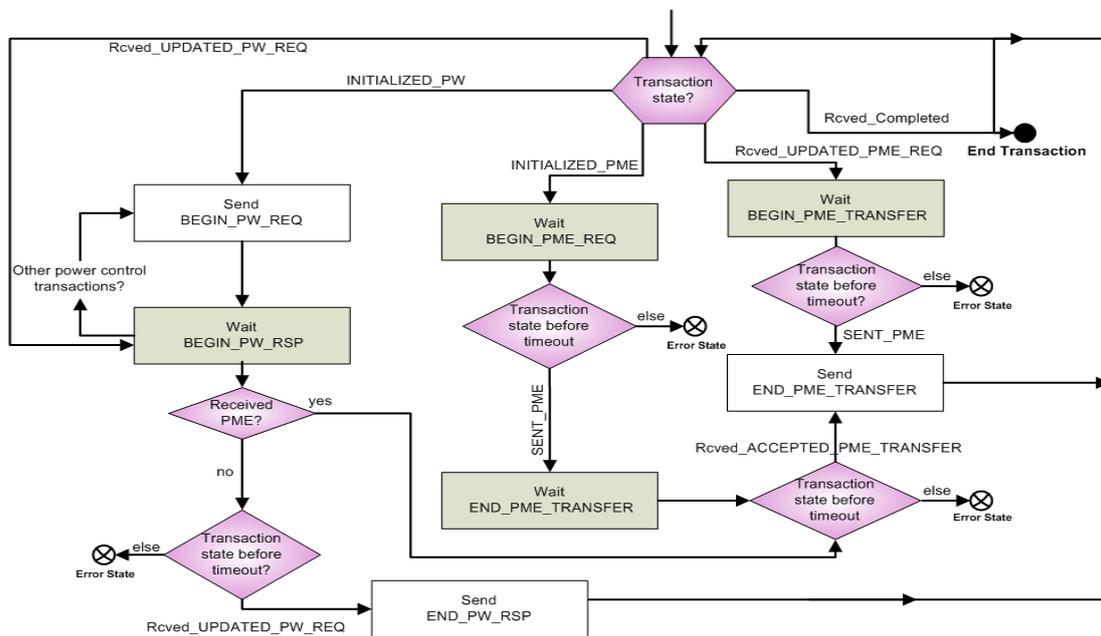
This TLM 2.0 modeling feature can be considered to model the Request Capable Target (RCT) concept. Therefore, in the context of our work, each PDMgIF target defined as a RCT will use the TLM 2.0 non-blocking transport interface calls on the backward path (i.e. `nb_transport_bw()` method call) in order to initiate a PME transaction. Figure 6.23(b) illustrates this feature and depicts the PME transaction sequencing rules between a PDMgIF initiator and target during a PME transaction transfer.

Given the sequencing between timing points of each channel shown in Figure 6.23(a) and Figure 6.23(b), the PDMgIF protocol behavior on the initiator and target sides can



⊗ An error state that may indicate a timeout expired error or a transaction state error.

(a) Finite State Machine for the Initiator Side of the PDMgIF Protocol



⊗ An error state that may indicate a timeout expired error or a transaction state error.

(b) Finite State Machine for the Target Side of the PDMgIF Protocol

Figure 6.24: Mapping Channels' FSMs to Initiator and Target Finite State Machines

be determined. Figure 6.24(a) and Figure 6.24(b) show a high-level representation of the state machines for respectively the initiator and target sides. States of each state machine correspond either to calling the TLM2.0 `nb_transport` interface methods or to waiting for the arrival of a TLM 2.0 `nb_transport` interface call from targets. More precisely, on the initiator side, states correspond to either sending PDMgIF transactions by calling to the `nb_transport_fw()` method or to waiting for calls to the `nb_transport_bw()` method from targets (Figure 6.24). On the target side, states correspond to either sending PDMgIF transactions by calling to the `nb_transport_bw()` method or to waiting for incoming PDMgIF transactions in the form of calls to the `nb_transport_fw()` method from initiators (Figure 6.24). Naturally, the PDMgIF interconnect is considered as both a PDMgIF initiator and target. Transitions between states in Figure 6.24 are conditioned by a transaction status or a PME reception.

As it can be observed in Figure 6.24(a) and Figure 6.24(b), rules for the temporal relationship between phases of a power control transaction and that of a PME transaction have been defined. For instance, Figure 6.24(a) depicts the case when a PDMgIF initiator (i.e. the PMU's power domain) receives a PME transaction while it is waiting for the **BEGIN_PW_RSP** phase of a power control transaction. Here, the initiator has to urgently treat the PME transaction and perform this PME transaction state transition to the **END_PME_TRANSFER** phase before handling a potentially received **BEGIN_PW_RSP** phase.

6.3.3.4 The PDMgIF Protocol Interconnect Structure Behavior Definition

Figure 6.25 depicts the internal structure and behavior of a SystemC TLM 2.0 PDMgIF interconnect model. It includes the following modules: Identifiers Decoder, Target Arbiter, PDMgIF Initiator and PDMgIF Target. The Identifiers Decoder routes each transaction from a power domain to another for both the forward and backward paths. For that, it uses a map that matches each power domain identifier with its corresponding power sockets. Like each PDMgIF initiator, the PDMgIF interconnect includes a PDMgIF Initiator module that derives from the PDMgIF initiator base module. Moreover, like each PDMgIF target, the PDMgIF interconnect includes a PDMgIF Target module that derives from the PDMgIF target base module. The behavior of each of the PDMgIF initiator and target base modules is depicted by respectively Figure 6.24(a) and Figure

6.24(b).

Both initiator and target base modules include an active part that contains the protocol channels state machines for initiating the outgoing transactions. While the active part of the PDMgIF initiator derives the forward path of a transaction, the active part of the PDMgIF target derives the backward path. The initiator and target base modules also contain a reactive part that processes the incoming transactions by implementing the related `nb_transport` transport interface. Depending on the received phase, this method notifies the adequate FSM in the active part. Therefore, a synchronization layer (events and arrays of ongoing transactions status) is required between the two parts of each base module. As shown in Figure 6.25, a custom behavioral part is added to customize the phase transitions sequencing defined in the base modules active parts.

The Target Arbiter module handles target arbitration requests. These requests consist in PME transactions initiated by a RCT PDMgIF target via the TLM 2.0 backward path. The Target Arbiter module decides which RCT power domain gets the bus based on the `PRIORITY` attribute (see Table 6.5). A PME transaction with a high priority level transmits timing-critical information and must be granted the PDMgIF interconnect once received. In order to guarantee that all RCT power domains can access the PDMgIF interconnect, each of them shall obey the following rule: a power domain that has trans-

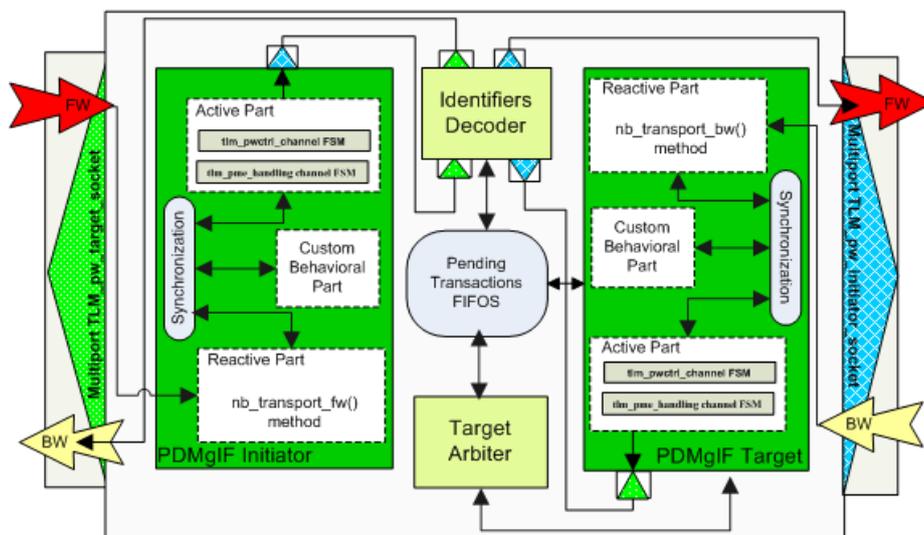


Figure 6.25: The Internal Structure and Behavior Modeling of the PDMgIF Interconnect Using the TLM 2.0 Standard Transport Interfaces

mitted a high priority PME transaction can only transmit henceforth a low priority PME transaction until another power domain issues a high priority PME.

6.3.4 Application on a Case-Study

Performance and flexibility of the TL PDMgIF interface have been tested with a TL white-box virtual prototype for an ADPCM-based audio application shown in Figure 6.26(b) [10]. The test consists in recording and playing a 5-second voice message. To record a voice message, linear audio samples are first stored in the SRAM memory. Then, a block of 10 samples is transferred from the SRAM memory to the G711 encoder in order to encode them using the G711 voice-compression algorithm. The resultant encoded samples are transferred back to the SRAM once their G711 compression is completed. This step is repeated until the end of linear samples. At this point, the G726 encoding process is performed in the same way as the G711 encoding one. Blocks, each including ten G711 encoded samples, are successively copied from the memory to the G726 encoder to get compressed using the G726 voice-compression algorithm. Each encoded block is then stored in the SRAM memory. To listen to a recorded message, the reverse procedure of recording is executed starting by the G726 decoding and ending with the G711 decoding.

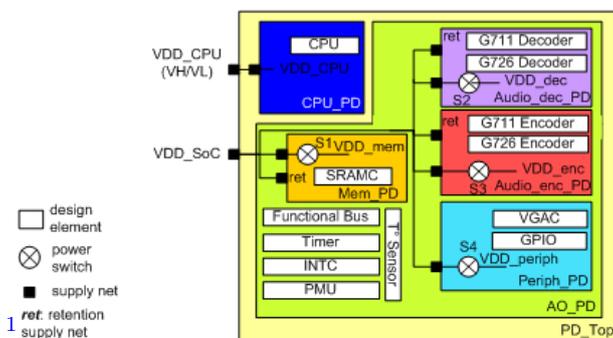
In addition to this sequential execution form, we have also tested the pipelined execution form in which a previously G711 encoded block will be processed by the G726 encoder while a new block is being encoded using the G711 encoder. The same pipelined execution principle is applied to the decoding part.

As shown in Figure 6.26, three different power architecture alternatives have been considered. Each alternative has been first defined using the PwARCH utility (see section 6.1). PDMgIF interfaces and power domains layers are then added according to our modeling approach in order to control the specified power architecture. Figure 6.26(b) shows an example of a power domain managed structure (corresponding to the first power architecture alternative illustrated by Figure 6.26(a)) layered on top of the initial functional platform. In alternative 2 (6.26(c)), each audio codec sub-module is put in a single power domain and the SRAMC belongs to the always-on power domain (AO_PD). Alternative 3 (Figure 6.26(d)) is the same as alternative 2 except the SRAMC module is put on a

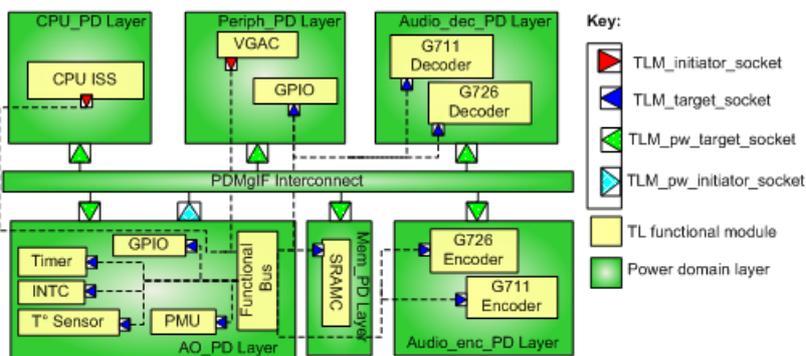
¹power architecture specifications using the PwARCH utility

single power-gated domain that can be switched-off while encoding or decoding samples.

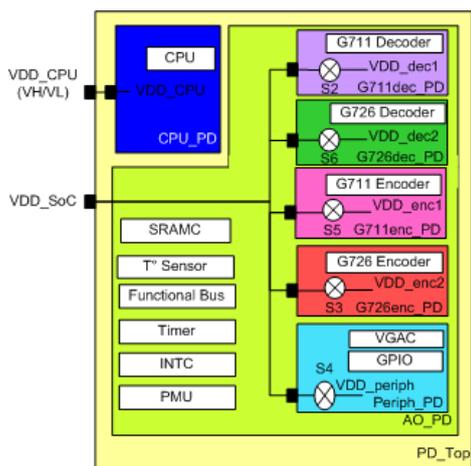
Each TL functional platform execution version (sequential or pipelined) has been simu-



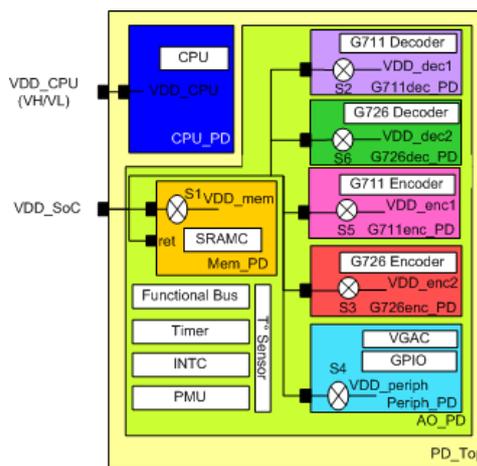
(a) Alternative 1 ¹



(b) Building Power Domains Layers and PDMgIF Interfaces (Alternative 1)



(c) Alternative 2 ¹



(d) Alternative 3 ¹

Figure 6.26: The Considered Power Architecture Alternatives

lated with the three power architecture alternatives while considering three different power management strategies for each alternative. The considered power management strategies are: scenario-based, reactive and scenario-tracking strategies. The general principles and rules of each of these strategies have been detailed in Section 4.1.4 of the Chapter 4.

Here is a brief reminder of these power management strategies: a scenario-based strategy relies on the specification of a static power state table (PST) which summarizes possible system power modes. This PST-based strategy is originally adopted by the UPF standard [30]. An example of a PST for the power architecture alternative in Figure 6.26(a) is given by Table 6.6. Here, the Record system power mode corresponds to the record voice scenario where both the G.711 and G.726 encoding are performed. Therefore, the Audio_enc_PD power domain (including the G711 and G726 encoders) must be powered-on before this scenario execution. In general, when using the scenario-based strategy, transactions on the PDMgIF interconnect consist only in power control transactions.

PD State Pw Mode	PD_Top	AO_PD	CPU_PD	Mem_PD	Audio_enc_PD	Audio_dec_PD	Periph_PD
allon	ON	ON	VH	ON	ON	ON	ON
alloff	OFF	OFF	OFF	OFF	OFF	OFF	OFF
Record	ON	ON	VL	OFF	ON	OFF	OFF
Play	ON	ON	VL	OFF	OFF	ON	OFF
Transfer_mem	ON	ON	VH	ON	-	-	OFF

- : Do not care

Table 6.6: An Example of a PST for the Power Architecture Alternative 1

In a reactive strategy, the PMU only responds to each RCT power domain requesting a power state change through a PME transaction. A scenario-tracking strategy is similar to the scenario-based one since the PMU still uses a PST. However, PME transactions that simply inform the PMU about a system functional state are allowed. This information helps the PMU to decide about the right PST power mode to set. As a simple example, consider that while the system in Figure 6.26 is recording a message, the temperature sensor issues a PME transaction to request a state change of the Audio_enc_PD power domain due to the detection of an excessive heating. Here, the PMU has to stop recording and just play the encoded samples. For that, it has to switch-off the Audio_enc_PD power domain and switch-on the Audio_dec_PD power domain instead.

Figure 6.27 shows the obtained energy savings for each alternative compared to the

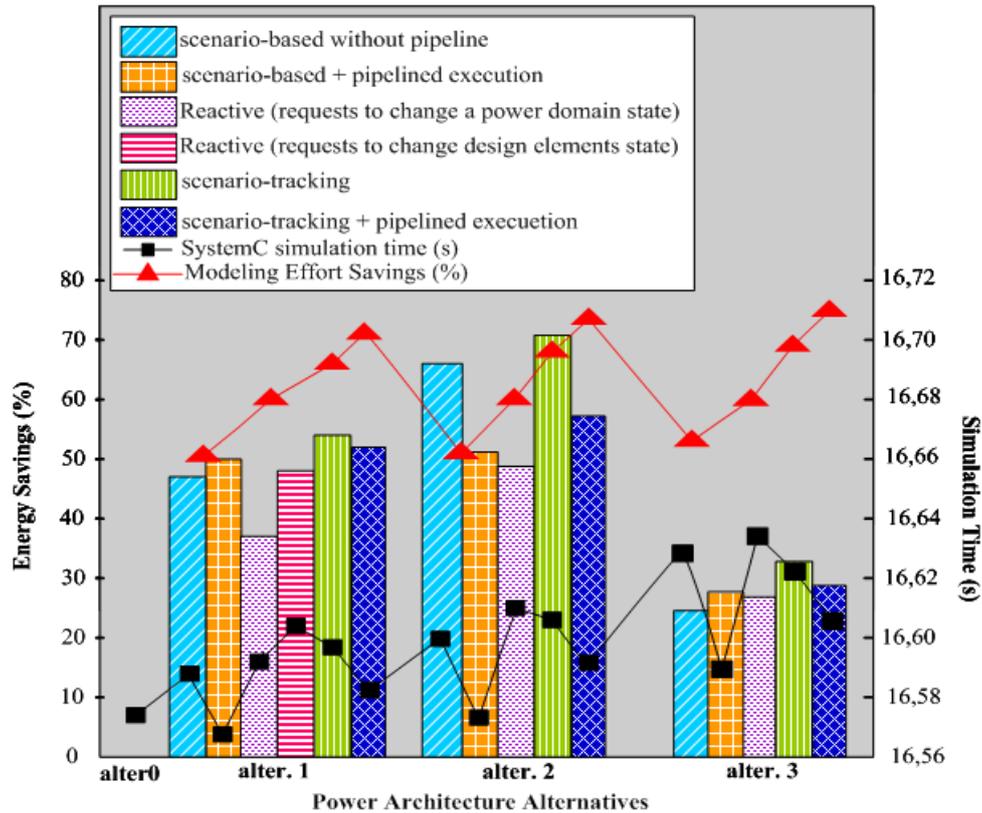


Figure 6.27: Energy savings, modeling effort savings and simulation time for the various power management strategies and power architecture alternatives

initial non-partitioned functional platform (alter.0 in Figure 6.27). These results highlight the ability of our PDMgIF protocol model to handle various power management solutions. In our case-study example, the scenario tracking strategy and the power architecture alternative 2 represent together the most energy-efficient power management solution for this platform as it saves energy by up to 70%.

Figure 6.27 shows also the modeling effort savings achieved by using the common PDMgIF protocol interface instead of SystemC signals. Modeling effort refers to the source code lines and ports number added for power domain management. These results show that our PDMgIF interface achieves flexibility to consider all types of power domain management strategies with a reduced modeling effort. As shows Figure 6.27, modeling effort is saved by up to 70% with the PDMgIF use compared to the signal-based management use for the three power architecture alternatives when applying a pipelined

execution and scenario-tracking strategy. This is due to the PDMgIF high flexibility and fast reuse whatever the applied power architecture and management strategy.

Figure 6.27 gives also the simulation time required by each strategy to record and play the same voice message according to each alternative. Differences between elapsed simulation times are due to the PDMgIF latencies and time penalties required for power state transitions. Note that only a small simulation time overhead, lower than 6%, is incurred by our PDMgIF-based modeling approach. This enforces the ability of our PDMgIF-based approach to rapidly explore different power architecture and domain management alternatives at Transaction-Level.

6.3.5 Locality and Scalability

In large Systems-on-Chip designs, the number of power domains is increasing and hierarchically structuring them is required to handle the system power states explosion problem. This increases both design and verification complexity. Low power format standards support specification of hierarchical SoC power domains structure. Management of these power domains is based on a top-level power state table specification that combines power domain states based on the states of their supply nets. According to such a power state table, a centralized power management unit operates.

However, It could be anticipated that more custom IP blocks will be created with power management features and local power control. When such IP blocks are integrated into a large SoC with their own low power information, a hierarchical power domain composition and management must be considered in order to take into account the hierarchical architecture of power management units. In the general case, such a structure allows divide and conquer principle use. Indeed, it represents a good solution to reduce the design and verification complexity implied by the use of single centralized power domain management unit. Moreover, conversely to a centralized power management structure, a hierarchical power management structure allows the exploitation of a natural hierarchy of power domains activation in a design. From a physical point of view, a hierarchical power management structure reduces the power spent in the power management controls and events when they are frequently issued on long wires. Drawbacks of the Texas Instruments's Power, Reset and Clock Manager (PRCM) listed in the State-of-the-Art Chapter (see Section 2.1.5.3) proves in general the defects of a centralized power domain

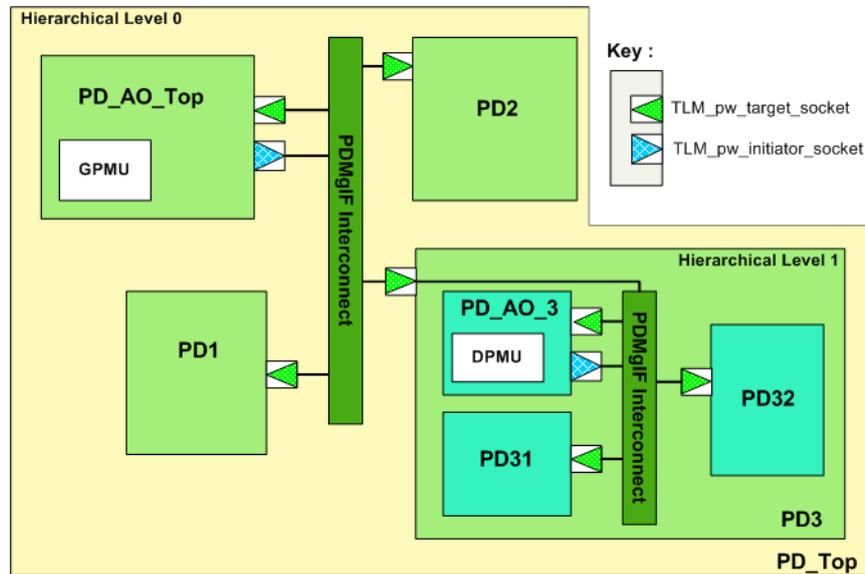


Figure 6.28: Using the PDMgIF Interface in a Hierarchical Power Domain Management Structure

management.

A good reusable and modular solution to construct a hierarchically organized power management structure is to separate the global PMU (GPMU) functionality into smaller distributed power managers (DPMUs). Each DPMU is spatially closer to the domains it controls and its activity is exerted on a specific container power domain. A DPMU only controls states of the power domains nested in this container. The use of the PDMgIF protocol interface between power domains at each hierarchy level as depicts Figure 6.28 promotes an easy and rapid interfacing of a power domain managed IP block or sub-system with an existing power domain managed system. This allows arranging hierarchically DPMUs in a tree structure such that each sub-tree is a container power domain controlled by a DPMU and wrapping an arbitrary number of nested power domains. These nested power domains set is locally managed by a DPMU and can be treated as a power domain at the next higher level of hierarchy. Figure 6.29 illustrates an example of a three-level of hierarchy of PDMgIF-based power management tree structure.

In order to illustrate transmission of power domain management control over the PDMgIF interconnect between different levels of hierarchy, let us consider the example of Figure 6.28 where a PST is used by each power management unit as a main power man-

	PD_Top	PD_AO_Top	PD1	PD2	PD3
A	ON	ON	ON	ON	OFF
B	ON	ON	OFF	ON	OFF
C	ON	ON	OFF	ON	D

Table 6.7: A Power State Table Attached to PD_Top

	PD3	PD_AO_3	PD31	PD32
OFF	OFF	OFF	OFF	OFF
D	ON	ON	OFF	ON

Table 6.8: A Power State Table Attached to PD3

agement strategy. For instance, to set the global system power mode "C" (Table 6.7), the GPMU transmits a power control transaction to the PD3 over its PDMgIF power slave socket to set its local state to the state "D". Since PD3 is a container and locally managed power domain, this control transaction is first received by a second PDMgIF interconnect wrapped by PD3 that simply routes this transaction to the PD3's PDMgIF master power domain to get handled by the PD3's DPMU component. Here, the DPMU recognizes the command from the higher hierarchical level and operates immediately to set the PD3's overall "D" state. For that, using its proper PST specification (Table 6.8), it transmits appropriate power control transactions to the PD3's nested power domains over the PD3's PDMgIF interconnect.

At first glance, an additional field in the `tlm_pwctrl` PDMgIF extension payload is required in order to designate a complex power domain state (i.e. state "D" of the PD3 power domain in Table 6.7) through a power control transaction. Nevertheless, a hierarchical power domain control might also require a careful handling of interactions between local power management units and the global one in order to respect dependencies between specific power domains states. For instance, supposing that the ON state of the PD32 power domain nested in PD3 cannot be set unless the power state of the PD2, being at a higher level of hierarchy than PD32, is already set to ON by the GPMU. Here, the order of sending power control transactions over the PDMgIF interconnect to set a global system power mode may simply resolve this dependency problem. For instance, to set the

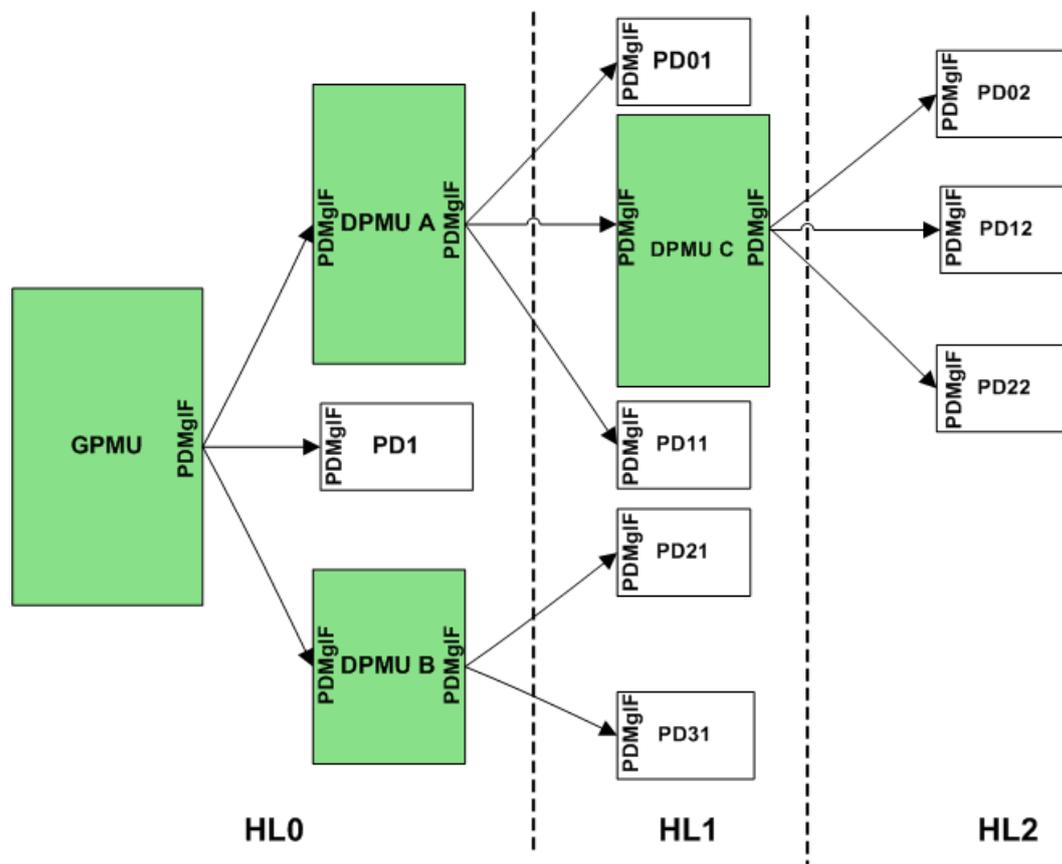


Figure 6.29: Example of Three-Level Hierarchical Power Domain Management Tree Structure

"C" power mode according to the Table 6.7, the GPMU must effectively change the PD2 state to ON before ordering the PD3 state change to "D".

In the general case, a "super" power domain's PMU ideally detains a list of dependencies between the different power domains that are under its control. In order to avoid deadlock situations between power domains controls due to a disrespect of power domains states dependencies, the "super" power domain's PMU must transmit power control transactions in a specific order using this dependency list.

Cases where dependent power domains belong to different hierarchical levels and are under the control of different DPMUs are more complex to handle. Here, the PMU in a higher hierarchical level than these DPMUs must be informed about such dependency cases and best handle them. For instance, in Figure 6.29, a dependency between the PD22

power domain at the hierarchical level 2 and the PD31 power domain at the hierarchical level 1 must be handled by the GPMU. For that, a PME transaction, that requests to urgently check this dependency respect and handle it if not already managed, must be routed over the PDMgIF interfaces from the DPMUC to the GPMU passing through DPMU A. Here, the PDMgIF interface must be extended so that reliable and safe interactions between DPMUs over this interface can be allowed. For instance, a data field informing about the current request hierarchical level and another data field indicating the type of requested dependency (`SLEEP_DEP`, `WKUP_DEP`, `PW_DEP`) are examples of possible extensions of the `tlm_pme_handling` payload.

6.3.6 Concluding Remarks

The USLPACom utility includes a TLM 2.0 simulation model of a new and flexible inter-power-domain protocol interface, called PDMgIF, used along with, either the PAL or the PwARCH USLPAL utilities. A great benefit of this interface is the easy reuse and the platform-independency. It allows an easy integration of a power-domain-managed architecture into a functional SoC model and enables power domains reuse in different platforms. Separation of functional and power concerns promotes this easy integration. The PDMgIF proposed features represent a potential extension of the UPF and CPF standards that miss power architecture control semantics.

Nevertheless, a more formal study of this proposed protocol properties is strongly needed in the future. This study would allow checking this protocol completeness and correctness and solving its potential ordering and deadlock freedom issues. In addition, we have discussed the current PDMgIF protocol scalability when applied to a SoC having a large power domains number and depicting a high power management overhead due to a frequent change of power domains states. We have shown how this PDMgIF eases the hierarchical organization of power domain management units in a tree-structure and offers a plug-and-play solution for locally managed container power domains. Nevertheless, the presented extension ideas of the PDMgIF protocol in order to best handle interactions between distributed power domain management units and dependencies among power domains belonging to different hierarchical levels are still to deepen and validate in the future.

Chapter 7

Conclusions and Prospects

7.1 Summary of Contributions	258
7.2 Prospects	262
7.2.1 Extending the USLPAF Framework With Additional Power-Aware Simulation Semantics	262
7.2.2 Thermal Behavior Analysis and Management Based on Power-Aware Simulation	264
7.2.3 Automating LPDISE	264
7.2.4 A Toolset for PMPs Identification and Off-Line Simulation and Validation	265
7.2.5 Complementary Studies on Power-Aware Verification	266
7.2.6 Towards a Standard Structure for Easy Integration and Reuse of IPs' Power Intent and Control Features	267
7.2.7 Validation of System-Level Results at Lower Levels of Abstraction than TLM	268
7.3 Author's Publications Related to This Thesis	269

7.1 Summary of Contributions

THIS dissertation studies at TLM the relationship between a power design, a functional design and a power management strategy controlling these two designs in order to achieve the highest energy savings. It proposes a complete and unified framework, called

USLPAF, which enables the exploration of this relationship towards deciding at TLM about an optimized energy-efficient power management solution including an energy-efficient power architecture and its control strategy.

This framework depicts a strong emphasis on separating functional and power-aware concerns. Indeed, this separation of concerns methodology is well-suited for a transaction-level of modeling where priority is mainly given to functional validation of the embedded software on a TL functional virtual prototype. Non-functional modeling features should be allowed to be easily added or omitted dependently on simulation purposes. In addition, this methodology meets the original goal to decouple the functional behavior from the power-aware one in existing low power format standards. Nevertheless, while these existing standards operate at low levels of abstraction starting from RTL, the USLPAF framework has been conceived to operate rather at TLM, analogously to the Unified Power Format standard. This was the major challenge throughout the development of USLPAF. To fit a TLM use, the USLPAF abstracts UPF semantics relevant to this level of abstraction. This choice to stick to an existing industry standard and use it as a support to build this framework was not arbitrary. Its main advantage is to facilitate connecting the USLPAF TLM power-aware flow to classic RTL low power design flow. Connection is ensured through the generation of RTL-based UPF files from their corresponding TL abstract specifications. With few refinements to the generated UPF files through mainly adding RTL- and chip-specific commands (e.g. ports connected to voltage regulators and to the Power Management Integrated Circuit (PMIC)), these files can be more rapidly created and used as an input of RTL simulation tools.

Actually, USLPAF goes beyond this objective to also propose new semantics that are lacking in low power design standards, but that are still compatible with those defined by these standards. Therefore, the new proposed concepts in this dissertation can be considered as potential extensions of these standards.

In order to reliably achieve its different challenging objectives, the USLPAF framework includes a set of modular approaches and modeling techniques that form the main contributions of this thesis. We recall them in the following:

- **A well-structured Transaction-Level power-aware methodology**

At the heart of the USLPAF framework lies the USLPAM methodology. This methodology defines a well-structured and multi-stage way to build a TL power-aware platform

from its functional SystemC/TLM model. The USLPAM different stages range from TL power intent specification to the inference, simulation and verification of the power-aware behavior implied by this intent. Being widely inspired by the Unified Power Format low-power standard, these different stages preserve a separation of power and functional concerns when combining the existing functional behavior with the added power-aware one and use abstract UPF-like power-aware simulation and verification semantics throughout the USLPAM flow. In order to be adequately applied on each type of functional TL virtual prototypes regardless of the accessibility degree offered by the prototype, a set of fundamental principles on which this methodology is based are presented in the form of essential requirements. These requirements must be absolutely fulfilled by each implementation approach of the USLPAM methodology.

- **A method for identifying the power management points of power domains:**

Power Management Points (PMPs) are defined in this thesis as locations in the functional source code where the system power mode can be changed. According to a power domain partitioning, each power domain is assigned a set of PMPs in respect of inter-power domains functional and structural dependencies. Specification of these points relies on the modeling of each SystemC/TLM component behavior as an Extended Finite State Machine (EFSM) followed by the conversion of each functional EFSM to a power-aware one in which requirements on power domains states are added. The set of PMPs of components in a same power domain form this power domain's PMPs and helps the PMU to efficiently control this power domain's state. Being the second stage of the USLPAM methodology, the identification of PMPs is performed offline based on analyzing the output traces of a TL functional platform simulation. It prepares the rest of the USLPAM stages, specifically the specification of a low-power infrastructure, the implementation of a power management strategy and the specification and placement of power-aware properties checks.

- **A dynamic contract-based power-aware verification approach**

The USLPAM methodology in this thesis includes a power-aware assertion-based verification process. A set of power-aware properties, which are mainly related to the low-power structure and its effects on the normal operation of the functional model, have been defined using a contract-based reasoning and classified into four classes of contracts. When applying the USLPAM flow, these contracts are checked incrementally through simulation in the form of specific types of assertions. A method to build power-aware monitors and

automate source code annotation at appropriate locations with contract-based instrumentation calls has been proposed in this thesis. This method can be used with the different standalone utilities of the USLPAL Library presented in this dissertation.

- **A flexible TL power domains management protocol interface**

The USLPAL library of the USLPAF framework includes the USLPAcom utility that provides built-in features of transactional power management communications between power domains. These features correspond to a Transaction-Level simulation model of a new power domain management protocol interface, called PDMgIF, proposed in this dissertation. A great benefit of this interface is the easy reuse and the platform-independency. It allows an easy integration of a power-domain-managed architecture into a functional SoC model and enables power domains reuse in different platforms. Separation of functional and power concerns promotes this easy integration. The PDMgIF proposed features represent a potential extension of the UPF and CPF standards that miss power architecture control semantics. The usability of this PDMgIF interface model and its potential extensions to handle the hierarchical power domains management case have also been discussed.

- **A source code instrumentation approach for the USLPAM application on white-box types of virtual platforms**

To take advantage from the full source code availability of a white-box virtual prototype, we have proposed a source code instrumentation approach based on the use of the PwARCH utility of the USLPAF framework. This utility eases the instrumentation process throughout the USLPAM flow while meeting all this methodology's requirements. While locations in the source code are almost easily identified and instrumented with only few lines of codes, an MDE approach has been included into the methodology flow to ease instrumenting the main platform source code with a semantically and syntactically correct power architecture structure, allowing hence the rapid exploration of different energy-efficient power intent alternatives throughout the entire USLPAM flow.

This USLPAF utility can be used in standalone or in conjunction with the USLPAcom utility. For the standalone use, power domains are changed state when the PMU calls simple functions implemented in the power components classes of the PwARCH utility. However, in the case of the USLPAcom utility use, these function calls are replaced by power-aware transactional communications over the PDMgIF interface while

the PwARCH utility remains needed to specify TL power intent.

- **A power-aware wrapper-based approach for the USLPAM application on black-box types of virtual platforms**

This thesis delineates the main constraints of the USLPAM application on black-box virtual prototypes covering mainly the power intent specification and power-aware contracts checking. Faced to these constraints that prohibit the use of instrumentation as an implementation approach, we have proposed an alternative approach to implement power-aware wrappers for black-box virtual platforms that meet all the USLPAM requirements. These wrappers layer the power-aware simulation and verification capabilities on top of each black-box functional block allowing a UPF-like separation of concerns. Modularity of this approach is enforced by the use of the PAL utility provided by the USLPAL library as this utility helps customizing the required behavior of each power-aware layer. Similarly to PwARCH, the PAL utility can be used in standalone or in conjunction with the USLPACom utility.

7.2 Prospects

Future research directions where the work presented in this thesis will be useful are numerous. They may include the points listed in the sequel.

7.2.1 Extending the USLPAF Framework With Additional Power-Aware Simulation Semantics

- **Adding extensions of the PDMgIF for hierarchical power domain management to the USLPACom Utility**

Section 6.3.5 of this dissertation has outlined the scalability of the PFMgIF protocol interface integrating the USLPACom utility of the USLPAF framework: although PDMgIF eases the hierarchical organization of power domain management units in a tree-structure and offers a plug-and-play solution for locally managed container power domains, extending it to best handle interactions between distributed power domain management units and dependencies among power domains belonging to different hierarchical levels would most likely be necessary. In this context, further and deeper studies on the hierarchical

power domain management and possibilities to extend the USLPACom Utility are required.

- **Modeling in SystemC-TLM of clock and reset domains, clock-gating and DVFS-oriented capabilities**

The USLPAF framework presented in this thesis uses mainly the industry-standard UPF as a support to specify and simulate power intent at TLM. This standard targets in particular power gating as a power management technique due to the complexity implied by this technique concerning the design and management of additional interfaces crossing power domains boundaries as well as power domains local states and inter-power domains dependencies. For that, this dissertation focuses more specifically on adding power gating oriented capabilities at TL.

However, the challenge of the joint validation of a power intent and a functional virtual platform should also cover the modeling in SystemC-TLM of clock and reset domains, clock-gating-oriented and DVFS-oriented capabilities. Enabling TL power intent specifications for such additional power management techniques and simulating their direct impact on the functional behavior of the TL platform would help to find a power management solution potentially more energy-efficient than that found with the only application of power gating.

Actually, the lack of a standard support, such that of UPF, to model simulation and verification semantics related to these additional features represents a major difficulty. Nevertheless, these added features may be seen as potential extensions of existing low power designs. Therefore they should be compatible as much as possible to the power gating oriented semantics defined by these standards. A study of structural and behavioral relationships between power architectures and power controller oriented management strategies dedicated for each power management technique (power gating, DVFS, clock-gating) could help anticipate this compatibility.

- **Study of hybrid power management behavior along with power architecture**

To provide a TL virtual prototype that is more faithful to the final real system, many virtual prototyping industrial tools make available examples of virtual platforms with an Operating System (OS) (such as embedded linux, μ cos or android). The OS runs in simulation and schedules the embedded application tasks on virtual hardware resources in order to achieve performance objectives (such as real time constraints and energy savings).

Most of these Operating Systems incorporate an OS-oriented (i.e. software) power manager enabled to manage statically or dynamically the system power consumption through efficiently scheduling software tasks.

Conversely, in this thesis, we were particularly interested in Power Controller (PC)-oriented power management that rather focuses on power domains states control according to the workload being executed during simulation. So, an underlying problem that necessitates to be carefully studied as a perspective of this dissertation is:

In the hybrid power management case, where a PC-directed power manager is in charge of scheduling power domains states to maximize total energy savings, and an OS-directed power manager is simultaneously in charge of scheduling the software application tasks for the same purpose, how to correlate these two power managers behaviors while avoiding deadlocks and conflicts between their decisions and keeping the functional and power system states coherent?

7.2.2 Thermal Behavior Analysis and Management Based on Power-Aware Simulation

The analysis of dynamic thermal behavior in complex embedded IC technology becomes of prime importance because refined thermal strategies need to be developed to avoid system performance degradation if hot spots appear at runtime. From the power architecture intent it is possible to get the sequence of activation of each domain in the abstracted architecture corresponding to the abstract execution of the target application. Thus at this level a dynamic thermal analysis could be realized if a thermal model reflecting the power architecture intent is developed. With the virtual platform technology, a power trace resulting of activation of domains by the power manager could be produced. With this power trace, a dynamic evolution of system temperature could be calculated which provides back the input to the thermal management strategy implemented in the system.

7.2.3 Automating LPDISE

A design space exploration (DSE) approach that automates the LPDISE iterations in the proposed USLPAM methodology flow is missing in this dissertation and can be addressed in future works. Based on design constraints and properties extracted from an abstract

performance evaluation step, such a DSE approach is required to explore potential power intent alternatives. Exploration should be done in terms of domain decomposition of the whole system architecture according to specific requirements of low power techniques (power gating, AVS, DVFS, clock and reset) while taking performance, design costs and power into account and covering maximum power intent energy-efficient candidates. In the context of the power gating management technique for instance, such a DSE approach would aim to find an optimal clustering of hardware blocks of a chip into power domains that implement efficient low power strategies and require a minimum power interfaces (isolation cells, retention registers and level shifters).

7.2.4 A Toolset for PMPs Identification and Off-Line Simulation and Validation

This dissertation has presented a formalized method for capturing specific power control and verification requirements according to the existing functional behavior of the system components and a specific power domain partitioning alternative. These requirements are captured by the enrichment of EFSM-based behavioral models of each component with power domain state transitions requirements. The enriched models of a same power domain's components helps defining the different power management points (PMPs) per power domain. Actually, these PMPs represent contracts between the functional system behavior and the added power-aware one that must not be violated when combining the two behaviors. Note that this modeling method is still manual in this dissertation which makes it error-prone and tedious. Nevertheless, it can be the basis of a toolset that enhances it through implementing:

- An automatic building of functional components' EFSM models from their SytemC/TLM description.
- An automatic conversion of these functional EFSMs to power-aware ones and components' PMPs identification.
- Execution of power domains' EFSM models and validation of coherence among power domains PMPs.

7.2.5 Complementary Studies on Power-Aware Verification

To the best of our knowledge, the assertion-based power-aware checking framework proposed in this dissertation is the first research work interested in checking functional/power coherence through monitoring a set of pre-defined power-aware properties during a TL simulation. Using a component-based reasoning to specify these properties and assertion-based contracts to implement them enforces our approach originality. Nevertheless, a set of enhancements could still be brought to the current verification framework. In this direction, two studies listed in the following could be carried out:

- **Automatic generation of power-aware monitors**

For simple designs, manually writing power-aware monitors source code may be feasible. However, in most industrial platforms with a large number of functional components and power domains, writing and debugging power-aware monitors manually would be high-cost and error-prone. Therefore, automating the generation of monitors from formal specifications of requirements in general has always been of primary importance for industrials. Similarly to our needs, a good idea would be to propose a practical mechanism for automatic generation of SystemC/TLM power-aware monitors from power-aware EFSM-based models of a platform's components, used primarily in this dissertation for power domains PMPs identification. This mechanism should guarantee coverage-driven power-aware checking. In other words, it should ensure that generated power-aware monitors would detect all finite executions of the power managed behavioral model that violate power-aware properties.

- **Using the Property Specification Language (PSL) standard for TL power-aware checking**

The Property Specification Language (PSL) IEEE standard [8] defines powerful semantics for semi-formal specifications applied to assertion-based verification. In recent works, some layers of this language have been enriched in order to enable assertions expression for SystemC [141] and SystemC/TLM designs [76] [127]. However, these efforts have not yet tackled the problem of PSL use for TL power-aware properties verification as treated in this dissertation. Thus, extending this standard to formally specify the power-aware requirements of a SystemC/TLM functional model defined in this thesis and use these specifications with power-aware assertion monitors represents an open innovative direction of research.

7.2.6 Towards a Standard Structure for Easy Integration and Reuse of IPs' Power Intent and Control Features

Some IP blocks already include few power management features that are not easily understood or captured by this IP user unless this IP provider gives minimal information that best summarizes such features while structuring it in a standard way. This would enormously facilitate not only this IP integration within different industrial tool flow, but also the specification of its power intent either with support to a specific format (i.e. UPF and CPF) used in the design flow or according to an appropriate process such as the USLPAM methodology flow proposed in this thesis.

In particular, an IP block may be delivered with its own power controller. Well-structured information on the internal and external interfaces of this controller within this IP, as well as the different power features of this IP used by the power controller (i.e. the different Operating Performance Points (OPPs) of the IP for the multi-voltage scaling power management technique use), would facilitate the use of the PDMgIF interface presented in this thesis to properly and rapidly interface the existing power controller of the IP with the full system global power manager.

The power-aware black-box approach presented in this thesis has underlined the difficulties to apply partial retention of internal and non-memory-mapped registers of black-box IPs on power-down. Typically, in addition to information on the registers structure of a packaged black-box IP, information on this IP's registers whose states must be retained on power-down is an important design feature that must be delivered by the IP provider in order to facilitate to this IP end-user its debug, reuse and enrichment with other features, either functional or non-functional.

A good solution to deal with this lack of well-structured information on IPs power management features is to extend the syntax of the IP-XACT [7] standard. By doing so, such additional and IP vendor-specific information would be represented in a standard way easing as much as possible the adherence to existing design flows. Obviously, this solution implies also the development of adequate tools supporting access to this specific information and interpreting it properly. In a TLM context specifically, it would be also interesting to enable the use of these IP-XACT extensions as a support to specify power intent alternatives and integrate them automatically into existing TL virtual prototyping

tools.

7.2.7 Validation of System-Level Results at Lower Levels of Abstraction than TLM

Another direction for future work would be to validate, at lower levels of abstraction than TLM, the energy savings results obtained at TLM using the USLPAF framework. In this context, three fundamental questions need to be tackled:

- Is the UPF file, that has been generated in this work using an MDE approach, semantically and syntactically correct when simulated using low power design tools starting from RTL? Is the RTL-based power-aware behavior, that is inferred into the HDL functional description based on this UPF specification, does not really alter the RTL system functionality and remains coherent with the HDL functional design as it has been guaranteed at TLM? A detected error in both cases could indicate failures or gaps either in UPF semantics abstraction at TLM, or in the transformation rules used by the MDE generation process, or in the modeling techniques proposed in this dissertation.
- Does a power management solution, that has been elected at TLM using the USLPAM methodology as the most energy-efficient solution, remain so throughout the rest of the low power design flow?
- The latter question generates another main and classical issue: What about the power estimation accuracy obtained at TLM using power domain based models proposed in this thesis? Do we get a small margin of error when comparing the power consumption values obtained at TLM and at lower levels?

To answer this question it is necessary to have relevant and sufficiently precise models of power consumption/energy of the platform's IPs. However, these patterns of consumption must be more related to the IPs' PMPs considered at TLM. Thus, estimation studies at TLM cited in Chapter 2 find their meanings here. The relationships between these estimation techniques and power intent modeling in this thesis can also be a subject of study.

The different challenging issues raised from this set of prospects have initiated the French National ANR Project HOPE ¹ (Hierarchically Organized Power/Energy management).

¹an ANR HOPE Project bearing reference ANR 12 INSE 0003 , <http://anr-hope.unice.fr/>

7.3 Author's Publications Related to This Thesis

Below is the list of publications on the work done by the author of this dissertation.

- **Journal Papers:**

[113] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. Using unified power format standard concepts for power-aware design and verification of systems-on-chip at transaction level. *Circuits, Devices Systems, IET*, 6(5): 287-296, 2012.

[114] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. PDMgIF: A flexible protocol interface for transaction-level power domain management. *Computers & Digital Techniques, IET*, 2013.

- **Conference Papers:**

[111] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. A methodology for power-aware transaction-level models of systems-on-chip using upf standard concepts. In Jose L. Ayala, Braulio Garcia-Camara, Manuel Prieto, Martino Ruggiero, and Gilles Sicard, editors, *PATMOS*, volume 6951 of *Lecture Notes in Computer Science*, pages 226-236. Springer, 2011.

[110] Ons Mbarek, Amani Khecharem, Alain Pegatoquet, and Michel Auguin. Using model driven engineering to reliably accelerate early low power intent exploration for a system-on-chip design. In Sascha Ossowski and Paola Lecca, editors, *SAC*, pages 1580-1587. ACM, 2012.

[112] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. Black-box and white-box early power intent simulation and verification: Two novel approaches. In *DASIP*, pages 1-8. IEEE, 2012.

[115] Ons Mbarek, Alain Pegatoquet, Michel Auguin, and Housseem Eddine Fathallah. Power-aware wrappers for transaction-level virtual prototypes: A black box based approach. In *VLSI Design*, pages 239-244. IEEE, 2013.

Chapitre 7'

Conclusions et Perspectives (In French)

7'.1 Résumé des Contributions

CETTE thèse étudie au niveau transactionnel la relation entre un design de puissance, un design fonctionnel et une stratégie de gestion d'énergie contrôlant ces deux designs dans le but d'atteindre la plus grande économie d'énergie possible. Elle propose un environnement unifié et complet, appelé USLPAF, qui permet l'exploration d'une telle relation afin de décider au niveau TLM une solution de gestion d'énergie optimale et efficace en énergie. Cet environnement met l'accent sur la séparation des aspects fonctionnels et ceux orientés puissance. En effet, une telle méthodologie de séparation convient bien à un niveau de modélisation transactionnel où la priorité est principalement donnée à la validation fonctionnelle du logiciel embarqué sur un prototype virtuel fonctionnel modélisé au niveau transactionnel. Les aspects non-fonctionnels devraient être autorisés à être facilement ajoutés ou omis dépendamment des fins de simulation. En outre, cette méthode répond à l'objectif initial de découpler le comportement fonctionnel de celui orienté puissance visé par les normes de conception faible puissance existants. Néanmoins, si ces normes existantes fonctionnent à de faibles niveaux d'abstraction à partir de RTL, l'environnement USLPAF a été conçu pour fonctionner plutôt au niveau TLM, d'une façon analogue à ces normes de conception faible puissance. C'était le défi majeur tout au long de l'élaboration de USLPAF.

Pour s'adapter à une utilisation TLM, le USLPAF fait l'abstraction des sémantiques UPF qui sont pertinentes à ce niveau d'abstraction. Ce choix de s'en tenir à une norme

de l'industrie existante et l'utiliser comme un support pour construire ce cadre n'était pas arbitraire. Son principal avantage est de faciliter la connexion du flot de conception TLM orienté puissance de l'environnement USLPAF au flot classique de conception de puissance au niveau RTL. La connexion est assurée grâce à la génération de fichiers UPF basé RTL à partir de leurs correspondantes spécifications abstraites au niveau transactionnel. Avec quelques modifications aux fichiers UPF générés à travers principalement l'ajout de commandes spécifiques RTL (par exemple les ports connectés à des régulateurs de tension et le circuit intégré de gestion de l'alimentation (PMIC)), ces fichiers peuvent être plus rapidement créés et utilisés comme des entrées à des outils de simulation RTL. En fait, USLPAF va au-delà de cet objectif pour aussi proposer de nouvelles sémantique qui manquent dans les normes de conception de faible puissance, mais qui sont toujours compatibles avec celles définies par ces normes. Par conséquent, les nouveaux concepts proposés dans cette thèse peuvent être considérés comme des extensions possibles de ces normes.

Afin de réaliser efficacement ces différents objectifs ambitieux, l'environnement USLPAF comprend un ensemble d'approches modulaires et de techniques de modélisation qui forment les principales contributions de cette thèse. Nous les rappelons dans ce qui suit :

- **Une méthodologie de conception orientée puissance au niveau transactionnel bien structurée**

Au coeur de l'environnement USLPAF réside la méthodologie USLPAM. Cette méthodologie définit une manière bien structurée et multi-étages pour construire une plateforme faible consommation au niveau transactionnel à partir de son modèle SystemC/TLM fonctionnel. Les différentes étapes de USLPAM, allant de la spécification d'intention de puissance au niveau TL à l'inférence, la simulation et la vérification du comportement orienté puissance résultant de cette spécification. Largement inspiré par la norme de faible puissance UPF, ces différentes étapes permettent de préserver la séparation entre les aspects fonctionnels et de puissance lorsque l'on combine le comportement fonctionnel existant avec le comportement de gestion de puissance, et utilisent tout au long du flot USLPAM des sémantiques de vérification et de simulation orientées puissance qui sont abstraites et semblables à celles définies par UPF. Pour être appliqué de manière adéquate sur chaque type de prototypes virtuels transactionnels fonctionnels, et ce indépendamment du degré d'accessibilité offerte par le prototype, les principes fondamentaux sur lesquels cette méthodologie est basée sont présentés sous la forme d'exigences essentielles de USLPAM.

Ces exigences doivent être absolument respectées par chaque approche de mise en œuvre de la méthodologie USLPAM.

- **Une méthode pour identifier les points de gestion d'énergie des domaines d'alimentation**

Les points de gestion d'énergie, notés PMP, sont définis dans cette thèse comme étant les emplacements dans le code source du modèle fonctionnel où le mode de consommation d'énergie du système peut être modifié. Pour un partitionnement en domaines d'alimentation, chaque domaine est attribué un ensemble de PMPs tout en respectant les dépendances structurelles et fonctionnelles entre les différents domaines d'alimentation. La spécification de ces points s'appuie sur la modélisation du comportement de chaque composant SystemC/TLM comme une Extended Finite State Machine (EFSM), suivie par la conversion de chaque EFSM fonctionnelle à une EFSM orientée consommation d'énergie dans laquelle des exigences sur les états énergie du domaine d'alimentation du composant sont ajoutées. L'ensemble des PMPs des différents composants d'un même domaine d'alimentation forme les PMPs de ce domaine et permet à l'unité de gestion d'énergie (PMU) de contrôler efficacement l'état d'énergie de ce domaine. Tout en étant la deuxième étape de la méthodologie USLPAM, l'identification des PMPs est effectuée hors ligne en se basant sur l'analyse des traces d'exécution du modèle transactionnel de la plateforme fonctionnelle. Cette étape prépare le reste des étapes de USLPAM, notamment la spécification d'une infrastructure de faible puissance, la mise en œuvre d'une stratégie de gestion d'énergie et la vérification de propriétés orientées consommation d'énergie.

- **Une approche de vérification dynamique orientée consommation d'énergie basée sur le concept de contrat**

La méthodologie USLPAM dans cette thèse comporte un processus de vérification orienté consommation d'énergie et basé sur l'utilisation d'assertions. Des propriétés orientées consommation d'énergie, qui sont principalement liées à la structure de faible consommation et ses effets sur le fonctionnement initial du modèle fonctionnel, ont été définies selon un raisonnement basé sur les contrats et ont été classées en quatre catégories de contrats. Lors de l'application du flot USLPAM, ces contrats sont vérifiés progressivement en simulation sous la forme de certains types d'assertions. Une méthode pour construire des moniteurs orientés consommation d'énergie et automatiser l'annotation du code source avec des vérifications de contrats aux endroits appropriés a été proposée dans cette thèse. Cette méthode peut être utilisée avec les différents utilitaires autonomes de la bibliothèque

USLPAL présentés dans cette thèse.

- **Une interface protocolaire flexible de gestion des domaines d'alimentation au niveau transactionnel**

La bibliothèque USLPAL de l'environnement USLPAF inclut l'utilitaire USLPAcom qui fournit des aspects intégrés de communications au niveau transactionnel entre les domaines d'alimentation en vue de la gestion de leurs états d'énergie. Ces aspects correspondent à un modèle de simulation au niveau transactionnel d'une nouvelle interface protocolaire de gestion des domaines d'alimentation, appelée PDMgIF, proposée dans cette thèse. Un grand avantage de cette interface est la facilité de sa réutilisation et son indépendance de toute plateforme fonctionnelle.

Elle permet une intégration aisée d'une architecture en domaines d'alimentation dans un modèle fonctionnel de système sur puce. Elle permet aussi la réutilisation de domaines d'alimentation dans différentes plateformes. D'ailleurs, la séparation des aspects fonctionnels et de consommation d'énergie favorise cette intégration facile. Les fonctionnalités proposées par PDMgIF représentent un potentiel d'extension des normes UPF et le PCF, qui manquent déjà des sémantiques de contrôle de l'architecture de consommation d'énergie. La facilité d'utilisation de ce modèle d'interface PDMgIF et ses extensions potentielles dans le cas de la gestion hiérarchique des domaines d'alimentation ont également été discutés.

- **Une approche d'instrumentation du code source pour l'application de USLPAM sur les types boîte blanche de plateformes virtuelles**

Pour profiter de la disponibilité du code source d'un prototype virtuel de type boîte blanche, nous avons proposé une approche d'instrumentation du code source basée sur l'utilisation de l'utilitaire PwARCH de l'environnement USLPAF. Cet utilitaire facilite le processus d'instrumentation à travers le flot USLPAM tout en répondant à toutes les exigences de cette méthodologie. Alors que des endroits dans le code source sont presque facilement identifiés et instrumentés avec quelques lignes de codes, une approche MDE a été incluse dans le flot de la méthodologie afin de faciliter l'instrumentation du code source de la plateforme avec une structure d'architecture de consommation d'énergie correcte sémantiquement et syntaxiquement. Cela permet ainsi l'exploration rapide des différentes alternatives d'intention de conception faible puissance économes en énergie tout au long du flot de USLPAM.

Cet utilitaire USLPAF peut être utilisé en mode autonome ou en conjonction avec

l'utilitaire USLPacom. Pour l'utilisation autonome, les domaines d'alimentation changent d'états quand le PMU appelle des fonctions simples mises en oeuvre dans les classes de l'utilitaire PwARCH. Toutefois, dans le cas de l'utilisation d'utilité USLPacom, ces appels de fonction sont remplacés par des communications transactionnelles pour gestion des domaines d'alimentation passées à travers l'interface PDMgIF, tandis que l'utilitaire PwARCH reste nécessaire pour spécifier l'intention de conception faible consommation.

- **Approche basée sur l'utilisation d'un "wrapper" orienté consommation d'énergie pour l'application de USLPAM sur les types boîte noire de plateformes virtuelles**

Cette thèse définit les principales contraintes de l'application de USLPAM sur des prototypes virtuels type boîte noire. Ces contraintes couvrent principalement la spécification de l'intention de conception faible consommation et la vérification de contrats orientés consommation d'énergie. Face à ces contraintes, interdisant l'utilisation d'instrumentation comme une approche de mise en oeuvre, nous avons proposé une approche alternative pour la mise en oeuvre de « wrappers » orientés consommation d'énergie pour des plateformes virtuelles type boîte noire répondant à toutes les exigences de USLPAM.

Ces "wrappers" ajoutent les capacités de simulation et de vérification orientés consommation d'énergie au-dessus de chaque bloc fonctionnel en boîte noire, tout en permettant une séparation des aspects à l'instar d'UPF. La modularité de cette approche est achevée grâce à l'utilisation de l'utilitaire PAL fournie par la bibliothèque USLPAL. En effet, cet utilitaire permet de personnaliser le comportement requis de chaque couche orientée consommation d'énergie. Comme dans le cas de PwARCH, l'utilitaire PAL peut être utilisé en mode autonome ou en conjonction avec l'utilitaire USLPacom. Comme dans le cas de PwARCH, l'utilitaire PAL peut être utilisé en mode autonome ou en conjonction avec l'utilitaire USLPacom.

7.2 Perspectives

Les directions de recherche futures où le travail présenté dans cette thèse sera utile sont nombreuses. Elles peuvent inclure les points énumérés dans la suite :

7.2.1 L'extension de l'environnement USLPAF avec des sémantiques de simulation supplémentaires orientées consommation d'énergie

- **L'ajout d'extensions de la PDMgIF pour la gestion hiérarchique des domaines d'alimentation à l'utilitaire USLPACom**

La Section 6.3.5 de cette thèse a souligné l'évolutivité de l'interface protocolaire PFMgIF intégrant l'utilitaire USLPACom de l'environnement USLPAF : bien que PDMgIF facilite l'organisation hiérarchique des unités de gestion de domaines d'alimentation dans une structure arborescente et offre une solution plug-and-play pour gérer localement les domaines d'alimentation de type conteneurs, l'étendre afin de mieux gérer les interactions entre les unités de gestion des domaines d'alimentation distribuées et les dépendances entre les domaines d'alimentation appartenant à différents niveaux hiérarchiques serait très probablement nécessaire. Dans ce contexte, de nouvelles et plus profondes études sur la gestion hiérarchique des domaines d'alimentation et les possibilités d'étendre l'utilitaire USLPACom sont nécessaires.

- **Modélisation en SystemC-TLM des domaines d'horloge et de "reset" et des capacités de "clock-gating" et de "DVFS"**

L'environnement USLPAF présenté dans cette thèse utilise principalement le standard UPF de l'industrie en tant que support afin de spécifier et de simuler l'intention en consommation d'énergie au niveau TLM. Ce standard cible en particulier la technique "power gating" comme technique de gestion d'énergie en raison de la complexité impliquée par cette technique au niveau de la conception et de la gestion des interfaces supplémentaires chevauchant les limites des domaines d'alimentation, ainsi que d'états locaux des domaines d'alimentation et les dépendances entre eux. Pour cela, cette thèse se concentre plus spécifiquement sur l'ajout de capacités orientées "power gating" au niveau TLM.

Cependant, le défi de la validation conjointe d'une intention de consommation d'énergie et d'une plateforme virtuelle fonctionnelle devrait également couvrir la modélisation en SystemC-TLM des domaines d'horloge et de "reset" et des capacités en "clock-gating" et "DVFS". Permettre la spécification en TLM de l'intention de consommation d'énergie pour ces techniques supplémentaires de gestion d'énergie et la simulation de leur impact direct sur le comportement fonctionnel de la plateforme transactionnelle aiderait à trouver une solution de gestion d'énergie potentiellement plus économe en énergie que celle trouvée

avec la seule application de la technique "power gating". En fait, l'absence d'un standard, tel que UPF, à utiliser comme support afin de modéliser les sémantiques de simulation et de vérification liées à ces techniques supplémentaires représente une difficulté majeure.

Néanmoins, ces fonctionnalités supplémentaires peuvent être considérées comme des extensions potentielles aux standards de conception de faible consommation existants. Elles devraient donc être compatibles autant que possible aux sémantiques "power gating" définies par ces standards. Une étude des relations structurelles et comportementales entre les architectures de consommation d'énergie et des stratégies de gestion dédiées pour chaque technique de gestion d'énergie (power gating, DVFS, clock gating) pourrait aider à anticiper cette compatibilité.

- **Etude d'un comportement de gestion d'énergie hybride avec une architecture de consommation d'énergie**

Pour fournir un prototype virtuel au niveau transactionnel qui est plus fidèle au système réel final, de nombreux outils industriels de prototypage virtuel mettent à disposition des exemples de plateformes virtuelles avec un système d'exploitation (OS) (comme linux embarqué, *μcos* ou Android). Le système d'exploitation fonctionne en simulation et ordonnance les tâches de l'application embarquée sur les ressources matérielles virtuelles afin d'atteindre des objectifs de performance (tels que les contraintes de temps réel et les économies d'énergie). La plupart de ces systèmes d'exploitation intègrent un gestionnaire d'énergie orienté système d'exploitation (c.à.d. logiciel) qui permet de gérer de manière statique ou dynamique la consommation d'énergie du système grâce à la planification efficace des tâches logicielles.

Par contre, dans cette thèse, nous étions particulièrement intéressés par un gestionnaire d'énergie orienté plutôt contrôleur de consommation (PC), qui se concentre sur le contrôle d'états des domaines d'alimentation en fonction de la charge de travail en cours d'exécution. Donc, un problème sous-jacent qui nécessite d'être soigneusement étudié dans une perspective de cette thèse :

Dans le cas d'une gestion d'énergie hybride, où un gestionnaire d'énergie orienté contrôleur de consommation est en charge d'ordonner les états des domaines d'alimentation pour maximiser les économies d'énergie totales, et un gestionnaire d'énergie orienté système d'exploitation est simultanément en charge d'ordonner les tâches logicielles de l'application embarquée pour le même but, comment corréliser les comportements de ces deux gestionnaires d'énergie tout en évitant les blocages et les conflits entre leurs décisions

et conservant les états fonctionnels et d'énergie du système cohérents ?

7.2.2 Analyse du comportement thermique et de gestion basées sur la simulation orientée consommation d'énergie

L'analyse du comportement thermique dynamique dans des circuits intégrés et embarqués complexes devient d'une importance primordiale parce que des stratégies thermiques raffinées doivent être développées afin d'éviter la dégradation des performances du système si des points de surchauffe apparaissent à l'exécution. A partir de l'architecture de consommation d'énergie, il est possible d'obtenir la séquence d'activation de chaque domaine d'alimentation correspondant à l'exécution abstraite de l'application cible. Ainsi, à ce niveau, une analyse thermique dynamique pourrait être réalisée si un modèle thermique qui reflète l'architecture de consommation d'énergie est développé. Avec la technologie de prototypage virtuel, une trace d'activation des domaines d'alimentation par le gestionnaire d'énergie pourrait être produite. Avec cette trace, l'évolution dynamique de la température du système pourrait être calculée, ce pourrait fournir une entrée à la stratégie de gestion thermique mis en œuvre dans le système.

7.2.3 Automatiser LPDISE

Une démarche d'exploration de l'espace de conception (DSE) qui permet d'automatiser les itérations LPDISE dans la méthodologie USLPAM est absente dans cette thèse et peut être abordée dans les travaux futurs. En se basant sur les contraintes de conception et les propriétés extraites d'une étape d'évaluation de performance, une telle approche DSE est nécessaire pour explorer des alternatives potentielles d'intention de consommation d'énergie. L'exploration doit être faite en terme de décomposition en domaines d'alimentation de l'ensemble du système en fonction des besoins spécifiques des techniques de gestion d'énergie (power gating, AVS, DVFS, horloge et reset) tout en prenant la performance et les coûts de conception et de consommation d'énergie en compte et couvrant un maximum de candidats d'architecture de consommation efficaces en énergie. Dans le cadre de la technique power gating par exemple, une telle approche DSE viserait à trouver un regroupement optimal des blocs matériels d'une puce dans des domaines d'alimentation qui mettent en oeuvre des stratégies efficaces de gestion d'énergie et nécessitent un mini-

mum d'interfaces supplémentaires (cellules d'isolement, registres de maintien et décalage de niveau).

7.2.4 Un ensemble d'outils pour l'identification, la simulation hors ligne et la validation des PMPs

Cette thèse a présenté une méthode formalisée pour la capture des exigences de contrôle et de vérification des domaines d'alimentation en fonction du comportement fonctionnel des composants du système et d'un partitionnement donné en domaines d'alimentation. Ces exigences sont capturées par l'enrichissement des modèles comportementaux EFSM de chaque composant avec des exigences d'états de domaines d'alimentation. Ces modèles enrichis de composants d'un même domaine d'alimentation contribuent à définir les différents PMPs de ce domaine. En fait, ces PMPs représentent les contrats entre le comportement du système fonctionnel et celui de l'architecture de consommation d'énergie lorsque l'on combine les deux comportements. Notez que cette méthode de modélisation est encore manuelle dans cette thèse ce qui la rend source d'erreurs et fastidieuse. Néanmoins, elle peut être améliorée à travers un ensemble d'outils qui mettent en oeuvre :

- Une construction automatique de modèles EFSM des composants fonctionnels à partir de leur description SystemC/TLM.
- Une conversion automatique de ces EFSMs fonctionnels en EFSMs orientées consommation d'énergie, aussi une identification automatique des PMPs.
- Exécution des modèles EFSM des domaines d'alimentation et la validation de la cohérence entre les PMPs de ces différents domaines.

7.2.5 Des études complémentaires sur la vérification orientée consommation d'énergie

Au meilleur de nos connaissances, l'environnement de vérification orienté consommation d'énergie et basé sur l'utilisation d'assertions proposé dans cette thèse est le premier travail de recherche s'intéressant à vérifier la cohérence fonctionnelle/énergie à travers la surveillance au cours d'une simulation TLM d'un ensemble de propriétés prédéfinies orientées consommation d'énergie. L'utilisation d'un raisonnement basé sur des composants afin de spécifier ces propriétés, et de contrats vérifiés en tant qu'assertions pour la

mise en œuvre de la vérification de ces propriétés, constituent les points originaux de notre approche. Néanmoins, un ensemble d'améliorations pourraient encore être apportées à cet environnement de vérification. Dans ce contexte, deux études énumérées ci-après pourraient être effectuées :

- **La génération automatique de moniteurs orientés consommation d'énergie**

Pour les modèles SystemC/TLM de plateformes simples, écrire manuellement le code source des moniteurs orientés consommation d'énergie peut être faisable. Cependant, dans la plupart des plateformes industrielles ayant un grand nombre de composants fonctionnels et de domaines d'alimentation, l'écriture et le débogage manuels des moniteurs seraient très coûteux et source d'erreurs. Par conséquent, l'automatisation de la génération du code des moniteurs à partir de spécifications formelles d'exigences a toujours été d'une importance primordiale pour les industriels. De même pour nos besoins, une bonne idée serait de proposer un mécanisme pratique pour la génération automatique de moniteurs décrits en SystemC/TLM à partir des modèles EFSM orientés consommation d'énergie des différents composants d'une plateforme. Ce mécanisme devrait garantir une couverture maximale des vérifications. En d'autres termes, il devrait assurer que les moniteurs générés détecteraient toutes les exécutions finies du modèle comportemental qui violent les propriétés orientées consommation d'énergie.

- **L'utilisation du standard de spécification des propriétés (PSL) pour une vérification orientée consommation au niveau transactionnel**

La norme IEEE PSL (Property Specification Language) [8] définit des sémantiques puissantes pour les spécifications semi-formelles appliquées à la vérification basée sur des assertions. Dans des travaux récents, certaines couches de ce langage ont été enrichies afin de permettre l'expression d'assertions pour les modèles en SystemC [141] et SystemC/TLM [76] [127]. Cependant, ces efforts n'ont pas encore abordé le problème d'utilisation de PSL pour la vérification des propriétés orientées consommation d'énergie au niveau transactionnel tel que traité dans cette thèse. Ainsi, étendre cette norme afin de spécifier formellement les exigences orientées consommation d'énergie d'un modèle fonctionnel SystemC/TLM et d'utiliser ces spécifications avec des moniteurs d'assertions représente une direction de recherche innovatrice.

7.2.6 Vers une structure standard pour une réutilisation et intégration facile de l'architecture et du contrôle en énergie d'une IP

Certains blocs IP incluent déjà quelques fonctionnalités de gestion d'énergie qui ne sont pas faciles à comprendre ou à capturer par l'utilisateur de l'IP à moins que le fournisseur de cette IP donne un minimum d'information résumant au mieux ces caractéristiques et ce tout en les structurant d'une manière standard. Cela faciliterait considérablement, non seulement l'intégration de cette IP au sein de différents flots d'outils industriels, mais aussi la spécification de l'intention en consommation d'énergie, soit en utilisant un format spécifique (comme UPF et CPF) ou selon un processus bien approprié tel que le flot de la méthodologie USLPAM proposée dans cette thèse.

En particulier, une IP peut être livrée avec son propre contrôleur d'énergie. Des informations bien structurées sur les interfaces internes et externes de ce contrôleur au sein de cette IP, ainsi que les différentes caractéristiques d'énergie de cette IP utilisée par ce contrôleur (comme les différents points fonctionnels de performances (OPPs) de l'IP pour la technique de gestion d'énergie DVFS), faciliterait l'utilisation de l'interface PDM-gIF présenté dans cette thèse afin d'interfacer correctement et rapidement le contrôleur d'énergie de l'IP avec le gestionnaire d'énergie global du système.

L'approche boîte noire orientée consommation d'énergie présentée dans cette thèse a souligné les difficultés d'appliquer la rétention partielle des registres internes et non mappés en mémoire des IPs de type boîte noire durant leur mise hors tension. En général, en plus des informations sur la structure de registres d'une IP boîte noire, des informations sur les registres de IP dont les états doivent être conservés lors d'une mise hors tension est une caractéristique de conception importante qui doit être livrée par le fournisseur de l'IP afin de faciliter à l'utilisateur d'une telle IP son débogage, sa réutilisation et son enrichissement avec d'autres aspects, que ce soit fonctionnels ou non fonctionnels.

Une bonne solution pour faire face à ce manque d'information bien structurée sur les fonctionnalités de gestion d'énergie dans une IP est d'étendre la syntaxe du standard IP-XACT [7]. Ainsi, ces informations supplémentaires et spécifiques au fournisseur de l'IP seraient représentées de façon standard facilitant autant que possible l'adhésion aux flots de conception existants. Évidemment, cette solution nécessite également le dévelop-

pement d'outils adéquats supportant l'accès à cette information spécifique et sa correcte interprétation. Dans un contexte TLM plus précisément, il serait également intéressant de permettre l'utilisation de ces extensions IP-XACT comme un support pour spécifier des alternatives d'intention de consommation d'énergie et les intégrer automatiquement dans les outils de prototypage virtuel TL existants.

7'.2.7 Validation des résultats obtenus à un niveau d'abstraction inférieur au niveau TLM

Une autre direction pour les travaux futurs serait de valider, à des niveaux d'abstraction inférieurs au niveau TLM, les résultats d'économies en énergie obtenus en utilisant notre environnement TLM USLPAF. Dans ce contexte, trois questions fondamentales doivent être abordées :

- Le fichier UPF, qui a été généré dans ce travail en utilisant une approche MDE, est-il sémantiquement et syntaxiquement correcte lorsqu'il est simulé à l'aide d'outils de conception de faible consommation au niveau RTL ? Est-ce que le comportement orienté consommation d'énergie inféré dans la description matérielle fonctionnelle grâce à la spécification UPF, n'a pas vraiment modifié la fonctionnalité RTL du système et reste cohérent avec le fonctionnement des blocs matériels tel qu'il a été garanti au niveau TLM ? Une erreur détectée dans les deux cas pourrait indiquer des défaillances ou lacunes soit dans l'abstraction des sémantiques UPF au niveau TLM, ou dans les règles de transformation utilisées par le processus de génération MDE, ou dans les techniques de modélisation proposées dans cette thèse.
- Est-ce une solution de gestion d'énergie, qui a été élue au niveau TLM en utilisant la méthodologie USLPAM comme la solution la plus économe en énergie, reste ainsi dans le reste du flot de conception de faible consommation ?
- La dernière question génère un autre problème principal et classique : Qu'en est-il de la précision de l'estimation de la consommation d'énergie obtenue à l'aide de modèles basés sur les domaines d'alimentation au niveau TLM proposés dans cette thèse ? Obtenons-nous une petite marge d'erreur lorsque l'on compare les valeurs de consommation d'énergie obtenues au niveau TLM et celles obtenues à des niveaux inférieurs ?

Pour répondre à cette question, il est nécessaire d'avoir des modèles pertinents et suffisamment précis de la consommation d'énergie des IP de la plateforme. Cependant, ces

modèles doivent être davantage liés aux PMPs des IPs considérés au niveau TLM. Ainsi, les études d'estimation au niveau TLM citées dans le chapitre 2 trouvent leurs significations ici. Les relations entre ces techniques d'estimation et la modélisation de l'intention de consommation d'énergie dans cette thèse peuvent aussi être un sujet d'étude.

Les différentes questions posées dans cet ensemble de perspectives ont lancé le projet national français de recherche HOPE¹ (Hierarchically Organized Power/Energy management) financé par l'ANR.

7.3 Publications de l'auteur liées à cette thèse

Ci-dessous la liste des publications sur le travail réalisé par l'auteur dans cette thèse.

- **Revue internationale avec comité de lecture :**

[113] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. Using unified power format standard concepts for power-aware design and verification of systems-on-chip at transaction level. *Circuits, Devices Systems, IET*, 6(5) : 287-296, 2012.

[114] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. PDMgIF : A flexible protocol interface for transaction-level power domain management. *Computers & Digital Techniques, IET*, 2013.

- **Conférences internationales avec actes :**

[111] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. A methodology for power-aware transaction-level models of systems-on-chip using upf standard concepts. In Jose L. Ayala, Braulio Garcia-Camara, Manuel Prieto, Martino Ruggiero, and Gilles Sicard, editors, *PATMOS*, volume 6951 of *Lecture Notes in Computer Science*, pages 226-236. Springer, 2011.

[110] Ons Mbarek, Amani Khecharem, Alain Pegatoquet, and Michel Auguin. Using model driven engineering to reliably accelerate early low power intent exploration for a system-on-chip design. In Sascha Ossowski and Paola Lecca, editors, *SAC*, pages 1580-1587. ACM, 2012.

[112] Ons Mbarek, Alain Pegatoquet, and Michel Auguin. Black-box and white-box early power intent simulation and verification : Two novel approaches. In *DASIP*, pages 1-8. IEEE, 2012.

¹un projet ANR HOPE ayant comme référence ANR 12 INSE 0003 , <http://anr-hope.unice.fr/>

[115] Ons Mbarek, Alain Pegatoquet, Michel Auguin, and Housseem Eddine Fathallah. Power-aware wrappers for transaction-level virtual prototypes : A black box based approach. In VLSI Design, pages 239-244. IEEE, 2013.

Appendix A

Using an MDE Approach for the Enhancement of the USLPAM Simulation-Based Flow

A.1 Automatic Generation of "PowerMain" and UPF Codes Using Our MDE-Based Approach

In the proposed MDE approach for USLPAM enhancement in the Section 6.1.2, the same model used to generate the "PowerMain" code section of the most energy efficient power intent alternative is reused to define another M2T transformation that automatically generates corresponding UPF code. The Figure 6.10 shows that once the most energy-efficient power intent is found, the MDE approach stage is again processed to generate the corresponding UPF code. The different steps of our MDE approach are illustrated by Figure A.1. Each step is performed using a specific tool based on the Eclipse environment. The overall transformation chain as depicted in Figure A.1 is explained in detail in the following.

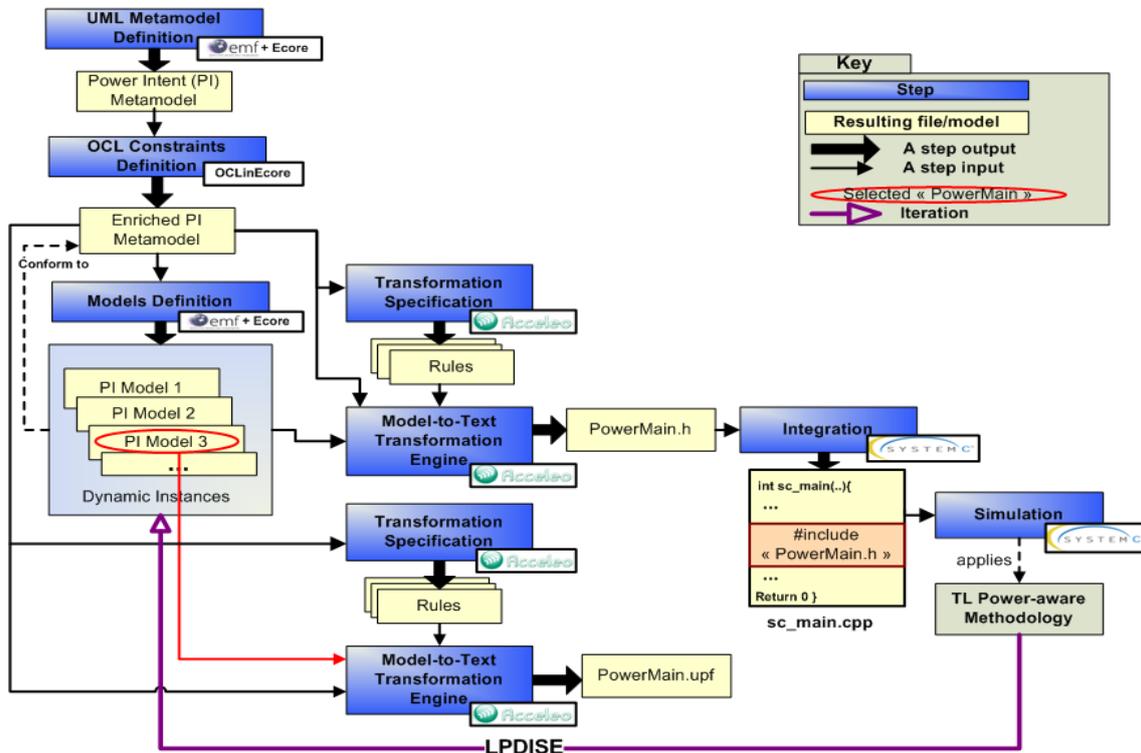


Figure A.1: Generation and Integration process

A.1.1 Automating "PowerMain" Code Generation

The first step in any MDE approach is the definition of metamodels (see section 2.1.2). A metamodel called "Power Intent" (PI) has been elaborated only once using the UML formalism [19] and the graphical editor of the Eclipse Modeling Framework (EMF) [6]. As shown in Figure A.2, the PI metamodel defines the different concepts that can be used in a "PowerMain" code section and naturally figure as PwARCH classes (e.g. power domains, power state tables (PST), power transitions (PSTrans), supply nets, power switches, design elements and observers). Relations between the UPF standard semantics, PwARCH utility and the high-level model used in this work are illustrated by Figure A.3. The UPF standard is naturally used for an RTL-based power specification. To allow a TL-based power specification and evaluation, abstract UPF semantics, as well as structural constraints have been extracted from the UPF standard semantics to be implemented as a part of PwARCH. Among extracted structural constraints, we distinguish between explicit properties which

are directly extracted from the UPF language and standard semantics, and implicit properties which are rather indirectly deduced.

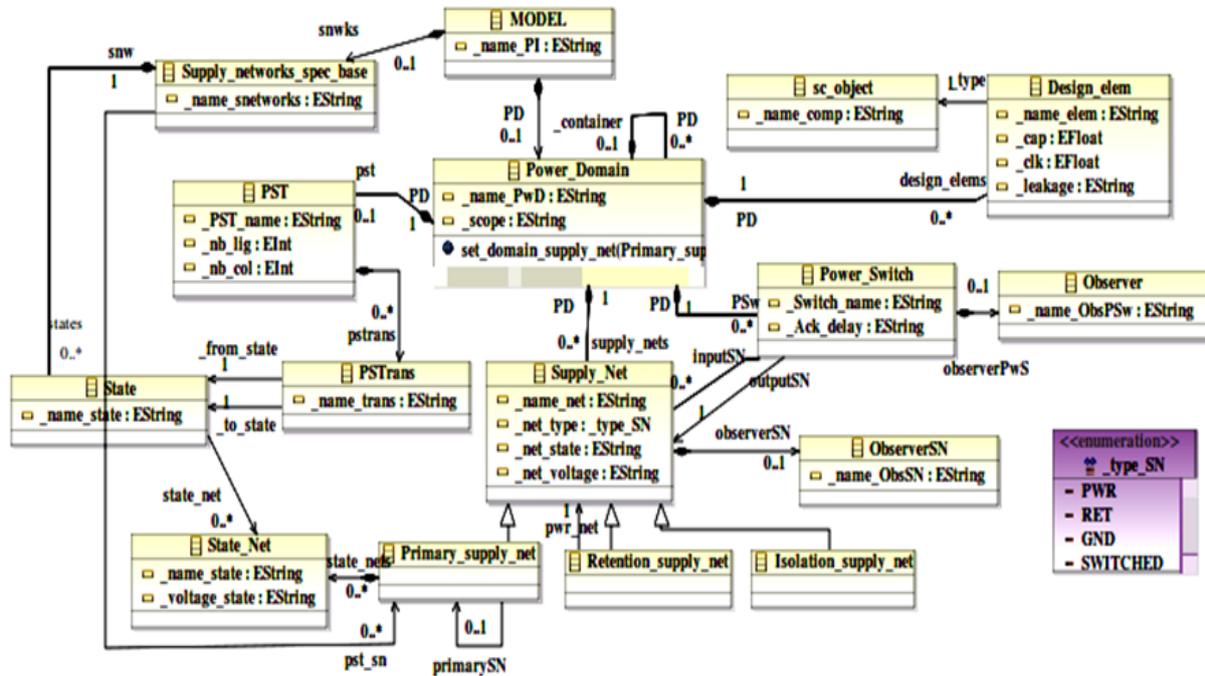


Figure A.2: The Power Intent (PI) Metamodel

Explicit properties mainly consist in the different relationships between UPF concepts. As shown in Figure A.2, this kind of constraints is expressed in the PI metamodel using composition relations and cardinality concepts [19]. For instance, a power domain contains all other UPF power concepts except power transition concept which is rather attached to a PST object. Other relationships were additionally specified. For instance, a relationship is required between a design element and a power domain UPF concept. Note also that the PI metamodel contains only the part of PwARCH classes used to describe a system power intent in a "PowerMain" code. Additionally, only their attributes and methods which are inevitably used in a "PowerMain" code are defined (having always the same semantics as in PwARCH). Implicit properties (Figure A.3) mainly concern structural coherence in a power intent specification. They include simple structural properties (e.g. ensuring that each entered local state in a power state table must be already defined as a valid state of the corresponding supply net). But, they concern as well more sophisticated

ones such as the definition of an illegal combination of power domains' states for a power mode in a power state table: for instance, once specifying an output supply net of a power switch S1 (in PD1) being also an input to a second power switch S2 (in PD2), combining a sleep state for PD1 with a wake-up state for PD2 will be forbidden in any power mode specification inside a power state table.

We have classified implicit properties as class 1 contracts and are fully implemented in PwARCH as types of assertions as previously mentioned. In our MDE-approach, this kind of constraints is considered by enriching the PI metamodel with a set of constraints using the Object Constraint Language (OCL) [14] (Figure A.3). These constraints represent hence conditions and restrictions imposed on some attributes and methods of the PI metamodel classes. As depicts Figure A.1, these constraints are defined once and the resulting enriched PI metamodel is used afterward to structurally elaborate correct models. In order to automate the generation of a "PowerMain" code, the model representing instances of the enriched PI metamodel classes needs to be defined and then transformed to code. This model is simply obtained using EMF dynamic instance creation option [6]. In this way, all structural constraints imposed by implicit and explicit constraints are validated when building a model. Before proceeding to M2T transformation, transformation rules must be

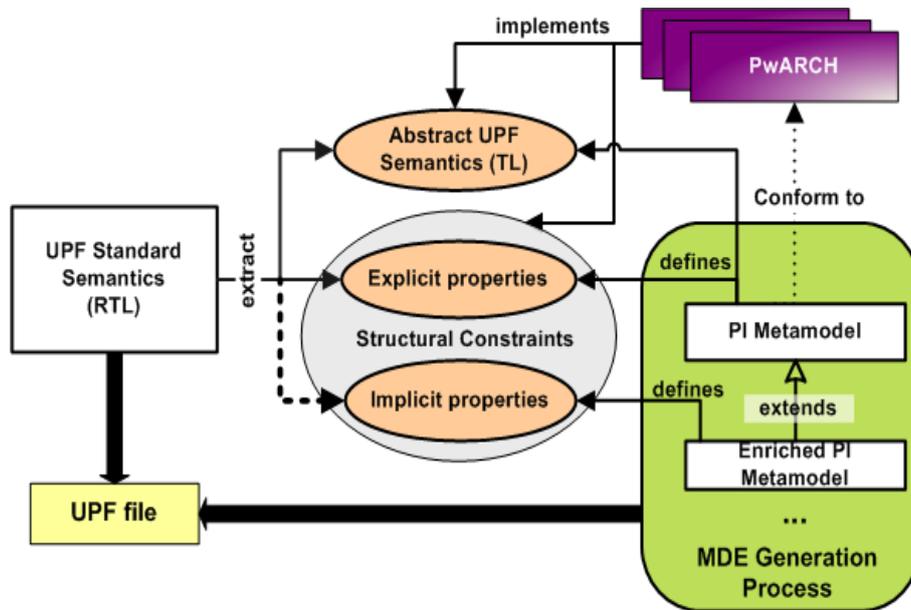


Figure A.3: Relationships Between UPF Standard, PwARCH and PI Metamodel

specified. For that, a template file is written using Acceleo editor tool [1] to configure the generated "PowerMain" code on the previously defined model. Having the enriched PI metamodel as input, the transformation specification is done only once and before generating any "PowerMain" codes (Figure A.1). This demonstrates the generic aspect of a template file. Indeed, to handle the variable number of class instances required in each new "PowerMain" alternative, loop instructions and filters are used to dynamically create the instances and configure their target code. Using Acceleo model-driven code generator, an execution chain created using the enriched PI metamodel, the defined model and the template file as inputs, can be launched to generate the "PowerMain" source code. The generated file is then simply included in the main code of the SystemC hardware platform (Figure A.1). Using this automatic methodology, the power intent specification stage is performed efficiently. In order to evaluate different power intent specification alternatives, new EMF models corresponding to the new intended power intent specification must be specified. As the enriched PI metamodel and transformation rules do not change, exploring different power intent alternatives can be done hence with reduced effort.

A.1.2 Automating UPF Code Generation

Once iterations for LPDISE are finished, the most efficient power intent specification for the target system is selected. Hence, to generate a UPF code corresponding to the selected specification, a new generation chain illustrated by Figure A.1 has been elaborated. For that, the same enriched PI model used to generate the "PowerMain" is reused for a new M2T transformation engine. However, new transformation rules have also to be defined as input to this engine.

As illustrated by Figure A.3, these rules must ensure obtaining from abstract semantics used at Transaction-Level a correct UPF file ready for RTL-based simulation (i.e. as if it is directly defined using the UPF standard file). This is a challenging step in UPF code generation and three cases are handled to perform it. First, a correspondence between abstract UPF concepts in "PowerMain" and UPF commands must be done (case (1)). For instance, a power switch can be created in a "PowerMain" without specifying its control signals. In fact, the Transaction-Level PMU uses function calls instead of RTL signals in order to control a power switch.

However, in a UPF standard specification, control signals must be specified for a power switch.

Furthermore, some UPF commands must be deduced from the abstract semantics in "PowerMain" (case (2)). For instance, in a "PowerMain", only supply nets can be specified to keep a fast simulation. However, supply ports as well as connections between these ports and adequate supply nets are required in a UPF specification and can be merely deduced from the supply nets specifications.

Finally, we believe that UPF protection elements (isolation cells and level shifters) are not so relevant at a Transaction-Level: all signals related to isolation cells are not available at Transaction-Level, and level shifters do not affect the functionality of the design because from a logical perspective they are just buffers (see section 2.1.5). As a consequence, UPF protection elements do not figure in a "PowerMain" code. However, power-aware tools need information about isolation and level shifting strategy so as to automatically infer them where they are required. For that, a UPF code must include such specifications using the UPF standard semantics. In our case, these UPF commands and their related options are deduced from the "PowerMain" code (case (3)): for each switched power domain, an isolation strategy and control is specified. Level shifters placement is predicted from the power state table specification. Recommendations in [96] have been followed to set isolation and level shifting strategies using UPF.

A.2 Performance Enhancement Results

Using our proposed MDE approach, the "PowerMain" codes for the different alternatives of Figure 6.7, as well as the UPF code corresponding to the PI (b) alternative (Figure 6.7), have been automatically generated. Then we compared time and effort investments for both the manual approach (Figure 6.1) and the automated approach (Figure 6.10). Figure A.4 and table A.1 show obtained results, mainly based on the Source Lines Of Code (SLOC) metric to measure the size of codes (using LoCmetrics application), the source development time (by considering the standard typing speed : 33 words per minute), and the time required to process some MDE generation steps.

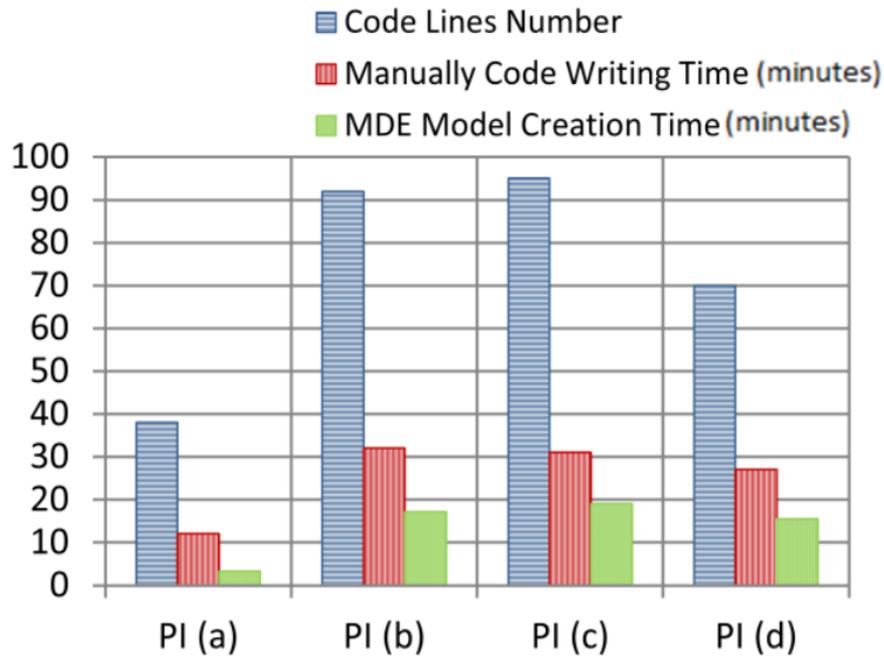


Figure A.4: Comparison of Results Between Manual Writing and Automatic Generation of "PowerMain" Codes

Table A.1 shows required effort for the different steps of our MDE approach. The effort required to define the PI metamodel and the different templates is a factorized effort because done only once, they remain unchanged and are only reused to process the remaining steps of our MDE-based approach. However, at each iteration (for the same or a different case study), the model definition step must be performed again. Of course, the effort required to create a MDE-based PI model depends on the PI to specify. Figure A.4 presents a comparison between time taken to create each MDE-based PI model and that required to manually define each alternative (without considering the time of verification required in both approaches). As it can be seen, up to 50% of time can be saved using a MDE-based approach. It is worth noticing that with the use of a well-defined "PowerMain" template, we were able to generate "PowerMain" codes identical to those manually written.

Furthermore, manually writing a new "PowerMain" code (generally using copy-paste) from a previously one is a tremendously used method which unquestionably accelerates code development time. However, this approach is error-prone and may

		PowerMain	UPF
PI Metamodel Definition Time (minutes)		12	
Code Generator Execution Time (minutes)		1	
Acceleo Template	Code Lines Number	80	110
	DefinitionTime (minutes)	15	20

Table A.1: Required Effort to Perform Generation Process

increase the validation time. Although enriching the PI metamodel with OCL constraints is not a trivial task, this is also done once. Moreover, the time and effort for debugging each PI specification at the simulation-based verification stage are also reduced. Debug at this stage is no more required since the verification of a PI specification is totally shifted to the MDE-based model creation step. At this specific step, a PI model can be created only if all contracts specified as OCL constraints are respected by this model. A designer can hence have a greater confidence in the structural correctness of the generated "PowerMain" codes.

Enhancing USLPAM with an MDE-based approach only accelerates the first stage while verifying structural properties. But, such an enhancement does not alter the benefit of the methodology. Indeed, by using the enhanced methodology flow instead, the PI (b) alternative has been decided as the best solution for the studied SoC as well. For that alternative, a UPF file was automatically generated. In this case, the comparison of the code lines' number between the produced UPF file (271 lines) and the UPF template file (110 lines) shows that the effort is reduced more than twice.

In fact, among 62 generated UPF commands, 24 were inferred using both the abstract UPF semantics of the PI (b) model and the rules specified in the UPF template file. This is automatically performed through MDE-based commands deduction. Here are the inferred commands: supply ports creation, supply nets to supply ports connection, states of supply ports, top-level power domain specification, level shifting and isolation strategies settings. With the use of abstract UPF semantics in the

PI (b) model, specific UPF commands with specific options can be obtained with a simple translation. However, some other UPF options cannot be obtained this way. Here are for instance options for the `create_power_switch` UPF command [30]: on the one side, control and supply ports for power switches are not explicitly defined in the PI metamodel since this latter only defines abstract UPF semantics. On the other side, `on_state` and `off_state` options can be partially deduced from the PI metamodel semantics. In the generated UPF file for PI (b), 15 options of this nature were automatically set for three power switches. The table A.2 gives some lines of code of the PI (b) "PowerMain" section of code, and their corresponding generated UPF commands. The UPF commands and options in the corresponding UPF file that were deduced from the abstract UPF specification are colored in red. Nevertheless, the most important benefit of automating UPF code generation using our MDE approach consists in the high degree of confidence the designer can have in the correctness of the generated UPF file. Indeed, due to implicit and explicit properties added to the PI metamodel, defining a UPF-file is no more error-prone: the generated UPF file is henceforth correct regarding to rules and semantics defined by the UPF language and standard [30]. As a consequence, this reduces significantly the verification and validation cost of a UPF power specification at levels of simulation lower than Transaction-Level.

APPENDIX A. USING AN MDE APPROACH FOR THE ENHANCEMENT OF THE USLPAM SIMULATION-BASED FLOW

PowerMain	UPF
Power_Domain * PD_top_level = new Power_Domain (NULL,"PD_top_level","top");	set_design_top PD_top_level create_power_domain PD_top_level -scope top
primary_supply_net * VDDSoC = new primary_supply_net ("VDDSoC", "power_net", PD_top_level);	create_supply_port VDDSoC create_supply_net VDDSoC -domain PD_top_level connect_supply_net VDDSoC -ports VDDSoC
Power_Switch * SW1 = new Power_Switch (PD_Memory,"sw1"); (SW1-> input_supply_nets).push_back (VDDSoC); SW1-> SetOutput_supply_net (VDDRAM_SW); VDDRAM_SW-> net_valid_states ["RAM_ON"]=0.8; VDDRAM_SW-> Add_State_Input_Net_Value ("RAM_ON", VDDSoC); VDDRAM_SW-> net_valid_states ["RAM_OFF"]=0.0;	create_power_switch SW1 -domain PD_Memory -input_supply_port {VDDSoC VDDSoC} -output_supply_port {VDDRAM_SW VDDRAM_SW} -control_port {Ctrl_in VDDSoC} -on_state {RAM_ON VDDSoC {Ctrl_in}} -off_state {RAM_OFF {! Ctrl_in}}
create_pst ("PST_top",PD_top_level,"5","VDDSoC","VDDCPU","VDDR AM_SW","VDDVGA_SW","VDDGPIO_SW");	create_pst PST_top -supplies {VDDSoC VDDCPU VDDRAM_SW VDDPeriph_SW VDDGPIO_SW}
add_pst_state ("PST_top","initialize","5","ON","ON_H","RAM _ON","VGA_OFF","GPIO_OFF");	add_pst_state initialize -pst PST_top -state {ON ON_H RAM_ON PER_OFF GPIO_OFF}
PSTrans * Tr0 = new PSTrans ("PST_top","initialize","all_on");	describe_state_transition Tr0 -object PD_top_level -from {initialize} -to {all_on}
	set_isolation SW2_iso -domain PD_Periph -isolation_power_net VDDSoC -isolation_ground_net GND -clamp_value {0} -applies_to outputs set_isolation_control SW2_iso -domain PD_Periph -isolation_signal Iso_Periph -isolation_sense high -location automatic
	set_level_shifter schift_up -domain PD_CPU -applies_to outputs -threshold 0 -location fanout -rule both

Table A.2: Analogy Between Some Code Lines of the PI(b) "PowerMain" and the Corresponding UPF Commands

Bibliography

- [1] *Acceleo, MDA Generator Home*. <http://www.acceleo.org/pages/home/en>. 290
- [2] *Accelera: Universal Verification Methodology (UVM) 1.0 EA User's Guide*. Accelera, MAy 2010. <http://www.accellera.org/activities/vip/uvm1.0ea.tar.gz>. 177
- [3] *Aceplorer tool*: <http://www.doceapower.com/product-services/aceplorer.html>. 81, 85
- [4] *Advanced Configuration Power Interface (ACPI) Specification, Revision 4.0a*, April, 2010: <http://www.acpi.info/>. 65
- [5] *Chip Vision Tool, "Orinoco: A High-Level Power Estimation and Optimization Tool Suite"*, <http://www.chipvision.com>. 78
- [6] *Eclipse Modeling Framework (EMF)*. <http://www.eclipse.org/modeling/emf/>. 287, 289
- [7] *IEEE standard for IP-XACT standard structure for packaging, integrating, and reusing IP within tool flow*. *IEEE Std 1683TM*, 2009. 38, 232, 267, 280
- [8] *IEEE Standard for Property Specification Language (PSL), 1850-2010*. 178, 266, 279
- [9] *IEEE Std 1800-2009, IEEE Standard for System Verilog - Unified Hardware Design, Specification, and Verification Language*, December 2009. 178
- [10] *ITU-T website*: <http://www.itu.int/itu-t/recommendations/rec.aspx?id=10651>. 224, 249
- [11] *Low power static checker*. 180
- [12] *Mentor Graphics Questa Simulator*, <http://www.mentor.com/products/fv/questa-power-aware-simulator/>. 85, 180

- [13] *MIPI Alliance Dpecification for System Power Management Interface (SPMI), version 1.00.00, 27 October 2008: <http://www.mipi.org/specifications/system-power-management-interface>. xii, 59, 62, 236, 237*
- [14] *Object Constraint Language (OCL) Specification. <http://www.omg.org/spec/OCL/2.0/>. 289*
- [15] *OMAP35xx Applications Processor, Power, Reset, and Clock Management, Texas Instruments OMAP Family of Products, February 2008, <http://maemo.jacekowski.org/docs/>. xii, 52, 59, 62*
- [16] *OMG, "MOF Query /Views/Transformations", 2005, <http://www.omg.org/cgi-bin/doc?ptc/2005-11-01>. 36*
- [17] *OMG, "Systems Modeling Language (SysML)", Object Management Group, vol. v1.2, Jun. 2010. 83*
- [18] *OMG. Uml profile for schedulability, performance, and time, 2002. 83*
- [19] *OMG Unified Modeling Language (UML), <http://www.uml.org/>. 29, 287, 288*
- [20] *Open Verification Methodlogy Website. <http://www.ovmworld.org>. 177*
- [21] *OVP website, <http://www.ovpworld.org/>. 80*
- [22] *PCI Bus Power Management Interface Specification, Revision 1.2. March, 2004: <http://www.pcisig.com/specifications/conventional/pcipm1.2.pdf>. 66, 237*
- [23] *PCI Express Base Specification, Revision 1.1, PCI-SIG: <http://www.pcisig.com/specifications/pciexpress/base/>. 66, 237*
- [24] *Power System Management Protocol Specifications: <http://pmbus.org/specs.html>. 59*
- [25] *UML profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, 2009. 83, 84*
- [26] *Virtualizer tool, Synopsys Inc., <http://www.synopsys.com>. 82*
- [27] *WEST Team LIFL, Lille, France. Graphical array specification for parallel and distributed computing (GASPARD-2), 2005. <http://www.lifl.fr/west/gaspard/>. 84*
- [28] *Pktool TLM Documentation. PKtool 2.2 Framework extension for transaction level power analysis (related to beta-9 release). 2011. 81*

-
- [29] *S. I. Initiative, Common Power Format (CPF) 2.0 Specification. Silicon Integration Initiative (Si2), Inc., <http://www.si2.org>. 2011.* 5, 19, 68, 182
- [30] Ieee standard for design and verification of low power integrated circuits. *IEEE Std 1801-2009* (27), C1–218. 5, 19, 68, 137, 181, 182, 214, 251, 294
- [31] ABRIL, A., MEHREZ, H., PÉTROT, F., GOBERT, J., AND MIRO, C. Energy estimation and optimization in architectural descriptions of complex embedded systems. In *Proceedings of Microtechnologies for the New Millennium 2005 : VLSI Circuits and Systems* (Sevilla, Espagne, 2005). 79
- [32] ADAM, R., STUART, S., JOHN, P., AND JEAN-MICHEL, F. *Transaction Level Modeling in SystemC*. Open SystemC Initiative, 2005. 30
- [33] AGGARWAL, M., AND BHARTI, R. Asynchronous serial communication protocol (without flow control) using tlm 2.0 example of non memory based protocol. In *GreenSocs initiative* (2009). 49
- [34] AHUJA, S., MATHAIKUTTY, D., LAKSHMINARAYANA, A., AND SHUKLA, S. K. Accurate power estimation of hardware co-processors using system level simulation. In *SOC Conference, 2009. SOCC 2009. IEEE International* (sept. 2009), pp. 399–402. 78
- [35] ALLILAIRE, F., AND IDRISSE, T. Adt: Eclipse development tools for atl. In *Proceedings of the Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA-2)* (2004), Computing Laboratory, University of Kent, Canterbury, Kent CT2 7NF, UK, pp. 171–178. 36
- [36] ARPINEN, T., SALMINEN, E., HÄMÄLÄINEN, T. D., AND HÄNNIKÄINEN, M. Marte profile extension for modeling dynamic power management of embedded systems. *Journal of Systems Architecture: the Euromicro Journal* 58, 5 (Apr. 2012), 209–219. 84
- [37] BAILEY, S., SRIVASTAVA, A., GORRIE, M., AND RUDRA, M. To retain or not to retain: How do i verify the state elements of my low power design? In *Proceedings of DVCon* (2008), pp. 11–17. 86
- [38] BANSAL, N., LAHIRI, K., AND RAGHUNATHAN, A. Automatic power modeling of infrastructure ip for system-on-chip power analysis. In *Proceedings of the 20th International Conference on VLSI Design held jointly with 6th*

- International Conference: Embedded Systems* (Washington, DC, USA, 2007), VLSID '07, IEEE Computer Society, pp. 513–520. [78](#)
- [39] BANSAL, N., LAHIRI, K., RAGHUNATHAN, A., AND CHAKRADHAR, S. T. Power monitors: A framework for system-level power estimation using heterogeneous power models. In *Proceedings of the 18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design* (Washington, DC, USA, 2005), VLSID '05, IEEE Computer Society, pp. 579–585. [78](#)
- [40] BEMBARON, F., KAKKAR, S., MUKHERJEE, R., AND SRIVASTAVA, A. Low power verification methodology using upf. In *Proceedings of DVCon* (2009), pp. 228–233. [86](#)
- [41] BEN ATITALLAH, R., NIAR, S., MEFTALI, S., AND DEKEYSER, J.-L. An mp soc performance estimation framework using transaction level modeling. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (Washington, DC, USA, 2007), RTCSA '07, IEEE Computer Society, pp. 525–533. [80](#)
- [42] BEN ATITALLAH, R., PIEL, É., NIAR, S., MARQUET, P., AND DEKEYSER, J.-L. Multilevel mp soc simulation using an mde approach. In *SoCC* (2007), pp. 197–200. [84](#)
- [43] BENINI, L., BERTOZZI, D., BOGLIOLO, A., MENICHELLI, F., AND OLIVIERI, M. Mparm: Exploring the multi-processor soc design space with systemc. *J. VLSI Signal Process. Syst.* *41*, 2 (Sept. 2005), 169–182. [79](#)
- [44] BENINI, L., BOGLIOLO, A., AND DE MICHELI, G. Readings in hardware/software co-design. Kluwer Academic Publishers, Norwell, MA, USA, 2002, ch. A Survey of Design Techniques for System-Level Dynamic Power Management, pp. 231–248. [xii](#), [58](#), [66](#)
- [45] BENINI, L., BOGLIOLO, A., PALEOLOGO, G. A., AND DE MICHELI, G. Policy optimization for dynamic power management. In *Proceedings of the 35th annual Design Automation Conference* (New York, NY, USA, 1998), DAC '98, ACM, pp. 182–187. [xiii](#), [102](#), [106](#)

- [46] BERGAMASCHI, R. A., AND JIANG, Y. W. State-based power analysis for systems-on-chip. In *Proceedings of the 40th annual Design Automation Conference* (New York, NY, USA, 2003), DAC '03, ACM, pp. 638–641. [77](#)
- [47] BERGERON, J. Advances in low power verification. In *Proceedings of the 13th international symposium on Low power electronics and design* (New York, NY, USA, 2008), ISLPED '08, ACM, pp. 327–328. [58](#)
- [48] BONA, A., ZACCARIA, V., AND ZAFALON, R. System level power modeling and simulation of high-end industrial network-on-chip. In *Proceedings of the conference on Design, automation and test in Europe - Volume 3* (Washington, DC, USA, 2004), DATE '04, IEEE Computer Society, p. 30318. [76](#)
- [49] BONFIETTI, A., BENINI, L., LOMBARDI, M., AND MILANO, M. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010* (2010), pp. 897–902. [118](#)
- [50] BOUHADIBA, T. *42, A Component-Based Approach to Virtual Prototyping of Heterogeneous Embedded Systems*. Thesis report, University of Grenoble, sep 2010. [39](#), [179](#)
- [51] BOUHADIBA, T., MARANINCHI, F., AND FUNCHAL, G. Formal and executable contracts for transaction-level modeling in systemc. In *Proceedings of the seventh ACM international conference on Embedded software* (New York, NY, USA, 2009), EMSOFT '09, ACM, pp. 97–106. [179](#), [191](#)
- [52] BOUHADIBA, T., MOY, M., MARANINCHI, F., CORNET, J., MAILLET-CONTOZ, L., AND MATERIC, I. *Co-Simulation of Functional SystemC TLM Models with Power/Thermal Solvers*. No. TR-2012-21. 2012. [82](#)
- [53] BROOKS, D., TIWARI, V., AND MARTONOSI, M. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th annual international symposium on Computer architecture* (New York, NY, USA, 2000), ISCA '00, ACM, pp. 83–94. [78](#)
- [54] BURTON, M., ALDIS, J., GÜNZEL, R., AND KLINGAUF, W. Transaction level modeling: A reflection on what tlm is and how tlms may be classified. In *FDL* (2007), pp. 92–97. [30](#)

- [55] CAI, L., AND GAJSKI, D. Transaction level modeling: an overview. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (New York, NY, USA, 2003), CODES+ISSS '03, ACM, pp. 19–24. [30](#)
- [56] CALDARI, M., CONTI, M., COPPOLA, M., CRIPPA, P., ORCIONI, S., PIERALISI, L., AND TURCHETTI, C. System-level power analysis methodology applied to the amba ahb bus. In *Proceedings of the conference on Design, Automation and Test in Europe: Designers' Forum - Volume 2* (Washington, DC, USA, 2003), DATE '03, IEEE Computer Society, p. 20032. [76](#), [77](#)
- [57] CALDARI, M., CONTI, M., CRIPPA, P., ORCIONI, S., AND TURCHETTI, C. Design and power analysis in systemc of an i2c bus driver. In *FDL* (2003), pp. 719–727. [76](#)
- [58] CHEN, Y. L. M. Y. Y. *Open Source Analysis and Practice of Embedded System Software: SkyEye and ARM-Based Development Platform*. Beijing Aeronautics and Astronautics University, 2000. [78](#)
- [59] CHOUALI, S., AND HAMMAD, A. Formal verification of components assembly based on sysml and interface automata. *Innov. Syst. Softw. Eng.* 7, 4 (Dec. 2011), 265–274. [178](#)
- [60] CLARKE, JR., E. M., GRUMBERG, O., AND PELED, D. A. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999. [176](#)
- [61] CONTI, M., VECE, G. B., AND COLAZILLI, S. Extension of systemc framework towards power analysis. In *Specification Design Languages, 2009. FDL 2009*. (sept. 2009), pp. 1–4. [81](#)
- [62] CORNET, J. *Separation of Functional and Non-Functional Aspects in Transactional Level Models of Systems-on-Chip*. Thesis report, INP Grenoble, 2008. [39](#), [111](#), [112](#), [119](#), [139](#), [191](#)
- [63] CORNET, J., MAILLET-CONTOZ, L., MATERIC, I., KAISER, S., BOUSSETTA, H., , T., MOY, M., AND MARANINCHI, F. *Co-Simulation of a SystemC TLM Virtual Platform with a Power Simulator at the Architectural Level: Case of a Set-Top Box*. Design Automation Conference (DAC), San Francisco, USA, Jun 2012, p. SESSION 10U: USER TRACK. [82](#)

- [64] CROFT, M., AND BAILEY, S. Is your low power design switched on? In *System-on-Chip, 2007 International Symposium on* (Nov.), pp. 1–4. [86](#)
- [65] CRONE, A., AND CHIDOLUE, G. Functional verification of low power designs at rtl. In *Proceedings of the 17th international conference on Integrated Circuit and System Design: power and timing modeling, optimization and simulation* (Berlin, Heidelberg, 2007), PATMOS'07, Springer-Verlag, pp. 288–299. [86](#)
- [66] CZARNECKI, K., AND HELSEN, S. Feature-based survey of model transformation approaches. *IBM Systems Journal* 45, 3 (July 2006), 621–645. [35](#)
- [67] DAMM, M., MORENO, J., HAASE, J., AND GRIMM, C. Using transaction level modeling techniques for wireless sensor network simulation. In *Proceedings of the Conference on Design, Automation and Test in Europe* (3001 Leuven, Belgium, Belgium, 2010), DATE '10, European Design and Automation Association, pp. 1047–1052. [49](#)
- [68] DANIEL D., G., JIANWEN, Z., RAINER, D., ANDREAS, G., AND SHUQING, Z. Specc: Specification language and methodology. In *Kluwer Academic* (Boston, 2000). [30](#)
- [69] DELP, G., MARSCHNER, E., AND BAKALAR, K. Understanding the low power abstraction. In *Proceedings of DVCon* (2010), pp. 204–210. [87](#)
- [70] DHANWADA, N., LIN, I.-C., AND NARAYANAN, V. A power estimation methodology for systemc transaction level models. In *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software code-design and system synthesis* (New York, NY, USA, 2005), CODES+ISSS '05, ACM, pp. 142–147. [79](#)
- [71] DHANWADA, N. R., BERGAMASCHI, R. A., DUNGAN, W. W., NAIR, I., GRAMANN, P., DOUGHERTY, W. E., AND LIN, I.-C. Transaction-level modeling for architectural and power analysis of powerpc and coreconnect-based systems. *Design Autom. for Emb. Sys.* 10, 2-3 (2005), 105–125. [79](#)
- [72] DHOUIB, S., SENN, E., DIGUET, J.-P., BLOUIN, D., AND LAURENT, J. Energy and power consumption estimation for embedded applications and operating systems. *Journal of Low Power Electronics* 5, 4 (2009), 416–428. [76](#)
- [73] DHOUIB, S., SENN, É., DIGUET, J.-P., LAURENT, J., AND BLOUIN, D. Model driven high-level power estimation of embedded operating systems com-

- munication services. In *Proceedings of the 2009 International Conference on Embedded Software and Systems* (Washington, DC, USA, 2009), ICESS '09, IEEE Computer Society, pp. 475–481. [76](#), [83](#)
- [74] DONLIN, A. Transaction level modeling: Flows and use models. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (New York, NY, USA, 2004), CODES+ISSS '04, ACM, pp. 75–80. [30](#)
- [75] EDWARDS, S., LAVAGNO, L., LEE, E. A., AND SANGIOVANNI-VINCENTELLI, A. Readings in hardware/software co-design. Kluwer Academic Publishers, Norwell, MA, USA, 2002, ch. Design of embedded systems: formal models, validation, and synthesis, pp. 86–107. [178](#)
- [76] FERRO, L., AND PIERRE, L. Formal semantics for psl modeling layer and application to the verification of transactional models. In *Proceedings of the Conference on Design, Automation and Test in Europe* (3001 Leuven, Belgium, Belgium, 2010), DATE '10, European Design and Automation Association, pp. 1207–1212. [266](#), [279](#)
- [77] FUMMI, F., MARTINI, S., PERBELLINI, G., AND PONCINO, M. Native iss-systemc integration for the co-simulation of multi-processor soc. In *Proceedings of the conference on Design, automation and test in Europe - Volume 1* (Washington, DC, USA, 2004), DATE '04, IEEE Computer Society, pp. 564–569 Vol.1. [30](#)
- [78] GABRIEL, C., AND B., R. Upping verification productivity of low power designs. In *Proceedings of DVCon* (2008), pp. 3–10. [86](#)
- [79] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. [222](#)
- [80] GEIB, J.-M., GRANSART, C., AND MERLE, P. Corba, des concepts à la pratique. *Masson ed., Paris* (1998). [94](#)
- [81] GHENASSIA, F. *Transaction-Level Modeling with Systemc: Tlm Concepts and Applications for Embedded Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. [30](#)

-
- [82] GIAMMARINI, M., CONTI, M., AND ORCIONI, S. System-level energy estimation with powersim. In *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on* (dec. 2011), pp. 723–726. [81](#)
- [83] GIVARGIS, T. D., VAHID, F., AND HENKEL, J. Instruction-based system-level power evaluation of system-on-a-chip peripheral cores. In *Proceedings of the 13th international symposium on System synthesis* (Washington, DC, USA, 2000), ISSS '00, IEEE Computer Society, pp. 163–169. [76](#)
- [84] GLOUCHE, Y., LE GUERNIC, P., TALPIN, J.-P., AND GAUTIER, T. A boolean algebra of contracts for assume-guarantee reasoning. *Electron. Notes Theor. Comput. Sci.* 263 (June 2010), 111–127. [179](#)
- [85] GOMEZ, C., DEANTONI, J., AND MALLET, F. Multi-View Power Modeling based on UML, MARTE and SysML. In *SEAA - 38th Euromicro Conference on Software Engineering and Advanced Applications* (Cesme, Turquie, Oct. 2012), pp. 17–20. RR-7934 RR-7934. [84](#), [85](#)
- [86] GROSSE, P., DURAND, Y., AND FEAUTRIER, P. Power modeling of a noc based design for high speed telecommunication systems. In *Proceedings of the 16th international conference on Integrated Circuit and System Design: power and Timing Modeling, Optimization and Simulation* (Berlin, Heidelberg, 2006), PATMOS'06, Springer-Verlag, pp. 157–168. [77](#)
- [87] GROTKER, T. *System Design with SystemC*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. [30](#)
- [88] GUNZEL, R. GreenSocs Inc. (<http://www.greensocs.com/>). [49](#)
- [89] HAGNER, M., ANICULAESEI, A., AND GOLTZ, U. Uml-based analysis of power consumption for real-time embedded systems. In *Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications* (Washington, DC, USA, 2011), TRUST-COM '11, IEEE Computer Society, pp. 1196–1201. [84](#)
- [90] HAZRA, A., MITRA, S., DASGUPTA, P., PAL, A., BAGCHI, D., AND GUHA, K. Leveraging upf-extracted assertions for modeling and formal verification of architectural power intent. In *Proceedings of the 47th Design Automation Conference* (New York, NY, USA, 2010), DAC '10, ACM, pp. 773–776. [87](#)

- [91] HELMSTETTER, C., AND PONSINI, O. A comparison of two systemc/tlm semantics for formal verification. In *MEMOCODE* (2008), IEEE Computer Society, pp. 59–68. [191](#)
- [92] INITIATIVE. SYSTEMC MODELING LANGUAGE: [HTTP://WWW.SYSTEMC.ORG.](http://www.systemc.org), O. S. [30](#), [38](#)
- [93] JADCHERLA, S., BERGERON, J., AND INOUE, Y. Verification methodology manual for low power (vmm-lp). Synopsys, p. 226. [87](#)
- [94] JULIEN, N., LAURENT, J., SENN, E., AND MARTIN, E. Power estimation of a c algorithm based on the functional-level power analysis of a digital signal processor. In *Proceedings of the 4th International Symposium on High Performance Computing* (London, UK, UK, 2002), ISHPC '02, Springer-Verlag, pp. 354–360. [76](#)
- [95] KALNINS, A., BARZDINS, J., AND CELMS, E. Model transformation language mola. In *Proceedings of MDFAFA 2004 (Model-Driven Architecture: Foundations and Applications 2004)* (2004), pp. 14–28. [36](#)
- [96] KEATING, M., FLYNN, D., AITKEN, R., GIBBONS, A., AND SHI, K. *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007. [52](#), [86](#), [120](#), [160](#), [172](#), [216](#), [291](#)
- [97] KOPETZ, H. Component-based design of large distributed real-time systems. In *Journal of IFAC, Pergamon Press* (1997), pp. 53–60. [178](#)
- [98] KUEHNLE, M., WAGNER, A., AND BECKER, J. A statistical power estimation methodology embedded in a systemc code translator. In *Proceedings of the 24th symposium on Integrated circuits and systems design* (New York, NY, USA, 2011), SBCCI '11, ACM, pp. 79–84. [77](#)
- [99] LAFAYE, M., PAUTET, L., BORDE, E., GATTI, M., AND FAURA, D. Model driven resource usage simulation for critical embedded systems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012* (March), pp. 312–315. [32](#)
- [100] LANG, L. *Hierarchical Methods for Power Intent Specification*. EETimes Design Article, 2012. [73](#)

- [101] LAURENT, J., JULIEN, N., SENN, E., AND MARTIN, E. Functional level power analysis: An efficient approach for modeling the power consumption of complex processors. In *DATE* (2004), pp. 666–667. [76](#)
- [102] LAURENT, J., SENN, E., JULIEN, N., AND MARTIN, E. High-level energy estimation for dsp systems. In *PATMOS' 01* (2001), IEEE, pp. pp 311–316. 10 pages. [76](#)
- [103] LE TALLEC, J.-F. *Extraction de Modèles pour La Conception de Systèmes sur Puce*. Thesis report, University of Nice Sophia-Antipolis, 2012. [32](#)
- [104] LEBRETON, H., AND VIVET, P. Power modeling in systemc at transaction level, application to a dvfs architecture. In *Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI* (Washington, DC, USA, 2008), ISVLSI '08, IEEE Computer Society, pp. 463–466. [80](#)
- [105] LEE, I., KIM, H., YANG, P., YOO, S., CHUNG, E.-Y., CHOI, K.-M., KONG, J.-T., AND EO, S.-K. Powervip: Soc power estimation framework at transaction level. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference* (Piscataway, NJ, USA, 2006), ASP-DAC '06, IEEE Press, pp. 551–558. [79](#)
- [106] LI, S.-C., LIAO, W.-T., LEE, M.-S., HSIEH, W.-T., AND LIU, C.-N. A practical power model of amba system for high-level power analysis. In *VLSI Design, Automation and Test, 2009. VLSI-DAT '09. International Symposium on* (april 2009), pp. 347–350. [76](#)
- [107] LU, Y.-H., AND DE MICHELI, G. Comparing system-level power management policies. *IEEE Design Test* 18, 2 (Mar. 2001), 10–19. [106](#)
- [108] MARANINCHI, F., AND MOREL, L. Logical-time contracts for reactive embedded components. In *Proceedings of the 30th EUROMICRO Conference* (Washington, DC, USA, 2004), EUROMICRO '04, IEEE Computer Society, pp. 48–55. [179](#)
- [109] MARKUS, W., AND SAM, T. *Accelerating the Development of TLM-2.0 Models Using Model Authoring Kits (MAKs)*. Synopsys Inc. [81](#)
- [110] MBAREK, O., KHECHAREM, A., PEGATOQUET, A., AND AUGUIN, M. Using model driven engineering to reliably accelerate early low power intent explo-

- ration for a system-on-chip design. In *SAC* (2012), S. Ossowski and P. Lecca, Eds., ACM, pp. 1580–1587. [210](#), [269](#), [282](#)
- [111] MBAREK, O., PEGATOQUET, A., AND AUGUIN, M. A methodology for power-aware transaction-level models of systems-on-chip using upf standard concepts. In *PATMOS* (2011), J. L. Ayala, B. García-Cámara, M. Prieto, M. Ruggiero, and G. Sicard, Eds., vol. 6951 of *Lecture Notes in Computer Science*, Springer, pp. 226–236. [194](#), [269](#), [282](#)
- [112] MBAREK, O., PEGATOQUET, A., AND AUGUIN, M. Black-box and white-box early power intent simulation and verification: Two novel approaches. In *DASIP* (2012), IEEE, pp. 1–8. [229](#), [269](#), [282](#)
- [113] MBAREK, O., PEGATOQUET, A., AND AUGUIN, M. Using unified power format standard concepts for power-aware design and verification of systems-on-chip at transaction level. *Circuits, Devices Systems, IET* 6, 5 (2012), 287–296. [194](#), [269](#), [282](#)
- [114] MBAREK, O., PEGATOQUET, A., AND AUGUIN, M. Power domain management interface: Flexible protocol interface for transaction-level power domain management. *Computers Digital Techniques, IET* (2013). [235](#), [269](#), [282](#)
- [115] MBAREK, O., PEGATOQUET, A., AUGUIN, M., AND FATHALLAH, H. E. Power-aware wrappers for transaction-level virtual prototypes: A black box based approach. In *VLSI Design* (2013), IEEE, pp. 239–244. [223](#), [269](#), [283](#)
- [116] MCMILLAN, K. L. *Symbolic Model Checking: an Approach to the State Explosion Problem*. PhD thesis, Pittsburgh, PA, USA, 1992. UMI Order No. GAX92-24209. [176](#)
- [117] MEYER, B. *Object-Oriented Software Construction*, 1st ed. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988. [178](#)
- [118] MEYER, B. Applying "design by contract". *Computer* 25, 10 (Oct. 1992), 40–51. [97](#), [150](#), [178](#)
- [119] MEYER, B. *Eiffel: the language*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992. [97](#)
- [120] NEFFE, U., ROTHBART, K., STEGER, C., WEISS, R., RIEGER, E., AND MÜHLBERGER, A. Energy estimation based on hierarchical bus models for

- power-aware smart cards. In *Proceedings of the conference on Design, automation and test in Europe - Volume 3* (Washington, DC, USA, 2004), DATE '04, IEEE Computer Society, p. 30300. 76
- [121] NIEMANN, B., AND HAUBELT, C. Formalizing tlm with communicating state machines. In *FDL* (2006), ECSI, pp. 285–293. 191
- [122] OLIVEIRA, M. F. D. S., BRIÃO, E. W., NASCIMENTO, F. A., AND WAGNER, F. R. Model driven engineering for mpsoC design space exploration. In *Proceedings of the 20th annual conference on Integrated circuits and systems design* (New York, NY, USA, 2007), SBCCI '07, ACM, pp. 81–86. 83
- [123] OLIVEIRA, M. F. D. S., DE BRISOLARA, L. B., CARRO, L., AND WAGNER, F. R. Early embedded software design space exploration using uml-based estimation. In *Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping* (Washington, DC, USA, 2006), RSP '06, IEEE Computer Society, pp. 24–32. 83
- [124] OPEN SYSTEMC INITIATIVE. SYSTEMC TRANSACTION LEVEL MODELING LIBRARY 2.1.0, . <http://www.systemc.org>. xi, 7, 20, 42, 45, 94, 100, 206, 220, 223, 240, 241, 245
- [125] PEDRAM, M. *Power Aware Design Methodologies*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. 52, 102
- [126] PETER H., F., BRUCE, L., AND STEVE, V. An overview of the sae architecture analysis design language (aadl) standard: A basis for model-based architecture-driven embedded systems engineering. In *volume 176/2005 of IFIP International Federation for Information Processing* (Springer Boston, 2005), pp. 3–5. 29, 83
- [127] PIERRE, L., FERRO, L., AMOR, Z. B. H., BOURGON, P., AND QUÉVRE-MONT, J. Integrating psl properties into systemc transactional modeling - application to the verification of a modem soc. In *SIES* (2012), IEEE, pp. 220–228. 266, 279
- [128] QU, G., KAWABE, N., USAMI, K., AND POTKONJAK, M. Function-level power estimation methodology for microprocessors. In *Proceedings of the 37th Annual Design Automation Conference* (New York, NY, USA, 2000), DAC '00, ACM, pp. 810–813. 77

- [129] RETHINAGIRI, S. K., BEN ATITALLAH, R., DEKEYSER, J.-L., SENN, E., AND NIAR, S. An efficient power estimation methodology for complex risc processor-based platforms. In *Proceedings of the great lakes symposium on VLSI* (New York, NY, USA, 2012), GLSVLSI '12, ACM, pp. 239–244. [80](#)
- [130] RUDRA, M., AMIT, S., AND STEPHEN, B. Static and formal verification of power aware designs at the rtl using upf. In *Proceedings of DVCon* (2008), pp. 47–42. [86](#)
- [131] SENDALL, S., AND KOZACZYNSKI, W. Model transformation: the heart and soul of model driven software development. 42–45. [34](#)
- [132] SENN, É., LAURENT, J., JUIN, É., AND DIGUET, J.-P. Refining power consumption estimations in the component based aadl design flow. In *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on* (sept. 2008), pp. 173–178. [83](#)
- [133] SHEETS, M., BURGHARDT, F., KARALAR, T., AMMER, J., CHEE, Y. H., RABAEY, J., AND FUNCTIONALITY, A. A power-managed protocol processor for wireless sensor networks. In *in Proc. IEEE Symp. VLSI Circuits* (2006), pp. 262–263. [59](#), [237](#)
- [134] SHEETS, M. A. Standby power management architecture for deep-submicron systems. Thesis report, 2006. [59](#), [237](#)
- [135] SINHA, A., AND CHANDRAKASAN, A. P. Jouletrack: a web based tool for software energy profiling. In *Proceedings of the 38th annual Design Automation Conference* (New York, NY, USA, 2001), DAC '01, ACM, pp. 220–225. [75](#), [76](#)
- [136] SPINCZYK, O., GAL, A., AND SCHRÖDER-PREIKSCHAT, W. Aspectc++: an aspect-oriented extension to the c++ programming language. In *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet, mobile and embedded applications* (Darlinghurst, Australia, Australia, 2002), CRPIT '02, Australian Computer Society, Inc., pp. 53–60. [191](#)
- [137] SRIKANTH, J., JANICK, B., YOSHIO, I., AND FLYNN, D. *Verification Methodology Manual for Low Power*. Synopsys, 2009. [xiv](#), [180](#), [181](#), [188](#)
- [138] STEPHEN, B., GABRIEL, C., AND ALLAN, C. Low power design and verification techniques. In *Mentor Graphics, White Paper* (2007). [xii](#), [54](#)

- [139] STEVENS, P. Generative and transformational techniques in software engineering ii. Springer-Verlag, Berlin, Heidelberg, 2008, ch. A Landscape of Bidirectional Model Transformations, pp. 408–424. [35](#)
- [140] SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. [93](#), [98](#)
- [141] TABAKOV, D., KAMHI, G., VARDI, M. Y., AND SINGERMAN, E. A temporal language for systemc. In *FMCAD* (Portland, Oregon, USA, 2008), A. Cimatti and R. B. Jones, Eds., IEEE, pp. 1–9. [266](#), [279](#)
- [142] TIWARI, V., MALIK, S., AND WOLFE, A. Power analysis of embedded software: a first step towards software power minimization. *IEEE Trans. Very Large Scale Integr. Syst.* 2, 4 (Dec. 1994), 437–445. [75](#)
- [143] TRABELSI, C., BEN ATITALLAH, R., MEFTALI, S., DEKEYSER, J.-L., AND JEMAI, A. A model-driven approach for hybrid power estimation in embedded systems design. *EURASIP Journal on Embedded Systems 2011* (2011). [xi](#), [35](#), [84](#)
- [144] TRUMMER, C., KIRCHSTEIGER, C. M., STEGER, C., WEISS, R., DALTON, D., AND PISTAUER, M. Simulation-based verification of power aware system-on-chip designs using upf iee 1801. In *NORCHIP, 2009* (Nov.), pp. 1–4. [86](#)
- [145] VAN MOLL, H. W. M., CORPORAAL, H., REYES, V., AND BOONEN, M. Fast and accurate protocol specific bus modeling using tlm 2.0. In *Proceedings of the Conference on Design, Automation and Test in Europe* (3001 Leuven, Belgium, Belgium, 2009), DATE '09, European Design and Automation Association, pp. 316–319. [49](#)
- [146] VARANASI, A. *Course Grained Low Power Design Flow Using UPF*. Thesis report, Rochester Institute of Technology, Rochester, NY, August 2009. [85](#)
- [147] VECE, G. B., AND CONTI, M. Power estimation in embedded systems within a systemc-based design context: The pktool environment. In *Intelligent solutions in Embedded Systems, 2009 Seventh Workshop on* (june 2009), pp. 179–184. [81](#)

BIBLIOGRAPHY

- [148] WANG, Q. *The Evolution of Power Format Standards: A Cadence Viewpoint*. 2011. [xii](#), [72](#)
- [149] YE, W., VIJAYKRISHNAN, N., KANDEMIR, M., AND IRWIN, M. J. The design and use of simplepower: a cycle-accurate energy estimation tool. In *Proceedings of the 37th Annual Design Automation Conference* (New York, NY, USA, 2000), DAC '00, ACM, pp. 340–345. [78](#)
- [150] YOSSI, V., AND SHABTAY, M. *Why You Should Optimize Power at the Electronic System Level*. Mentor Graphics Datasheets. [81](#)
- [151] ZIEMANN, P., AND GOGOLLA, M. An extension of ocl with temporal logic. In *Critical Systems Development with UML* (2002), pp. 53–62. [179](#)