Using Model Driven Engineering to Reliably Accelerate Early Low Power Intent Exploration for a System-on-Chip Design

Ons Mbarek
LEAT, University of Nice Sophia Antipolis-CNRS
250-Rue Albert Einstein
Bâtiment-4, 06560
Valbonne, France
mbarek@unice.fr

Amani Khecharem
LEAT, University of Nice Sophia Antipolis-CNRS
250-Rue Albert Einstein
Bâtiment-4, 06560
Valbonne, France
khecharem@etu.unice.fr

Alain Pegatoquet
LEAT, University of Nice Sophia Antipolis-CNRS
250-Rue Albert Einstein
Bâtiment-4, 06560
Valbonne, France
pegatoquet@unice.fr

Michel Auguin
LEAT, University of Nice Sophia Antipolis-CNRS
250-Rue Albert Einstein
Bâtiment-4, 06560
Valbonne, France
auguin@unice.fr

ABSTRACT
Defining low power design intent for a system-on-chip (SoC) consists in specifying its power management architecture and strategy according to specific low power techniques such as power gating and multi-voltage scaling requirements. Choosing the most energy-efficient power intent for a final system contributes widely to reduce its overall power consumption. At Transaction-Level, a rapid exploration of different power intent alternatives can be made. In this paper, we present a Model Driven Engineering (MDE) approach to automate low power design intent specifications and accelerate Low Power Design Intent Space Exploration (LPDISE) using a Transaction-Level power-aware design methodology. This MDE approach mainly relies on high level abstraction of the Unified Power Format (UPF) standard concepts that fit a TLM approach use. Then, the MDE approach is applied to automatically generate a UPF code defining the most energy-efficient power intent and being a reference file for Register Transfer Level (RTL) design team. This task focuses on a smart deduction of adequate UPF commands from the high level abstraction semantics. The effectiveness of the proposed MDE approach is illustrated by an example.

Categories and Subject Descriptors
B.2.2 [Performance Analysis and Design Aids]: Simulation, Verification; D.2.1 [Requirements/Specifications]: D.2.m [Rapid Prototyping]; K.1 [The Computer Industry]: Standards

General Terms
Design, Performance, Verification, Languages

Keywords
Model Driven Engineering (MDE), Low Power Design Intent Space Exploration (LPDISE), IEEE 1801 (UPF) standard, low-power design, Transaction-Level of Modeling (TLM)

1. INTRODUCTION
Power gating and multi-voltage strategies are commonly used in today’s systems-on-chip to optimize their power consumption. To implement these low power strategies, a low power architecture design including multi-power domain partitioning and low power interfaces (e.g. isolation, retention and level shifting cells) is required. A Power Domain (PD) [20][9] represents a partition of functional blocks (IPs) that needs individual power control. Within each partition, each IP must share the same primary supply nets. Hence, a challenging task consists in finding the best power domain partitioning while respecting functionalities required by the embedded application. Using a Low Power Design Intent Space Exploration (LPDISE) approach helps to systematically achieve this task. LPDISE consists in exploring different power intent specifications of a SoC according to specific requirements of low power techniques. In this context, LPDISE mainly aims 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).

At the Electronic System Level (ESL), a large and rapid LPDISE can be performed toward early power management optimization and what-if power analysis. In [10], we proposed a power domain based methodology targeting this kind of power exploration at Transaction-Level [11]. The methodology uses a well-structured manner to iteratively instrument existing Transaction-Level models of SoCs with different power intent specification alternatives and evaluate the power savings of each during simulation. However, manually designing complex power intent can represent a real burden for power exploration using this proposed methodology. As a consequence, an automatic generation of each power intent specification alternative is of prime importance to facilitate and speed up LPDISE, hence to respect time-to-market constraints.

In this paper, we focus on the use of a Model Driven Engineering (MDE) [15] approach to first achieve this automation objective. Generally, the main concept in MDE is model transformations (MTs) [16]. By changing transformation rules from the same high-level model, different target implementations can be obtained.

(c) 2011 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of the national government of France. As such, the government of France retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
SAC 2012, March 26 - 30 2012, Riva (Trento), Italy
Copyright 2011 ACM 978-1-4503-0857-1/12/03 ...$10.00

This work is supported by the French National Research Agency (ANR) project HELP bearing reference ANR-09-SEGI-006
This MDE feature has been used to automatically generate a UPF standard [20] reference code describing the most energy-efficient power intent design once finishing LPDISE. As the power intent specifications generated from a high level model will then be used in a TLM context, only abstract UPF concepts need to be represented in this high level model [10]. In this case, generating a UPF code from this model requires defining transformation rules between UPF standard commands and the existing abstract UPF concepts of this model. For missing UPF commands in this model, they will have to be deduced from the existing UPF abstract concepts.

This paper is organized as follows: in the next section, our contributions with regard to the state of the art are highlighted. In section 2, main features of our proposed power domain based methodology are briefly described. Section 3 details the MDE-based approach used to automate power intent specifications and generate UPF files. In section 4, the effectiveness of the approach is analyzed by integrating it to the methodology flow and applying this new flow on a case study. Section 5 concludes this article and gives perspective works.

2. RELATED WORK

Several works have used MDE to cover different power related issues in embedded systems. Among these issues, early power estimation was widely addressed. In [4] and [5], UML diagrams and UML-SPT [13] profile 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 estimates. Aiming at selecting the most adequate modeling solution for application and architecture fulfilling best energy, cycles and memory requirements, a Design Space Exploration (DSE) tool was used to automatically explore different solutions. Authors in [17] and [7] 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) [18] 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 [3], authors present a MDE-based methodology to automatically generate MPSoC system model descriptions at different simulation levels. The Gaspard [22] MDE environment based on the Modeling and Analysis for Real Time and Embedded systems (MARTE) standard [21] was used in this work. The same MDE-based methodology was adopted in [19] 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. Our work is close to these two approaches since MDE is rather mainly used to automate code generation than to automate Design Space Exploration (DSE). Furthermore, the generated code in [3] and [19] is used to estimate power consumption during simulation. This is not the case in the MDE-based approach proposed in this paper. Indeed, codes generated using our approach are not only used for simulation-based power estimations and are rather connected to a more global simulation flow presented in Figure 1 and detailed in the next section. Contrary to all mentioned works, our MDE-based approach addresses also power management issues by modeling power management architecture of a system at a high-level of abstraction. A recent work [1] aims at modeling Dynamic Power Management (DPM) aspects in embedded systems by proposing an extended DPM MARTE profile. Unlike [3] and [19], the MARTE allocation profile is used in [1] to associate application functionalities with system power modes and not hardware components. This work is still at a conceptual level and needs a connection to simulation level for energy dissipation analysis. In our work, modeling power management requirements is done by not only describing system power modes like presented in [1], but also the power architecture defining such modes. This is performed without even using MARTE allocation profile. Indeed, the system application and hardware architecture do not need to be modeled in our case: in fact, the evaluation of different power management architectures and strategies are done during simulation. In addition, there is no need to regenerate the platform code since our approach [10] supposes that the simulation platform already exists. Similarly to our work, a system-level power intent exploration can be performed using the Aceplorer [6] commercial tool from DoceaPower. Indeed, the power components used to specify the power architecture are inspired from UPF standard [20] concepts as well. Hence, the tool capability to export the UPF files is justified. However, neither the LPDISE nor the power intent specifications are automated by the Aceplorer tool.

3. OVERVIEW OF OUR POWER DOMAIN BASED METHODOLOGY

The objective of the MDE approach presented in this paper is to automate some operations of the methodology presented in [10]. In this section, a brief description of the key features of this methodology is given.

3.1 A Transaction-Level Power-Aware Flow

Starting from a pure functional Transaction-Level model of a SoC, our methodology mainly aims at adding power management capabilities to this model in an instrumentation-based manner. Figure 1 depicts the overall four-stage flow of the methodology. There are three successive stages: at the first stage, relying on the transactional flow analysis, the designer starts by specifying a power intent alternative for the TL-model in terms of power domains partitioning, power distribution (supply networks and power switches) and system power modes. A system power mode is a combination of power domains states, where each power domain state is defined by its primary supply nets states. A Power Management Unit (PMU) is modeled at the second stage to manage the different system power modes at runtime. It includes a
Domain Power Controller (DPC) module for each power-gated domain in charge of changing its power state between sleep and wake-up. It also includes a Power Manager (PM) module used to set non-gated power domains states and request related DPCs to change their power domains states according to the requested system power mode. At the final stage, the new power-managed behavior of the TL-model is simulated and power equations are recursively updated as soon as a change in a domain state is detected. Running the simulation after each successive stage activates the power-aware verification process that checks specific coherence properties between added power features and existing functional ones. For that reason, the power-aware verification stage occupies an orthogonal position regarding to the rest of the methodology.

3.2 The PwARCH Library

The PwARCH library has been developed and used to implement each stage of the methodology (Figure 1). Figure 2 depicts the class structure of the library. The UPF standard [20] power concepts having a relevant behavior at Transaction-Level have been abstracted. Adopted UPF power concepts are power domains (PD), power switches (PSw), supply nets (SN) of type primary, retention or isolation, power state tables (PST) including the system power modes and, legal transitions (PSTrans) between these power modes. Design element (DE) concept has been added.

![Figure 2. PwARCH general class structure.](image)

A DE refers to a SystemC module and is hence characterized by power data coming from datasheet or low level simulations. A PD is a group of DEs. All these concepts are used to specify the power intent at the first sequential stage. Observers on power switches and supply nets are defined at this stage as well and are notified as soon as the state of their power element changes. However, observers on DE elements are defined at the final stage and used for verification purposes. They are notified when an activity is detected in their DE. Assertions class is also used at the verification stage. Domain Power Controller class is used to ease DPC modeling. For each specified power domain, a power monitor is attached to update power equations when a power switch or supply net observer is notified. Power monitors are used at the final stage to perform power estimations.

3.3 An Iterative Methodology for Power Intent Exploration

As shown on Figure 1, the proposed methodology is iterative. At each iteration, a new low power design intent alternative can be specified and evaluated following the different methodology stages. The different alternatives (examples are given in Figure 7) are then compared and the most energy-efficient one is retained. However, to provide an efficient support for early exploration of low power design intent, the power intent specification stage of the methodology still needs automation mechanisms to be easily and rapidly performed. Our instrumentation method at this stage consists in augmenting the main class of the TL-design with an additional “PowerMain” code section that uses the PwARCH library. Indeed, in this code section, abstract UPF power objects are instantiated. Additionally, some of their attributes are set. Observers on power switch and supply net objects are instantiated as well in the “PowerMain”. The instantiation is done in a specific order to meet the composition dependency rules among the different power objects. At each iteration, the “PowerMain” is different and has to be manually rewritten by the designer. This can represent a tedious and error-prone task if the power design intent is complex, for example with a large number of power domains and power objects belonging to different hierarchy levels. For that reason, generating automatically the “PowerMain” code section at each iteration is strongly required to help and speed up the power intent exploration step.

4. THE PROPOSED MDE APPROACH

Following a Model Driven Engineering (MDE) approach is a well-suited solution for our automation purpose. Indeed, by using Model Transformation (MT) [16] key aspect of the MDE development process, executable models (or codes) can be produced from high level models. In general, MDE is based on three main strongly related concepts: metamodels, models and model transformations. Starting from a high abstraction level, a metamodel reflecting the domain concepts and relationship between them is defined using a model description language. A model is defined according to a specific metamodel to which it conforms, hence representing an instance of it. A model transformation (MT) [16] is a compilation process that allows moving from an abstract model to a more detailed target model. So, before performing any MT, a set of transformation rules must be first specified. Each MT is performed using a transformation
engine based on a source model and transformation specification rules to generate a target model. A key characteristic in MDE approaches is that the specified transformation rules can be modified or extended allowing 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 used in our work are only Model-to-Text (M2T) transformations: only executable models (codes) are generated from a specific high-level model. As shown on the Figure 3, our methodology has been extended with a MDE initial stage. This stage automates the power intent specification by automatically generating the “PowerMain” code section. The MDE approach is applied at each iteration. According to our methodology, after specifying a system power intent alternative, the augmented design model is simulated to check for some contracts relying to specific properties [10]. 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, static verification of this kind of contracts would certainly be enough. For that reason, this step in the power-aware verification stage has been automated in this work by 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 first stage of the methodology and joined with it as shown in Figure 3.

Our proposed methodology allows also 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 [20], it can be fully transformed to 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.

In the following, we propose to reuse the same model in order to define another M2T transformation to automatically generate corresponding UPF code. The Figure 3 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 the Figure 5. Each step is performed using a specific tool based on the Eclipse environment. The overall transformation chain as depicted in Figure 5 is explained in detail in the next section.

4.1 Automating “PowerMain” Code Generation

The first step in any MDE approach is the definition of metamodels. A metamodel called “Power Intent” (PI) has been elaborated only once using the UML formalism [12] and the graphical editor of the Eclipse Modeling Framework (EMF) [8]. As shown on Figure 4, the PI metamodel defines the different concepts that can be used in a “PowerMain” code section and naturally figure as PwARCH classes. These concepts are abstract UPF concepts (e.g. power domains, power state tables (PST), power transitions (PSTrans), supply nets, and power switches) and other added concepts (e.g. design elements and observers).

Relations between the UPF standard semantics, PwARCH library and the high-level model used in this paper are illustrated by Figure 6. 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

![Figure 4. The Power Intent (PI) metamodel.](image-url)
structural constrained, we distinguish between explicit properties which are directly extracted from the UPF language and standard semantics, and implicit properties which are rather indirectly deduced.

Explicit properties mainly concern the different relationships between UPF concepts. As shown on Figure 4, this kind of constraints is expressed in the PI metamodel using composition relations and cardinality concepts [12]. For instance, a power domain contains all other UPF power concepts except power transition concept which is attached to a PST. 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 6) 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. Implicit properties are seen in our work [10] as 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 6). These constraints represent hence conditions and restrictions imposed on some attributes and methods of the PI metamodel classes. As depicts Figure 5, 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 [8]. 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 specified. For that, a template file is written using Acceleo editor tool [2] 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 5). 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

Figure 5. Generation and integration process.

Figure 6. Relationships between UPF standard, PwARCH and PI metamodel.
included in the main code of the SystemC hardware platform (Figure 5). 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.

4.2 Automating “UPF” Code Generation
Once iterations for LDPISE 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 5 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 illustrates Figure 6, 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 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 [9]. 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 [9] have been followed to set isolation and level shifting strategies using UPF.

5. RESULTS AND DISCUSSION
In order to demonstrate the benefit of our proposed MDE approach, the same case study used in [10] has been applied. The Figure 7 depicts different power intent (PI) specification alternatives in terms of power distribution and power domain partitioning. For space limitations, specifications of the power state table and the corresponding set of legal power state transitions for each PI alternative do not figure in this paper but can be obtained in [10]. These alternatives were evaluated in [10] by iteratively applying the proposed methodology with a manual writing of “PowerMain” codes. It was concluded that the (b) alternative was the most energy-efficient PI specification for the final system executing the Conway’s game of life application with about 58% of energy savings is observed compared to (d) alternative and 7.3% compared to (c).

Using our proposed MDE approach presented in this paper, the “PowerMain” codes for the different alternatives of Figure 7, as well as the UPF code corresponding to the PI (b) alternative, have been automatically generated. Then we compared time and effort investments for both the manual approach (Figure1) and the automated approach (Figure 3). Figure 8 and table 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. Table 1 shows

![Figure 7. The considered power intent (PI) alternatives.](image)

![Figure 8. Comparison of results between manual writing and automatic generation of “PowerMain” codes.](image)
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 8 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 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 our methodology with an MDE-based approach only accelerates the first stage while verifying structural properties. But, such an enhancement does not alter the benefit of our methodology. Indeed, by using the enhanced methodology flow version (Figure 3), we also determined that the PI (b) alternative is the best solution for the studied SoC. 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.

### Table 1. Required effort to perform generation process

<table>
<thead>
<tr>
<th>Step</th>
<th>PowerMain</th>
<th>UPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI Metamodel Definition Time (min)</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>Code Generator Execution Time (min)</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Template</th>
<th>Code Lines Number</th>
<th>Definition Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acceleo</td>
<td>80</td>
<td>15</td>
</tr>
<tr>
<td>Template</td>
<td>110</td>
<td>20</td>
</tr>
</tbody>
</table>

### Table 2. Analogy between some code lines of the PI (b) “PowerMain” and the corresponding UPF commands

<table>
<thead>
<tr>
<th>PowerMain</th>
<th>UPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power_Domain* PD_top_level = new Power_Domain(NULL,&quot;PD_top_level&quot;,&quot;top&quot;);</td>
<td>set_design_top PD_top_level</td>
</tr>
<tr>
<td>primary_supply_net* VDDSoC = new primary_supply_net (&quot;VDDSoC&quot;,&quot;power_net&quot;,&quot;PD_top_level&quot;);</td>
<td>create_supply_port VDDSoC</td>
</tr>
<tr>
<td>Power_Switch* SW1 = new Power_Switch (PD_Memory,&quot;sw1&quot;);</td>
<td>create_supply_port VDDSoC</td>
</tr>
<tr>
<td>(SW1-&gt;.input_supply_nets).push_back (VDDSoC); SW1-&gt;.Set_Output_supply_net (VDDRAM_SW); VDDRAM_SW-&gt;.net_valid_states [&quot;RAM_ON&quot;] = 0.8;</td>
<td>create_power_switch SW1 = domain PD_Memory</td>
</tr>
<tr>
<td>VDDRAM_SW-&gt;.Add_State_Input_Net_Value (&quot;VDDOn&quot;,VDDSoC); VDDRAM_SW-&gt;.net_valid_states [&quot;RAM_OFF&quot;] = 0.0;</td>
<td>-input_supply_port (VDDSoC VDDSoC)</td>
</tr>
<tr>
<td>create_pst (&quot;PST_top&quot;,PD_top_level,&quot;5&quot;,&quot;VDDSoC&quot;,&quot;VDDCPU&quot;,&quot;VDDRAM_SW&quot;,&quot;VDDVGA_SW&quot;,&quot;VDDGPO offseason&quot;);</td>
<td>-output_supply_port (VDDRAM_SW VDDRAM_SW)</td>
</tr>
<tr>
<td>add_pst_state(&quot;PST_top&quot;,&quot;initialize&quot;,&quot;5&quot;,&quot;ON&quot;,&quot;ON_H&quot;,&quot;RAM_ON&quot;,&quot;VGA_OFF&quot;,&quot;GPO offseason&quot;);</td>
<td>-control_port [Ctrl_in VDDSoC]</td>
</tr>
<tr>
<td>PSTrans* Tr0 = new PSTrans (&quot;PST_top&quot;,&quot;initialize&quot;,&quot;all_on&quot;);</td>
<td>-on_state [RAM_ON VDDSoC {Ctrl_in}]</td>
</tr>
<tr>
<td>describe_state_transition Tr0 = object PD_top_level</td>
<td>-off_state [RAM_OFF ({Ctrl_in}]</td>
</tr>
<tr>
<td>-from [initialize] -to [all_on]</td>
<td>add_port_state VDDRAM_SW</td>
</tr>
<tr>
<td>set_isolation SW2_is_domain PD_Periph</td>
<td>-state [RAM_0.8]</td>
</tr>
<tr>
<td>-isolation_power_net VDDSoC</td>
<td>-state [RAM_OFF 0.0]</td>
</tr>
<tr>
<td>-isolation_ground_set GND</td>
<td>-supplies (VDDSoC VDDCPU VDDRAM_SW VDDRaph SW VDDGPO offseason)</td>
</tr>
<tr>
<td>-clamp_value [0] -applies_to_outputs</td>
<td>add_pst_state initialize -pst PST_top</td>
</tr>
<tr>
<td>set_isolation_control SW2_is_domain PD_Periph</td>
<td>-state [ON_H RAM_0.8 ON_PER OFF GPO offseason]</td>
</tr>
<tr>
<td>-isolation_signal Iso_Periph</td>
<td>-description for transitions</td>
</tr>
<tr>
<td>-isolation_sense high -location automatic</td>
<td>set_level_shifter schifft_up -domain PD_CPU</td>
</tr>
<tr>
<td>-applies_to_outputs -threshold 0</td>
<td>-location fanout -rule both</td>
</tr>
</tbody>
</table>

* The asterisk (*) indicates that the command is automatically generated by the MDE-based approach.
command [20]: 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 2 gives some lines of code of the PI (b) “PowerMain” section of code, and their corresponding generated UPF commands.

However, 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 [20]. As a consequence, this reduces significantly the verification and validation cost of a UPF power specification at levels of simulation lower than Transaction-Level.

6. CONCLUSION AND FUTURE WORK
In this paper, we have presented a Model-Driven Engineering (MDE) approach to generate, at Transaction-Level, correct power intent specifications and a UPF standard file describing an energy-efficient structure of a SoC. This MDE approach enhances our proposed methodology since it accelerates the low power design intent space exploration by fully automating the specification of power intent alternatives while verifying in parallel related structural properties. An automatic generation of an efficient power intent specification fully described with UPF commands is also allowed using our MDE-based approach. As a main consequence, the development and debug time required to manually write correct UPF specifications are hence significantly reduced. In general, this work demonstrates the effectiveness of using a MDE approach integrated in a power-aware design flow. In a near future, we will focus on the automation of the low power design intent space exploration. Towards a more complete coverage during the power intent exploration in our methodology, an automatic and constrained selection of power intent solutions will then be aimed.

7. ACKNOWLEDGMENTS
We would like to thank our colleagues from INRIA of Sophia Antipolis (research team AOSTE) for the fruitful discussions we had with them on MDE-based techniques.

8. REFERENCES