PMS2UPF: An Automated Transition from ESL to RTL Power-Intent Specification

Miroslav Siro, Dominik Macko, Katarína Jelemenská
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
miroslav.siro@stuba.sk, dominik.macko@stuba.sk, katarina.jelemenska@stuba.sk

***Abstract*—High power density is the most crucial problem in deeply integrated hardware systems. Therefore, the power has to be reduced in such systems, what is most commonly achieved by the utilization of power management. Unfortunately, the standardized application of power management is quite complex and does not very well support the system level of design abstraction, which is increasingly used by the industry. An immediately applicable solution to this problem is to use increased automation in the design process, regarding the power management. This paper proposes a tool, called PMS2UPF, which can automatically generate the standard UPF (Unified Power Format) power intent based on the abstract power-management specification in SystemC/PMS. The automated transition between the two abstraction levels not only accelerates the design process, but also prevents possible introduction of human errors into the refined design.**

Keywords—design automation; electronic system level; high level synthesis; low power electronics; power control

I. Introduction

In the recent times, the power and the corresponding energy consumption are the key design issues when developing a modern hardware system. It does not matter if we want to prolong the battery life of mobile devices, to cut down the energy costs of server farms and data centers, or to prevent the overheating of highly integrated systems-on-chips (SoC). All of these goals have contributed to the rising significance of the power-management support integration into the hardware design process. It is nowadays supported in various ways: there are built-in power-management constructs in some hardware description languages (HDLs) [1], there are extension libraries for HDLs (PowerSC [2] or PMS – Power Management Specification [3]), or there even exist specification languages dedicated to capturing the power intent (UPF – Unified Power Format [4] or CPF – Common Power Format [5]).

The power management enables to adopt various power-reduction techniques [5], such as clock gating, voltage and frequency scaling, power shut-off, operand isolation, and others. One of the most basic concepts of power management is to split the system into the so-called power domains. The power domain represents a set of system modules with the same power supply. The idea is to dynamically change the operation state (power state) of the power domains and thus to save power or redirect it to other parts of the system. The standard way of power-management adoption is using the UPF or CPF power-intent specification alongside the HDL model, as illustrated in Fig. 1 (*P & R* item represents the place and route process). However, the commonly used design-automation tools support these formats at the register-transfer level (RTL) and at the lower abstraction levels, where current complex systems are difficult to manage manually. The design automation at the system level (ESL), which is nowadays a common design starting point (because of the ever-growing systems complexity), does not currently support power-intent synthesis. Instead, the UPF/CPF power-intent specification has to be designed manually to accompany the synthetized ESL functional specification. Since the standardly used design language at the ESL is SystemC [6], several standard and custom techniques to include power specification in SystemC have been developed (see Section II). However, these techniques either do not offer enough abstraction needed for ESL, or do not offer automated transition to the lower levels.

In this paper, a tool is introduced, called PSM2UPF, which can automatically analyze the ESL specification model in SystemC/PMS, extract the abstract power-management specification, create additional power-intent components, required at the RTL, and generate the standard UPF specification supported by the commonly used power-analysis

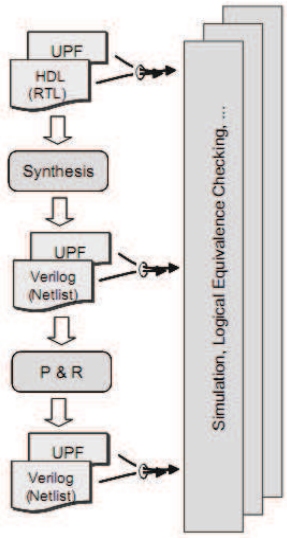

Fig. 1 Typical use of UPF in low-power design flow [4].

This is an accepted version of the published paper:
M. Siro, D. Macko and K. Jelemenská, "PMS2UPF: An automated transition from ESL to RTL power-intent specification," 2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Dresden, 2017, pp. 140-144.
doi: 10.1109/DDECS.2017.7934558
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7934558&isnumber=7934552

tools. In the next section, the state of the art is presented. In Section III, the UPF synthesis method implemented in the proposed tool is described. Section IV is focused on the experimental evaluation and verification of the tool. And finally, the conclusions are summarized in Section V.

II. Related Work

To support the ESL, the UPF standard has been recently updated [7]. It has standardized a way, in which a designer can specify power consumption of an IP (Intellectual Property) block in multiple power states. This enabled the design-automation tools to analyze and to estimate power at the ESL. However, it is not standardized how the power information is obtained. It is most commonly obtained from the previously implemented designs. Therefore, it is not suitable for the top-down design approach. Also, the specification of power management uses low-level details (e.g. power-supply networks), which are not suitable for the ESL.

PowerSC [2], TLM Power3 [8], and PKtool [9] represent extension libraries for SystemC that enable a designer to monitor and estimate power consumption. However, these libraries do not provide functions to manage it, and therefore the resulting estimations might be highly inaccurate when the power management is used in the system. On the other hand, Power Modelling Framework [10] enables to model power-management at the ESL. The main limitation of the method is that it uses customized power-intent modelling with no connection to the standard power-intent specification at the lower levels. Thus, the verification of power-intent equivalency is difficult to achieve.

PwARCH [11] represents a framework for ESL power-management modelling, which is based on the abstracted UPF specification. The power-intent specification is separated from the functional specification, which is not very convenient for a designer at such an early design stage. Moreover, transition to the RTL is mostly manual. The LP-HLS methodology [12] seems to be more automated. It is based on a generic power-management module description at the system level, which is then used to generate CPF directives for implementation. However, it is focused only on the power shut-off technique, which is not sufficient in the complex SoC designs.

In PMS library [3], the abstract UPF concepts are used as well, this time directly integrated into the SystemC functional specification. The power-management high abstraction level enables a designer to focus on the functionality. However, the power management is only specified at the ESL, not modelled. Therefore, the power can be analyzed only later, in the RTL model. For this reason the fast and automated UPF synthesis tool would be required to enable fast design-space exploration, based on the design alternatives comparison.

A. Comparison of PMS and UPF

The PMS [3] extension library is based on the UPF concepts. It means that it also uses power domains to group the system components, always operating in the same power states. However, the power state is not dependent only on the power supply, but also on the operation frequency of the component. In UPF, a power state is defined by a combination of values of control signals for the power-management elements (e.g. power switches or isolation cells). These elements are abstracted away in PMS, and therefore they must be introduced during the high-level synthesis (targeted in this paper). For further simplification of the power-management specification, PMS defines the power mode. It is a specific combination of power states in individual power domains. A designer can then switch the power mode of the whole system rather than set appropriate power state (control-signals values) of each power domain. This is an abstract view of the UPF power-state table.

To summarize, PMS enables a designer to assign components to power domains, to specify power states in which the components of the domain can operate, to define a voltage-frequency pair for each active power state, to specify power modes in which the system can operate, and to specify switching among power modes.

In PMS several predefined power states are available and each of them implies application of some power-reduction technique. These predefined power states and their implication to UPF constructs are briefly described below.

1) NORMAL

In this power state no explicit power-reduction technique is used, so the UPF power domain is supplied by the system main supply net (i.e. the primary supply net of the top domain). The components of the domain in this state are operating at the primary supply voltage and at the basic frequency of the system (the frequency cannot be modelled in UPF).

2) HOLD

This state represents the usage of clock-gating and operand-isolation techniques. All input signals (clock included) of a power domain in this state are isolated, meaning that isolation cells and isolation control constructs must be specified in UPF.

3) DiffLevel

This state means that a performance level different to the normal one is used in the domain. It means that the voltage, or frequency, or both are different to those used in the *NORMAL* power state. This power state enables to apply the voltage and frequency scaling power-reduction technique, or to use multiple fixed voltages/frequencies in the system. Multiple possible voltages in the power domain imply the power switch in the UPF specification. If there is a high difference between the voltages of interconnected power domains, the level shifter is also required.

4) OFF

This power state represents an application of the power shut-off technique. The power supply of the domain is switched-off; therefore, the power switch is required in UPF. Moreover, inputs and outputs of the power domain must be isolated to prevent unnecessary power dissipation; therefore, the isolation cells are also required in UPF.

5) OFF_RET

In this power state, the power shut-off with the state retention is applied. There are the same implications to UPF as in case of the *OFF* power state; however, the retention must be set for the domain in this state in UPF.

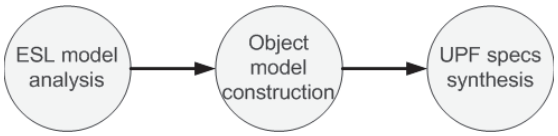

Fig. 2 Power intent transformation steps.

III. ESL TO RTL TRANSITION PROCESS

As it was mentioned in the introductory section, the PMS2UPF tool transforms the power management, specified in SystemC with PMS constructs, to the equivalent UPF specification. This transformation process consists of three basic steps illustrated in Fig. 2. The first step is the analysis of the input specification files, which identifies the important constructs in the design (e.g. modules, power domains). In the next step, the objects of important design structures are created, which are necessary for RTL power-intent specification (supply nets, power switches, isolation cells etc.). The created object model is then used in the last step for generation of the UPF specification. These steps are executed in this order. The following subsections contain a more detailed description of these steps.

A. ESL Model Analysis

This step involves parsing of the SystemC source code mixed with the PMS specification. An example of input SystemC/PMS specification is provided in Fig. 3. In the figure, a part of the SystemC module constructor is illustrated, in which the *system0* module contains four components (*CPU* – representing microprocessor, and *M1*, *M2*, *M3* – representing memories). There are two power domains specified, *PD1* and *PD2*, each has assigned components and power states it can operate in. There are three power modes specified (*PM1*, *PM2*, and *PM3*), which represent a combination of power states in power domains. For example, the specification of *PM1* implies that *PD1* operates in *NORMAL* power state and *PD2* operates in *DiffLevel(1)* power state. It means that when the system is switched into this power mode, the power states of power domains are correspondingly adjusted. Finally, there are voltage and frequency values assigned to the *NORMAL* and *DiffLevel(1)* power states. More on the SystemC/PMS specification can be found in [3].

```
SC_CTOR(system0): CPU("CPU"), M1("M1"), M2("M2"), M3("M3")
{
     CPU(clk,Abus,Dbus,reset,MEMrq,Rnw);
     M1(clk,Rnw,MEMrq1,Abus,Dbus);
     M2(clk,Rnw,MEMrq2,Abus,Dbus);
     M3(clk,Rnw,MEMrq3,Abus,Dbus);

      PD1.AddComponent("CPU");
      PD2.AddComponent("M1");
      PD2.AddComponent("M2");
      PD2.AddComponent("M3");

      PD1 = PD(NORMAL, OFF_RET, OFF);
      PD2 = PD(NORMAL,DiffLevel(1), OFF_RET);

      PM1 = PM(NORMAL,DiffLevel(1));
      PM2 = PM(OFF_RET,NORMAL);
      PM3 = PM(OFF,OFF_RET);
      POWER_MODE = PM1;
      sigPM = 1;
      SetLevel(NORMAL,1.2,10);
      SetLevel(DiffLevel(1), 1.6, 10);
```

Fig. 3 SystemC/PMS source-code fragment.

For the code-analysis purpose, we have developed a custom parser. The parser loads all source files and extracts module definitions from them. Each module is stored in a dedicated string structure, what simplifies the further analysis. These module-representing strings are further split into a list of smaller tokens by using semicolons and composed brackets. Several iterations through the tokens are then necessary in order to extract required information (e.g. sub-modules, ports, channels, PMS constructs) from these tokens, which is then used to fill the objects in the next transformation step.

Several issues had to be solved in implementation of the parser:

- Some information required for UPF commands cannot be extracted from the PMS specification. Therefore, the SystemC design has to be also analyzed.
- Module and port identifiers are not known beforehand; therefore, declarations and definitions of the modules have to be recognized. The parser iterates over the tokens multiple times, which prolongs the analysis process. There is a potential for future optimization of this algorithm.
- The designer's coding style in the input files is not known beforehand (e.g. whitespaces, brackets, number of lines and number of modules per file). Therefore, a robust solution has to be developed, which tokenizes the files and purifies tokens from redundant characters (whitespaces, semicolon, brackets).
- Some tokens include comments or SystemC constructs that do not influence the power management. Therefore, the parser has to distinguish the required tokens from the unnecessary ones.

B. Object Model Construction

The goal of this step is to create a list of objects that include all the information required to create a proper UPF specification at the RTL. The object model will therefore contain not only the information extracted in the previous step, but also the more-detailed UPF-required information. There are dedicated objects created that represent individual UPF commands. It must be noted that only the fractional information about those objects is explicitly specified in PMS specification; therefore, the missing data has to be determined by analysis of relations of the specified objects. To be more specific, in the analyzed ESL specification, there is no information regarding the supply nets, power switches, isolation cells, level shifters, retention cells, or power-state table. However, this information can be deduced based on the abstract power states, structural dependencies of the system components (e.g. hierarchy, communication), and relations among power domains implied by the abstract power modes.

For example, in case of the *PM2* power mode in Fig. 3, the first power domain is powered down and the second one is on. It implies that the first power domain requires a power switch to take it down. To prevent unnecessary switching power to be

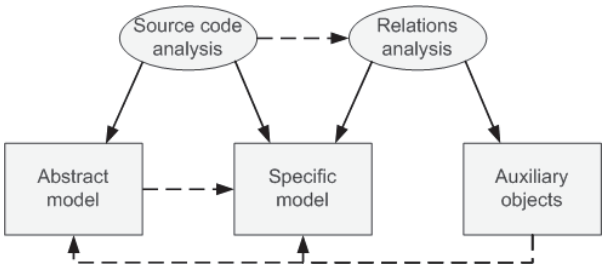

Fig. 4 Internal object model composition.

consumed, the input signals of this domain should be isolated; therefore, the isolation cells are added in front of the input ports. Eventually, the output ports must also be isolated in case of inter-domains communication, in order to prevent wrong values propagation. Such a deduction is fully automated, what saves time and avoids potential human errors.

The internal object model consists of three parts (see Fig. 4):

- *Abstract model* – includes object prototypes representing some specific parts of the SystemC design. For example, each type of module is represented by one object prototype, which is then used to create individual instances of the module.
- *Specific model* – includes objects representing instances of the modules and other constructs of the SystemC design with the PMS specification. These objects contain all data required for UPF generation.
- *Auxiliary objects* – have only the supplementary roles for creation of abstract and specific models. For example, there is an object representing a connection of a port to a channel. These objects just simplify the implementation.

C. UPF Specification Synthesis

The developed UPF generator sequentially iterates through the lists of created objects, representing UPF specification, and generates the required UPF commands based on the objects. After the commands are generated, they are concatenated and written to a text file. However, in order to generate a correct UPF specification, we have to follow some constrains:

- The correct order of the UPF commands has to be preserved. For example, the power domains have to be created before the supply nets can be generated.
- The structural dependencies among some UPF commands have to be respected. For example, the number of power switch control ports is dependent on the number of voltage levels the corresponding power domain can operate at.

An example of a UPF code fragment, obtained as an output from the PMS2UPF tool is provided in Fig. 5. More specifically, the provided UPF code includes specification of power domains (*PD1*, *PD2*, and implicit *PD_TOP*), the supply net and the supply port *sn_1*, and one power switch *ps_system0_PD1*. The tool also generates isolation cells, level shifter cells, retention cells, and power-state table. These are

```
set_design_top /system0
set_scope .

create_power_domain PD_TOP -elements /system0
create_power_domain PD1 -elements /system0/CPU
create_power_domain PD2 -elements { /system0/M1 /system0/M2 /system0/M3 }

create_supply_port sn_1
create_supply_net sn_1 -domain PD_TOP
create_supply_net sn_1 -domain PD1 -reuse
create_supply_net sn_1 -domain PD2 -reuse
connect_supply_net sn_1 -ports sn_1
 ...
create_power_switch ps_system0_PD1 \
 -domain PD1\
 -input_supply_port { in1 sn_1 }\
 -output_supply_port { out system0_PD1_V }\
 -control_port { cp_0_0 cp_0_0_s }\
 -off_state { OFF_RET  { !cp_0_0 }}\
 -on_state { NORMAL in1 { cp_0_0 }}
```

Fig. 5 An example of the generated UPF specification fragment.

not shown in the figure. Even when the abstract power-intent specification using the PMS library is quite simple (such as the one in the example in Fig. 3) a complete UPF specification can require hundreds of lines. For simplified illustration purpose, full output is not provided. Even from this simple example, one can see that the proposed automation is really helpful. In average the UPF complexity is about five times higher compared to the PMS complexity [3].

IV. Experimental Evaluation

For an experimental evaluation of the proposed tool, we have designed a simple case-study SoC, consisting of one microprocessor *mu0* and three memories *ram0* (RTL description of the components is available in [13]). They are interconnected using a memory controller, which selects the memory with which the microprocessor will communicate based on the address at the address bus. We have described this system in SystemC code in order to mimic top-down design and to use the proposed ESL to RTL transition process. The SoC architecture overview is provided in Fig. 6. This system also corresponds to the examples provided in Fig. 3 and Fig. 5. We have incorporated into this system various power-management specifications in PMS, synthesized the UPF specifications and verified the results. Correctness of the generated UPF specifications has been verified using the professional verification tool Questa Power Aware Simulator [14], especially its UPF static checks.

In Table I, experimental results are provided for six test cases. The first column contains test-case number. *PD* represents the number of power domains in the PMS

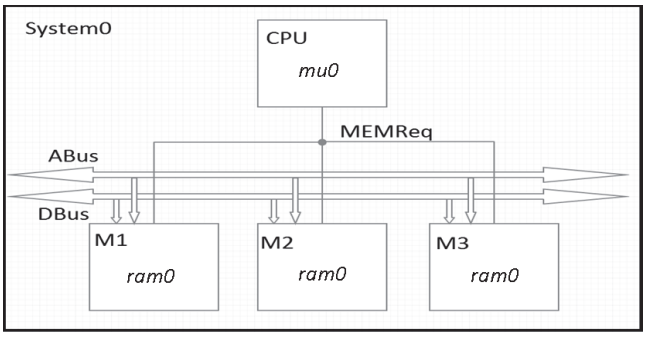

Fig. 6 The case-study system architecture overview.

TABLE I. TESTING RESULTS

Test case	PD	PM	PS	UPF chars	SW	ISO	SN	RESULT
1	3	3	NORMAL, HOLD	2145	0	3	2	correct
2	4	4	NORMAL, HOLD, OFF, OFF_RET, DIFF_LEVEL(1), DIFF_LEVEL(2)	5091	4	2	4	correct
3	3	3	NORMAL, OFF, OFF_RET	5113	4	4	2	correct
4	2	3	NORMAL, OFF, OFF_RET, DIFF_LEVEL(1)	3421	2	2	3	correct
5	0	0	(NORMAL)	497	0	0	2	correct
6	1	7	NORMAL, HOLD, OFF, OFF_RET, DIFF_LEVEL(1), DIFF_LEVEL(2), DIFF_LEVEL(3)	3214	1	1	5	correct

specification, *PM* represents the number of power modes in the PMS specification, and *PS* contains power states that have been used in the PMS specification. The next column (*UPF chars*) represents the number of characters generated in the UPF specification, *SW* represents the number of generated power switches, ISO represents the number of generated isolation cells and SN represents the number of supply nets in the UPF specification. The last column states whether the UPF specification is correct as reported by the utilized verification tool. We can see in the table that the tool has synthesized the correct UPF specification in all the test cases.

The only available comparable tool is the one used in [15], called PMHLS. The difference between these two tools is that PMHLS is a complex experimental system, which has not been developed for actual SystemC designs but for artificial PMS samples. As we have shown in our experiments, PMS2UPF is able to work with a real-life SoC. PMHLS can use only single-file input and PMS2UPF operates with unlimited designs. Moreover, PMS2UPF can be easily used in a complex design environment because of its simple command line interface. This cannot be done with PMHLS.

V. CONCLUSIONS

In this paper the PMS2UPF tool has been described, which speeds-up the transition between ESL and RTL abstraction levels during the design process. More specifically, it is dedicated to the power-intent specification; therefore, it is usable in low-power systems designs. The proposed tool enables the designers using the SystemC/PMS abstract specification to easily convert their power intent into the UPF standard. It replaces a time-consuming manual synthesis process, and thus it not only saves time but also reduces the possibility of introducing human errors into the design. It simplifies the validation and debugging process. The designed tool provides a simple command line user interface, which makes it easy to use in more robust development platforms, but also it provides a graphical user interface to be usable as a standalone tool.

Future enhancements can be oriented towards optimization of the analysis algorithm, reducing the number of iterations through the tokens, or to support other output formats, such as CPF – to be used in Cadence's design toolchain.

ACKNOWLEDGMENT

This work was supported by the Slovak Scientific Grant Agency (VEGA 1/0836/16), the Slovak Research and Development Agency (APVV-15-0789), and the Ministry of Education, Science, Research and Sport of the Slovak Republic within the Research and Development Operational Programme for the project "University Science Park of STU Bratislava", ITMS 26240220084, co-funded by the European Regional Development Fund.

REFERENCES

[1] D. Macko and K. Jelemenská, "Managing digital-system power at the system level," in IEEE Africon 2013 Sustainable Engineering for a Better Future, 2013, pp. 179-183.

[2] F. Klein, R. Azevedo, L. Santos, and G. Araujo, "SystemC-based power evaluation with PowerSC," in Electronic System Level Design: An Open-Source Approach, S. Rigo, R. Azevedo and L. Santos, Eds. Springer, 2011, pp. 129-144.

[3] D. Macko, K. Jelemenská, and P. Čičák, "Power-management specification in SystemC," in 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 259-262.

[4] IEEE Standard for Design and Verification of Low Power Integrated Circuits, IEEE Standard 1801-2013, May 2013.

[5] Cadence Design Systems, A Practical Guide to Low Power Design: User Experience with CPF, 2009.

[6] IEEE Standard for Standard SystemC Language Reference Manual, IEEE Standard 1666-2011, Jan. 2012.

[7] IEEE Standard for Design and Verification of Low-Power, Energy-Aware Electronic Systems, IEEE Standard 1801-2015, Dec. 2015.

[8] D. Greaves and M. Yasin, "TLM POWER3: Power estimation methodology for SystemC TLM 2.0," in Models, Methods, and Tools for Complex Chip Design, LNEE, vol. 265, J. Haase Ed. Springer, 2014, pp. 53-68.

[9] "PKtool: Power estimation in SystemC," [Online]. Available: http://pktool.sourceforge.net/

[10] H. Lebreton and P. Vivet, "Power modeling in SystemC at transaction level, Application to a DVFS architecture," in IEEE Computer Society Annual Symposium on VLSI, 2008, pp. 463-466.

[11] O. Mbarek, A. Pegatoquet, and M. Auguin, "Using unified power format standard concepts for power-aware design and verification of systems-on-chip at transaction level," IET Circuits, Devices & Systems, vol. 6, no. 5, pp. 287-296, 2012.

[12] A. Qamar, F.B. Muslim, J. Iqbal, and L. Lavagno, "LP-HLS: Automatic power-intent generation for high-level synthesis based hardware implementation flow," Microprocessors and Microsystems, vol. 50, pp. 26-38, May 2017.

[13] A. Rogers, "Designing a simple system-on-a-chip in under 60 minutes with the mu0 microprocessor and Xilinx tools," 2003. [Online]. Tutorial. Available: http://www.ece.uah.edu/~lacasa/tutorials/mu0/mu0tutorial.html

[14] Mentor Graphics, "Questa Power Aware Simulator: Verify active power management." [Online]. Available: https://www.mentor.com/products/fv/questa-power-aware-simulator

[15] D. Macko, K. Jelemenská, and P. Čičák, "Power-management high-level synthesis," in The 23rd IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 2015, pp. 63-68.