Synthesis of Hardware Systems Power Management

Miroslav SIRO*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
siro.miroslav@gmail.com

Abstract. Power consumption is a problem in highly integrated hardware systems, mainly because of the rising power density and associated overheating of the chip. Therefore, the power must be managed in such hardware designs. However, the standard way of power management is quite complex and a way to simplify it is to use more automation in the design process. Our work is focused on automatic generation of the power-management specification of hardware systems at the register-transfer abstraction level (RTL). In this paper, we describe the design and implementation of a tool, called PMS2UPF, which automatically transforms abstract power-management specification from the SystemC standard, enhanced by the PMS library, to the more-detailed UPF (Unified Power Format) standard language. The automation of this transformation accelerates the design process, specifically the transition from the system level of abstraction to the RTL.

1 Introduction

Nowadays, the power consumption is one of the key considerations in hardware system development. It is very important in various kinds of hardware systems, like mobile systems where battery life is essential, servers where power costs are not negligible or general highly integrated hardware systems, where overheating is a problem. Therefore, need for power management arose. There are multiple approaches to hardware system design that have power management in mind, like built-in features of hardware description languages (HDL), external libraries for these languages or completely new languages dedicated to power management specification. The most well-known language for this purpose is UPF (Unified Power Format), [1]. These approaches provide various methods of power management, like clock gating, voltage and frequency scaling, power shut-off, operand isolation and others.

In this paper, we focus on automation of power management in hardware system design using SystemC standard and UPF language. There are multiple SystemC [2] libraries that handle power monitoring, estimation, modelling or management. Some of them are TLM Power 3 [3], PK

* Master degree study programme in field: Software Engineering
Supervisor: Dr. Dominik Macko, Institute of Computer Systems and Networks, Faculty of Informatics and Information Technologies STU in Bratislava

Tool [4], Power SC [5], Power Modelling Framework [6] PwARCH [7], or PMS [8]. We focus particularly on the PMS library.

PMS is a SystemC library that provides methods of abstract power management specification. Its central idea is the same as of the UPF language and that is a division of the hardware system into power domains.

The power domain [1] is one of the most basic concepts of power management. It is a group of modules of a hardware system that are powered in the same manner. These modules are usually located physically close to each other and powered by the same supply nets. If a component is a part of the power domain, then all its subcomponents are part of it too. Power domains are hierarchically structured and one power domain can be a part of the other. Every domain has one power state that is active and a list of power states that the domain can possibly achieve. Also, it is necessary to create a power state table, which specifies, which combinations of power states of various power domains are allowed [9].

In this paper, we propose a tool named PMS2UPF which automatically converts power management specification specified by SystemC design and functions of PMS library to commands of the more-detailed UPF language, which is an internationally recognized standard of power management specification. This automation significantly accelerates development process and reduces risks of introducing a human error to the design.

In the next section we introduce a work related to the problem area of our research. In Section 3 we describe a proposed transformation process, including the analysis of the input files with SystemC design and PMS specification, internal data model and generation of the UPF specification. In the last section, we make the conclusions from this work and plans for further work.

2 Related work

As mentioned before, there are numerous SystemC libraries that enable power monitoring, estimation, modelling or management. We introduce each of the mentioned libraries in this section.

TLM Power 3 [3] provides classes that allow a hardware system designer to estimate power consumption of the designs. The library requires the designer to modify classes of the design in a way to inherit from the basic class of TLM Power 3: *pw_module*. The TLM Power 3 uses two approaches towards power estimation: estimation by regime of individual modules or estimation by power per transaction.

Another mentioned library is PK Tool [4], which allows a designer to estimate power consumption of the design. Basic class of the library is *power_module*. An instance of this class is created for every module of SystemC design. PK Tool allows the designer to use models of power consumption to make predictions. It also allows user to specify custom power consumption models. Another feature of this library are *Augmented Signals* which can extract dynamic data from the signals.

PowerSC [5] is a library that allows a designer to monitor power consumption. It monitors a switching activity in the system design and provides macros to extract necessary data from the design.

These three libraries allow designer to monitor and estimate power consumption but they do not provide functions to manage it. These estimations might however become less accurate when power management is specified.

Next three libraries, Power Modelling Framework, PwARCH and PMS however allow the designer to specify power management.

Power Modelling Framework [6] allows user to model the power consumption of the designed system. It allows user to specify and change the power state of the system by using

function *new_power_phase(phase).* This library also allows the designer to use Dynamic Voltage and Frequency Scaling (DVFS) by using function *new_power_mode(mode*).

PwARCH [7] allows the designer to model and verify power management specification. It adopts multiple concepts from UPF and abstracts them to the transaction layer. It supports power domains, power switches, and global power state table. After designing power management specification, a PMU (Power Management Unit) needs to be modelled as well.

Finally, there is the PMS (Power Management Specification) [8], which we have mentioned before. This library is inspired by UPF, but allows the designer to use system level of abstraction. Its main concept of power management specification is a power domain. It is however much easier to use than UPF. Benefit of this library is that it allows verification of power specification in the early stages of development which makes it easy and cheap to detect and correct errors.

We decided to focus our work on PMS library because it does not only allow to specify power management on the system level of abstraction, but also significantly simplifies it.

3 The proposed synthesis process

As stated above, PMS2UPF transforms a SystemC and PMS specification to the UPF specification. This transformation consists of three basic steps (See Figure 1).

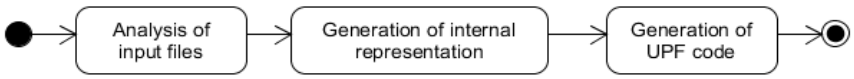

Figure 1. Three steps of transformation from PMS specification to UPF specification.

The first step is the analysis of input files, which loads all the specification files and identifies the important constructs in the design. The next step creates objects of important design structures and identifies relationships among them. The last step generates the power-intent specification in the UPF language based on the created internal object representation. These steps are executed sequentially. Each of these three steps mentioned above will be explained in more detail in the following sections.

3.1 Analysis of input files and creation of internal representation of the design

Analysis of input files requires parsing of SystemC source files mixed with PMS specification. For this task we have implemented custom parser of the code. Example of an input file with PMS specification can be seen in Figure 2.

```
#include "systemc.h"
#include "pms.h"

SC_MODULE(System){
    sc_signal<data> data_register;
    PowerDomain PD1;
    PowerDomain PD2, PD3;
    PowerMode PM1;
    PowerMode PM2;
    Component1 C1;
    Component2 C2;
    Component3 C3;
    ...
    ...
    SC_CTOR(System):C1("C1"), C2("C2"), C3("C3"){
        PD1 = PD(NORMAL, OFF);
        PD2 = PD(NORMAL, DIFF_LEVEL(1), HOLD);
        PD3 = PD(NORMAL, OFF_RET);
        PD1.AddComponent("C1");
        PD1.AddComponent("C3");
        PD2.AddComponent("C2");
        PD3.AddComponent("data_register");
        PM1 = PM(NORMAL, DIFF_LEVEL(1), NORMAL);
        PM2 = PM(OFF, HOLD, OFF_RET);
        POWER_MODE = PM1;
        SetLevel(NORMAL, 0.9, 50);
        SetLevel(DIFF_LEVEL(1), 1.1 V, 0.1 GHz);
        ...
    }
    ...
};
```

Figure 2. SystemC source file with PMS specification [8].

This parser opens all source files and reads module definitions from them. Each module is stored in separate string. This way we create mapping of one string to one module, which simplifies further analysis. Each of these strings is separated by characters *semicolon*, *opening* and *closing composed bracket* (;{}) to a list of smaller tokens. These tokens are then read multiple times and each iteration extracts some data from these tokens and creates objects representing these data. These data include sub-modules, ports, inner channels, connections between modules and PMS specification itself (example illustrated in Figure 2). Based on the extracted data we are able to create objects representing more-detailed UPF specification. Each object is going to represent one UPF command. Information about some of those objects is not explicitly specified by PMS specification therefore our tool needs to determine which objects need to be created. Examples of such objects are power switches, supply nets, isolation cells, level shifters, retention cells or power state table. Aim of this analysis is to create list of objects that contain all necessary information to create proper UPF specification. This process is illustrated in Figure 3.

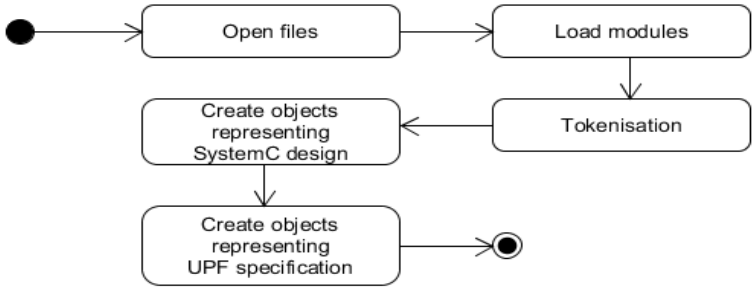

Figure 3. Process of analysis of SystemC source files.

During implementation of the analysis we have encountered several problems:

- A part of the information necessary to generate UPF commands is impossible to be extracted from the PMS specification alone. Therefore we have to analyse the SystemC design too.
- Names of the modules and ports are not known beforehand. Therefore we must search for declarations and definitions of the modules. We need to iterate over the tokens multiple times, which prolongs the analysis process. Implementation of this part is subject to further optimisation.
- The form of the input files (i.e. the designer's coding style) is not known beforehand. Locations of whitespaces, number of lines and number of modules per file is not known. Therefore we had to create more robust solution that tokenises the files, excludes tokens that contain no relevant information and purify tokens of characters (whitespaces and others) that do not hold any useful information. This processing simplifies loading of all necessary data to create objects representing SystemC/PMS design and UPF specification.
- Not all tokens contain relevant information. Some of them might be comments or they describe parts of SystemC design that are not needed in our internal representation. Therefore we must be able to determine which tokens are necessary to analyse and which are not.

Internal data model that contains representation of SystemC design and UPF specification consists of three parts:

- Abstract model: Abstract model contains prototypes for each object that represent some part of SystemC design. For example each module type is represented by one object and serves as a basis to create every instance of the module.

- Specific model: Objects of specific model represent every instance of the module or other element of the SystemC design or PMS specification. These objects hold all information necessary for generating of the UPF specification.
- Auxiliary objects and classes: These objects have supplementary roles at creation of abstract or specific model. For example, there is a class that represents connection of port to the channel. These classes do not represent anything from SystemC design or UPF specification but they simplify the implementation.

3.2 UPF Generation

In this section, we describe a method of generating the UPF specification. The UPF generator sequentially iterates through the lists of objects representing UPF specification and generates a UPF command based on each object. Finally, all generated UPF commands are concatenated to one long string and written into a text file. However, we must adhere to the following constraints of the UPF language:

- We need to keep a correct order of the UPF commands. For instance, we need to create power domains at first and only then we can generate supply nets.
- Structure of some UPF commands is dependent on other UPF commands. For example the number of power switch control ports depends on the number of power states that corresponding power domain can achieve.

The following source code illustrates some parts of the UPF output of the PMS2UPF tool, namely power domains, some supply nets and one power switch. A complete UPF specification even when based on a simple PMS specification can, take hundreds of lines. Therefore we are not going to present full example of the output.

```
create_power_domain PD1 -elements { DUT.System.C1 DUT.System.C3 }
create_power_domain PD2 -elements DUT.System.C2
create_power_domain PD3

create_supply_port VSS
create_supply_net VSS -domain PD1
create_supply_net VSS -domain PD2
create_supply_net VSS -domain PD3
connect_supply_net VSS -ports VSS

create_supply_port sn_1
create_supply_net sn_1 -domain PD1
create_supply_net sn_1 -domain PD2
create_supply_net sn_1 -domain PD3
connect_supply_net sn_1 -ports sn_1

…

create_power_switch ps_PD1
 -domain PD1
 -input_supply_port { in0 VSS }
 -input_supply_port { in1 sn_1 }
 -output_supply_port { out PD1_V }
 -control_port { cp_6_0 cp_6_0_s }
 -off_state { OFF in0 { ! cp_6_0 }}
 -on_state { NORMAL in1 { cp_6_0 }}
```

In the code above, we can see that from information about power domains and power states, our tool generates supply nets and power switches. It also generates isolation cells, level shifter cells, retention cells and power state table. Those are however not shown in the example above.

4 Conclusions

In this work, we have proposed and implemented a tool that simplifies usage of the PMS library. The developed tool allows the designers using the PMS library to easily convert their power management specification to the UPF standard. This increases speed of development process and reduces the possibility of introduction of human errors into the design. The reason is that with the PMS2UPF tool a hardware system designer and PMS user does not need to transform the power management specification from SystemC to UPF manually. This significantly simplifies the validation and debugging process. Our tool provides a simple command line user interface which makes it easier to use in conjunction with other programs and thus to automate even larger portion of the design process.

In the future work we plan validate the resulting UPF specification by using the Modelsim [10] tool, so we can guarantee validity of the generated UPF code. We also plan to optimize our tool so the transformation process can run faster, create a graphical version of the user interface and provide more options to the user. For example, allow the system designer to specify which particular version of UPF (1.0 or 2.0) our tool should provide as an output.

Acknowledgement: This work was partially supported by the Slovak Scientific Grant Agency (VEGA 1/0616/14 "Methods for the design and verification of digital systems with low power consumption using formal specification languages").

References

[1] IEEE: Standard for Design and Verification of Low-Power Integrated Circuits 1801. IEEE, (2013).

[2] IEEE: Standard for Standard SystemC Language Reference Manual 1666. New York, IEEE, (2011).

[3] Greaves, D. and Yasin, M.: TLM POWER 3: Models, Methods and Tools for Complex Chip Design, LNEE, (2014).

[4] Vece, G., Conti, M.: PKtool: Power estimation in SystemC. Available at http://www.deit.univpm.it/PKtool/

[5] Klein, F. et al.: SystemC-based power evaluation with PowerSC. Electronic system level design: An open-source approach, Springer, (2011).

[6] Lebreton, H. and Vivet, P.: Power Modeling in SystemC at Transaction Level, Application to a DVFS Architecture, Grenoble: IEEE Computer society, (2008).

[7] Mbarek, O., Pegatoquet, A. and Auguin, M.: Using unified power format concepts for power-aware design and verification of systems-on-chip at transaction level. IET Circuit Devices Syst Vol. 6. Nice, IET, (2012), vol. 6, pp. 287-296.

[8] Macko, D., Jelemenská, K. and Čičák, P.: Power-Management Specification in SystemC, IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits and Systems. IEEE, Bratislava, (2015), pp. 259-262.

[9] Power Format Initiative. A practical Guide to Low Power Design.: Power Forward, (2012).

[10] Mentor Graphics. mentor.com. Available at https://www.mentor.com/products/fv/modelsim/