Managing digital-system power at the system level

Dominik Macko, Katarína Jelemenská
Slovak University of Technology
Faculty of Informatics and Information Technologies
Bratislava, Slovakia
macko@fiit.stuba.sk; jelemenska@fiit.stuba.sk

***Abstract*— The increasing trend in using mobile devices causes the power consumption to become the key aspect in modern digital systems design. The current flow for low-power design involves the use of some power-reduction technique at the RTL (Register Transfer Level) or lower levels. This paper describes an extension of current low-power design flow. It proposes the power-intent specification based on UPF (Unified Power Format) in an abstract form and integrates it into a system-level model. Such an abstract level gives us the opportunity to manage power with higher efficiency.**

Keywords—consumption; design; hardware; power; system

I. INTRODUCTION

Process technologies below 90 nm make power consumption the key factor constraining electronic design [1]. To address the continuously increasing power-reduction requirements, there have been many techniques developed, such as clock gating, power gating, or voltage scaling. The current low-power design flow involves the application of such techniques in an RTL (Register Transfer Level) or lower-level digital system model (see section II). These techniques impact all aspects of integrated-circuits development (i.e. design, implementation, and verification) increasing thus the ever growing complexity of current digital systems designs even more. In order to increase the efficiency of system development, the more abstract level above the RTL, so called electronic system level (ESL), should be adopted [2]. Since the higher level offers the opportunity of more efficient system development, the low-power design flow should also utilize the system-level advantages.

This paper proposes an extension of typical low-power design flow. It moves the power-intent specification in UPF (Unified Power Format – see section II) up to the system level. The paper is organized as follows. The next section introduces the current way of power-intent specification in the digital system design process. In section III, the novel methodology for low-power design is proposed, the specification model selected for the methodology is introduced, and the proposed model extensions are briefly described. Before the conclusion, a showcase example is illustrated.

II. MEANS OF POWER-INTENT SPECIFICATION

To help designers with adoption of advanced power-reduction techniques (see Table I), two main low-power standards were developed – UPF (Unified Power Format) [3] and CPF (Common Power Format) [4]. The usage of these formats to express power intent during the SoC design process is important for the designers as well as for the EDA (Electronic Design Automation) tools and IP (Intellectual Property) components providers. These formats provide unified mechanisms for expressing the same power intent through typical design stages (e.g. functional simulation, synthesis, place and route, or formal equivalence checking) and for ensuring that the verification consistently interprets the functional semantics implied by the power intent [4].

Fig. 1 shows the typical use of UPF in low-power design flow (CPF-based flow is basically the same). The HDL (RTL) item represents an HDL (Hardware Description Language) model at the RTL; Verilog (Netlist) represents a model at lower levels – after synthesis and after place-and-route (P & R) process. As the figure shows, power-intent in the UPF is separated from the main functional model. It is preserved throughout the whole design flow and it participates in verification process (simulation or equivalence checking) of the main model.

TABLE I. OVERVIEW OF SEVERAL COMMONLY USED POWER-REDUCTION TECHNIQUES.

Technique	Description
Clock gating	Disables blocks or clock tree parts not in use.
Multiple supply voltages	Operates different blocks at different, fixed supply voltages. Signals that cross voltage domain boundaries are level-shifted.
Adaptive voltage scaling	Operates different blocks at variable supply voltages. Uses in-block monitors to determine frequency requirements, and adjusts voltage on-the-fly to satisfy them.
Dynamic voltage/frequency scaling	Operates different blocks at variable supply voltages/frequency. Uses look-up tables to adjust voltage/frequency on-the-fly to satisfy varying performance requirements. Signals that cross voltage domain boundaries are level-shifted [5].
Power gating	Turns off supply voltage to blocks not in use. Significantly reduces the leakage. Block outputs float.
Power gating with retention	Stores system state prior to power-down. Avoids complete reset at power-up, which reduces power-up/reset delay and power consumption.
State retention power gating	Stores the system state in local registers. When on standby or idling, gates the clock, and the register saves the data. State retention registers use both a continuous power supply and a switchable supply. Other logic can be powered down [5].
Save and restore power gating	As State retention power gating, but uses a memory array.

This is an accepted version of the published paper:
D. Macko and K. Jelemenská, "Managing digital-system power at the system level," 2013 Africon, Pointe-Aux-Piments, 2013, pp. 179-183.
doi: 10.1109/AFRCON.2013.6757781
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6757781&isnumber=6757586

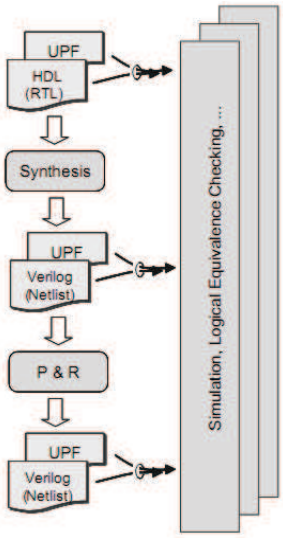

Fig. 1. Typical use of UPF in low-power design flow [3].

The ever increasing complexity makes the adoption of more abstract level above the RTL in the low-power design flow unavoidable. Despite of the undeniable benefits of the developed power standards, the specification of power intent in a separated file using different specification language, inconsistent with functional specification, is undesirable at ESL. Such an abstract system specification should be as simple as possible. For designers, it is therefore more convenient to specify the system requirements in one style and using one language, especially at the system level. All the specifications (e.g. functionality, time, or power intent) should participate in the abstraction refinement process in order to achieve target system specification. During the high-level synthesis phase, when already enough details are described, the standardized power format file (UPF or CPF) should be automatically generated in order to be used at RTL and lower levels.

III. LOW-POWER DESIGN FLOW EXTENSION

The key idea of proposed novel methodology lies in an extension of the current low-power design flow illustrated in Fig. 1 in a way that will enable to utilize the advantages of system level modeling (e.g. shorter specifications, subsystems intercommunications, or faster verification). The current design flow steps remain intact, thus the traditional design/verification methods and tools can still be used at the lower levels. The proposed low-power design flow methodology extending the current design flow to the system level is illustrated in Fig. 2.

This methodology starts from crude system specification at the ESL. The power-intent specification in the standard UPF format at such abstract level would disrupt the system specification simplicity. However, the UPF concept is not entirely omitted, but rather integrated into the system functional specification in an abstract form. The specification participates in the abstraction refinement process. The specification at respective refinement stages is verified using the verification technique, known as formal equivalence checking. Therefore, the verification can proceed continuously, even when simulation is not yet possible due to the lack of details (e.g. communication protocol with the system

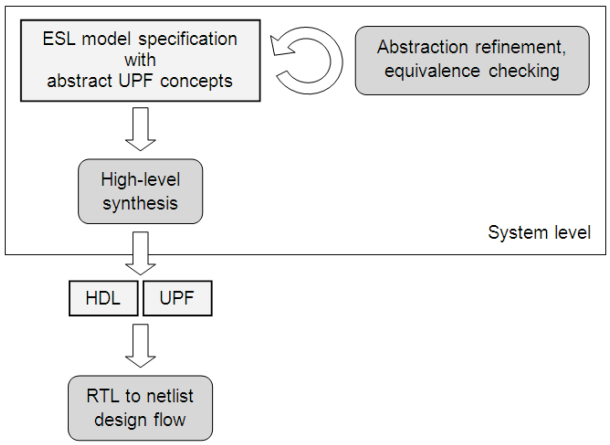

Fig. 2. Proposed low-power design flow using abstract UPF.

environment). When the specification is refined to the RTL and contains sufficiency of details, it can be converted into the RTL model (typically described in an HDL). This process is called high-level synthesis. The proposed methodology assumes that during this process the specified power intent will be extracted and the standard UPF representation will be automatically generated along with the HDL model. Then, the low-power design flow continues as shown in Fig. 1.

According to the author's knowledge, no ESL specification model is available supporting all three important concepts required by this methodology at the same time. These concepts include an abstraction refinement, formal equivalence checking, and power-intent specification. In order to put the proposed methodology into the use, a suitable system-level model has to be selected and enriched by the missing capabilities. The structure and simplicity of the specification model described in [7] makes it suitable for integration of power-intent specification since it already supports the concept of abstraction refinement along with formal verification.

A. System-level specification model

The HSSL specification language has been developed at the Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava [7, 8]. It is intended for system-level specification and modeling of reactive systems. The language supports basic concepts used for system specification in the behavioral style, such as hierarchical behavioral decomposition, behavioral refinement, behavioral reuse, timing, exception handling, concurrency and synchronization. In addition, the underlying mathematical model enables formal verification of the specification [8]. The top-level structure in the HSSL language is a system, consisting of inputs and outputs, represented by the input/output variables, the state variables representing an internal state of the system, and finally the sets of agents and processes describing the system behavior. In addition, the sets of constants and types are defined in the *System* entity (see Fig. 3).

Regarding the Fig. 3, the white color items belong to the original specification model as described in [7], the grey color items represent several additional parts of the model proposed and presented in this paper, and the grey-white color items are

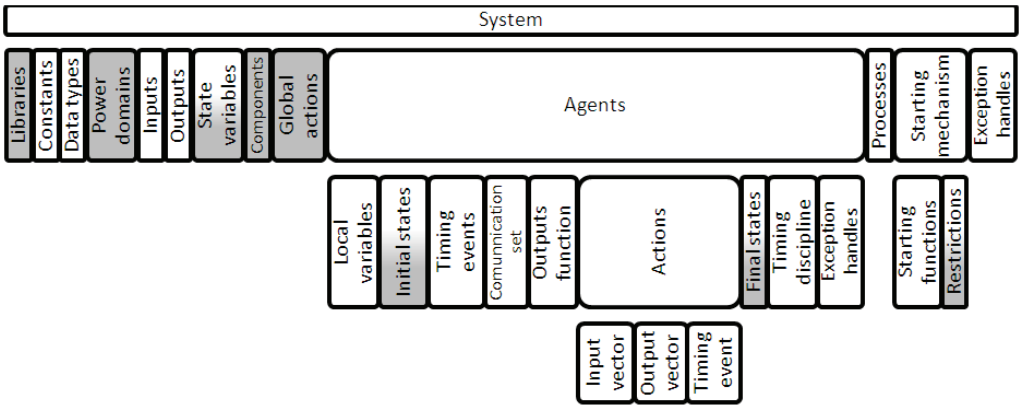

Fig. 3. The structure of HSSL's specification model.

partially extended parts with the proposed extensions also presented here.

The basic entity used to specify a partial behavior is called an agent. *Agent* describes the communication of the system with its environment between two well-defined states. The communication is represented by *communication formula* consisting of a set of actions. An *action* represents status of inputs and outputs of the specified system in particular time given by an event. A more complex behavior can be expressed by a *process* that prescribes an execution order of the agents. It allows sequential and concurrent composition of the agents, repetitive behavior, interrupt and conditional branching.

The *timing discipline* in the HSSL language is based on event-driven concept in which events represent significant discrete-time points along the continuous time interval. Except events, quantitative timing constraints can be set over the set of these events to achieve the behavior (the i/o communication) specified by the agent in more detailed and precise way. These constraints, called *timing rules*, are defined inside the agent in a way to be independent from the functional specification [8].

B. The model extensions and power-intent integration

Regarding SoC design, there were several important features missing in the model that need to be added before the actual power-intent integration. These features include structural decomposition and reuse specification. They can be achieved by adding the new entities into the specification model (as proposed in Fig. 3), namely the libraries, components, and global actions.

Library is a commonly used structure supporting reuse of previously created specifications. It is typically used in conjunction with concept of structural decomposition of the system into several communicating subsystems. This feature is very important regarding the complexity of modern highly integrated systems. The content of the library can consists of the whole subsystems, constants and defined types, or even individual generic agents. Therefore, it is possible to reuse not only the created subsystems, but also partial behaviors (agents). The support for intra-system reuse was also added by means of *global actions*. These actions can be used inside different agents of the system, thus the meaning of these actions is specified only once.

The structural decomposition requires specifying precisely which subsystems are to be used inside a system under development. For this purpose, the *components* entity was added into the model. It contains also the typical mapping of inputs and outputs of the components.

In order to integrate power intent specification into the system-level model, it is necessary to choose what details it should contain. Several UPF concepts could be used in an abstract form. One of the most important concepts of power management is so called *power domain* [9]. It is a collection of instances (components) with the same power supply. Different power supply of system blocks imply the possibility of different *power states* of these blocks. It means that an unused block can be powered-off or can be in stand-by mode while other can normally operate or operate at different voltage level. In general, it is a technique resulting in huge power savings, thus splitting the system into several power domains is one of the most important power-related decisions. For clarification, our methodology assumes that system blocks can be in the same power domain only if they are always in the same power state.

Often it is necessary to retain current state of the block that is to be powered-off. Power state which models this situation is called *off_ret*. The retention slows-down the operation (saving and loading the state values to and from the memory) or excessively increases the chip area (additional retention cells). Therefore, this concept is suitable only for blocks that are not used for longer period of time (trade-off between speed, area and power consumption). The block that needs to retain its state, but retention is not suitable, can stop its operation without being powered-off (to save at least a dynamic power). This concept is known as clock-gating. In highly abstract specification, such a concept needs to be abstracted from clock. It can be modeled by power state called *hold*. This concept can be implemented by gating the clock signal or gating the control signals of asynchronous circuits during the high-level synthesis stage (we can refer to it as block gating).

Our methodology assumes these power states:

- normal – block operates at the basic voltage level
- off – block is powered-off without state retention
- off_ret – the block state is retained before powering-off
- hold – block stops its operation but remains powered

- diff_voltageX – block operates at voltage level different from the basic one, X is just the ordinal number of voltage level (i.e. diff_voltage1 is different as well from basic voltage level as from diff_voltage2)

IV. SHOWCASE EXAMPLE

The use of such model extensions is shown on the following example. It represents a system *Sys* which consists of three interconnected components *LFSR* for data generation, *COUNT* for counting of the generated data, and *BCD* for data binary code conversion. In addition, the system contains one register for keeping the bcd count. The functionality of the model is not important for our showcase, thus the focus of analysis is targeted to power intent specification.

```
System Sys{
    Power Domains{
        PD_1(normal, off);
        PD_2(normal, diff_voltage1, hold);
        PD_3(normal, off_ret);
    }
    Inputs{ bool clk, ctrl; }
    Outputs{ data bcd_out; }
    State Vars{
        data BR(PD_3); //buffer register
        power_states PS=(normal,normal,normal);
    }
    Components{
        lfsr LFSR(PD_2);
        counter COUNT(PD_1);
        bcd_converter BCD(PD_1);
        Maps{
            LFSR.clk = clk;
            COUNT.clk = clk;
            COUNT.data_in = LFSR.data;
            BCD.clk = clk;
            BCD.data_in = COUNT.data_out;
            BR = BCD.data_out;
        }
    }
    Agent First{
        LV{ bool sig; }
        DefaultTE: up(clk, 1);
        CS{ Start.Final }
        Action Start{ iv:sig=ctrl; }
        Action Final{ }
        FS{
            PS= if(sig==1) (off, hold, off_ret)
            else (normal,diff_voltage1,normal);
        }
    }
    Agent Second{
        DefaultTE: up(clk, 1);
        CS{ Start.Final }
        Action Start{ iv:sig=1; }
        Action Final{ ov:bcd_out=BR; }
    }

    Process Global{
        loop { do First while (ctrl == 1); Second }
    }
    Starting Structure{ Stf Global: eon; }
}
```

The power domains are assigned to the subsystem instances in the components entity of the model and they can be also assigned to the state variables of the system. These domains are represented by a set of power states which the domain blocks can be in. These are specified in the entity power domains. The subsystem *LFSR* is assigned to the power domain *PD_2* which supports *normal, diff_voltage1, hold* power states. Instances *COUNT* and *BCD* are assigned to *PD_1* power domain, meaning they can be both in *normal* or both in *off* state at the same time. Register *BR* is in *PD_3* domain, thus it works either in *normal* state or it can be powered off, but its state is retained.

Power states of the entire system can be specified in a way similar to other system state variables. A special variable type is used, called *power_states*. Such variable has to contain its name and initial states for all power domains in the specified order. The power states changes are modeled in *final state* of an agent. In our example, the initial power state is *(normal, normal, normal)*. The agent *First* changes this state according to input variable value of *ctrl*. If the *ctrl* value is 1, then the power states change to *(off, hold, off_ret)*. It means that blocks in *PD_1* domain are powered-off without the state retention, operation of blocks in *PD_2* is stopped, and the state of the sequential registers in *PD_3* is retained before blocks in this domain are powered-off.

During the phase of high-level synthesis, the power intent is extracted from the functional model and the UPF standard file is generated. The code below shows power-intent specification extracted from our example.

Firstly, the power domains need to be created. The top-level power domain is added which represents the domain of the entire system.

```
create_power_domain PD_top
create_power_domain PD_1 -elements {COUNT BCD}
create_power_domain PD_2 -elements {LFSR}
create_power_domain PD_3 -elements {BR}
```

The second step is to create supply ports, create supply nets, connect the nets with ports, and set the primary supply nets for power domains. Beside the main supply, there is additional port *VDD2_port* generated as a result of *diff_voltage1* state of *PD_2* power domain.

```
create_supply_port VDD_port -domain PD_top
create_supply_port VSS_port -domain PD_top
create_supply_port VDD2_port -domain PD_top
create_supply_net VDD_top -domain PD_top
create_supply_net VSS_top -domain PD_top
create_supply_net VPD1 -domain PD_1
create_supply_net VSS1 -domain PD_1
create_supply_net VPD2 -domain PD_2
create_supply_net VSS2 -domain PD_2
create_supply_net VPD3 -domain PD_3
create_supply_net VSS3 -domain PD_3
connect_supply_net VDD_top -ports {VDD_port}
connect_supply_net VSS_top -ports {VSS_port}
connect_supply_net VSS1 -ports {VSS_port}
connect_supply_net VPD2_top -ports {VDD2_port}
connect_supply_net VSS2 -ports {VSS_port}
connect_supply_net VSS3 -ports {VSS_port}
set_domain_supply_net PD_top -primary_power_net VDD_top -primary_ground_net VSS_top
set_domain_supply_net PD_1 -primary_power_net VPD1 -primary_ground_net VSS1
```

set_domain_supply_net PD_2 -primary_power_net VPD2 -primary_ground_net VSS2
set_domain_supply_net PD_3 -primary_power_net VPD3 -primary_ground_net VSS3

The domains containing the blocks that can be powered-off or that work at several voltage levels need power switches. The signals controlling these switches are added into the functional model. In our example, control signals *PD_1_SW_ctrl* and *PD_3_SW_ctrl* have the same value as *ctrl* input variable. *PD_2_SW_ctrl* value is inverted from *ctrl*.

create_power_switch PD_1_SW -domain PD_1 -input_supply_port {vin VDD_top} -output_supply_port {vout VPD1} -control_port {ctrl_sig PD_1_SW_ctrl} -on_state {PD_1_ON vin {!ctr_sig}} -off_state {PD_1_OFF {ctr_sig}}
create_power_switch PD_2_SW -domain PD_2 -input_supply_port {vin1 VDD_top} -input_supply_port {vin2 VPD2_top} -output_supply_port {vout VPD2} -control_port {ctrl_sig PD_2_SW_ctrl} -on_state {PD_2_ON vin1 {!ctr_sig}} -on_state {PD_2_DIFF vin2 {ctr_sig}}
create_power_switch PD_3_SW -domain PD_3 -input_supply_port {vin VDD_top} -output_supply_port {vout VPD3} -control_port {ctr_sig PD_3_SW_ctrl} -on_state {PD_3_ON vin {!ctr_sig}} -off_state {PD_3_OFF {ctr_sig}}

In the communication with the block that can be powered-off, signals (inputs and outputs of the subsystem) have to be isolated. The isolation is necessary to prevent floating values between domains. Between blocks that work at different voltages, the voltage level has to be shifted. The level shifters change the signal between these blocks from one voltage level to another. Since this additional logic is necessary in some determinable block interfaces, there is no need to specify it explicitly. They are generated as follows. Inputs and outputs of the block in power domain *PD_1* are isolated and inputs/outputs belonging to block in *PD_2* are level-shifted.

set_isolation isolation_1 -domain PD_1 -isolation_power_net VDD_top -isolation_ground_net VSS_top -clamp_value latch -applies_to both
set_isolation_control isolation_1 -domain PD_1 -isolation_signal iso1 -isolation_sense high
set_level_shifter shifter_2 –domain PD_2

The following command provides the state retention for all sequential cells in *PD_3*.

set_retention retention_3 -domain PD_3 -retention_power_net VDD_top -retention_ground_net VSS_top
set_retention_control retention_3 -domain PD_3 -save_signal {save_reg high} -restore_signal {restore_reg high}

The power state *hold* representing the clock/block gating cannot be specified in UPF. High-level synthesis implements this technique directly into the functional model as stated before.

V. Conclusions and Further Work

The paper is devoted to the problem of power management at the system level. In section III, the extension of current low-power design flow to the system level is proposed as well as the method of specifying the power-management intent in the early stage of the design process. Such an approach helps the designer to make more suitable architectural decisions and therefore to have more impact on eventual power consumption. The method expects the functional and power-architecture models to be unified in order to keep the system specification as simple as possible (including one specification model and one specification style). The subsections A and B discuss shortly the selected specification model and the required specification model extensions. The proposed extensions help to utilize the concepts of structural decomposition, reuse, domains, retention and block gating in a highly-abstracted form. The example in section IV illustrates the usefulness of this methodology. The power-reduction techniques such as clock gating (block gating), voltage scaling and power gating (with or without retention) can be specified directly in the system-level model and in a concise manner. The extended methodology for low-power design only supplements the existing one. There are other efficient power-reduction techniques that cannot be applied at the system level because of their conjunction with lower levels, e.g. restructuring of the logic circuit structure, transistor resizing, or substrate biasing [6].

The further work will include the development of an evaluation technique that could help designer to decide between several possible power-architectures without the need for simulation. Such a technique along with the methodology proposed in this paper would result in more effective low-power design flow (faster design process with fewer re-spins, reduced power consumption, faster verification). The feasibility of the methodology will be showed on a case study – effect on power consumption.

Acknowledgment

This work was partially supported by the Slovak Science Grant Agency (VEGA 1/1008/12 “Optimization of low-power design of digital and mixed integrated systems“) and COST Action IC 1103 MEDIAN.

References

[1] S. Bailey, G. Chidolue and A. Crone, “Low power design and verification techniques,” Mentor Graphics, 2007.

[2] The International Technology Roadmap for Semiconductors: Design. ITRS, 2011 edition, 2011.

[3] IEEE Standard for Design and Verification of Low Power Integrated Circuits. IEEE, 2009. IEEE Std 1801-2009.

[4] S. Carver, A. Mathur, L. Sharma, P. Subbarao, S. Urish and Q. Wang, “Low-power design using the Si2 common power format,” IEEE Design & Test of Computers, vol. 29, no. 2, pp. 62-70, 2012.

[5] R. Goering, “Low power design,” [Online; accessed January 13, 2013]. http://www.leepr.com/PDF/SCDsource_STR_LowPower.pdf

[6] Power Forward Initiative, A Practical Guide to Low Power Design: User Experience with CPF. Power Forward, 2012.

[7] N. Fristacky, J. Kacerik, T. Bartos and M. Kardos, “Behavioral specification model and language for digital systems,” Slovak University of Technology, 2000. Technical report.

[8] N. Fristacky, J. Kacerik and T. Bartos, “A mixed event-value based specification model for reactive systems,” in SoC Methodologies and design Languages, Kluwer academic publishers, 2001.

[9] F. Bembaron, S. Kakkar, R. Mukherjee, A. Srivastava, “Low power verification methodology using UPF,” Proc. of Design & Verification Conference & Exhibition (DVCon), 2009, pp. 228-233.