Power Intent from Initial ESL Prototypes: Extracting Power Management Parameters*

David Lemma[1] Mehran Goli[1] Daniel Große[1,2] Rolf Drechsler[1,2]
[1]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[2]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
{lemma,mehran,grosse,drechsle}@informatik.uni-bremen.de

***Abstract*—The increase in digital circuit complexity not only stems from sophisticated functionality, but also from power concerns. Power concerns are addressed via the realization of power intent (the specification for power). Unlike functional specifications, power intent is generally implicit within an initial ESL Prototype. For power intent to become explicit, it needs to be expressed in terms of Power Management parameters. These parameters are major indicators of the efforts involved in realizing the power intent.**

We introduce an automated method to extract two Power Management parameters (number of Control Signals and Power Modes) from ESL prototypes. These parameters are extracted in a two-step process. First, relevant structural and behavioral information of the prototype is retrieved and translated into an activity profile. Following this, an analysis is performed on the activity profile to extract the power management parameters. The effectiveness and efficiency of the method is demonstrated by its application on several ESL benchmarks.

I. Introduction

The influence of power concerns has grown to such an extent that *Design Space Exploration* (DSE) at system level is now heavily based on power/performance tradeoff [1]. The architectural choices need to comply with a power budget as well as with the functional requirements, without increasing the complexity. One way to approach the challenge posed by above scenario is through *Power Aware Design*, which is a set of methodologies to fully incorporate power concerns into the design workflow [2].

At the *Electronic System Level* (ESL), development is usually done on a SystemC-based *Virtual Prototype* (VP) [3]. A VP is basically a simulatable model of the described digital hardware, which allows for an estimation of the impact of possible design options arising from the DSE process. Within these design options there are usually different approaches to manage the power concerns.

It has been shown that reducing the power consumption of a SoC (to meet the power budget) is easier to achieve at higher levels of abstraction during initial design phases [4]. Hence, ESL is a natural choice for the evaluation of different approaches to manage the power concerns. However, the traditional workflow at the ESL is functionality based, so an updated power aware workflow is needed.

*This work was supported in part by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative, and by the German Academic Exchange Service (DAAD).

978-1-5386-7656-1/18/$31.00 ©2018 IEEE

Customarily, the DSE process focuses on how power concerns impact an initial functionality focused VP [5], [6]. However, initial functionality also impacts the way power concerns should be managed. The extent and characteristics of this latter impact are, unfortunately, implicit within the initial VP. The power intent (the specification for power) is the means by which this impact is made explicit.

More precisely, power intent has been referred to as describing "the partitioning of a design into power domains" as well as "the control signals that are used to control these power domains" [7]. Thus, knowing these elements (power domains, control signals), plays an important role in the architecture realizing the power intent [8].

The number of *Control Signals* (CS) and the number of *Power Modes* (PM) are the most descriptive power management parameters. The former control the power domains (to perform clock/power gating or voltage switching, among others). The latter indicate the finite states of the system, (e.g. Normal Mode, Performance Mode, Initialization Mode, Idle Mode and etc.).

The number of CS and PM typically define the *Power Management Unit* (PMU), which realizes the power intent by managing the switching of the power domains into different states to reach the desired power modes. The PMU itself is commonly specified following the *Unified Power Format* (UPF) standard [9], which offers fully synthesizable semantics for the power management logic. Therefore the aforementioned parameters allow designers to estimate the effort required for the realization of the power intent. To estimate the effect of realizing the power intent, two main challenges must be overcome:

- revealing the implicit impact of the VP functionality on the power intent and
- estimating the effort required to realize the power intent via the PMU.

The first challenge is addressed by the process of Design Understanding. The second challenge is overcome by extracting the power management parameters which have a direct impact on both the PMU power overhead [10], [11] and the time and effort that PMU verification will require [12], [13], [14].

In this paper, we present an automated, non-intrusive method to extract the aforementioned power management parameters. The method comprises two steps. In the first step, we take advantage of the *GNU Debugger* (GDB) to extract the run-time behavior of the VP and translate it into an activity

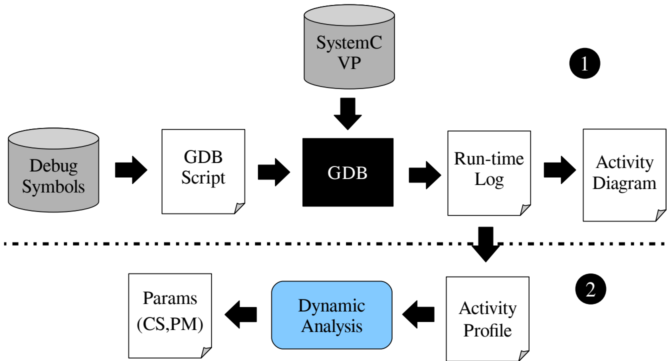

Fig. 1: Flow of the proposed method

profile. In the second step, to make the implicit impact of functionality on power concerns actually explicit, the activity profile produced in the first step are analyzed to extract the number of CS and PM, which will be used for the PMU.

The remainder of the paper is structured as follows. Section II briefly discusses the basis for the challenges. Section III presents the two steps of our method, while Section IV shows the results of applying the proposed method on several ESL prototypes. Section V concludes the paper.

II. Preliminaries

At the ESL, the de facto VP modeling language is *SystemC* (in essence, a library of classes on top of C++), which allows the simulation of concurrent events. In addition to being modeled using SystemC [15], VPs are typically created (but not always) following *Transaction-Level Modeling* (TLM) approach [16]. Regardless of the possible use of TLM, the inspection of VPs requires a design understanding process to be conducted on the VP code.

The static and dynamic information of a VP is revealed by the way its various components have been brought together. Both the static and dynamic facts of a prototype and their relation can be condensed into an activity profile. Although, the activity profile represents the complete structure of a given VP, it does not provide the designers with relevant information to unveil the power intent. To unveil the power intent, there is the need of tools that appropriately parse the VP to extract power management parameters defining such power intent.

Since the power intent is usually realized through a Power Management Unit (PMU), the power management parameters specify the PMU's structure. These parameters are the number of Control Signals (CS) and Power Modes (PM) and they are the major indicators of the required effort to realize they power intent. It is therefore extremely convenient to extract these parameters from the VP as early as possible.

Given the convenience derived from knowing the aforementioned power management parameters, we present a method to extract them from ESL prototypes in an efficient automated manner. With our method, power intent is unveiled early in the design process. Additionally the realization efforts of the power intent become discernible.

III. The proposed method

As illustrated in Fig. 1, the proposed method consists of two steps:

1) generating activity profile from VP design's simulation log using GDB and
2) extracting the power management parameters (CS and PM) by analyzing the generated activity profile.

In the following, each step of the proposed method is explained in detail and illustrated using a motivating example *Pipe* design [3]. The Pipe example is (as it name suggest) a simple pipelined System C VP with three stages performing successive algebraic operations (addition, subtraction, division, exponentiation) on a self generated input. It contains four modules (`numgen`, `stage1`, `stage2`, `stage3`) with the latter three (`stage1`, `stage2` and `stage3`) computing the mathematical operations on the input generated by the former (`numgen`). For a closer look at the design, the reader is referred to Fig. 2, which shows a part of the code for its structure.

A. Extracting Activity Profile

In order to extract the activity profile, we take advantage of GDB to access the run-time behavior of a given VP model. This first step is shown in the top part of Fig. 1. It consists of two input blocks (`SystemC VP`) and `Debug Symbols`), one control file (`GDB Script`), one application (`GDB`) and three output files (`Run-time Log`, `Activity Diagram` and `Activity Profile`). Our analysis in this step is inspired by the design understanding approach in [17], but unlike it, we do not retrieve the whole state of variables or transactions in the VP, as it is not necessary.

Rather, we program GDB to only retrieve and log the functions' activity of all modules' instances. To do this, we first analyze the debug symbols (seen on the left of Fig. 1) of the model to retrieve the static information. This information includes the modules' name and their corresponding member functions and methods. We use this information to program the debugger (creating a script command file for GDB) in order to trace its functions' activities at run-time. This is depicted in Fig. 1 by the blocks `GDB Script` and `GDB`.

The tracing process is performed by setting breakpoints at the beginning of each module's function in the DB script file. During execution of the SystemC VP (represented in Fig. 1) by the uppermost block of the same name whenever the function is activated, its corresponding breakpoint stops the execution and the state of execution (e.g. the root and instance name of modules that this function belongs to) is logged in the *run-time Log* file, which is represented in Fig. 1 by the block of the same at the right hand side.

Execution is continued and as the next breakpoints are hit, vast amount of information is extracted. After the execution of the VP model is completed and the *run-time Log* file is generated, it is then translated into a structural model, the *Activity Profile* (represented in Fig. 1) by the right hand side block of the same name), where activities of modules' functions are presented with regards to the simulation time.

```
1  struct stage1 : sc_module {
2  //...
3  void stage1::addsub() {
4   double a, b;
5   a = in1.read();
6   b = in2.read();
7   sum.write(a+b);
8   diff.write(a-b); }
9  struct stage2 : sc_module {
10 //...
11 struct stage3 : sc_module {
12 //...
13 void stage3::power() {
14  double a, b, c;
15  a = prod.read();
16  b = quot.read();
17  c = (a>0 && b>0)? pow(a, b) : 0.0;
18  powr.write(c); }
19 struct numgen : sc_module {
20 //...
21 void numgen::generate() {
22  static double a = 134.56;
23  static double b = 98.24;
24  a -= 1.5;
25  b -= 2.8;
26  out1.write(a);
27  out2.write(b); }
28 //...
```

Fig. 2: Part of *Pipe* design implemented in SystemC

Formally, the function's activity (FA) of a module in the *Activity Profile* is defined as follows:

Definition: $FA = (M, I, F, T)$ in which:

- T is a list of discrete timestamps from 0 to the end of the execution time of the prototype,
- M is the root name of the module,
- I is the instance name (or number) of module M and
- F is the function of the module M being executed.

Finally, to give a better overview of the designs' behavior, an activity diagram (rightmost block in Fig. 1) of the entire system is also generated from *Run-time Log*.

Fig. 3 and Fig. 4 shows a portion of the activity profile and diagram of the *Pipe* design (motivating example), using the default workload scenario (which provides full functional coverage), respectively. In Fig. 3, the extracted information is presented based on set of sequence numbers. Each sequence includes the information related to the function of module's instance that is activated during execution run followed by name of module and function, instance number of module and the simulation time stamp. In Fig. 4, the *ST* and *Name* axis show the execution time (simulation time stamps) and the name active components (including function's name and its module's root and instance number), respectively. As an instance, it shows that function `power` of module `stage3` with instance number `0x7fffd62` is only activated at the period of [1000, 2000] and [3000, 4000] during the execution.

Please note that, the activity profile is generated based on each functionality of the design. The term functionality refers to the tasks that a given ESL design is supposed to perform. For example, the entire *Pipe* design has only one functionality. It receives the inputs and performs three stages computation to obtain the final result.

```
1  Seq-0: [sc_main, NULL, 0ns]
2  Seq-1: [numgen::generate, 0x7fffd0, 0ns]
3  Seq-2: [stage1::addsub, 0x7fffd10', 0ns]
4  ...
5  Seq-4: [stage3::power, '0x7fffd62, 1000ns]
6  ...
```

Fig. 3: Part of generated *Pipe* design's activity profile

B. Power management parameter (CS,PM) extraction

In this step, the *Activity Profile* is analyzed via the *Dynamic Analysis* block in the lowest half of (Fig. 1, step 2) to extract the power management parameters (number of CS and PM). To extract the number of PM, an unique time-based pattern identification is performed within the activity profile. To obtain the number of CS, the number of coinciding modules per identified pattern is retrieved. There are then two algorithms which we explain in the next paragraphs.In this explanation we read and refer to the elements of the activity profile on a module basis (instead of on a function basis) for simplicity purposes, as there is only one function per module.

To extract the number of PM, the principle guiding the analysis is unique pattern identification. Looking at the activity profile in Fig. 4, we notice two distinct and unique time-based patterns. As mentioned in the previous step, the activity profile reveals patterns of activity for the modules. As an instance, we can see in Fig. 4 that modules `stage1`, `stage2` and `numgen` share the same active/inactive periods. We name this pattern `PM1`. However, `stage3` has different activity periods, starting from simulation time 1000. We name this pattern `PM2`.

To extract the number of CS, the number of coinciding modules per identified PM pattern is retrieved. For this retrieval, the underlying assumption is that a module is either active or inactive at any given timestamp. If any two modules are simultaneously active (or inactive) for each and every identified time-based pattern, then they can be controlled by the same CS. From the inspection of Fig. 4, we see that `stage2`, `stage1` and `numgen` are active for `PM1`, but not for `PM2`. These modules shall be controlled by `CS1`. Module `Stage3` is active for PM1 and inactive for `PM2`, which means it shall be controlled by a different CS (`CS2`).

The result of applying the algorithm for both CS and PM is visible in the outer Y and X axis of Fig. 4, respectively. The parameters `CS1` and `CS2` are depicted as governing *stage3* and *stage2*, *stage1* and *numgen*, respectively. At timestamp 0, the PM (shown under the timestamp in the X axis) is `PM1`, with `stage2`, `stage1` and `numgen` being active and `stage3` inactive. This PM is reached when CS1 is on and CS2 is off.

At timestamp 1000 and at intervals of 2000 units of time thereafter, the PM is `PM2`, for which only `stage3` is active (which means that `CS1` is off and CS2 is on). At timestamp 2000 and at intervals of 2000 units of time thereafter, the PM is again `PM1`. The overall repeating pattern shown in Fig. 4 is repeated throughout the entire execution. Please note that we eliminate *sc_main* function form our analysis (since it is a SystemC artifact and not a true synthesizable module of any

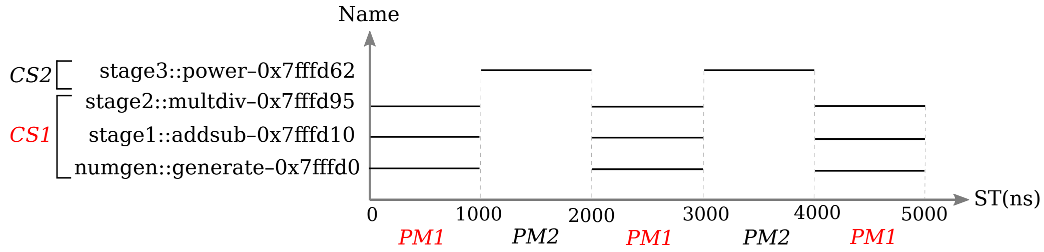

Fig. 4: Part of the activity profile of the 3-stage Pipe example

given design), which reduces the number of CS retrieved and simplifies the identification of PM.

A further reduction in the number of CS retrieved can happen if the corresponding algorithm uses a threshold value to assess whether modules coincide in each identified PM pattern. Such a threshold allows for a strict or a lax application of the process to assess coinciding modules per identified PM pattern. In essence, when the threshold is 0 (strict application), the algorithm will group certain modules under a single CS only if those modules' active or inactive status is the exact same for all time stamps.

IV. Experimental Evaluation

In this section we present the experimental evaluation of our proposed method. First, the results of applying the proposed method to several ESL prototypes are presented in section IV-A. Second, to illustrate the benefits of the method, the extracted power management parameters are used to select the best alternative between two implementations of a Hamming Encoding/Decoding system in section IV-B. Finally, we give briefly discuss the characteristics of the method based on the obtained results in IV-C.

A. Results for ESL prototypes

Table I shows the experimental results of all case studies (ESL prototypes). The first two columns list the type and name of each ESL prototype, respectively. Column *LoC* presents the prototypes' lines of code. Column *#Func* shows the number of functions that have been executed for each instance of a module. Columns *#CS* and *#PM* list the number of extracted control signals and power mode for each design, respectively. The execution time of the proposed method is reported in column *ET* including the extraction of the activity profile *ET* (step 1), the extraction of the power management parameters *P2* (step 2) and total execution time *Total* of the method (the sum of step 1 and step 2) Column *CET* shows the time required for the standard compilation (done via GCC) and execution of the ESL prototype without any of the method's steps being applied. In summary, Table I shows that for a given ESL prototype the proposed method is able to extract the power management parameters in a reasonable time frame in comparison to *CET*. The extracted power management parameters, together with the activity profile from which they are extracted, effectively help designers estimate the effort required for power intent realization of existing or third party ESL Prototype during the early design phases.

B. Hamming Encoding/Decoding system prototypes: design alternatives

In this section, we show how the proposed method can be used for a task in Design Space Exploration task – selecting the best (having the lowest realization effort) VP between two alternatives with the same functionality. Both *Hamming-comb* and *Hamming-seq* VPs provide designers with encoding/decoding for the (15,4) Hamming code. The difference is that *Hamming-seq* is a sequential circuit, while *Hamming-comb* is combinational. A part of generated activity diagrams of both *Hamming-seq* and *Hamming-comb* are shown in Fig. 5 and Fig. 6, respectively. Comparing both diagrams shows that the number of extracted functions and their activities during execution are different in both VPs. This difference results in different parameters (depicted in Table I) leading to two distinct efforts to realize the power intent (i.e. the power management parameters lead to two different PMUs).

We assume that a designer uses a PMU to realize the power intent. In such a situation, the number of CS plays an important role in the realization effort required. At a level like the ESL. an extra two CS for a given prototype when compared to another may not seem significant, but it is important as the power intent is realized via the PMU. The reason for this is that a *Control Signal* typically entails more *Register Transfer Level* (RTL) management logic, which increases the area overhead of the PMU, as well as making its verification more time consuming. These factors underscore the impact that a modest increase in number of *Control Signals* in an ESL prototype can have on the efforts required to realize power intent.

C. Discussion

As our analysis depends on the complete activity of all design's functions, the testcases for each VP must provide high functional coverage. We believe that this assumption is fair due to the following reason. In practice, very often (functional) coverage of a VP is measured to ensure that each functionality has been exercised at least by one testcase. Essentially, high coverage is an indicator showing the VP works properly. If the coverage is initially insufficient, more tests need to be added, either manually or by employing an automated test creator (which is out of scope of this paper). The activity profile is generated for each functionality of a design, meaning that we consider every possible combination of functions' activities within the design. Thus, once the number of testcases reaches this coverage, further increasing the number does not have an

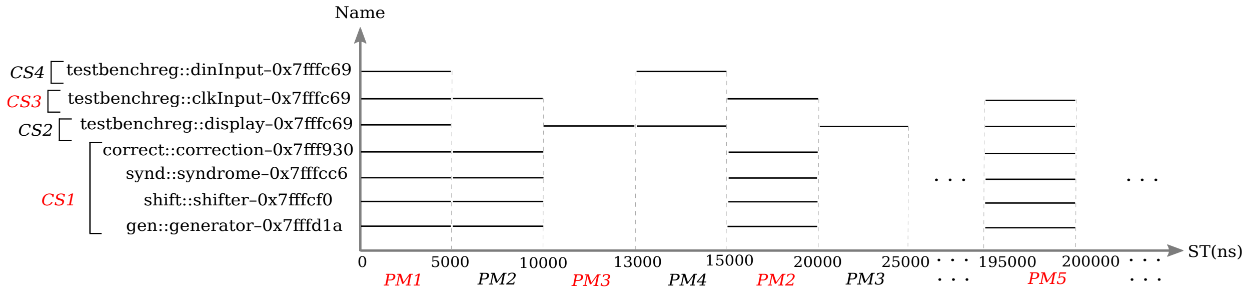

Fig. 5: Part of the activity profile for Hamming-seq

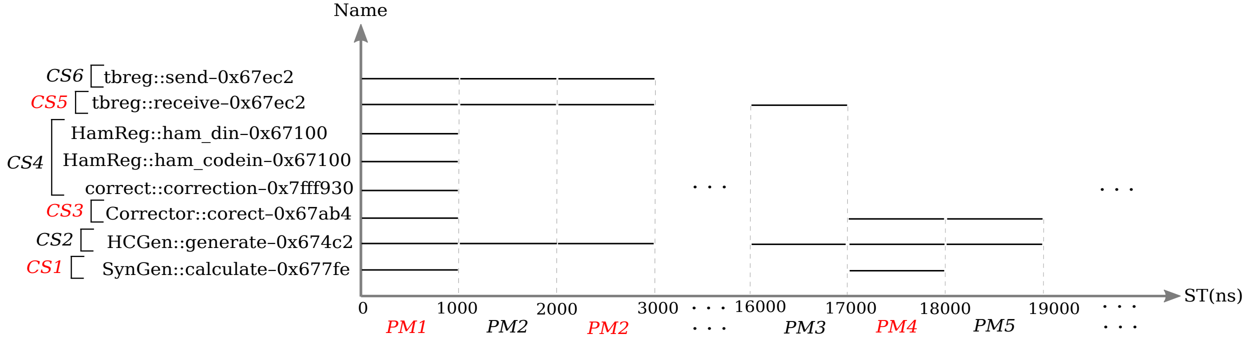

Fig. 6: Part of the activity profile for Hamming-comb

TABLE I: Experimental Results for all ESL prototypes

Type	ESL Design	LoC	#Func	#CS	#PM	ET (s) Step1	ET (s) Step2	ET (s) Total	CET (s)
SystemC	3-stage Pipe[1]	211	4	2	2	3.9	0.1	4	3.5
	Hamming-seq[2]	368	7	4	5	8	0.1	8.1	4.9
	Hamming-comb[3]	563	8	6	5	9	0.2	9.2	6.9
	FIR Filter[1]	834	3	3	4	5.3	0.5	5.8	4.2
	VGA Controller[4]	856	4	3	3	16.8	1.1	17.9	3.8
	Packet Switch[1]	1020	10	10	74	11.4	1.9	13.3	8.5
	RISC CPU[1]	1960	12	7	18	13	2.1	15.1	12.5
	Simple Bus[1]	2100	9	7	8	43	2.6	45.6	5.1
TLM-2.0	Example-5[5]	650	21	14	15	49.3	1.2	50.5	2.1
	Example-6[5]	713	36	34	152	51.7	1.2	52.9	2.2
	AT-example[5]	2942	41	29	19	117.6	3.3	120.9	21
	Locking-two[5]	3831	42	35	32	139.4	3.6	143	24.3

[1]Provided by [3] [2]Provided by [18] [3]Provided by [19] [4]Provided by [20] [5]Provided by [16] **LoC**: Line of Code **#Func**: number of executed Functions in active instances of modules **#CS**: number of Control Signals **#PM**: number of Power Modes **ET**: Execution Time **Step1**: step1 of the proposed method **Step2**: step2 of the proposed method **CET**: Standard Compilation and Execution Time by GCC without applying the method.

effect on the results of our analysis. Extra testcases will just cover a given functionality pattern that is already known.

It is also important to highlight that the level of granularity at which the method works (targeting functions within instances of modules) can be changed. As a rule, for every coding style (Cycle Accurate, Approximately Timed, Loosely Timed) used in SystemC designs, the finer the granularity the more accurate the results yielded by the method. In addition, the method working with a finer granularity potentially enables the DSE process to address power concerns in a more sophisticated manner. A finer granularity can, for instance, enable the use of power reduction techniques such as *Dynamic Voltage Frequency Scaling* (DVFS) when realizing the power intent via the PMU.

V. Conclusion

In this paper, an automated method for the extraction of power management parameters from initial ESL prototypes has been presented. The extracted power management parameters are major indicators of the effort required to realize the power intent from ESL prototypes. The method consists of two steps that enable designers to make the implicit power intent of a prototype explicit via the extracted power management parameters. Several ESL benchmarks have been run to evaluate the effectiveness of the proposed method.

We plan to extend our research to use the extracted power management parameters to produce ready to fill templates for the PMUs. This extension would prove the usefulness of knowing the parameters in early design stages.

REFERENCES

[1] T. Xie and B. Wilamowski, "Recent advances in power aware design," in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*. IEEE, 2011, pp. 4632–4635.

[2] O. S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1055–1069, 2003.

[3] Accellera Systems Initiative. OSCI SystemC 2.3. [Online]. Available: http://www.accellera.org/

[4] Y. Samei, "Automated power-aware system-level design with the mavo framework," Ph.D. dissertation, University of California, Irvine, 2014.

[5] O. Mbarek, A. Khecharem, A. Pegatoquet, and M. Auguin, "Using model driven engineering to reliably accelerate early low power intent exploration for a system-on-chip design," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. ACM, 2012, pp. 1580–1587.

[6] 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, 2017.

[7] E. Sperling. Defining Power Intent. Semiconductor Engineering. [Online]. Available: https://semiengineering.com/defining-power-intent/

[8] D. Lemma, D. Große, and R. Drechsler, "Natural language based power domain partitioning," in *2018 IEEE 21st International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2018, pp. 101–106.

[9] IEEE, "IEEE standard for design and verification of low-power, energy-aware electronic systems - 1801-2015," 2016.

[10] D. Macko, K. Jelemenská, and P. Cicák, "Power-efficient power-management logic," in *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014 24th International Workshop on*. IEEE, 2014, pp. 1–7.

[11] D. Macko, "Rapid power-management exploration using post-processing of the system-level simulation results," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 2017, pp. 1–6.

[12] C. C.-H. Hsu and C. H.-P. Wen, "Speeding up power verification by merging equivalent power domains in rtl design with upf," in *Test Conference in Asia (ITC-Asia), 2017 International*. IEEE, 2017, pp. 168–173.

[13] A. Hazra, R. Mukherjee, P. Dasgupta, A. Pal, K. M. Harer, A. Banerjee, and S. Mukherjee, "Power-tructor: An integrated tool flow for formal verification and coverage of architectural power intent," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 11, pp. 1801–1813, 2013.

[14] V. Herdt, H. M. Le, D. Große, and R. Drechsler, "Towards early validation of firmware-based power management using virtual prototypes: A constrained random approach," in *Forum on Specification and Design Languages*, 2017, pp. 1–8.

[15] D. Große and R. Drechsler, *Quality-Driven SystemC Design*. Springer, 2010.

[16] J. Aynsley, "SystemC TLM-2.0 base protocol checker," https://www.doulos.com/knowhow/systemc/tlm2, accessed: 2018-01-30.

[17] M. Goli, J. Stoppe, and R. Drechsler, "Aiba: An automated intra-cycle behavioral analysis for systemc-based design exploration," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*. IEEE, 2016, pp. 360–363.

[18] Hamming-Code-SystemC. [Online]. Available: https://github.com/SamTheDev/Hamming-Code-SystemC

[19] HammingCodeForFPGA. [Online]. Available: https://github.com/ChDuhil/HammingCodeForFPGA

[20] B. C. Schafer and A. Mahapatra, "S2CBench: Synthesizable SystemC benchmark suite for high-level," *IEEE Embedded Systems Letters*, no. 3, pp. 53–56, 2014.