Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2012, Article ID 439727, 12 pages
doi:10.1155/2012/439727

Research Article

Modeling and Implementation of a Power Estimation Methodology for SystemC

Matthias Kuehnle,[1] Andre Wagner,[1] Alisson V. Brito,[2] and Juergen Becker[1]

[1] *Institute for Information Processing Technology, KIT, 7602 Karlsruhe, Germany*
[2] *Department of Informatics, Federal University of Paraiba (UFPB), 58051-900 João Pessoa, PB, Brazil*

Correspondence should be addressed to Alisson V. Brito, alissonbrito@dce.ufpb.br

Received 19 March 2012; Revised 12 May 2012; Accepted 18 June 2012

Academic Editor: Massimo Conti

This work describes a methodology to model power consumption of logic modules. A detailed mathematical model is presented and incorporated in a tool for translation of models written in VHDL to SystemC. The functionality for implicit power monitoring and estimation is inserted at module translation. The translation further implements an approach to wrap RTL to TLM interfaces so that the translated module can be connected to a system-level simulator. The power analysis is based on a statistical model of the underlying HW structure and an analysis of input data. The flexibility of the C++ syntax is exploited, to integrate the power evaluation technique. The accuracy and speed-up of the approach are illustrated and compared to a conventional power analysis flow using PPR simulation, based on Xilinx technology.

1. Introduction

The need for more abstract system on chip development techniques is evident due to rising system complexity. Consequently, accurate system evaluation in less time will increase the productivity. According to the Semiconductor roadmap, especially the consideration of energy consumption is becoming more important and is also a limiting factor for many applications [1, 2]. Modeling strategies are driven by system and software engineers on the one hand and hardware engineers on the other hand. The first group develops RTL models written in hardware description languages (HDLs) since they are the basis for synthesis tools. The second group uses transaction level models (TLMS), most commonly written in SystemC [3] since these models enable fast system simulation. SystemC is a library based on the object-oriented programming language C++. A TLM specification extends SystemC to separate communication from computation. TLM improves modeling and simulation speed. The simulation speed depends on the level of abstraction [4]. Also, modeling at different abstraction levels is possible. This increases the flexibility of SystemC.

A remaining problem is the trade-off between accuracy and simulation speed and with that, the link and synchronization between the two layers. Translation tools are solving this problem to some extent. They inherit some limitations in the translation of syntax constructs that do not have direct counterparts. The presented tool extends this feature list. The main goal of this work, however, is the integration of a power analysis methodology into the translation process. The power estimation methodology estimates switching activities in DSP units such as adders or multipliers according to actual input data. The approach is based on a statistical methodology. It implements the measurement functionality implicitly into the SystemC model by defining overloaded operators, in the sense of object-oriented programming. These can be differently characterized based on technology parameters. The operators can be automatically integrated at system translation. DSP units are considered, since a power analysis of such systems shows that the major part of the dynamic power dissipation is consumed in the data-processing part of the architecture. In addition, the power dissipation is highly data dependent. Therefore a fast but reasonably accurate estimation of the dynamic

power dissipation of such data-driven kernels is of high interest and the analysis using representative input data is essential. From the hardware point of view, the additional flexibility inherent in reconfigurable architecture and easy-to-use implementation flows make reconfigurable computing, especially FPGA architectures, attractive for power-aware computing solutions. Since CPUs are not sufficiently energy efficient (power consumption up to 200 W) and, on the other hand, ASICS are unaffordable for a low market volume, reconfigurable computing is considered an alternative especially for data-driven applications. Their benefit has been shown, for example, in [5, 6].

The described situation in the system development landscape motivates the development of a strategy for abstract, hence faster data-dependent power analysis for LUT-(lookup-table-) based systems in this work. The results were presented in [7] and are compared here to a standard power estimation flow. Beside it a detailed mathematical modeling strategy for power consumption estimation is presented. To ease and accelerate the development cycle, the power estimation method is embedded in a translation tool, so that the process of power estimation is transparent for the user and can be validated.

The remainder of this work is structured as follows. Section 2 summarizes related work. Section 6 embeds the strategy in an overall tool flow. Section 3 discusses the strategy to evaluate power dissipation for LUT-based hardware based on a statistical model to estimate toggle rates. This is related to a hardware mapping analysis of representing macroblocks (adders and multipliers) for data processing. Section 7 describes the implementation of the technology into the VHDL to SystemC converter. Section discusses the results. Section 9 concludes and gives an outline on future extensions.

2. Related Works

Code translation tools can help guaranteeing the consistency of translated and original models across languages and speeding up development time, since tool-supported translation takes seconds instead of hours to days for manual translation. In the system development cycle, top-down approaches [8–10] are used in HLS (high-level synthesis) tools; bottom-up approaches [11, 12] are used for IP reuse and module abstraction to achieve faster simulation models. In many hardware engineering problems, optimization is necessary on RTL Level. However, readability and efficiency of the translated code are two major problems of the HLS code translators. Because of that a bottom-up strategy is followed in this work. The authors in [13] further illustrates the effects of IP reuse on design time, hence motivates the bottom up approach.

The presented methodology differs from the existing solutions since it targets, beside correct code translation, the automatic integration of further functionality: bus interface wrapping and a power estimation methodology. The power estimation methodology estimates switching activities in DSP units such as adders and multipliers according to actual input data. The approach is based on probability theory. With that it differs from approaches that need to run a conventional time-consuming power estimation flow (e.g., XPA from Xilinx) or other estimators that are based on synthesis results without considering input data (e.g., XPE from Xilinx).

The approach presented in [14] is also implemented for SystemC. It enables the usage of different power models. In comparison with this work, it uses a strategy based on logging the execution of special modules and signals extended from regular SystemC ones, instead of a probability model, as presented here. In [15] a SystemC class library is proposed to the calculate the energy consumption of hardware described with SystemC TLM, and the power model was based on experimental results performed in laboratory, while approach of the presented work is based on the translation of modules from VHDL to SystemC RTL and on probability models. Further works concern the extension of the approach also to TLM.

The work [16] presents an architectural level framework for power analysis, based on parameterized power models of common structures of microprocessors, but does not consider any probability model, as presented in this work. In [17] a methodology is presented for simulation and verification of low-power systems using SystemC. It is based on disabling modules during execution to simulate the power switch-off used by the technique of power gating. The technique for disabling SystemC modules at simulation time is detailed in [18]. At the same time, these related works are not specific to reconfigurable architectures, in contrast to our work, which considers specific characteristics of FPGAs.

3. Power Evaluation Strategy

This project uses an activity-based power estimation and calculation based on the switching frequency of the inputs, outputs, and the internal signals of the individual sub-modules. The proposed power estimation and calculation is tailored for signal processing units where the major part of the power consumption is produced by multiply-accumulate instructions, which is computed in hardware by ripple carry adders and field multipliers based on ripple carry adders. To guarantee that the synthesized hardware for computing MAC is realized on the given adder and multiplier structures, a hardware model on register transfer level of the processing unit is needed. Both implementation and calculation are optimized for area consumption on FPGAs with little slice count. The accuracy of the estimation is based on two areas.

First there is the realization on hardware which is done by the synthesis tools for a specific LUT-based FPGA. Later in this section synthesis results for given FPGAs are examined for getting the fan-out parameters which are needed for a exact calculation of the power dissipation of each element. Further general rules for optimal synthesis of ripple carry adders and field multipliers are formulated.

Secondly, input data is analyzed for predicting the switching frequency on the expected synthesized hardware structure. This means that the probability distribution of the

input data has to be computed. The probability distribution of the output data is derived from that. The input data should be taken from the original implementation. This means, for example, in case of an audio decoder, a corresponding audio stream should be used. For better comprehension an complete workflow for power estimation is shown.

The first step to be done is the analysis of the input data. For example, the input data is uniformly distributed on the complete accepted range with the boundaries 0 and u, where u is a natural number. This assumption is suitable if the distribution of the input data is unknown. By the way this concept was first formulated by Gauss and is known as the Gaussian indifference principle.

Next the binary coding of the numbers between 0 and u is analyzed. The value of the MSB is distributed as followed:

(i) in the range between 0 and $u/2$, the value of the MSB is 0,

(ii) in the range between $u/2$ and u, the value of the MSB is 1.

With the condition of the uniform distribution, all numbers in range are selected with the same frequency. This means, for the probability that the MSB bit is set to 1,

$$p(\text{MSB}) = \frac{u/2}{u} = \frac{1}{2}. \tag{1}$$

The analysis of the next less significant bit is made the same way. The value of the (MSB-1) bit is the follows:

(i) in the range between 0 and $u/4$, value of (MSB-1) is 0,

(ii) in the range between $u/4$ and $u/2$, value of (MSB-1) is 1,

(iii) in the range between $u/2$ and $3u/4$, value of (MSB-1) is 0,

(iv) in the range between $3u/4$ and u, value of (MSB-1) is 1,

The calculation of the probability that (MSB-1) bit is set to 1:

$$p(\text{MSB} - 1) = \frac{2 * (u/4)}{u} = \frac{1}{2}. \tag{2}$$

Based on the calculations for the MSB and the (MSB-1) bit, a prediction can be made.

For every step you make from MSB towards LSB, the number of intervals in which the bit is set to 1 is doubled but the length of the intervals is halved. In consequence this means that the accumulated length of the intervals in which the bit is set to 1 is equal for all bits. Because of this fact the probability that a bit on the input is set to 1 is calculated for all bits:

$$p = \frac{1}{2}. \tag{3}$$

After the computation of the probabilities on the input vector, the probabilities for the output vectors of the ripple carry adder and the field multiplier can be calculated. First the probability of the occurrence of carry bits of the ripple carry adder is determined.

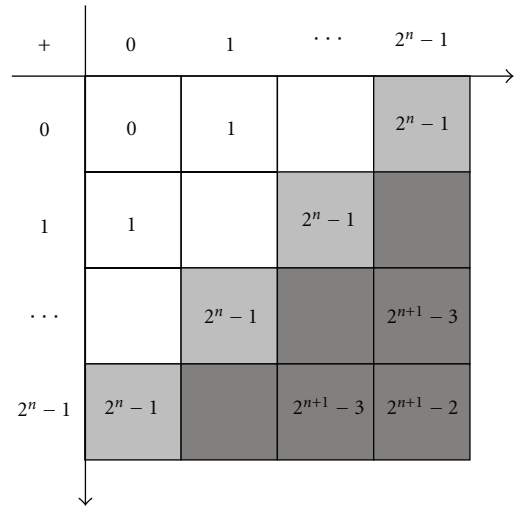


Figure 1: Carry generation in the adder.

For the possibility that a carry-on location n is generated, all input vectors of the adder between 0 and $2^n - 1$ are relevant. Figure 1 shows the addition of two summands in this range.

Each cell in this figure corresponds to the sum of the row and column number. To calculate the the probability that the carry is set, it is suitable to count the number of cell in which a carry is generated and divide this count by the total number of cells. This figure has 2^{2n} entries of which $2^{2n-1} - 2^{n-1}$ generate the carry. The division leads to

$$p_{c,n} = \frac{2^{2n-1} - 2^{n-1}}{2^{2n}} = \frac{1}{2} - 2^{n-1}. \tag{4}$$

Additionally the following statement can be made: because the generation nth carry is only possible if the upper bound of the accepted input values is 2^n, all the probabilities for all carry bits on locations higher than $ld(u)$ can be set to zero.

After calculating the probability of the carry bits, the probability of the sum bits of the adder can follow. The boolean equation of the sum on location n is

$$\begin{aligned} s_n = {} & \left(i1_n \wedge in2_n \vee \overline{i1_n} \wedge \overline{i2_n}\right) \wedge c_n \\ & \vee \left(\overline{i1_n} \wedge in2_n \vee i1_n \wedge \overline{i2_n}\right) \wedge \overline{c_n}. \end{aligned} \tag{5}$$

Transferring this equation to the probability domain leads to

$$\begin{aligned} p_{s,n} = {} & \left(p_{i1,n} * p_{i2,n} + \overline{p_{i1,n}} * \overline{p_{i2,n}}\right) * p_{c,n} \\ & + \left(\overline{p_{i1,n}} * p_{i2,n} + p_{i1,n} * \overline{p_{i2,n}}\right) * \overline{p_{c,n}}. \end{aligned} \tag{6}$$

Setting $p_{i1,n} = 0.5$, $p_{i2,n} = 0.5$ (uniform distribution) and $p_{c,n}$ equals

$$p_{b,n} = 0.5. \tag{7}$$

Like the statement for carry bits with location near the MSB, a likewise statement can be made for the sum bits: because the generation nth sum bit is only possible if the upper bound of the accepted input values is 2^n, all the probabilities for all carry bits on locations higher than $ld(u)$ can be set to zero. Similar to these calculations, likewise calculations for other probability are possible. The following paragraph describes the approach for calculating the probability of sum and carry bits with calculated nonequal probabilities for each bit of the adder input. The given input probabilities are named $p_{i1,n}$ and $p_{i2,n}$. In this project a 32-bit adder was analyzed so the valid indices are in the range from 0 to 31. With this new assumption, the proceeding is as follows.

First the boolean equation of the carry bit is annotated:

$$\overline{cn} = \left(i1_n \wedge \overline{i2_n} \vee \overline{i1_n} \wedge i2_n\right) \wedge c_{n-1} \vee \overline{i1_n} \vee \overline{i2_n}. \quad (8)$$

Based on this equation, a transformation to probability domain is made:

$$\overline{p_{c,n}} = \left(p_{i1,n} * \overline{p_{i2,n}} + \overline{p_{i1,n}} * p_{i2,n}\right) * p_{c,n-1} + \overline{p_{i1,n}} * \overline{p_{i2,n}}. \quad (9)$$

With the extra knowledge that the carry bit in the least significant adder is never set (this means $p_{c,n}$ is zero), all the carry probabilities can be calculated in a recursive manner. With the formula for calculating the $p_{s,n}$ probabilities the examination of the ripple carry adder with nonequal distributed input vectors is completed.

Based on these observations a derivation for the calculation of the signal probabilities of the field multiplier is possible.

The idea is to connect the calculations for the ripple carry adder with individual input probabilities. According to Figure 2 , the input probability p_{i1n} of one adder is the same value like the $ps(n+1)$ probability of the previous level. With the extra assumption of uniform distributed input vectors of the adder and the knowledge that a AND-gate is equal to a multiplication of its input probabilities, the $p_{i2,n}$ probability equals the multiplication of the probabilities of the inputs of the multiplier; that is, $0.5 * 0.5 = 0.25$. If the distribution is restricted to the range 0 to u, all $p_{i2,n}$ with a index and level higher than $ld(u)$ have to be set to zero.

Also these calculations can be performed on other input probability distributions. After the calculation of the probabilities of occurrence, the transition to the switch frequency is made. The normalized statistical switching frequency is given by

$$f_{\text{norm,stat}} = p(1-p). \quad (10)$$

That definition of the frequency is equal to the probability of the occurrence of a positive transition of the signal. By means of this frequency definition, the dynamic power dissipation of CMOS circuits is described as follows:

$$P_{\text{dyn,stat}} = C * U^2 * f_{\text{norm,stat}} * f_{\text{base}} * \text{fanout}, \quad (11)$$

where f_{base} defines the bit rate on the observed signal. The parameter C for the input capacity and the supply voltage

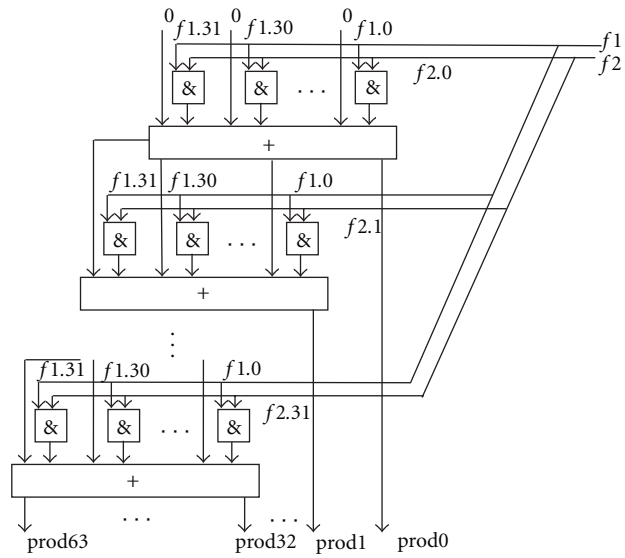


Figure 2: Structure of a field multiplier.

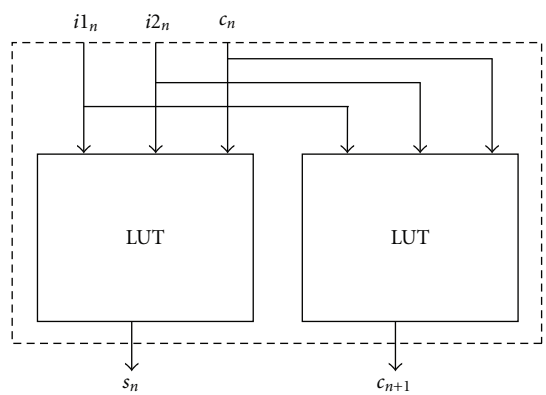


Figure 3: Optimized elementary cell for three-to-four-input LUT.

U can be extracted from the data sheet while the fanout parameter which describes the number of inputs to be driven is dependent on the circuit synthesis on the FPGA. For getting the fanout parameters of the individual signals, the synthesis result of Xilinx ISE is analyzed. The following pictures show the elementary cells of the adder for different FPGAs.

Figure 3 shows the synthesis result of the basic cell of a ripple carry adder for a LUT-FPGA with three to four inputs and one output, Figure 4 shows the synthesis result of the basic cell of a ripple carry adder for a LUT-FPGA with five to six inputs and one output, Figure 5 shows the synthesis result of the basic cell of a field multiplier for a LUT-FPGA with four to five inputs, and Figure 6 shows the synthesis result for a LUT-FPGA with six to seven inputs.

Out of the synthesis result, it is obvious that the maximal size of a elementary cell is limited by the LUT fanin which calculates the carry for the next elementary cell. The elementary cell synthesized for the adder equals a radix-floor $((n-1)/2)$ adder which is shown in Figure 7.

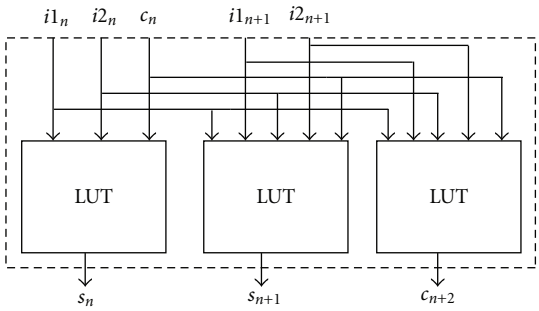


Figure 4: Optimized elementary cell for five-to-six-input LUT.

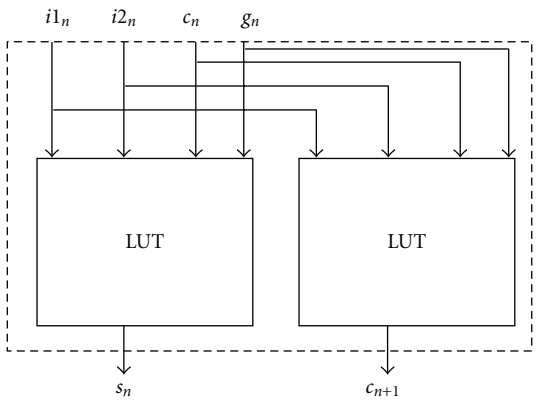


Figure 5: Optimized elementary cell for four-to-five-input LUT.

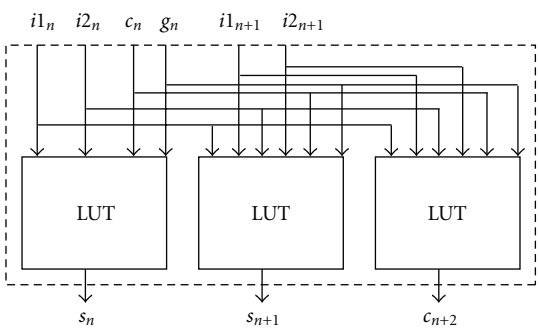


Figure 6: Optimized elementary cell for six-to-seven-input LUT.

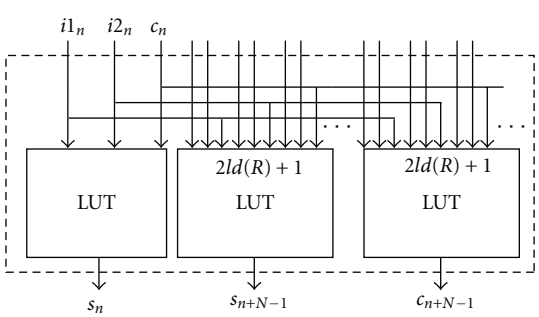


Figure 7: The high radix adder mapped to LUTs.

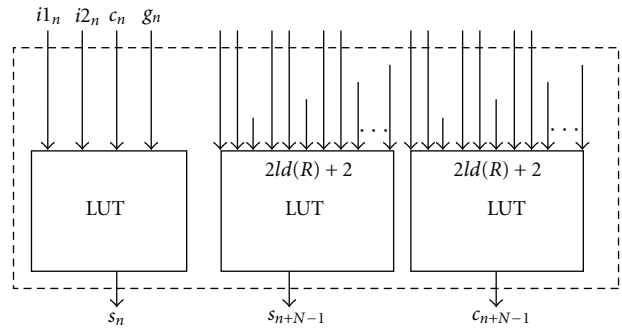


Figure 8: The high-radix adder with gate mapped to LUTs.

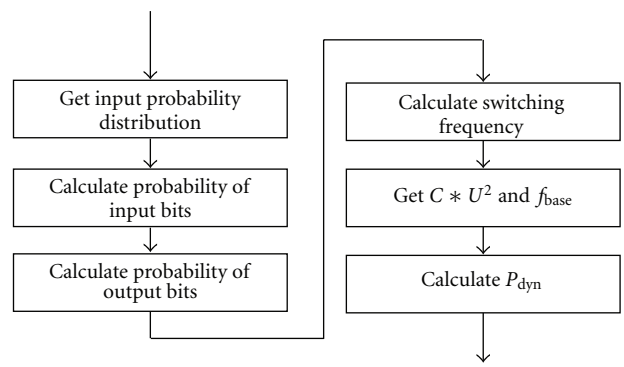


Figure 9: Setup of a new power estimation.

Similar to the high-radix adder elementary cell, the elementary cell of the multiplier consists of a high-radix adder and set of AND-Gates which are controlling the optional addition of summand no. 2 (shown in Figure 8).

With help of this circuit diagrams, it is possible to get the fan-out parameters for the power dissipation formula. The overall workflow can be seen in Figure 9 and is described as follows.

4. Improving Probability Analysis by Carry-Computation

Another possibility for the quality of the probabilities of signal occurrence is the replacement of signal by transfer of a specifically calculated value. It raises the question of which carry bit achieved through the replacement of an exact signal calculation has the best effect. The first consideration is the simplified assumption that all bits are equally distributed, then, for the probability of occurrence for each carry bit of the formula:

$$p^N(c,n) = \frac{1}{2} - \frac{1}{2^{n+1}}. \tag{12}$$

In case that the n_1th bit is calculated exactly for the signal, the maximum difference applies to

$$\begin{aligned}
\mathrm{Diff}_{c,n_1} &= \sum_{n=0}^{N-n_1} \left| p_{c,n}^{N} - /p_{c,n}^{N} \right|, \\
\mathrm{Diff}_{c,n_1} &= \sum_{n=0}^{N-n_1} 1 - 2 * p_{c,n}^{N}, \\
\mathrm{Diff}_{c,n_1} &= \sum_{n=0}^{N-n_1} \frac{1}{2}^{n}, \\
\mathrm{Diff}_{c,n_1} &= \frac{1}{2} - \frac{1}{2}^{N-n_1}.
\end{aligned} \tag{13}$$

This maximum deviation occurs with the probability of $p_{cn}^{N}a$. Hence, for the probable deviation,

$$\begin{aligned}
p_{\mathrm{Diff}_{c,n_1}} &= \mathrm{Diff}_{c,n_1} * p_{c,n_1}^{N}, \\
p_{\mathrm{Diff}_{c,n_1}} &= \left(\frac{1}{2} - \frac{1}{2^{n+1}}\right) * \left(\frac{1}{2} - \frac{1}{2}^{N-n_1}\right).
\end{aligned} \tag{14}$$

To get the maximum deviation probability, the first derivative of $p_{c\mathrm{diff},n_1}$ should be used:

$$\begin{aligned}
p'_{\mathrm{Diff}_{c,n_1}} &= \ln(2) * \frac{1}{2}^{n_1} \left[\frac{1}{2} - \frac{1}{2}^{N-n_1}\right] - \ln(2) \\
&\quad * \frac{1}{2}^{N-n_1} \left[\frac{1}{2} - \frac{1}{2}^{n_1+1}\right] \neq 0.
\end{aligned} \tag{15}$$

It can be seen from the equation that it must apply for a zero:

$$\begin{aligned}
N - n_1 &\overset{!}{=} n_1 + 1, \\
n_1 &= \frac{N-1}{2}.
\end{aligned} \tag{16}$$

With an additional graphic, it can be seen that it is found at the point where only the maximum of the function is located. Now it is possible to find the point in the equation that determines the maximum:

$$\max\left(p_{\mathrm{Diff}_{c,(N-1)/2}}; \mathrm{Diff}_{c,(N-1)/2} * p_{c,(N-1)/2}^{N}\right) \leq 0.25. \tag{17}$$

At this point it becomes clear why the exact calculation of the carry is not worthwhile: the probability of occurrence of a input signal bit is already, by assumption, 0.5 and can be determined without calculation; the calculation of a carry must be made in each run of the adder and helps only to avoid errors in average of 0.25. The distribution of the most probable estimation error for different length adders is presented in Figure 10. The calculation of a carry bit, however, is costly. The n_1th carry is calculated as follows:

$$\begin{aligned}
\mathrm{Carry}_{n_1} = {} & ([\mathrm{Summand}\ \ 1 \bmod (1 \ll i)] \\
& + [\mathrm{Summand}\ \ 2 \bmod (1 \ll i)]) \div (1 \ll i),
\end{aligned} \tag{18}$$

the total of

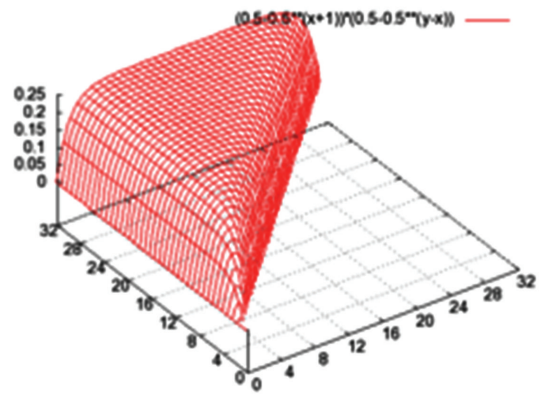


Figure 10: Distribution of the most probable estimation error for n-bit adder.

(i) 3 identical left-shift operations

(ii) 1 division

(iii) 2 modulo (division with remainder).

5. Interface: Power Loss Estimation

In order to generate as little code overhead, a basic C data type was created for unsigned integers `unsigned int` for the equivalent class `sc_int_power`, which is equipped with the same operator overloads like the original data type. These include all the arithmetic operators that are required for signal processing (addition, subtraction, multiplication, division), the assignment operator, and the operator for streaming output with `cout`. In addition, a cast operator for `unsigned int` casting in a safe situation in which only the numeric value of the `sc_int_power` object must be such as that with an array indexing. The real power estimation takes place only in the multiplication and addition operators, because these functions are the main focus of this work. The functions occurring in this energy consumption model are stored by the class static member `power`.

To take advantage of the presented class `sc_int_power` in a SystemC project, it should be compared to a standard implementation requiring the following additional code fragments:

(1) include the header file `power.h`;

(2) call the static function `set_calc` to indicate the accuracy of calculations;

(3) call the static function `set_mu` set parameters for the estimation;

(4) call the static function `set_lut` to adjust the power estimation of the target FPGA.

Anything else due to the replication of the functionality of `unsigned int` in `sc_int_power` must be ignored. It is also possible to define a simple design already in the VHDL source code with pragmas, such as the type `integer` in the types `sc_int_power` by the modified V2SC during translation. Instead of the function calls in an equivalent

SystemC project boundary bonds are defined by the above pragmas as follows:

(1) `set powerestimation_lut number;`

(2) `set powerestimation_mu number;`

(3) `set powerestimation_calc 0 or CALC_ALL;`

(4) `powerestimation on:` all of the following integers in `sc_int_power` translated;

(5) `powerestimation off:` all of the following integers in `int` translated.

These pragmas are recognized when translated from VHDL to SystemC and automatically converted into the corresponding SystemC functions. By this procedure, additional user settings are generated into the source code.

5.1. The Class prob_bit. The class `prob_bit` implements the functions to estimate the amount of time for the inputs, outputs, and internal signals of a 32-bit adder to be high and the estimation of the switching frequency. To fill the individual estimated, six arrays are implemented:

(1) `input` contains the estimation of bit = "1" for input bits;

(2) `input_rising_edges` contain the estimation for the switching frequency of the input;

(3) `output` contain the estimation of bit = "1" for the sum bit;

(4) `output_rising_edges` contain the estimation for the switching frequency of the sum bits;

(5) `signal` contains the estimation of bit = "1" for the carry bit;

(6) `signal_rising_edges` contains the estimation for the switching frequency of the carry bit.

To store a new estimate for bit = "1" in each field, the functions `add_input_prob`, `add_output_prob` and `add_signal_prob` provided that again over the functions `get_input_prob`, `get_output_prob`, and `get_signal_prob` can be retrieved. To simplify these getter and setter methods, the class has the static methods `in1`, `in2`, `carry` and `out`, which return the corresponding array index in the fields. For example, an estimate probability of bit = "1" for the 3rd bit of the adder `adder1` is set to 0.5 with the following call:

```
adder1.add_input_prob(prob_bit::in2(3),0.5).
```

The estimated switching frequencies on the power dissipation of the adder can have the class of functions `input_weight_function`, `signal_weight_function`, and `output_weight_function`, corresponding fan-ins for the individual inputs, outputs, and signal. By multiplying the switching frequencies associated with the switching frequency, we have the following relationship:

$$P_v = C_{\text{LUT}} * V^2 * (f * \text{fanin}). \tag{19}$$

Thus, all weighted switching frequencies must be multiplied only by the same factor in order to infer the loss of performance. The actual value for each weighting function depends on the specific type of LUT-optimized design.

The weighting factors provided with the switching frequencies can be accessed through the functions `get_input_power`, `get_output_power`, and `get_signal_power`. The parameters of these functions can also have the static function `in1`, `in2` be `carry` and out to select the desired bits.

These functions only refer to individual bits of each type. A new group of functions was defined to access the accumulated probabilities of occurrence of bit = "1" and the weighted switching frequencies. The functions are named following the same pattern as the getter and setter methods:

(1) `get_weighted_input_sum` accumulates all the probabilities of occurrence of the inputs;

(2) `get_weighted_input _power_sum` accumulates all the switching frequency of the inputs;

(3) `get_weighted_signal_sum` accumulates all the probabilities of occurrence of the signals;

(4) `get_weighted_signal_power_sum` accumulates all the switching frequency of the signals;

(5) `get_weighted_output_sum` accumulates all the probabilities of occurrence of the outputs;

(6) `get_weighted_output_power_sum` accumulates all the switching frequency of the outputs.

It is also possible to accumulate over the functions `get_weighted_all_sum` and `get_weighted_all_power_sum` all weighted probabilities of occurrence or all the weighted switching frequencies. For further support of statistics on different `prob_bit` objects, two other functions were created for each port and signal to return the probability of occurrence and the switching frequency, as follows

(1) `print_input_all_prob` returns the probability of occurrence of all inputs;

(2) `print_input_all_power` outputs the switching frequency of all inputs;

(3) `print_signal_all_prob` returns the probability of occurrence of all signals;

(4) `print_signal_all_power` outputs the switching frequency of all signals;

(5) `print_output_all_prob` returns the probability of occurrence of all outputs;

(6) `print_output_all_power` returns the frequency switching of all outputs.

The class uses a custom constructor with three parameters for the transfer of the number of input and output signals, which also allows the formation of any combinational logic result.

5.2. The Class exact_bit. The class `exact_bit` implements functions for the accurate determination of probability of occurrence and switching frequency of 32-bit adders and is analogue of the class `prob_bit` which estimates these parameters. In order to not duplicate code and to keep

the naming of the methods most consistently, the class exact_bit inherits from the parent class prob_bit and can largely use the methods of the parent class. The difference from the parent class makes itself felt in the determination of the exact switching frequency. During the estimation in the class prob_bit, an estimation of the switching frequency can be made directly from occurrence probability, this is not an exact determination because positive edges can be made only in the context of the last state of the adder. For this reason, the class attribute last_state holds a pointer to an associated exact_bit object that stores the last state of the input and output signals. With this additional information, it is now the exact determination of the number of positive edges occurred and therefore the exact determination of the switching frequency possible.

The determination of the accumulated amount of time in which one remains on high bit (equivalent to the probability of occurrence) is also possible without the context of the last condition; therefore the implementation of the class prob_bit is used. All other functions such as statistical functions to accumulate or print functions can also be used without modification, as there are the fields' input, input_rising_edges for the stored values to access from the direct calculation.

5.3. Addition Class. The class addition provides the methods for estimation and exact determination of the probabilities of occurrence and switching frequencies of individual bits in a prepared 32-bit ripple-carry adder, which are used in the class sc_int_power. Addition has the functions power_add_approx2, power_add_approx2_complex, and power_add_exact2, where the first two estimate the switching frequency and the last ones accurately determine the switching frequency of the individual inputs, broadcast, and outputs.

The two estimators power_add_approx2 and power_add_approx2_complex differ in the specification of the probability of occurrence of the input bits. For the case when the adders are used as a single unit, it can be assumed as a multiplier, according to the principle of indifference when all input vectors are equally distributed within an interval. The function power_add_approx2 can be used to get the average of the input vectors and a statistic value that the prob_bit object returns. There is the case when the adder is part of a multiplier, the perception of the equal distribution is no longer made, and the function power_add_approx2_complex is used. This function takes an input statistic in the form of a prob_bit counter object and returns the output statistics again in a prob_bit object.

The function for accurate calculation of the switching frequency power_add_exact2 was implemented, two terms for the associated switching frequencies, whilst the output statistics are in an exact_bit object are used as parameters. The exact determination of the required associated switching frequency is associated to exact_bit object with the latest state of the adder stored by the class in the private attribute last_state.

5.4. Multiplication Class. The class Multiplication provides methods for accurate determination and estimation of the probability of occurrence and the switching frequency in a 32-bit multiplier, which are required in the parent class sc_int_power. The class has the functions power_multiply_approx2 for the estimated and the power_multiply_exact2 for accurately determined switching frequency. The parameters of both functions are analogous to the methods of the class Addition. The estimator gives the mean value for the factors, while the function is used to the exact determination of the two factors to be multiplied. Because of the considered carry of the multiplier, the class Addition takes into account the wiring structure and take intermediate AND gates completely.

5.5. Random Generator for gcc. First, to test the estimation of the classes Addition and Multiplication, the built-in random generator of C was used with the function rand from stdlib library. In the first test the estimations had similar variations (about 10%) to the exact values calculated as in the subsequent tests with the Mersenne-Twister random number generation. Due to the fact that the C standard requires no algorithm for random number generation, the author does not know which algorithm is used in the current implementation by gcc.

5.6. Random Generator LFSR. In a second test the classes Addition and Multiplication were used for a linear feedback shift register. In the present implementation, the first 32 CRC polynomials were stored; that is, it can pseudo-randomize sequences to generate numbers in intervals up to 2^{32}. Tests, however, showed that the generated pseudo-random numbers are distributed very unevenly, most obviously in direct comparison with the C-random number generator or the Mersenne-Twister. Due this statistical property, this random number generator was considered unsuitable and was not included in further tests.

5.7. Mersenne-Twister Random Number Generator. The GNU suite delivers a Mersenne-Twister random number generator. This type of random generator works on an extremely long interval of $2^{19937-1}$ and its most important feature is the uniform distribution of all output bits. Due to these excellent statistical properties, the Mersenne-Twister random number generator has been selected as a reference for testing the power loss estimation. In a comparison to integrated random in C, the Mersenne-Twister cuts marginally better due to the fact that in the random number generator rand of C, the least significant bits are not equally distributed.

5.8. Random NDIST Based on gcc. To test the implementation of the power estimation with normally distributed random variables, the C-function NDIST was used, which approximates to function rand, but with uniformly distributed random numbers. With the aid of the central limit theorem, which states that the mean of a sufficiently large number of independent random variables, each with finite mean and variance, will be approximately normally

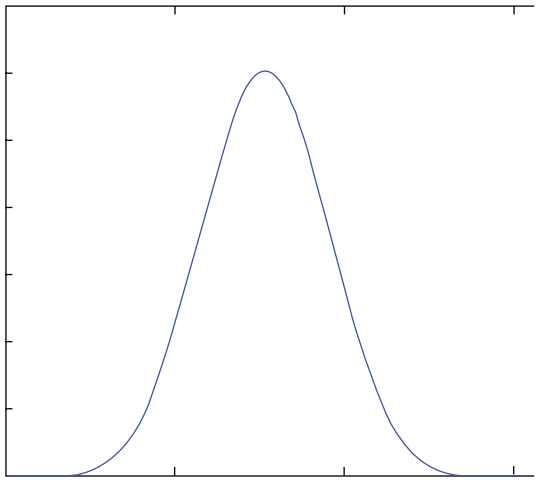


Figure 11: Normal distribution.

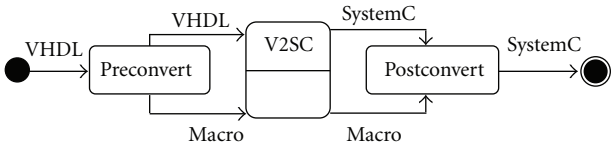


Figure 12: Combination of wrapper and V2SC.

Table 1: V2SC features.

Feature	Included
If	Yes
While	Yes
For	Yes
Function	Yes
Procedure	Yes
Packet	Yes
Record	Partly
Array	Partly

Table 2: V2SC + wrapper extras.

Feature	Included
Alias	Yes
Cast functions	Yes
Generate	Yes
Port map	Yes
Records	Yes
Arrays	Yes
Configuration	No
File IO	No

distributed, six different random numbers are added and displayed as a normally distributed random number. The actual number of six random numbers to be accumulated was determined experimentally. The chart of Figure 11 shows the theoretical distribution of the generated numbers in this six-accumulated random variable.

5.9. Generating a Large Random Number Set. The problematics when comparing different implementations of the estimation with the classes `Addition` and `Multiplication` in comparison with the VHDL implementation are to regenerate the random numbers used in each run. For this reason a common set of uniform distribution of $2 * 1000$ random numbers 0–1024 was generated and from this set all the relevant statistical characteristics of was calculated the probability of occurrence of individual bits. But the numbers from 0 to 1, which are the basis for determining the switching frequency, change. In addition to this, statistical characteristics were raised to what extension they can differ from the requirement to assess the validity of the estimation result.

6. Module Translation and Integration

An integrated system simulation can be accomplished in homogeneous or heterogeneous environments. The adoption of homogeneous system simulations has several advantages: (i) it achieves faster simulation since synchronization tasks between different simulation kernels can be omitted; (ii) the usage of SystemC instead of for example, VHDL and the possibility to simulate models on higher abstraction levels further increment the simulation speed and additionally; (iii) since SystemC models are inevitable for system simulations and RTL models are necessary for synthesis flows, a module translation helps in keeping synchronism among the different module implementations and reduces error sources. On the other hand, this strategy requires a module translator. The VHDL to SystemC translator (V2SC) that was used in this work as basis has been developed by [19].

It was designed for SystemC 1.0 and basically supports constructs that can be directly translated from VHDL to SystemC. In this work, the translator has been extended to be compatible with the actual SystemC version 2.2 and to translate synthesisable VHDL constructs. Table 1 presents the features provided by V2SC and Table 2 shows a selection of extensions included as a contribution of this work. Further convertible VHDL syntax elements are listed in [20]. The converter has been verified against a set of VHDL modules, among them a IMDCT (for calculating the inverse modified discrete cosine transform) module including an AHB master and a APB slave interface with a complexity of >5000 lines of code, a FFT module (for calculating the fast Fourier transform) with about 250 lines of code, and a GCD module (for calculating the greatest common divisor) with about 100 lines of code.

The extensions are implemented with the compiler building tools flex and bison in the same way as the original converter. Furthermore the macroprocessor m4 and the text manipulation tool sed were used. The whole extension is build as a wrapper around V2SC (Figure 12) and is divided in two parts. A preconvert filter simplifies syntax elements by converting them to V2SC compatible constructs, or the filter masks constructs such that they can pass the V2SC

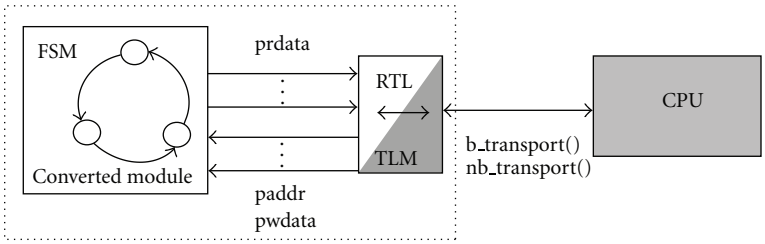


Figure 13: Interface conversion for generating a heterogeneous accuracy simulation model.

Table 3: Simulation speed comparison.

SystemC	SystemC with `int`	SystemC with `sc_int_power` (approx.)	SystemC with `sc_int_power` (exact)
Addition	1	0.5	0.03
Multiplication	1	0.5	0.001

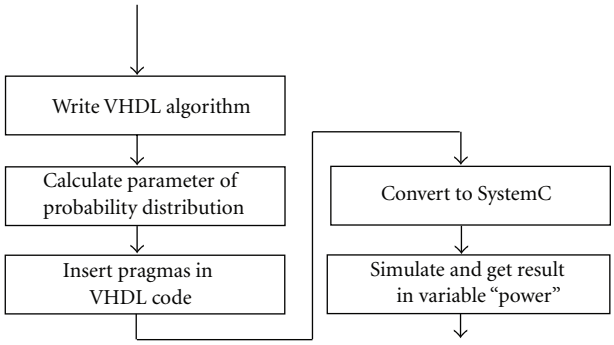


Figure 14: Power estimation workflow.

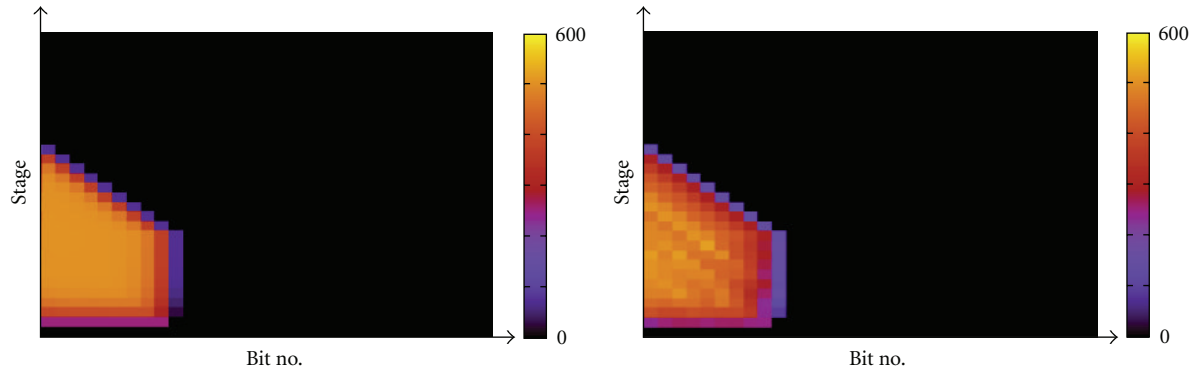
without modification, if the V2CS cannot handle them. The masked blocks are converted by the postconverter to SystemC constructs.

The converted IPs own a pin accurate RTL interface whereas the system simulator provides TLM interfaces. To decrease the design time, a protocol conversion has been integrated in the translation tool. The modified V2SC enables connecting VHDL RTL IPs to a SystemC TLM system by implementing a library that contains design patterns for pin accurate to TLM functional units according to the interface specifications. Appropriate modules are exemplary implemented for AMBA, AHB, and APB. The resulting translated modules can be directly connected to a AMBA-TLM 2.0 base system (see Figure 13).

7. Implementation of Power Estimation Features

The extension contains beside the compatibility pack, as described in Section 6, a tool for the automated integration of the previously discussed power estimation and calculation. This section explains a strategy, with which the conventional flow may be omitted if a certain decrease of accuracy can be accepted. Generally, the extension of functional parts with monitoring capabilities is an efficient way for automated system analysis. Since accurate power analysis flows are one the most time-consuming steps in the development process, their integration in a SystemC environment, with the techniques discussed in Section 3, are expected to speed up the evaluation process. Conventional accurate power analysis tools need mainly two kinds of input information. On the one hand a model of the placed and routed design allows to calculate capacitive loads of each net by evaluating fan-out numbers and the characteristics of the primitives (LUTs for FPGA technology) and with that the energy consumption per net per activity. On the other hand dynamic activities are collected during post-place and route simulation in value Change Dumps (VCD). Especially the accurate simulation with enabled signal tracing is a very time-consuming step.

The power evaluation extension of the modified V2SC can be enabled by the flag `-powerestimation` of preconvert. After setting this flag, preconvert converts the VHDL-type integer in the new-user-defined type `sc_int_power` and not to the standard SystemC-type `int`. `sc_int_power` is a new type which is constructed in this project to do all power estimation and calculation. It acts a replacement for `int` and its behaviour is similar to it. To get this functionality, all common arithmetic operators (+,−,∗,/) are overloaded while the addition and the multiplication operator definitions contain the power estimation and calculating algorithms. Additionally all arithmetic operators are overloaded for mixed-type operation with `int`. Another feature is the overloaded cast operator of `int`, which is especially for indexing arrays. With a view to the use of `sc_int_power` as a complete SystemC data type, the streaming operator ≪, the test of equality ==, the test of inequality !=, the assignment operator =, and the function `sc_trace` are overloaded. Thus in this manner defined type can be used as a template class in `sc_signal<>`, for example. To get a high prediction quality of the power estimation, the algorithms have to be parameterised. This is done by the static class methods `set_mu` (defines the average value of the input vectors) and the `set_lut` (for optimizing the design to architecture). The static class method `set_calc` defines a switch between estimation (value is zero) and calculation (value is equal `CALC_ALL` constant). The result of the calculation respectively estimation can be retrieved



(a) Approximation (b) Exact calculation

Figure 15: Toggle distribution estimation.

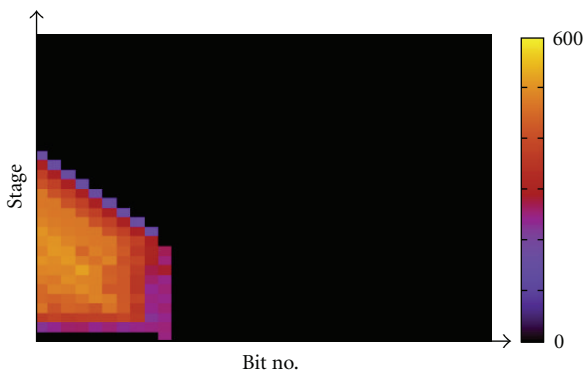


Figure 16: Reference from XPA tool.

from the static variable `power`. For improvement of the manageability of the power analyser tool the parametrisation of the algorithms is also possible in VHDL code by special comments called pragmas. These pragmas consist of the prefix "powerestimation" followed by the parametrisation function (i.e., `set_calc`, `set_mu` or `set_lut`) and the value to set. The following example shows the proceeding.

The original VHDL code:

```
(1) signal a,b,c: integer;
(2) --powerestimation set_lut 5
(3) --powerestimation set_calc 0
(4) --powerestimation set_mu 10.0
(5)
(6) c<=a+b.
```

Is translated in SystemC code:

```
(1) #include "power.h"
(2) ..
(3) sc_int_power a,b,c;
(4) c=a+b;
(5) //the powerestimation result
(6) //is stored in sc_int_power::power
(7) return 0;
```

The whole workflow is like that presented following that Figure 14.

8. Results

8.1. Accuracy of Toggle Estimation. The accuracy of the power estimation algorithms in `sc_int_power` to the also implemented exact calculation is about 13% for a average of more than 100 single additions or multiplications. Figure 15 shows the estimated toggle counts on the left and the exact calculated toggle count on the right of the partial sums (summands no. 1 of each stage) for a data set of 1000 two factors namely summands which are equally distributed in the range between 0 and 1024.

In comparison to the Xilinx XPower tool (see Figure 16), an almost similar accuracy was reached. On the following picture is the count of toggles for the partial sum bits visualized.

8.2. Speed Improvement. The integration of the proposed power estimation slows the calculation in contrast to the calculations on the standard type `int` about the factor two, but the also implemented exact power calculation slows the additions and multiplications about the factor 32, namely, 1024 (see Table 3). But the estimation is only slightly worse than the exact determination of the power dissipation; however the estimation is about the factor 16, namely, 512 faster. In sum this means that the proposed estimation is a good tradeoff between accuracy and simulation speed.

9. Conclusion and Future Work

This work proposes a methodology for switching activity estimation, taking into account the underlying hardware structure. The methodology has been exercised for MAC units. Different FPGAs have been used to show the portability of the method to other technologies. The loss of accuracy of 13% in the case of the MAC unit compared to post-place

and route simulation results comes along with a simulation speed up of a factor up to 1024. The transparent implementation of that methodology into a VHDL to SystemC converter further accelerates and eases the development process. The general approach of the methodology can be also applied to other regular computation structures. With the implementation of further computational units (e.g., dividers or other adder architectures) and the support for other FPGA architectures, an analysis of complex data paths and a faster evaluation of design alternatives are envisioned.

References

[1] N. Dhanwada, I. C. Lin, and V. Narayanan, "A power estimation methodology for SystemC transaction level models," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and Systems Synthesis (CODES+ISSS '05)*, pp. 142–147, September 2005.

[2] N. Dhanwada, R. A. Bergamaschi, W. W. Dungan et al., "Transaction-level modeling for architectural and power analysis of PowerPC and CoreConnect-based systems," *Design Automation for Embedded Systems*, vol. 10, no. 2-3, pp. 105–125, 2005.

[3] M. Lang, "System C for Embedded System Design," Seminar, 2006, http://dl.acm.org/citation.cfm?id=339657.

[4] S. Boukhechem and E. B. Bourennane, "TLM platform based on systemC for STARSoC design space exploration," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS '08)*, pp. 354–361, June 2008.

[5] S. Hauck and A. Dehon, *Reconfigurable Computing the Theory and Practice of FPA-Based Computing*, Elsevier, 2008.

[6] N. Voros, A. Rosti, and M. Hübner, *XDynamic System Reconfiguration in Heterogeneous Platforms*, Elsevier, 2009.

[7] M. Kuehnle, A. Wagner, and J. Becker, "A statistical power estimation methodology embedded in a SystemC code translator," in *Proceedings of the 24th Symposium on Integrated Circuits and Systems Design (SBCCI '11)*, pp. 79–84, IEEE Computer Society, 2011.

[8] Forte, "Cynthesizer," 2011, http://www.forteds.com/products/cynthesizer.asp.

[9] M. Graphics, "Catapultc," 2011, http://www.mentor.com/esl/catapult/overview.

[10] Cadence, "C-to-silicon compiler," 2011, http://www.cadence.com/products/ sd/silicon compiler/pages/default.aspx.

[11] N. Bombieri, "Hif suite 2.0: hdl translating and manipulation tools," 2009, http://hifsuite.edalab.it/.

[12] University of Cincinnati, "Savant," 2011, http://www.clifton-labs.com/savant/.

[13] M. e. a. Bocchi, "A system level IP integration methodology for fast SOC design," in *Proceedings International Symposium on System-on-Chip*, pp. 127–130, 2003.

[14] G. B. Vece and M. Conti, "Power estimation in embedded systems within a SystemC-based design context: the PKtool environment," in *Proceedings of the 7th Workshop on Intelligent Solutions in Embedded Systems (WISES '09)*, pp. 179–184, June 2009.

[15] M. Giammarini, M. Conti, and S. Orcioni, "System-level energy estimation with Powersim," in *Proceedings of the 18th IEEE International Conference on Electronics, Circuits and Systems (ICECS '11)*, pp. 723–726, December 2011.

[16] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 83–94, 2000, http://portal.acm.org/citation.cfm?id=339657.

[17] G. S. Silveira, A. V. Brito, and E. U. K. Melcher, "Functional verification of power gate design in systemc RTL," in *Proceedings of the 22nd Symposium on Integrated Circuits and Systems Design (SBCCI '09)*, I. S. Silva, R. P. Ribas, and C. Plett, Eds., ACM, September 2009.

[18] A. V. Brito, M. Kühnle, M. Hübner, J. Becker, and E. U. K. Melcher, "Modelling and simulation of dynamic and partially reconfigurable systems using System C," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '07)*, pp. 35–40, 2007.

[19] U. Tuebingen, 2001, http://www-ti.informatik.uni-tuebingen.de/systemc/.

[20] A. Wagner, *Diplomarbeit Randbedingungen der HW-Modellierung auf RTL-und Systemebene*, 2011.