Area and Power Consumption Estimations at System Level with SystemQ 2.0

Sören Sonntag
Infineon Technologies
Intellectual Property Reuse
Munich, Germany
soeren.sonntag@infineon.com

Wenjian Wang
Infineon Technologies
Intellectual Property Reuse
Munich, Germany
wenjian.wang@infineon.com

ABSTRACT

Systems-on-Chip (SoC) integrate a complete electronic system in a single integrated circuit. SoCs typically comprise processors, hardware accelerators, memories, and on-chip interconnects. These increasingly complex systems must fulfill many requirements, such as high data throughput, low latency, small area, as well as low power consumption and dissipation.

In this paper we show how to evaluate an SoC at Electronic System Level (ESL). We use our performance evaluation framework SystemQ 2.0 not only to analyze common performance metrics, e. g. throughput, latency, and resource utilization, but also to perform area and power estimations at system level. The foundation of our estimations is a large amount of data from synthesized and physically implemented hardware components. From that we build a set of formulas to be integrated into SystemQ.

In a case study we show the area and power consumption estimations of a complex SoC interconnect. We reveal how the area and power data are gathered and integrated into SystemQ. Based on real test cases we compare the transistor-level data with the system-level results from SystemQ. It will be shown that the error for the area estimations is up to 6.3 % for single components. The complete system is tested with two standard-cell libraries, whereas the error is 17.0 % and 28.1 %, respectively. The power estimation error is 11.5 % at component level.

Categories and Subject Descriptors

B.8.2 [**Integrated Circuits**]: Design Aids—*Placement and routing, Simulation*; I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis

General Terms

Design, Algorithms, Performance

SIMUTools 2009 March 2-6, Rome, Italy

Keywords

Area and Power Estimation, Synthesis, Electronic System Level, Modeling, SystemQ

1. INTRODUCTION

System-on-Chip (SoC) has become a common design technique in the integrated circuits industry. SoC offers many advantages in terms of cost and performance efficiency. SoCs are increasingly complex systems that are highly integrated comprising processors, hardware accelerators, memories, and on-chip interconnects. Several performance requirements must be fulfilled. In addition, tight area and power consumption constraints have to be met.

Area is an important factor which directly affects the cost of a chip. Especially for large volumes, constant development expenses become negligible compared to the manufacturing costs. The smaller the die size the more dies can be placed on a single wafer which in turn lowers the cost per die. Smaller dies also increase the yield (percentage of good dies) and reduce packaging costs.

As the circuit speed and integration density continue to rise, power consumption becomes another critical topic in SoC design. Especially for portable devices, lower power consumption means higher energy efficiency, longer battery life, and fewer cooling requirements. The self-heating of IC devices can also lead to malfunctions and therefore requires prevention through additional cooling efforts, such as power-consuming fans and weight-adding heat sinks. Lower power consumption has also become very desirable from an environmental point of view.

The power consumption of a circuit is also dependent on the amount of activity performed by the device. This makes predicting the power even more complicated and an even greater challenge for design development, compared to area estimations.

In the typical design process of SoCs, the description of circuit behavior starts from the system level and goes through lower abstraction levels with the assistance of Electronic Design Automation (EDA) tools. The higher the abstraction level, the more difficult it is to estimate the area and power consumption of a system accurately.

With the rapid increase of design complexity in SoCs, a sound estimation of area and power consumption at the very beginning phase of SoC design is helpful and important to properly plan the whole design, to avoid unnecessary iterations, and further to reduce the cost. The later a problem is found the more effort is required to correct it.

There have been many attempts on area and power con-

sumption modeling at different abstraction levels, such as wiring space estimation as well as register-transfer level area and power estimation (cf. Section 5). These models are useful for estimating the area and power of specific circuit structures, or they can be adopted by EDA tools to predict the area and power of a complete system. However, attempts to model the area and power directly at electronic system level are still rare.

In this paper we show how to integrate area and power consumption estimations in our Electronic System Level (ESL) performance evaluation framework SystemQ [9]. Our recently released version 2.0 allows the designer to early estimate area and power consumption at system level in conjunction with performance evaluation. Interdependencies between area and throughput or latency and power consumption can be easily examined even in the concept phase of the design process. Although we target at a low error for the estimations with respect to the physical implementation a rough estimation that reflects the parameter changes of the design is a major improvement for system engineers' work today.

Typically, the focus of related work is on regular structures like memories or non-configurable components like a simple processor. However, in our case study we present area and power consumption estimations for a complex multiprocessor SoC crossbar interconnect which is highly configurable regarding the number of master and slave interfaces as well as the various performance metrics.

The rest of this paper is organized as follows. In the next section we briefly introduce our ESL performance evaluation framework SystemQ while focusing on the new features of the recently released version 2.0, namely area and power consumption estimations at system level. A case study of evaluating a multiprocessor SoC interconnect system using our approach is shown in Section 3. The results are discussed in Section 4. Related work is illustrated in Section 5. We conclude our paper in Section 6.

2. SYSTEMQ 2.0

SystemQ is an Electronic System Level platform architecture evaluation framework [9]. It is based on queuing theory and implemented in SystemC [7]. SystemQ is targeted at modeling and evaluation of systems in data-flow oriented domains, such as network processing and System-on-Chip interconnect systems. Our modeling approach allows us to express scheduling decisions and workload dependent behavior since this is made explicit by the queuing model semantics.

2.1 Overview

Queuing theory has been used in operations research for more than 50 years. Its main focus is on stochastic analysis of system behavior. Queuing systems basically consist of *queues* and *servers*. A queue is a waiting room where requests are stored until a server becomes available to process one of the requests. The processing takes a specific amount of time after which the request leaves the queuing system. By analyzing queuing systems, several conclusions can be drawn, including the residence time of requests in the system, the average queue length, and the server utilization.

Queues store requests according to their queuing discipline, e. g. First Come First Served (FCFS), Processor Sharing (PS), and fixed priority. Thus, queues are used for explicit scheduling of requests. Queues neither modify requests nor consume simulation time. However, requests spend a certain amount of simulation time in the queue if other requests are getting serviced at the time of arrival.

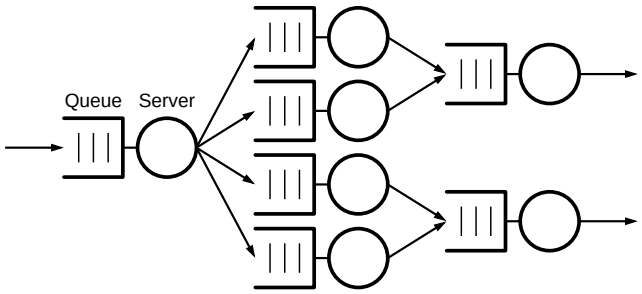

Figure 1: Simple queuing network consisting of seven queuing systems.

Servers consume an amount of time, called *service time*, while processing requests sequentially. In systems that use distinct request classes, servers can also alter the class of requests while processing them. Furthermore, servers may create or delete requests but do not store them since the latter is a distinct feature of queues.

For more realistic models *queuing networks* are built out of several queuing systems as shown in Figure 1. We use large networks with hundreds of queuing systems. By using simulation we can trace single requests through the system. This enables a detailed end-to-end delay analysis depending on the request class and the utilization of the system.

In order to stimulate wide acceptance among platform designers, the SystemQ framework is based on SystemC. Existing code bases for algorithms in C and C++ can be reused in the simulation models. SystemQ therefore supports a systematic path of refinement steps starting from plain performance models down to Transaction Level Modeling (TLM) and Register Transfer Level (RTL) exploiting SystemC's refinement methodology.

SystemQ has been successfully employed in more than ten recent SoC projects. The framework is available on various platforms like Linux, Solaris, Windows, and Cygwin.

2.2 New Features of SystemQ 2.0

Apart from the rich queuing system library, the SystemQ framework includes modular and extensible means for generating realistic traffic, analyzing patterns of received traffic, and tracing individual requests through the modeled system. Our new version 2.0 offers three new features that will be explained in the following sections.

2.2.1 Parameter Files

Parameters can be used to configure the simulation model during the model elaboration phase to increase flexibility. SystemQ offers a parameter file support where text based parameter files are read before the simulation starts. Based on these parameters, modules can be configured without recompilation. Parameters also allow efficient design space exploration using predefined or even automatically generated parameter files. One single compiled SystemC binary file can be used with dozens or hundreds of parameter files to explore a large design space.

The parameters are placed in a human-readable parame-

```
::Report_Interval        125 ms
Queue1::Capacity         256
Queue1::Area             1937.50 um2
Server1::Bitrate         800 Mbps
Server1::Response_Time   32 ns
Server1::Area            847.63 um2
```

Figure 2: Parameter file for a simple queuing system.

ter file. The syntax of the file is

```
<scope>::<parameter> <value> [<unit>]
```

where `scope` determines the scope of the parameter similar to the C++ operator of the same name. `parameter` denotes the name of the parameter, which is a user-defined alphanumeric string. The `value` and `unit` parts depend on the type of parameters as explained next.

How are the parameters in the parameter file linked to the simulation model? In fact, each module in the SystemQ model can add new parameters by calling the `add_parameter` function. This function allows the designer to link a parameter to a C++ class variable. Parameters may be of any basic C++ type, e. g. bool, double, and uint32. However, more fancy parameters are supported, such as time, bitrate, area, and power. These parameters are converted to C++ or SystemC, e. g. sc_time, types.

How does a parameter file look like? Imagine a SystemQ model consisting of a simple queuing system including one queue, called Queue1, and one server, called Server1. The queue has a capacity of 256 requests, its area is $1937.50\,\mu\text{m}^2$. The server has a bitrate of 800 Mbps, a minimum response time of 32 ns, and an area of $847.63\,\mu\text{m}^2$. Both modules display some statistical reports every 125 ms. The corresponding parameter file is listed in Figure 2.

As shown in the figure common parameters have a global scope and do not need to be defined for each particular module. This keeps parameter files clear and small.

SystemQ 2.0 also includes a parameter file caching mechanism. Every module can potentially have its own parameter file, but for large models with several hundreds of modules parameters are grouped in a small number of files. At the end of the design elaboration phase the following eight steps take place for each module in the design:

1. Open parameter file,
2. Remove comments,
3. Resolve includes and macros,
4. Split lines into tokens,
5. Check validity of values and units,
6. Find matching parameters for particular module,
7. Check parameter ranges,
8. Apply parameters to internal C++ variables.

The first five steps are independent of the module. Hence the results of step 5 are cached and if another module is using the same parameter file the preprocessed file can be reused. This significantly speeds up the simulation especially for larger designs.

2.2.2 Area Estimations

Area estimations are important for system engineers since the performance of the system interdepends on the area, i. e. the higher the clock speed the larger the design for a given technology. Other factors affecting the area are FIFO sizes, the bitwidths of the data and control paths, as well as the technology the model will be implemented in later on.

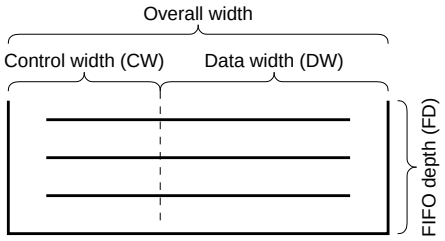

Figure 3: Influence of HDL generics on FIFO area.

Our approach is based on a detailed investigation of the SoC components. We use the Magma tool chain for both synthesis and physical implementation. Each component at Register-Transfer Level is highly configurable, having 10–25 HDL generics (hardware parameters). During component instantiation the generics are set and cannot be changed later on. HDL generics may influence the area of the component requiring us to synthesize and physically implement the component with different generics settings. As depicted in Figure 3 the area of the FIFO depends on the control width (CW), the data width (DW), and the FIFO depth (FD). The larger CW, DW, and FD the more area the component consumes. However, if the FIFO depth is only 1, there might be a performance bottleneck since the FIFO is toggling between empty and full.

In order to obtain significant results a vast number of synthesis runs has to be performed for each module: For all generics combinations, several frequencies, and different standard-cell libraries. In our work we analyzed six libraries with different properties in threshold voltage, cell size (tracks) and technology optimizations within the same technology node. For each component we have performed at least 220 synthesis runs. This seems to be a high effort but we have found out that synthesis results vary dramatically for some generics. Since the SoC modules are often reusable, the syntheses need to be performed only once for each type of module.

Later on the synthesis results are examined, grouped, and a set of formulas is generated, as shown in the following equation, that will be used in the SystemQ framework.

$$A_{estimated} = A_{base} \cdot P_{generics} \cdot P_{lib} \cdot P_{freq},$$

where $P_{generics}, P_{lib}, P_{freq} \geq 1$. The concept of the base area is the foundation of the area formula. The base area is used as a reference under a base setting (generics, library, frequency). By comparing with the base area, which is usually selected to be the smallest area, we know how much the setting of a factor (generics, library parameters, or frequency) changes the area. The parameters $P_{generics}$, P_{lib}, and P_{freq} are calculated by comparing the area under current settings with the base area, and therefore are greater than or equal to 1. $P_{generics}$ actually represents all the generics. For n generics, $P_{generics}$ can be rewritten as

$$P_{generics} = P_{g_1} \cdot P_{g_2} \cdot \ldots \cdot P_{g_n}.$$

The effect of any single factor on area also depends on the current settings of other factors. Therefore, the parameters usually vary under different test cases. In most cases the numbers are within certain range, and the mean value of the parameters can be taken. The calculation of the parameters

```
::Technology            40nm_Low_Power
::Frequency             333 MHz
FIFO::Data_Width        128
FIFO::Control_Width     48
FIFO::FIFO_Depth        16
FIFO::Base_Area         1193.7 um2
```

Figure 4: Parameter file for the FIFO example.

will be explained in more detail in Section 3.1. If the settings (generics, library, and frequency) under estimation are the same as the settings of the base area, then the parameters of these settings are set to one. If not, the corresponding parameters $P_{generics}$, P_{lib}, and P_{freq} are inserted to model the change in the area.

How can the system engineer choose the generics and library properties? This is naturally done by using the parameters in the SystemQ parameter file. For the FIFO example above an appropriate parameter file is shown in Figure 4.

Each queue and server module in the design is equipped with a *calculate_area()* function that resolves the parameters from the parameter file and applies their values to the set of area formulas. At the end of the simulation various statistics are printed among which the area values is shown. The designer can choose to display the area of basic queuing system components, i. e. queues and servers, queuing systems, sub-systems, or the complete design. For gathering all area data from the queuing systems we use a centralized approach. At the end of the simulation a *get_area()* function in the toplevel module recursively gathers the area information from its submodules by executing their *get_area()* functions. All area calculations are done at the end of the simulation which does not cause any overhead during the simulation.

2.2.3 Power Consumption Estimations

As the circuit speed and integration density continue to rise, power consumption becomes another critical topic in SoC design. Especially for portable devices, lower power consumption means higher energy efficiency, longer battery life, and less cooling requirements. Additionally, lower power consumption has also become very desirable from an environmental point of view.

Therefore, low power design is another target which engineers and IC companies have as a top priority. The power consumption of a circuit is also dependent on the amount of activities performed by the device. This makes predicting the power even more complicated and an even greater challenge for design development, compared to area estimation.

For our power estimation feature in SystemQ we used the synthesis results from our area investigations. We work at the transistor level where transistors have already been mapped to the appropriate technology, all components are placed and routed, and the clock tree is also inserted. At this detailed level we stimulate our modules with a large number of patterns to analyze the power consumption for each pattern.

The effort for the power investigation is much higher than for the area investigation since every design point for the area corresponds to many test cases for the power investigation. For system level simulations a general but fast power estimation approach is preferred. For a FIFO we consider the different transaction types that can be stored into it, e. g. single, burst8, and burst16 transactions. The simulation for each particular transaction type is repeated several times. The average value is then taken to eliminate errors in measurement.

In addition to the transaction types, the generics, library, and frequency are also varied in our large number of simulation runs to observe their effects on power consumption. Out of our investigations we derive a set of formulas for each module and integrate it into our SystemQ framework,

$$E_{estimated,i} = E_{base,i} \cdot P_{generics,i} \cdot P_{lib,i} \cdot P_{freq,i},$$

where $P_{generics,i}, P_{lib,i}, P_{freq,i} \geq 1$. $E_{estimated,i}$ and $E_{base,i}$ are the estimated and base power consumption for transaction i, $P_{generics,i}$, $P_{lib,i}$, and $P_{freq,i}$ are parameters of the generics, library, and frequency effects on power. As these effects vary according to different transactions, their parameters have to be calculated for each transaction respectively.

Analogue to the area estimations in SystemQ we implement a *get_power()* function that collects the power values from all subcomponents. However, power is a dynamic effect and therefore more calculations have to be performed which lead to a small run-time overhead. In some cases the designer is not interested in a detailed power profile. In that case, our power-calculation functions provide not only the power, which is timeless, but also the energy (power · time) measured in Ws.

Power estimations at system level bear several challenges. We distinguish three types of power, namely leakage power, idle power, and active power. Leakage power is a static dissipation caused by non-ideal insulators. In presence of a clock signal an additional idle power is consumed. Active power is consumed when transactions are ongoing.

In SoC design clock-gating techniques are used to minimize idle power. At system level we do not use a clock signal—and therefore cannot apply clock gating—since it would slow down the simulation significantly. However, to mimic clock gating we implement clock-on and clock-off events that are notified when the clock is switched on and off, respectively. Thus, we can track the presence of the clock and are able to measure power even more precisely.

3. CASE STUDY

The system-level modeling of SoCs is usually conducted on individual components. The interconnect system of a multiprocessor SoC is chosen as a case study. As the SoCs become more and more complex, the interconnect system of SoCs has evolved from bus structures to component-based systems, which are more advanced and perform complex functions. Unlike the processor which is fixed once the design is finished and the memory which has a regular structure, the interconnect system varies significantly depending the structure of the SoC. In addition, the interconnect system of SoC usually takes a considerable part of the area and power of the system. Therefore, it is critical and challenging to estimate area and power of the SoC's interconnect. Furthermore, the interconnect system can become a bottleneck of the system performance. With a fast estimation at the system level, it is possible to analyze the trade-offs among area, power and performance quickly with SystemQ.

Our area and power estimation modeling is performed on the XB07 crossbar [10] components. Compared with traditional bus structures, the XB07 crossbar can set up concurrent connections between different masters and slaves as well as bridging different clock domains in SoCs. The XB07

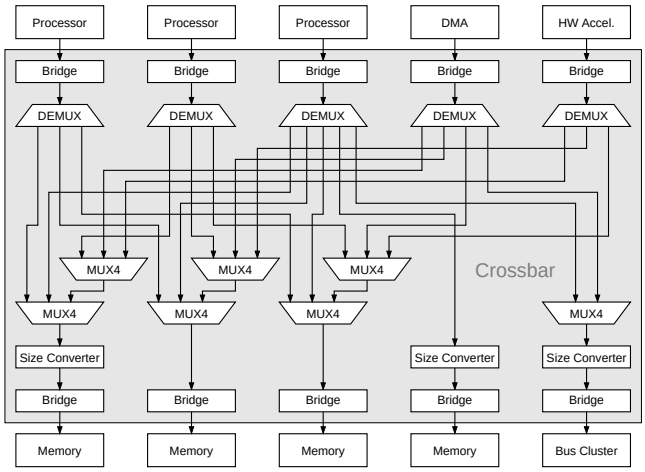

Figure 5: Multiprocessor SoC interconnect system.

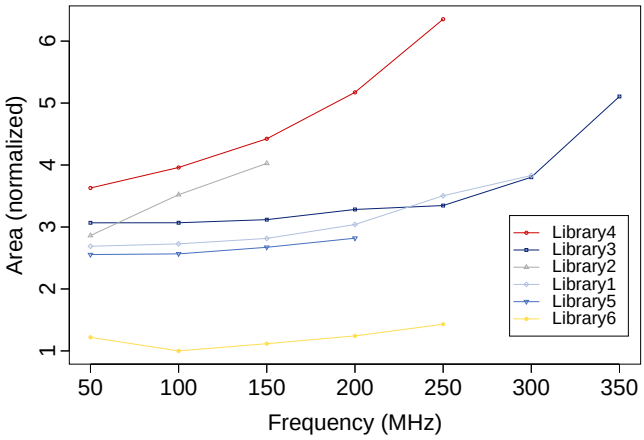

Figure 6: Area increase when clock frequency changes from 50 MHz to 400 MHz.

crossbar contains multiple sub-components such as multiplexers, de-multiplexers, size converters as well as various bridges. Depending on the setting of the SoC, the type and number of sub-components in a crossbar can be very different. Therefore, area and power consumption of the crossbar also vary significantly for different SoCs. First, we carried out the area and power analysis on 12 particular crossbar components. Then we performed the analysis on a complete crossbar system from a real design case as shown in Figure 5.

3.1 Area Investigation

Each crossbar sub-component is parameterized with 10 to 25 HDL generics with different impacts on area. The size of a crossbar also varies significantly corresponding to different standard-cell libraries as well as the frequency of operation. Therefore our investigation is performed on each crossbar component to calculate its $P_{generics}$, P_{lib}, and P_{freq}, and the overall area of the crossbar is then calculated by summing up the area of the sub-components. Within the same component, the library's effect on area is similar under different generics settings. However, the frequency's effect on area shows a significant difference depending on the generics setting.

To evaluate the generics effect of one component, generally the minimum and maximum value of each HDL generic is selected to evaluate its impact on area. Then with each setting, synthesis is performed on the RTL design, gates are mapped to the corresponding standard-cell library, and place and route is performed. Out of the large number of combinations of generics, 10 to 20 runs are mostly needed to investigate the generics effect on area for one component. The effect of each generic is measured by the changed area in percentage compared with the area under the smallest generics setting, which is the base area. Similarly, to evaluate the library's effect, the same generics settings are investigated with different libraries. The library which corresponds to the smallest area is chosen as the base library and the library effect is measured by comparing the area with the one under the base library setting. For the frequency effect, a range of 50 MHz to 400 MHz is tested with a step size of 50 MHz. To reduce the large number of analysis, the frequency investigation is performed on the largest and smallest generic setting with all different libraries. Figure 6 shows the area increase corresponding to different clock frequencies for the MUX component with six different libraries. As the timing constraints have to be met for valid synthesis, only limited maximum frequencies can be reached for some libraries.

The area under different settings is compared with the base area, and the area change resulting from specific factors is measured as a percentage number of the base area, such as $P_{generics}$, P_{lib}, and P_{freq} in Section 2.2.2. The area will be estimated according to the equation in Section 2.2.2. Since the area of a component is determined together by several factors including generics, library, and frequency, the effect of a specific factor on area differs depending on the current settings of the other factors. The most accurate solution is to include all the investigation results into a look-up table and interpolate the missing values, however, this approach requires significant effort to implement into SystemQ classes. A simplified approach is to take the mean values of all obtained parameters when they are close to each other. However, when the parameters show significant differences for certain settings, the values should be assigned manually according to the current settings. From the experimental results, it is proven that this approach also provides high accuracy.

The area estimation models are then implemented in an area calculation function of each SystemQ crossbar class. Parameters of HDL generics and library are also added in order to determine the setting under estimation. The base area of each component is also added as a parameter which can be modified as needed. $P_{generics}$, P_{lib}, and P_{freq} are taken from the investigation results of each component and inserted into the source code of the SystemQ crossbar components. When a crossbar is built from these sub-component classes, the area of each sub-component can be calculated separately according to its generics setting. Then a *get_area()* function from the top level class is called to inquire the area of each component in the crossbar and export the total crossbar area by summing up the area of all components.

The accuracy of our area estimation model is tested by comparing with the low-level area result under three different crossbar components, namely MUX, DEMUX, and DECODER. The average error is 5.6 %, 6.3 %, and 4.9 %, respectively. The error distribution for the MUX is shown

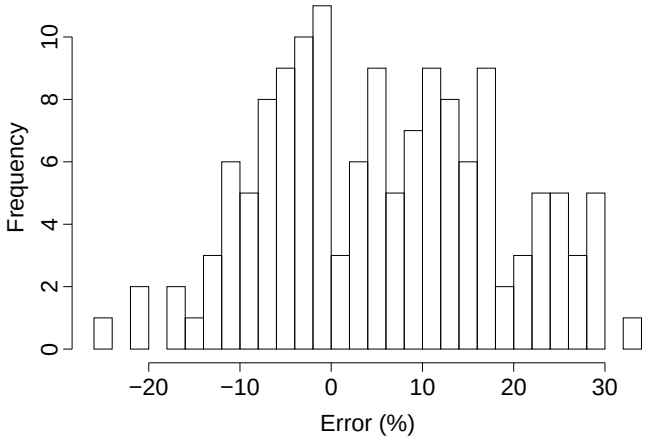

Figure 7: Error of estimated values compared to measured values.

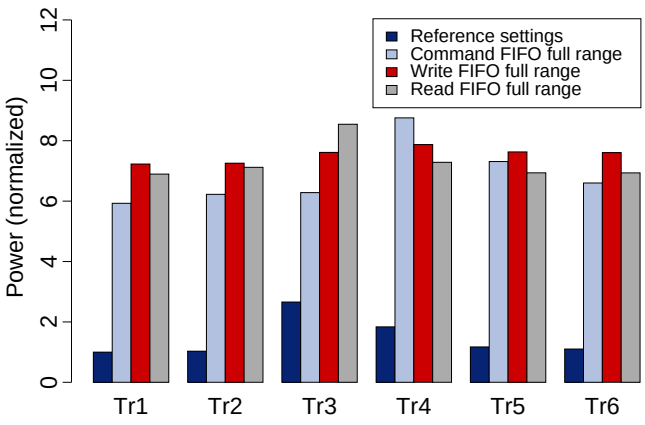

Figure 8: Generics effects on power consumption of different transactions (Tr).

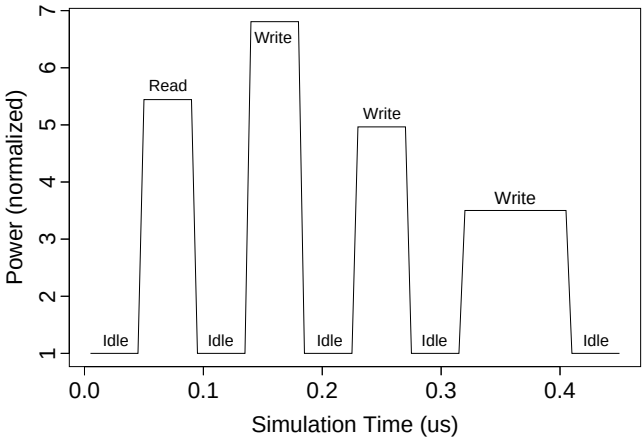

Figure 9: Power consumption of a sequence of mixed transactions simulated by SystemQ.

in Figure 7. In addition, the whole crossbar system shown in Figure 5 is built with SystemQ with specific settings. We synthesize and physically implement the system with two different voltage-threshold libraries. The estimation errors are 17.0 % and 28.1 %, respectively. The SystemQ estimations are too pessimistic in both cases.

3.2 Power Investigation

For the power consumption estimation, all the factors affecting the area also affect the power, including the HDL generics, library, and frequency. But unlike the area, the power estimation is a dynamic effect which also depends on the activity of circuits. At the system level, the activity of a crossbar can be classified into different transaction types such as read command, write command, read data, and write data. Even under the same settings, the power consumption differs significantly for different transaction types. Therefore, we need to evaluate the effects of generics, library, frequency as well as transaction types on the power consumption for each crossbar component.

To measure the power consumption a testbench is created for each component. Then a low-level power analysis tool (Talus Power Pro) is used to measure the power consumption of the component netlist with input stimuli for different transaction types. It is difficult to define the input vectors for all possible situations which the circuit might encounter. Even with all the possible input vectors defined, the number of the stimuli vectors will also be too large to perform power consumption simulation. Therefore, we determine the stimuli according to most general situations, which can set borders of power consumption for most input stimuli.

As in the area investigation, the parameters of $P_{generics}$, P_{lib}, and P_{freq} are taken to represent the effects of different settings on the power consumption. These parameters can be different depending on the transaction types. For example, as shown in Figure 8, when different generics are changed from the base generics setting, the power is increased to different extent on transaction states of Tr1 to Tr6. For the frequency effect, the major part of the power consumption is dynamic power, which is linear dependent on the current frequency. Therefore, the frequency effect in the power estimation model is simplified to P_{freq} which equals to the ratio of current frequency and the base frequency under which the base power is obtained.

The power estimation model is then implemented into the SystemQ crossbar classes. But unlike the area estimation, the power estimation is linked with the component activity. During the performance simulation, SystemQ identifies the current transaction and assigns a power consumption value depending on the transaction types and system settings. Since it is difficult to obtain the activity of the whole crossbar, the power consumption estimation model is tested on the MUX component. The power consumption is calculated over a total of 100 clock cycles where a sequence of mixed transaction types is issued, the SystemQ simulation result of the power consumption activity is shown in Figure 9. The average power consumption is then compared with the result from the low-level power analysis with the same settings and stimuli. The error of the SystemQ model is 11.5 % pessimistic compared to the result given by the power analysis tool.

4. DISCUSSION

The simple area and power estimation models implemented in SystemQ achieve relatively high accuracy compared to previous high-level area and power estimation models. The

accuracy can still be further improved in several ways as discussed later. With the new area and power estimation features of SystemQ, system engineers can evaluate the system performance in terms of trade-offs with area and power consumption quickly to select the optimum system structure. However, these models are based on a large amount of low-level synthesis results. The same analysis has to be performed on any new module or new library, which requires a lot of effort. Therefore, the analysis is preferably carried out on re-usable IP modules, where the results can be included by other systems as well.

4.1 Error Analysis

The estimation errors of 17.0 % and 28.1 % obtained by SystemQ on a total of 25 sub-components are both pessimistic compared to the real on-chip area. This is due to the fact that the hierarchical system is flattened by the synthesis tool so that global optimizations can be applied. SystemQ works on the hierarchical modules where only local optimizations can be applied. The system area is calculated by adding up the area of all sub-components. This leaves a safety margin for the system area estimation. With the global optimizations performed by the synthesis tool the real design will achieve a better result.

Area and power consumption are complicated design results, which are affected by many inter-related factors including HDL generics, library as well as frequency. It is unrealistic to explore the whole design space by testing every possible system setting. Our investigation is targeted at finding the most significant factors for area and power consumption and model their effects in simple formulas. To ensure a high accuracy, it is necessary to evaluate not only the effect from a single generic, but also the inter-dependent effects of several generics. However, due to limited time and resources, we could not investigate every possible combination. Instead, we emphasize on the significant generics for the area or power and test their interdependencies with a few synthesis runs. For those generics with a large range, a finer granularity can be achieved by performing more runs.

Another source of error comes from the modeling of these complicated effects. Especially for the HDL generics, the inter-dependency shows a complex irregular pattern which is difficult to model with simple additive or multiplicative formulas. More forms of formulas can be developed to better represent the complex generics' effects. Furthermore, the effect of one generic under different libraries can show a significant difference. However, to reduce the influence of the dependency on other factors, an average value has to be taken, which can be imprecise in some cases.

For power consumption an average number is taken of the upper and lower bound of the same transaction state but different data. This could deviate from the data in a real case. Also, at the system level many low-level configurations are not available which results in certain inaccuracies for the system-level estimation.

Last but not least, the synthesis tool increases the uncertainty of the area and power evaluation process. In our analysis, unexpected results were found out such as when a generic is increased, the area is decreased. The possible reason for this is that the optimization processes between two runs may be different. But this effect is relatively small and rare, and therefore always neglected.

Table 1: Trade-off analysis on the data path width of the crossbar for area, power, throughput, and latency (normalized).

Data width	Area	Power		Throughput	Latency
		Read	Write		
32 bit	1.00	1.00	1.00	1.00	1.00
64 bit	1.04	1.75	1.21	1.83	0.62

4.2 Trade-off Analysis

The SystemQ framework greatly shortened the analysis time compared with low-level analysis. Usually for a small component, the process from RTL to place and route needs at least half an hour, for large designs the process may take hours or even days. The calculation of area in SystemQ is just performed once at the end of simulation, and the power is calculated at every transaction state change during simulation. The calculation time is almost negligible. Once a system is built in SystemQ, settings such as generics, library, and frequency can be modified in the parameter file without re-compiling the system. Therefore, the system engineer can compare the trade-offs between structures under different settings, such as data path width or FIFO size.

An example is given below to compare the trade-offs of performance, area and power consumption for different structures. A complete data path of the crossbar is built with SystemQ. When we change the data path width from 32 bit to 64 bit while keeping the other setting constant, the performance index, the area and the power consumption change correspondingly as shown in Table 1. Compared with a 32-bit data path, the 64-bit structure increases the system throughput, lowers the latency of data, but at the same time it also increases the area and power consumption as a trade-off.

5. RELATED WORK

We find related work in two areas of research: 1) ESL frameworks and 2) area and power analysis tools:

5.1 ESL Frameworks

Artemis [8] is a modeling and simulation environment for embedded systems focusing on media processing. Artemis is a simulation environment that is based on trace-driven simulation. It is focused on coarse-grained operations that reflect data-dependent behavior. The application models are expressed using Kahn process networks (KPN). KPNs fit well into the application domain of media processing and therefore allow an efficient implementation of these applications. However, since KPNs are determinate, they cannot express time-dependent behavior and prevent modeling of, e. g. interrupts.

The Metro II framework [3] is based on the Metropolis [1] framework from the same authors. Based on a meta model specification language with formal semantics, the framework supports different models of computation and abstraction levels. The key property of Metro II is the separation of functionality and architecture. The meta-model language comprises four objects, namely processes, media, quantity managers, and netlists. Processes and media are similar to processes and channels of SystemC, respectively. Quantity managers supervise the access to shared media. They also assign physical quantities to events, e. g. time or power.

5.2 Area and Power Analysis Tools

For area estimation, there have been mature algorithms to predict the area from lower abstraction levels. PLEST [4] is a program for estimating the area of standard-cell layouts. It uses a simple probabilistic model for cell placement as well as interconnections. Given various gate-level parameters such as number of nets, total cell width and so on, PLEST is able to estimate the possible shapes of layout blocks. The error is tested to be below 10 % for the PLEST program. More techniques are developed later to explore the routability and improve the accuracy of wiring space estimations.

An RT-level model is proposed to measure the area of boolean functions in terms of gate counts in [6]. The area model based on transforming the multi-output boolean functions into an equivalent single-output function. The model is tested on some benchmark circuits and an average absolute error of about 20 % is achieved. However, this method can only be applied on combinational logic. Furthermore, measuring the area by the number of gates can be inaccurate because the standard-cell library used can also have a significant impact on the final area.

In power consumption estimations, many techniques are also developed to estimate power at layout, gate or behavior level. In [5] the general power consumption models on gate level are introduced on logic circuits, interconnections, clock distribution, and on-chip memories. Power estimation models at the gate level require the low level information of the circuit such as the capacitance of nodes, the switching activity at nodes and so on. Obtaining this information requires large amounts of simulation work.

Further models have been proposed at higher levels. In [2] an analytical model is derived to estimate the power consumption based on the information from gate-level simulations such as number of gates, switching activity and model coefficients. A behavior-level model is further derived to estimate the average switching activity by monitoring the switching activity of input and output nodes. The model is tested on different instruction states, the error of behavior-level power estimation is up to 58 %. The accuracy of this behavior-level estimation can be improved to 39 % by inserting internal nodes as test points. However, with this model, logic synthesis is still necessary to obtain the corresponding number of gates.

6. CONCLUSION

We have presented a disciplined approach to area and power-consumption estimations at electronic system level using our performance evaluation framework SystemQ 2.0. SystemQ allows system engineers to explore the design space of multiprocessor SoCs and to perform architecture performance evaluations. With its new features SystemQ is capable of pinpointing the interdependencies between area, power, and performance.

The foundation of our estimations is a large amount of data from synthesized and physically implemented hardware components. From that we have built a set of formulas and integrated them into SystemQ. We have shown the challenges that arise from applying low level effects to higher abstraction levels, especially the system level.

In a case study on a complex multiprocessor SoC interconnect system we have shown how the synthesis data can be integrated into SystemQ and how area and power consumption estimations can be performed. We have shown the background, challenges, and pitfalls of our approach. The results of our ESL estimations have been compared to the real transistor-level data. We have shown that the average error for the area estimations is up to 6.3 % for single components. For the complete system we examined two different standard-cell libraries. The error is 17.0 % and 28.1 %, respectively. The power estimation error is 11.5 % at component level.

Acknowledgments

This work has been developed in the project RapidMP-SoC. RapidMPSoC (project label 01M3085) is partly funded within the Research Programme ICT 2020 by the German Federal Ministry of Education and Research (BMBF).

The authors wish to thank W. Ullmann for providing us the synthesis environment as well as D. Margraf and R. Meijer for their support on power analysis and gate level simulation.

7. REFERENCES

[1] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, Apr. 2003.

[2] M. Caldari, M. Conti, P. Crippa, G. Nuzzo, S. Orcioni, and C. Turchetti. Instruction based power consumption estimation methodology. *9th International Conference on Electronics, Circuits and Systems, 2002.*, 2:721–724, Sept. 2002.

[3] A. Davare, D. Densmore, T. Meyerowitz, A. Pinto, A. Sangiovanni-Vincentelli, G. Yang, H. Zeng, and Q. Zhu. A next-generation design framework for platform-based design. In *DVCon 2007*, Feb. 2007.

[4] F. Kurdahi and A. Parker. PLEST: a program for area estimation of VLSI integrated circuits. *23rd ACM/IEEE conference on Design automation*, pages 467–473, 1986.

[5] D. Liu and C. Svensson. Power consumption estimation in CMOS VLSI chips. *IEEE Journal of Solid-State Circuits*, 29(6):663–670, 1994.

[6] M. Nemani and F. Najm. High-level area and power estimation for VLSI circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):697–713, 1999.

[7] Open SystemC Initiative (OSCI). `http://www.systemc.org/`, Oct. 2008.

[8] A. D. Pimentel. The Artemis workbench for system-level performance evaluation of embedded systems. *International Journal of Embedded Systems*, 1(7), 2005.

[9] S. Sonntag, M. Gries, and C. Sauer. SystemQ: Bridging the gap between queuing-based performance evaluation and SystemC. *Design Automation for Embedded Systems*, 11(2):91–117, Sept. 2007.

[10] S. Sonntag, H. Reinig, S. Linz, F. Pitter, and M. Ruhwandl. XB07: A highly reusable crossbar architecture for multiprocessor system on chip (MPSoC). In *IP Based Electronic System Conference (IP07)*, pages 307–311, Dec. 2007.