CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Bridging the Gap between Precise RT-Level Power/Timing Estimation and Fast High-Level Simulation

A method for automatically identifying and characterising combinational macros in synchronous sequential systems at register-transfer level and subsequent executable high-level model generation with respect to non-functional properties

Dissertation zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften

von

Dipl.-Inform. Kai Hylla

Gutachter:

Prof. Dr. Wolfgang Nebel
Prof. Dr. Wolfgang Rosenstiel

Tag der Disputation: 13. Januar 2014

Abstract

Knowing a system's power dissipation and timing behaviour is mandatory for today's system development and key to an effective design space exploration. Not only does battery lifetime or design of the power supply directly depend on the power dissipation of the system. Second-order effects such as thermal behaviour or degradation effects that are directly or indirectly affected by the power dissipation must be considered, too.

Various techniques for power estimation exist at different levels of abstraction. Low-level approaches provide accurate estimation results but require a lot of computational effort. High-level approaches however, allow fast and early estimates, but lack of a deeper knowledge and understanding of the hardware, implementing the behaviour. Therefore, they can only give rough estimates. What is missing is an approach allowing fast and early estimates with respect to as many relevant hardware artefacts and physical properties as possible.

This doctoral thesis tackles the problem of a fast, yet accurate power and timing estimation of embedded hardware modules at a high-level of abstraction. A comparatively time consuming low-level estimation is performed once in order to obtain an accurate estimate. By augmenting an executable high-level simulation model with this power and timing information, fast and comprehensive simulations at a high-level of abstraction using a large set of different use cases become possible. The abstraction gap between fast simulation and accurate estimation is closed.

This work describes a technique for automatically identifying and characterising combinational macros in synchronous sequential systems, such as co-processors or hardware accelerators. Using a high-level synthesis, a pure behavioural high-level system description is transformed into a cycle-accurate description at structural register-transfer level. So-called hardware basic blocks, comprising a set of jointly active RT components, are identified and characterised automatically. The characterisation uses sophisticated RT-level power models, which provide accurate power and timing estimates. The characterisation also considers as many relevant physical properties and synthesis artefacts as possible. These include scheduling and binding as well as parasitic functionality, for instance. Non-functional properties such as clock or controller power as well as static power dissipation are also considered.

Using the characterised macros, a power and timing annotated high-level simulation model is generated. This C++-based virtual prototype allows a fast, yet accurate estimation of the given design with respect to various use cases and test stimuli. Beyond that, the generated prototype can be embedded into a virtual system prototype allowing a design space exploration, far more complex and comprehensive than would be feasible by using a common estimation approach at register-transfer level.

Evaluation of the presented approach is performed using a set of several industrial and academic use cases. Results show that by having an average relative error per cycle of less than 6.93 % for most simulated clock cycles and a total error of around 1 %, a speed-up of approximately 160 $\times$ compared to an RT-level estimation is archived, while giving nearly cycle-accurate power estimates.

Zusammenfassung

Das Wissen um die Verlustleistung und des zeitlichen Verhaltens eines Systems ist unerlässlich für den heutigen Systementwurf. Es ist der Schlüssel für eine effektive Exploration des Entwurfsraumes. Nicht nur die Laufzeit der Batterie oder die Auslegung der Energieversorgung hängen von der Verlustleistung ab. Nachrangige Effekte wie z. B. das thermische Verhalten des Systems oder auch Alterungseffeke, welche direkt oder indirekt von der Verlustleistung abhängen, müssen ebenfalls berücksichtigt werden.

Es existieren verschiedene Techniken für die Abschätzung der Verlustleistung sowie des zeitlichen Verhaltens auf ganz unterschiedlichen Abstraktionsebenen. Verfahren auf niedriger Ebene ermöglichen genaue Vorhersagen, benötigen jedoch einen hohen Rechenaufwand. Verfahren auf hoher Ebene hingegen erlauben schnelle und frühzeitige Abschätzungen. Ihnen mangelt es jedoch an einen tieferen Verständnis der Hardware, welche das Verhalten implementiert. Daher können sie lediglich ungenaue Vorhersagen machen. Es fehlt ein Verfahren, welches schnelle und frühe Vorhersagen unter Berücksichtigung so vieler physikalischer Eigenschaften und Hardware-Artefakten wie möglich, erlaubt.

Diese Doktorarbeit adressiert das Problem schneller aber dennoch präziser Vorhersagen der Verlustleistung und des zeitlichen Verhaltens eingebetteter Hardware-Module auf einer hohen Abstraktionsebene. Eine vergleichsweise zeitaufwändige Abschätzung auf niedriger Ebene wird einmalig durchgeführt, um eine akkurate Abschätzung zu erhalten. Durch das Anreichern eines Modells auf hoher Abstraktionsebene mit den zuvor gewonnenen Informationen, werden schnelle und umfangreiche Simulationen auf hoher Abstraktionsebene unter Verwendung von einer Vielzahl an Anwendungsfällen möglich. Die Lücke zwischen schneller Simulation und akkurater Abschätzung wird geschlossen.

Diese Arbeit beschreibt ein Verfahren für das automatisierte Erkennen und Charakterisieren von kombinatorischen Makros in synchronen sequentiellen Systemen, wie z. B. Co-Prozessoren oder Hardware-Beschleunigern. Mit Hilfe einer High-Level Synthese wird eine, zunächst rein funktionale, High-Level Systembeschreibung in eine zyklengenaue Beschreibung auf struktureller Register-Transfer Ebene erzeugt. Sogenannte Hardware Basis Blöcke, welche eine Menge gemeinsam aktiver Register-Transfer Komponenten umfassen, werden automatisch identifiziert und charakterisiert. Die Charakterisierung nutzt fortgeschrittene Power-Modelle auf Register-Transfer Ebene, welche genaue Ergebnisse bezüglich zeitlichem Verhalten sowie der Verlustleistung liefern. Die Charakterisierung berücksichtigt darüber hinaus auch so viele physikalische Eigenschaften und Synthese-Artefakte wie möglich. Dazu zählen z. B. Scheduling, Binding so wie parasitäre d. h. ungewollte Funktionalität. Nich-funktionale Eigenschaften wie Verlustleistung durch den Controller oder den Takt ebenso wie Leckströme werden ebenfalls berücksichtigt.

Mittels der charakterisierten kombinatorischen Makros wird ein High-Level Simulationsmodell erzeugt. Dieses ist um Informationen bezüglich zeitlichem Verhalten sowie der Verlustleistung angereichert. Der so erzeugte, C++-basierte virtuelle Prototyp erlaubt eine schnelle

aber dennoch genaue Abschätzung des gegebenen Systems bezüglich verschiedener Anwendungsfälle und Teststimuli. Darüberhinaus kann der erzeugte Prototyp in einen virtuellen Systemprototypen eingebettet werden. Dies erlaubt eine deutlich komplexere und umfassendere Exploration des Entwurfsraumes, als unter Verwendung einer herkömmlichen Abschätzung auf RT-Ebene möglich gewesen wäre.

Die Evaluation des hier vorgestellten Ansatzes erfolgte unter Verwendung von verschiedenen Anwendungsfällen aus akademischen und industriellem Umfeld. Die Ergebnisse zeigen, dass bei einem relativen Fehler von weniger als 6.93 % für die meisten der simulierten Takte und einem Gesamtfehler von ca. 1 %, eine Beschleunigung von ungefähr 160 $\times$ im Vergleich zu einer herkömmlichen Simulation auf RT-Ebene erreicht werden kann, wobei annähernd Takt-genaue Vorhersagen bezüglich der Verlustleistung möglich sind.

We are like dwarfs standing upon the shoulders of giants, and so able to see more and see farther than the ancients.

(Bernard of Chartres, around 1120)

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Chapter 1

Introduction

Abstract

This chapter motivates the proposed characterisation and model generation process. It will explain why power estimation, especially on high levels of abstraction, is still an important topic and how it influences the design process. The chapter will show that despite shrinking technology sizes and transistor construction techniques like high-k and Fin-FET as well as power management techniques like power-gating, power dissipation and power density are still a matter of concern. Various old and new challenges regarding power dissipation are stated and it is described how these interfere with new design methods, processes, and tools. It will be shown that in upcoming designs, challenges like estimation of degradation effects, thermal simulation, etc., all relying on accurate power estimates, must be addressed. Power dissipation and thus power estimation will therefore still play a key role. Based on the named challenges, requirements for a sophisticated power and timing estimation process with respect to various functional and non-functional properties of embedded systems are identified and the contribution of this thesis for addressing the challenges is outlined.

After a brief history of semiconductor technology, the future development of semiconductor technology, based on the predictions made by the ITRS, is described. Todays and future challenges in electronic design automation will be shown and subsequently, this chapter will outline the scope of this thesis and its main contribution to the research community. Finally, the structure of this thesis is delineated.

One day before Christmas Eve in 1947 John Bardeen, William Shockley, and Walter Brattain, all three working at Bell Laboratories presented their supervisors the results of their work. It was a bulky piece of some centimetres in size, built from a chunk of germanium, a plastic triangle with gold foil at each of its sides, and a spring, pressing the apex onto the germanium. This device will later be known as the world's first transistor. Compared to a vacuum tube, used those days, a transistor is smaller, lighter, and more rugged, consumes less power, operates at lower voltages, produces less heat, and has a greater reliability. At that day the very impressive story of semiconductor technology started, which led to a world, were nearly every electronic device has more transistors than their inventors had dreamed of back in December 1947.

Ten years later in 1957 the first single crystal silicon was available and the following year the first field effect transistor was developed. It took another three years until the first *integrated circuit (IC)* was available from Texas Instruments. Until 1959 almost all electronic components performed only a one particular function. A circuit was created by wiring several of these components together. But then in 1960, Dawon Kahng also a Bell Laboratories employee invented the *metal-oxide-semiconductor field-emitting transistor (MOSFET)*, which started a rapidly developing semiconductor industry. One year later Fairchaild presented a camera, the first product using a monolithic IC. It took another year until the first transistor-transistor logic was invented. Again, one year past until the first IC using *p-channel metal-oxide semiconductor (PMOS)* technology was developed by RCA. Previously, only *n-channel metal-oxide semiconductor (NMOS)* transistors were used. Now, in the year 1963, it was possible to created circuits, using *complementary metal oxide semiconductor (CMOS)* technology. Decreasing feature sizes of the technology nodes allowed more and more complex circuits to be built and in 1965 Gordon E. Moore published what today is known as Moore's law [99]:

> The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. [...] Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65 000. I believe that such a large circuit can be built on a single wafer.

In mid-seventies, Moore amended his statement to a doubling every two years. According to today's interpretation of Moore's Law it states that the number of transistors on a standard processor doubles every eighteen months.

By the end of the 1960s, nearly 90 % of all manufactured components were ICs. In 1970 Cogar et al. at IBM fabricated a metal gate NMOS, which is particularly suitable for micro-electronic ICs. Only one year later Intel presented the first microprocessor. Another year later in 1972 Frederico Faggin begins to work on an 8-bit processor, the Intel 8080. Three years later, first commercial microprocessors were available with the 8080 and the 6800. In the same year the first personal computer was available. In the decade between 1970 and 1980 NMOS remained state of the art in commercial products. NMOS however has the disadvantage that a transistor is constantly conduction as long as a logical one is applied to the input. Gradually, this yields to problems with the power dissipation of the system. A disadvantage CMOS does not have due to its complementary structure i. e., either the pull-up or the pull-down network is not conducting. So CMOS gained more importance and it is still used since the 1980s. In

the years technology feature sizes dramatically decreased. At the same time the number of transistors per chip increased exponentially, ass predicted by Moore. With increasing numbers of transistors build on a single chip, the costs per transistor have drastically fallen. In 1954, five years before the first integrated circuit was invented, the average selling price of a single transistor was $5.52. Fifty years later, in 2004, the price had dropped to a billionth of a dollar or a nanodollar. One more year later, in 2005, not only a transistor, but a single Bit of *dynamic random access memory (DRAM)* was available for the same price [9]. In 2010 Intel stated at the *International Consumer Electronics Show (CES)* that the price of a single transistor is about the same price as a printed newsletter character and that about four million 32 nm transistors fit in the period at the end of this sentence [42].

Figure 1.1 compares gate length and transistor count for typical Intel desktop processors [43, 44]. It is clearly visible how well Moor's law and the *International Technology Roadmap for Semiconductors (ITRS)* gate length prediction fit to Intel's processor development.

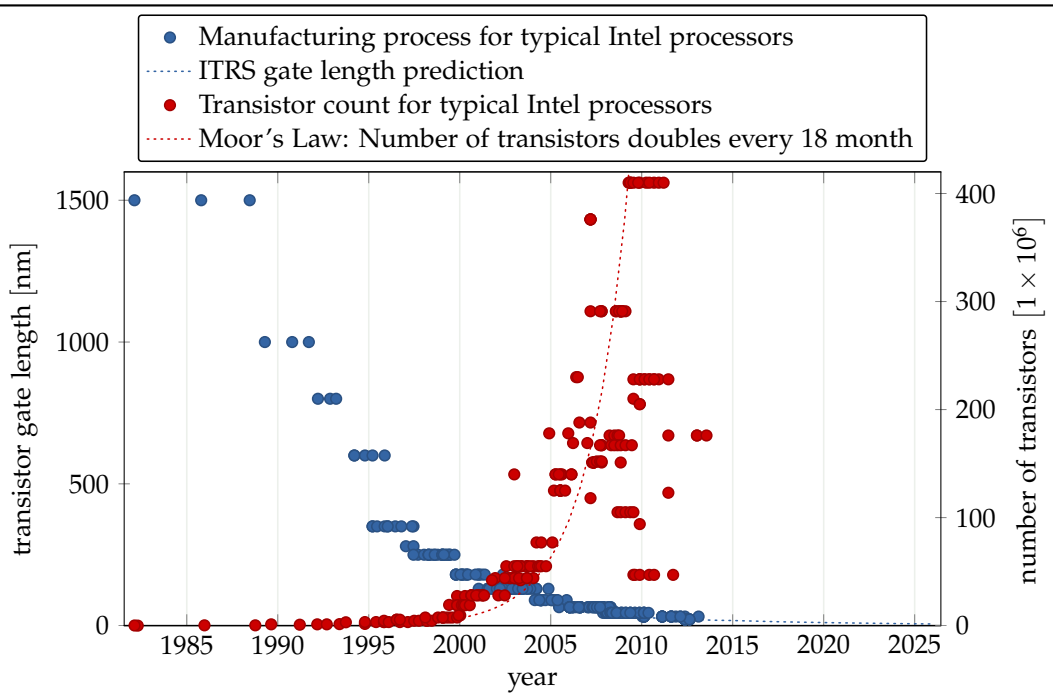


Figure 1.1: Transistor gate length versus node count — Intel's manufacturing process fits Moore's Law. Until 2005 processor's transistor count also fits Moor's Law. After 2005 a wider range of processors, containing some low-performance processors with small transistor count were introduced.

It is also visible, how much development cycles have shortened. The number of processor models and types introduced per year has steadily increased. At the same time, differences between the models also increased, clearly recognizable by the scattering starting from the year 2000.

1.1 Today's Challenges in Hardware Design

Already back in 2008 Rabaey et al. stated that there will be a change in the way computation takes place [110]. High-performance computing like climate simulation, particle physics, and material research as well as personal desktop computers still has a major part in the IT landscape. But the share of distributed systems like sensors nodes, environmental control, and especially mobile computing devices is massively increasing. By today, a high-end car has about 95 processors embedded in several sub-systems [79], for example. This number is to increase over the next years.

Using an increasing number of transistors to build the systems is also known as *More Moore*. With a larger transistor count, the system can handle different tasks. By incorporating different modules like processing units, co-processors and accelerators as well as memories on a single chip, so-called *Systems-on-a-Chip (SoCs)* are build. But not only an increasing number of transistors and more and more digital modules are incorporated to build the systems. There will also be a diversity of the systems. That is, future systems also known as *cyber-physical systems* will include non-digital parts like analogue and radio frequency, high-voltage power, sensors and actuators, or even bio-chips, containing a complete *Lab-on-a-Chip (LoC)*, where the last ones are especially interesting for environmental monitoring and health applications [9]. Wired or wireless networks allow the individual systems communicating with each other. This trend is known as *More than Moore*. While in the past systems were built in hardware completely i. e., extremely parallel and extremely low-power, recent systems are moving to more flexible application and domain specific processors at low frequencies. These processors are then combined with highly specialised hardware accelerators [79]. A right balance between performance and flexibility must be found.

All these different systems contain highly specialised parts with a dedicated purpose and a clearly defined functionality. Despite platform-based design [121], reuse of available *intellectual property (IP)* components, and evolutionary product lines, development of new systems still requires design of new parts implementing new functionalities. That is, if a system needs a new functionality it cannot be built by only re-using IP modules [123]. Only by using all these different techniques a new system can be build.

Shorter development cycles and a larger variety of implemented functionality require faster design and development iterations. Thus, faster tools are required that can handle more and more complex systems while requiring less time for estimation and optimisation. This is especially true when designing *application specific integrated circuits (ASICs)*, whose highly specialisation often prohibits reuse of complete and pre-existing platforms. The problems of the overall design process for complex and heterogeneous systems are addressed by the *Seventh European Framework Programme (FP7)* integrated project COMPLEX [1, 63, 65, 67], in which the techniques proposed in this thesis were incorporated. The COMPLEX project provides a system design process, able to deal with a wide variety of heterogeneous embedded systems, including hardware, software, and IP modules.

Generally spoken, an embedded system design process is a sequence of decisions that finally lead to a concrete hard- and/or software implementation of the indented functionality. The decisions are made with respect to the designer's expectations in terms of functional and non-functional properties of the system. These properties need to be captured and tracked

during the entire design process. All dimensions i. e., functional behaviour, timing, power, performance, memory, area, etc. of the design space need to be explored, in order to estimate the costs of the final system implementation. For this purpose a dependable and accurate model of the future system and workload scenario is necessary. In the past, embedded systems have only implemented simple functions, but nowadays complete, heterogeneous, and complex systems, like SoCs or more recently cyber-physical systems, can be build.

Besides the complexity of the design process for heterogeneous systems, there are some well-known issues that still must be regarded as well as some new concerns that must be tackled by today's design process. The most important ones are given in the following.

Power Dissipation With an increased range of functionality implemented in a single SoC, power dissipation of the chip also increased. Even though smaller technology feature sizes lowered the demand for power of a single transistor, the number of transistors used on a SoC increased dramatically. To face increasing power dissipation, various techniques have been developed, starting with establishing CMOS technology, usage of multi gates, or introducing other new transistor technologies. As will be shown in detail later, power dissipation is two-fold: dynamic power dissipation, caused by switching transistor capacitances; and static power dissipation as an artefact of the technology, used for implementing the design.

While scaling down technology feature size shifts the relation between the two types of power dissipation from dynamic to static power dissipation, the introduction of new technologies like *high-k* or *Fin-FET* transistors tremendously reduced static power dissipation. With current technologies, dynamic power has a larger share of the total power dissipation than its static counterpart. The ITRS predicts that this share will even increase in the next years [2–4]. Figure 1.2 shows the predicted development of dynamic and static power dissipation per transistor's gate-width for a high-performance technology.

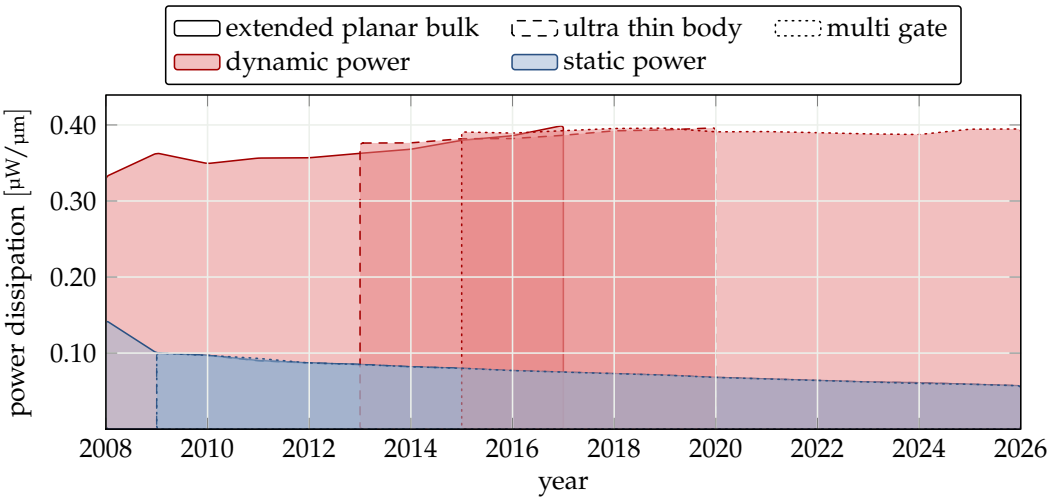


Figure 1.2: Dynamic and static power dissipation — Over the years different technologies emerge. But dynamic power dissipation (assuming 0.33 % active time) has always the major share of the total power dissipation.

Having nearly constant power dissipation per transistor, while building more transistors onto a single chip, the total power dissipation increases. In 2003 Gupta and Najm argued that power dissipation could exceed 30 to 50 W [69]. For processors this prediction had become true, as is visible from Figure 1.4. In current system design, power dissipation limits the performance of the system. Back in 2001 the ITRS predicted that by the year 2006 processors will run at 6 GHz, which never happened. Even today recent high-performance processors are not running faster than about 4 GHz, not at least because of the processor's power dissipation. Power efficiency and performance per Watt have become crucial metrics [79].

Most techniques for reducing dynamic and static power dissipation come with a certain overhead in terms of required chip area or they lower the operation speed of the system, for example. In order to perform a trade-off, it is required to see how the individual techniques or combinations of them influence the overall system behaviour. Power estimation is evolving from tasks like comparing design alternatives with respect to their power dissipation, to sophisticated use cases like chip-level power grid analysis, IR-drop calculation for static timing analysis, or hot-spot sensitive system floor planning [108]. It is also the basis for an estimation of the system's thermal behaviour and also the indirect cause for degradation and ageing effects, for example. All these effects cause emerging problems as will be shown later in this section.

Battery runtime Usually power dissipation was estimated in order to find the least power-consuming design. Lower power dissipation allows smaller batteries to be used, or, assuming the same battery capacity is provided, extends the runtime of a mobile system. These arguments still hold today, but more and more mobile devices enter the market that are providing a large computational power for a large range of applications, which on the other hand will drastically shortening the battery lifetime. Today's smartphones' batteries last about one to two days, while a couple of years ago a typical mobile phone's batteries lasted about 14 days. Besides the well-known mobile phones and mobile gaming devices, a new class of mobile computers have gained large parts of the consumer marked. These *net-books*, *tablets*, and *ultra-mobile PCs* are less than a sheet of paper in size, weight only slightly more than 500 g but provide the same amount of computational power like a common desktop computer only a couple of months ago. These mobile devices have to be very responsible with the available amount of energy in order to provide the required computation power while maintain a suitable long run-time.

Power Density Besides the ever increasing market demand for more computational power and longer run-times of mobile devices, the device's internal power density is an important technical concern. Although the number of transistors doubles every 18 months, shrinking sizes of the technology nodes cause the die size to remain nearly the same. It is predicted by the ITRS that scaling technology nodes down from 10 µm in the 1970s, 180 nm in 2000, 32 nm in 2009, and finally to 22 nm in 2011 will lead to a feature size of about 5.80 nm in 2026 [4]. Figure 1.3 shows the predicted development in the coming years.

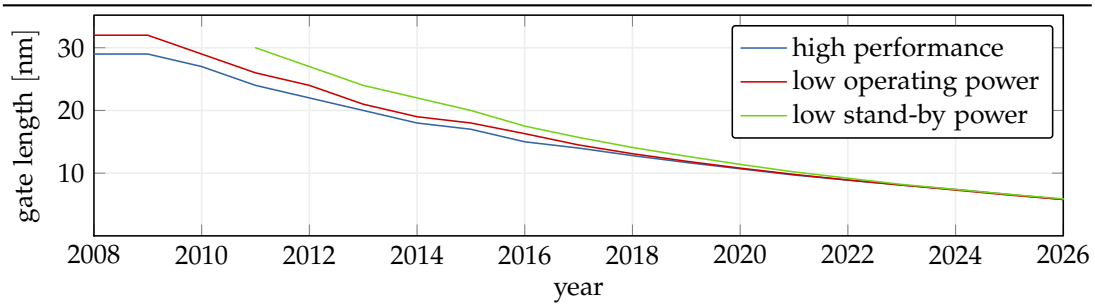


Figure 1.3: ITRS gate length prediction — While technology could be scaled down in the past decades, it is assumed that between 2020 and 2023 feature size reaches atomic level and thus will bring up a complete new class of problems and technical challenges.

While length and width of a single transistor shrink over the years, power per gate-width stays nearly the same, as Figure 1.2 on page 5 shows. If halving the length and width of a transistor, respectively, the power dissipation of the transistor is also halved. However, four times as many transistors can be placed on the same surface. This development leads to an increasing power density inside the die. Figure 1.4 shows this effect for typical Intel processors [43, 44]. The often cited scenario with ever and ever increasing power density as predicted by Borkar [29] did not occur. In 1999, when his article was published, a processor requires significantly less than 100 W/cm^2. Since then the power dissipation broke that barrier, but did not already achieve the power density of a rocket nozzle as predicted. On the contrary, since 2005 power density is slightly decreasing, showing the effort spend by the industry to tackle the problem. Ever increasing power density has been prevented by massive usage of parallel processing, for example. But power constrains still prevent the system from operating all its modules at full performance at the same time [50]. Even today, power dissipation is still a problem of concern, causing a large range of limitations for the devices to be built.

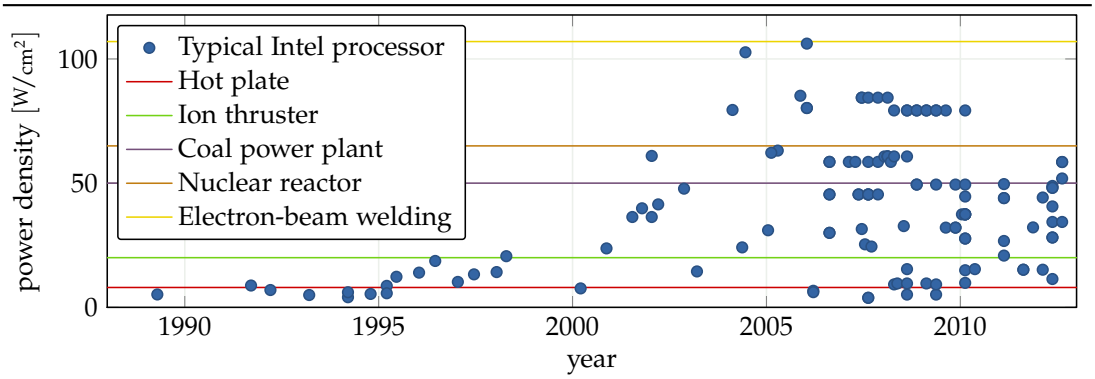


Figure 1.4: Processor power density — Before 2000 power density follows Moore's Law. Between 2000 and 2005, power density of typical consumer processors still increases, but mobile low-power processors were introduced. Since 2005 the power density of processors does not increase any more and a large variety of different processor types is available.

Heat and Cooling Each milliwatt dissipating during operation of the system can be considered to be converted into heat. The more power dense a certain area is, the hotter the chip gets at that particular point. The heat must be hauled from the inner of the chip to the outside of the package. The heat is emitted to the environment using heat spreaders, fans, water-cooling systems, or even more powerful cooling mechanisms. When building integrated 3D-stacks, heat becomes a serious issue. Hot-spots located in the inner of the stack cannot be cooled adequately, leading to an increased temperature of the surrounding parts of the stack. Expensive cooling techniques like special cooling layers between ordinary die layers and the introduction of thermal *through-silicon vias (TSVs)* are required, in order prevent the stack from damage. All these techniques increase the overall costs of the chip and its package.

Degradation and Ageing Besides increasing manufacturing costs, heat along with high power dissipation introduces new effects like degradation and ageing, which lower the yield during production as well as the expected lifetime of the system [30, 58, 113]. Even worse, smaller technology nodes are more susceptible for these effects than larger ones. Thus, degradation and reliability are currently of researchers' interest. Most degradation effects like electro migration, *hot-carrier injection (HCI)*, or *negative-bias temperature instability (NBTI)* depend on the temperature, the system is running at and thus the power dissipated by the system as well as the electrical currents flowing through the transistor.

1.2 Challenges and Requirements for High-Level Power Estimation

The previous section showed that many challenges in today's hardware design process are power related or are directly caused by the system's power dissipation. Therefore, reducing power dissipation is a key strategy in the future's design process. Recent research showed that the way energy is drawn from a battery has a high impact on the total amount of energy available to the system [89]. That is, in battery-powered systems the design target is not necessarily average power reduction, but battery lifetime extension [15, p. 140].

Nevertheless, if peak or average power dissipation or even battery lifetime is the optimisation goal, accurate power estimation is one key to solving today's and tomorrow's power-related problems in the design process. A new and sophisticated power estimation technique is required that is able to cope with today's complex embedded hardware designs. In today's full-custom hardware design, there are three major influences that must be regarded:

1. The overall workload of the system i. e., the *computational workload*.

2. The concrete hardware architecture, which is not predictable without having at least a net list at *register-transfer level (RTL)* or better an even more detailed view, introducing artefacts like parallelism or parasitic functionality influencing the power and timing behaviour.

3. The influence of physical properties like supply voltage, temperature, or process parameters significantly affecting dynamic as well as static power dissipation.

Figure 1.5 shows the power breakdown of a smartphone for various use cases. Analysis had been done by Carroll and Heiser for a smartphone with freely available specifications and with respect to typical use cases [37]. Even if parts like mass storage, display, or wireless connections such as *wireless local area network (WLAN)*, Bluetooth, etc. require a large amount of energy, computational components like the main processor core(s), hardware accelerators, or hardware implemented audio/video codecs still have a major impact and must be regarded, accordingly. The figure clearly shows that energy demand of a certain part of the overall system highly depends on the scenario in which the system is currently been used.

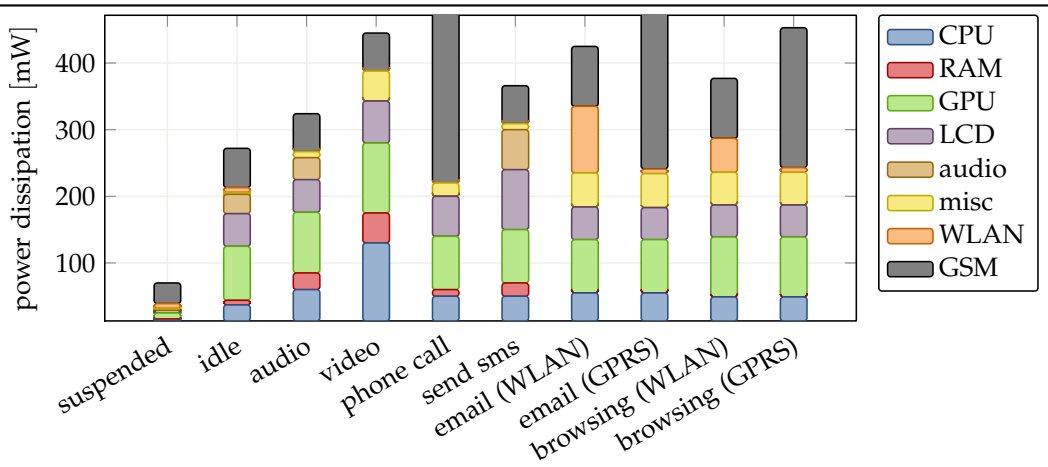


Figure 1.5: Power breakdown of a smartphone — Energy demand of the individual parts of the overall system highly depends on the use case the system is used within. For use cases *phone call* and *email (GPRS)* with 1040 mW and 601 mW, respectively the total power dissipation is very high and had been cropped in the figure.

The smartphone example sketches the character of today's embedded systems. They are complex and heterogeneous i. e., they are built from different types of sub-modules including hard- and software as well as IP modules. In these heterogeneous SoCs or cyber-physical systems, single parts of the system cannot be considered separately. They must be considered together and while interacting with each other in order to capture and make statements about the overall system behaviour and the particular part's individual workload [85]. Of course, a realistic workload must be applied to the system during estimation.

Due to the different nature of each module type, each one requires its own estimation technique and sophisticated characterisation tool. In the past, different sophisticated disciplines were developed, each one focusing on a certain module type. But today's design process requires that all modules must be considered jointly in order to obtain reliable estimations of the entire system. An estimation process must consider the overall system with all different types of modules, while interacting with each other. Due to the complex nature of the overall system estimation, this topic cannot be covert in a single thesis. More work is required, but has to be done with respect to the overall problem. This thesis focuses on ASICs, hardware co-processors, and accelerators. Despite the restriction on one module type, it must be possible to embed the proposed estimation process in a larger estimation framework, allowing to estimate the overall system.

Typical embedded systems consist of a particularly large number of transistors. The ARMv7-based SoC of an iPhone 5 is built in a 32 nm, high-k metal gate and low-power technology from Samsung. It has several millions of transistors on an area of 96.71 mm^2. It is obvious that timing and power dissipation of such large and complex systems cannot be estimated at transistor level. An abstraction is required to cope with the increasing complexity. Abstraction typically takes place in terms of a spatial abstraction. That is, several building blocks of a certain abstraction level are considered as building block in the next higher level. Figure 1.6 shows typical abstraction levels and how they build the next higher level.

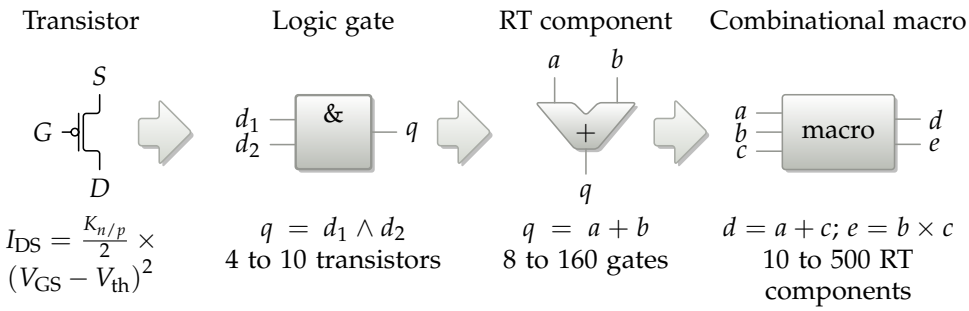


Figure 1.6: Abstraction levels — Abstraction takes place in terms of a spatial abstraction. That is, several building blocks of a certain abstraction level are considered as one building block in the next higher level. Characterisation of the new building block is performed by using the power models of the lower abstraction level.

In the past, the abstraction level for an early power and timing estimation had to rise from transistor to RT-level to cope with the ever increasing system size and complexity. While abstracting, each building block of level L_i is defined in terms of building blocks from level L_{i-1}. For lower levels of abstraction such as gate- or RT-level it is possible to provide a library of pre-characterised building blocks. The sets of building blocks in each level are small enough for characterising all possible peculiarities of all building blocks.

For logic gates the library provides operations like AND or OR with different bit widths, totalling in about 100 entries in the library. For *register transfer (RT)* components, the library provides operations like $+$, $\times$, multiplexing, or storing. Again, each operation is available for different bit widths and with different implementations such as *small* or *fast*, totalling in hundreds of different RT components in thousands of different sizes. So-called *soft macros* had been introduced to cope with the increasing number of components. With each higher level of abstraction, the models must become more flexible. All the pre-characterised blocks can then be instantiated from the library when composing the system.

With today's systems a barrier is reached that cannot be bypassed by simply raising the level of abstraction to the next level. At abstraction levels above RTL, it is no longer possible to provide such a pre-characterised component library. With an increasing level of abstraction, the number and complexity of building blocks in a particular level also increases. This is true for various natural systems and even Lego [38]. A combinational macro performs a large and possibly independent set of arithmetic operations. Apart from generic macros like glue-logic or simple arbitration logic, there are way too much possible operations that can be performed by a combinational macro, prohibiting a characterisation of all possible macros. It is required that the macros are identified and characterised for each system individually.

Since the library of low-level building blocks is generated only once and is then used for a very large number of systems, its generation i. e., the characterisation of all building blocks may take some time. For combinational macros the characterisation process must be fast, since the macros are characterised for each system individually. That is, a modified system implementation requires a re-characterisation thereof. But once all combinational macros have been characterised, the generated models can still be used for a larger range of use cases, improving the system evaluation and design space exploration.

1.3 Scope and Contribution of this Doctoral Thesis

This thesis tackles the problem of power and timing estimation of full-custom hardware accelerators and co-processors, embedded in a larger and heterogeneous system. Main goal is to provide a methodology that allows a fast, yet accurate power and timing estimation at a high level of abstraction. In this thesis, a novel approach is presented that:

- utilises the existing high-level synthesis and power estimation tool PowerOpt for performing a characterisation of full-custom hardware modules that are part of a larger and heterogeneous system,
- uses the estimates obtained during characterisation at RTL for creating a high-level power and timing augmented simulation model of the given hardware module, and
- utilises a fast compiled simulation with power and timing estimation capabilities for increasing estimation speed, while retaining suitable accurate estimation results.

The new high-level power and timing estimation process for embedded systems smoothly integrates into an existing design process. This provides maximal benefit while modifying the existing design process as little as possible. The new estimation process enables a design space exploration at a very high-level of abstraction based on accurate power and timing estimates obtained at RTL. Among other things, the main contributions of this thesis are:

1. A procedure for automatically identifying combinational macros in a given RT-level data path and its corresponding controller.
2. Two different techniques for characterising power dissipation and timing behaviour of the identified combinational macros.
3. A way for characterising synthesis artefacts like static power as well as dynamic power dissipation due to controller and clock-tree activity.
4. A method for characterising the aforementioned metrics based on structural properties only, allowing a subsequent estimation with respect to different supply voltages and clock frequencies.
5. A process for generating a power and timing annotated high-level model of the given hardware module, based on the previously identified and characterised macros and synthesis artefacts.
6. A technique allowing the generated virtual prototype to be embedded in a virtual system prototype, enabling an estimation of the entire system with respect to individual module interactions.

1.4 Structure of the Thesis

After some motivation given in this chapter, the following Chapter 2 introduces some basics and fundamentals, the presented approach relies on. These include the definition of an embedded system and the steps of a typical design process. Causes of static and dynamic power dissipation and their various subtypes are given. It is also outlined how power and timing estimation can be performed at lower levels of abstraction. These fundamentals give the basis for the content of this thesis and might be skipped by sophisticated readers.

Related work regarding power and timing estimation as well as model generation is given and assessed in Chapter 3. The various approaches are explained and it is reviewed how well they met the requirements to a high-level estimation process, identified earlier in this chapter. Based on the review results, the approaches are classified and suitable concepts for developing a new high-level power and timing approach are identified.

Main work of this thesis is given in Chapters 4 and 5, respectively. After defining the input model, the former one describes the development and usage of the newly developed estimation process, including identification and characterisation of so-called *hardware basic blocks*.

Chapter 5 however, addresses the generation of a power-aware executable high-level model. It describes how the combinational macros, previously identified and characterised, along with the controller's finite-state machine are transformed into C/C++ code, allowing a compiled simulation of the system.

The overall approach and all of its sub-parts are evaluated in Chapter 6, using a set of industrial and academic example systems. The evaluation will show the efficiency and the performance of the proposed approach. Finally, a conclusion and future work are given in Chapter 7.

Chapter 2

Fundamentals

Abstract

The basics for understanding cause and effect of power dissipation as well as basics concerning hardware design and high-level synthesis are given in this chapter. It gives an introduction into the topic of high-level synthesis, describes the causes of power dissipation, and explains power and timing estimation of embedded full-custom hardware modules at different levels of abstraction. It will also introduce all technical terms used in this thesis. These are the fundamentals for the further understanding of this thesis. Sophisticated readers may want to skip this chapter.

First, this chapter gives a definition of an embedded system and different design methodologies are shown. Some ways of describing and simulating an embedded system are outlined. The chapter then shows how a system, given at electronic system level is transformed and refined into a description at the electrical level. Different levels of abstraction and the corresponding synthesis steps are shown. This chapter also points out the different causes of dynamic and static power dissipation, respectively. Parasitic effects like glitches, hazards, or parasitic functionality are also introduced and explained. Finally some basics of power and timing estimation at different levels of abstraction are given.

BEFORE a new power and timing estimation process for full-custom digital hardware modules can be developed, a good understanding of the underlying basics is required, beginning with the specifics of embedded systems. In contrast to a general purpose computer such as a personal computer, an embedded system is limited in hard- as well as software functionality. Its key characteristic is that it performs a dedicated function. Because of its dedicated functionality, an embedded system typically provides only a very limited or even no human interface at all. As the name suggests, it is embedded as part of a larger system. Generally speaking, it can be said that there exists a large variety of embedded systems. They range from consumer electronics like portable multimedia players or mobile phones to vehicle control systems in avionics and automotive as well as autonomous audio/video surveillance systems, for instance. Except for some highly specialised systems, embedded systems are often mass-produced articles.

Niemann states that embedded systems typically have real-time constraints and are hard to program [106, sec. 1.1]. They often consist of hard- and possible software parts. Different types of implementations i. e., ASICs, *field-programmable gate arrays (FPGAs)*, or *digital signal processors (DSPs)* are possible when developing an embedded system. If the system contains software parts, they can be executed on an embedded processor or soft-core that is instantiated within the ASIC. It is also possible to develop or customise a special processor core with a reduced and application-specific instruction set i. e., an *application-specific instruction-set processor (ASIP)*.

If the system's application is safety-critical like in automotive, avionics, or health applications, special design steps like formal verification are necessary to check and met the safety requirements. If the system is part of a mobile and/or autonomous device, the system must be energy efficient in order to extent battery lifetime. Another way is to design systems capable of harvesting energy from their environment.

Bailey et al. pointed out that development of an embedded system is made across a number of levels of abstraction at the same time. The same holds for the different aspects of the system [11, p. 167]. It can be stated that the number of objects in a design decreases exponentially with higher levels of abstraction [56].

2.1 Technical Terms

Prior to explaining the technical basics of power dissipation and estimation, some technical terms must be introduced, starting with an explanation of how signal values are interpreted. Without loss of generality, there exist several techniques for interpreting and encoding signal values. In the reminder of this thesis it is assumed that a clock cycle starts at a rising edge of the clock signal. The falling edge occurs in the middle of the clock period. The clock cycle ends at the next rising edge. Regarding signal encoding, it is assumed that states of the controller's *finite-state machine (FSM)* as well as multiplexer select signals are encoded binary. Registers and handshake signals have an active-high semantic. An exception to this rule are memory control signals like chip- or write-enable, since these are historically active-low. Regarding power gating, an implementation using PMOS gating is assumed.

For lower levels of abstraction there is a common understanding of the used wording. Unfortunately this is not true for higher levels of abstraction. The following nomenclature is used within this thesis:

System / Virtual system prototype The, possibly pure functional, high-level description of the intended behaviour. It is typically given as C/C++ source code. The virtual system prototype is the power and timing augmented version of the system, built from the individual virtual prototypes of the system's modules.

Module / Virtual prototype A part of the overall system. Typically, a module is of a certain type like hardware, software, or IP. Focus of this thesis is on hardware modules. The virtual prototype is a power and timing annotated version of a module and is generated during model generation.

Design A specific instantiation of a full-custom hardware module. That is, a concrete implementation generated during high-level synthesis, using a specific set of synthesis and target parameters. During design space exploration, several different designs i. e., instances of a module are evaluated and compared against each other.

Process The behaviour inside a module that is running in parallel to the other processes of the same module and of course to the other modules of the system. A process consists of an RT data path and its corresponding controller.

Controller An FSM, defining the control flow of the process. The FSM's output symbol is used to control the data path, while the input symbol is used to obtain the data path's current state i. e., register values etc.

RT data path The behavioural implementation of a single process. The data path performs the computation by utilising RT-level functional units such as adders and multipliers as well as data-flow controlling blocks like multiplexers etc.

Operation A minimal part of a functional description. Typically a mathematical operation like an addition or multiplication.

Operator An RT component allowing the execution of a certain mathematical operation. Examples are adders, multipliers etc. Multiple operations can be mapped to the same operator.

RT component Besides the mentioned operators, additional RT components like multiplexers, registers, etc. are required. An RT component itself is built from several logic gates.

Logic gate / Standard cell Implements a logical function like AND, OR, etc. The standard cell in turn is built from transistors.

Transistor / Device A semiconductor device for switching electrical signals.

2.2 System Design Methodologies

Gajski et al. have identified three different methodologies of system design that can be assigned to different periods of time [55, pp. 18]. In the 1960s to 1980s *Capture and Simulate* was state of the art. Behaviour was specified without giving details of the implementation. The behavioural specification was split into blocks, which in turn will be refined down to gate or even transistor-level. Simulation and verification was then performed at this low level of abstraction [131].

From the late 1980s to the late 1990s a methodology called *Describe and Synthesise* was used [55]. First usable synthesis tools were available and the behaviour and structure could be captured at logic level. In other words, the design could be described in terms of Boolean expressions or FSMs. The synthesis tool then creates the net list implementation. Main benefit was that both, the logic level as well as the description in terms of standard cells could be simulated and that the equivalence between both could be verified.

Since the early 2000s, a methodology is used that could be called *Specify, Explore, and Refine* [55]. For the first time the level of abstraction increases to system level and models include both, hard- and software. The design process starts with an executable description of the system's behaviour. Exploration is performed at one level of abstraction. If requirements are met, the model is refined and exploration is repeated at the next lower level of abstraction.

The system itself can be built using three different approaches [55, pp. 35]. First, there is the *Bottom-Up* approach were the system is assembled from components and modules from a library. That is, before creating the system the component library must be available. It also must contain *all* possible components, as briefly outlined in Section 1.2. Bouyssounouse and Sifakis call this approach *Component-based Design* and mention that it is widely used especially when modelling large software systems [31, chap. 11]. They also broach the issue of component models for embedded systems, which helps formalising the components' interfaces.

The currently used design approach is typically a *bottom-up* assembly of independent subsystems to constitute a SoC. But with more complex SoCs, the design process starts to be more *top-down*, which is the second approach. It builds the system from a set of applications that is mapped onto a multi-processor platform [79]. A top-down approach refines a system only once the entire design is finished at one level. For reaching the next lower level of abstraction, each module and component is decomposed into smaller components [55]. Main drawback of this approach is that system metrics like power and timing are not available until the last step of the design process has been carried out. It is therefore difficult to perform design optimisations at earlier steps of the process.

A third way combines the benefits of the bottom-up and the top-down approach. It is known as the *Meet in the Middle* approach [55]. Here, the top-down approach is used at higher levels of abstraction, while the bottom-up approach is used at lower levels of abstraction. That is, a system is refined until components from a library are available. The level on which the top-down and the bottom-up approach met can vary. A well-known meet-in-the-middle approach is *Platform-based Design*. Platform-based design, as described by Bouyssounouse and Sifakis [31], starts by defining a so-called *platform*. The system is decomposed into processing elements and communication structures. By choosing specific modules from the library, an

instance of the platform is built. This helps to prevent redesigning the SoC from scratch for each generation or for each application. Typically, 70 to 80 % of the SoC does not have to be regenerated. Most modules can be reused [79], but not all functionality can be implemented by re-using IP modules [123]. New module types, which are not available from a library, must be refined to lower levels of abstraction, until components from a library are available. For the approach presented in this thesis, this level is RTL, for which there exists a corresponding component library.

Considering non-pre-existing hardware modules, using the meet-in-the-middle approach allows statements about the system's metrics as soon as RTL has been reached. But for today's complex systems it is still required to provide high-level models allowing an estimation of the entire system's implementation. One way to cope with the lack of a generic component library above RTL is to perform a design estimation and characterisation, whose result is a power and timing augmented high-level model. Classical *Front-End tools* allow capturing the system and developing the platform, whereas *Back-End tools* provide the ability to develop hard- as well as software. The characterisation and model generation approach presented here can be considered to be a reverse back-end tool. It does not provide a refinement of a particular module, but it uses the low-level model to create an enriched high-level model, which in turn provides fast, yet accurate statements about the system's metrics.

2.3 Hardware Design and Synthesis Process

As mentioned in Section 2.2, this thesis will utilise a meet-in-the-middle approach, which performs a top-down refinement of the system until components from a lower-level library are available. As discussed later, there are a lot of tools for high-level synthesis as well as power and timing estimation available. In this thesis the power optimising high-level synthesis and estimation tool PowerOpt is used. However, all techniques and methods mentioned in this thesis can be implemented in or applied to other tools and approaches as well.

Using the meet-in-the-middle approach, designing an embedded system typically starts with a non-formal, colloquial specification of the system to build. Only functionality, but not its implementation is described. Most companies start with a description embodied in an executable specification, but this starting point is already some type of an implementation solution that will have a strong influence on the final implementation [11]. This hypothesis is also supported by Chapter 6, in which the proposed approach is evaluated.

Based on the functional description a coarse-grained partitioning is performed. The system is split into blocks, each one implementing a certain part of the overall functionality. Interconnections between blocks represent communication between connected blocks. At this point early decisions about the hard- and software mapping are made. Each of the blocks represents a *processing unit*. This can be a processor, executing software tasks, a hardware accelerator or co-processor, or even a third-party IP module. It is also possible that multiple tasks are mapped onto the same processing unit. This is especially true for software tasks that are mapped onto the same processor core. Communication between blocks is mapped in the same way. Again, multiple communication relations can be mapped onto the same communication structure such as a bus or the like.

For each of these blocks, a description of the functions, implemented by the particular block, is created. This block and line structure along with block-associated descriptions are referred as *electronic system level (ESL)*, where the semi-automatic synthesis process starts. Such a typical hardware synthesis process is shown in Figure 2.1. An overview of the topic of high-level synthesis is also given by Coussy et al. [45], for example.

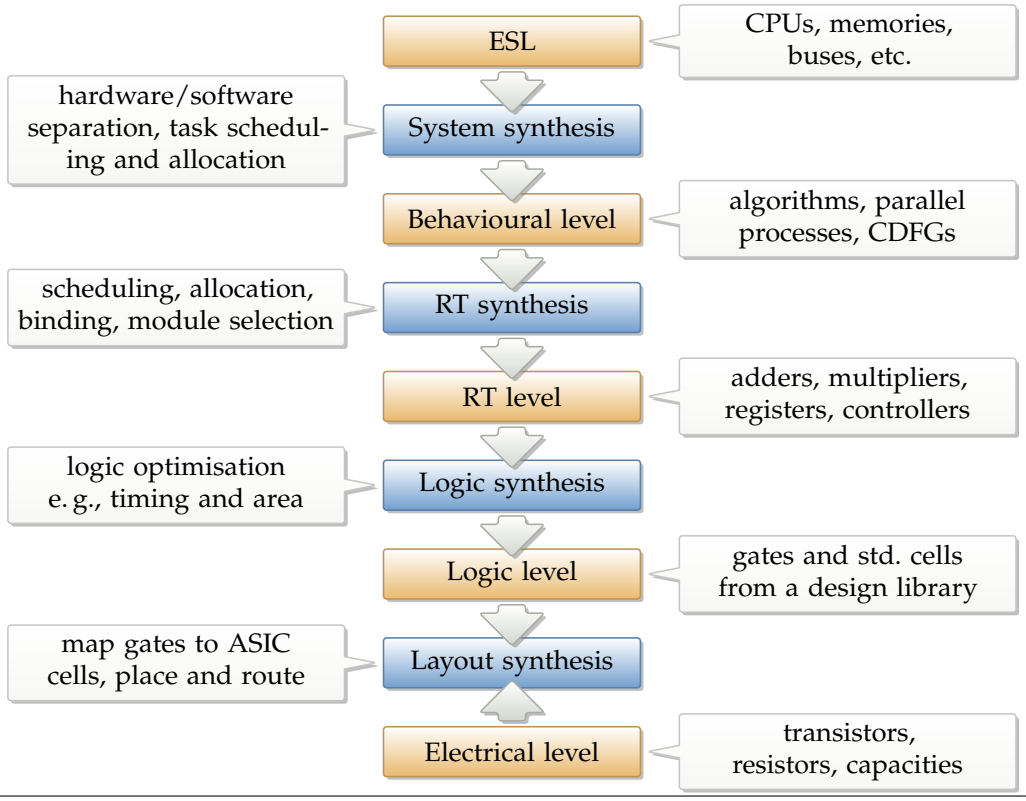


Figure 2.1: Typical synthesis process — The process starts at a very high level of abstraction. During the individual synthesis steps, the design is refined. The electrical level is represented as a database, containing pre-characterised cells that are instantiated during layout synthesis.

In the figure, typical design elements and tasks are given for each level of abstraction and synthesis step, respectively. In the following, each of them is outlined briefly.

Electronic system level At system or electronic system level, processors cores, memories, buses, IP cores, etc. are typical building blocks. The functionality is given as a set of communicating tasks that are distributed over the building blocks. Timing is only determined by external constraints, as defined in the specification, for example. Values are given in terms of very abstract data types.

System synthesis System synthesis transforms the system into a behavioural or algorithmic description. This includes identifying which parts of the system are implemented in

hardware and which ones are implemented as software. This is known as *hardware/software separation*. During synthesis, the execution order of tasks is determined, which is also known as *task scheduling*. Finally, *task allocation* is performed, which assigns the tasks to specific hardware resources.

Behavioural level At behavioural level, which is also known as algorithmic level, blocks communicating via signals form the system. The behaviour itself is implemented as algorithms, parallel processes, *control and data flow graphs (CDFGs)*, etc. Timing constraints are given either external or internal. Values are given more detailed but still as abstract data types.

RT synthesis During RT or high-level synthesis it is defined in which control step an operation is performed. This is known as *scheduling*. During *allocation* it is determined how many RT components are instantiated. During *binding*, operations are assigned to functional units. Moreover, the architecture used for a certain functional unit is selected during *module selection*. Several optimisations can be applied to improve timing, area, and power, for instance [128].

RT level At RTL, the system is seen as a set of registers, multiplexers, adders, and other functional units, as well as their interconnect. The behaviour can be represented by a set of RT functions. Timing is cycle-accurate and values have a bit semantic e. g., fixed-point. For this thesis this is the most important level of abstraction, since power estimation and characterisation is performed on this level. The description at RTL can be obtained from intermediate representation such as CDFGs, which are described in detail by Namballa et al. [102]. These CDFGs can also be hierarchical [137]. The register-transfer level is typically implemented in terms of a *finite state machine with data path (FSMD)*, as presented by Gajski et al. [54, sec. 2.4]. Such an FSMD consists of a state register, a next-state and an output logic as well as of a data path. The output function controls the data path i. e., the select signals of multiplexers, register-enable signals of registers, etc. A generic structure of an FSMD and its interaction with the surrounding system is shown in Figure 4.5 on page 67.

Logic synthesis During logic synthesis functional units are mapped to standard cells from a design library. Several optimisations for area and timing are performed.

Logic level At logic level, the system is seen as a set of logic gates, implementing simple logical functions. Several nets like the interconnect between the logic gates, the clock-tree and so forth are also considered. Timing information contains the real delay and values are seen as bits and bit vectors.

Layout synthesis Layout synthesis directly maps logic gates from the net list, available at logic level to an electrical implementation. Implementations of all logic gates are available from an ASIC cell library. Layout synthesis also performs the *place and route* phase. That is, placement of logic gates and their interconnection is done.

Electrical level On the electrical level, the system consists of transistors, resistors, and capacities. Individual functional units cannot longer be identified. Timing information is available as real delay. Values are seen as voltages and currents.

2.4 Power Dissipation

Power dissipation in a typical digital design can be split into different parts, which can be considered individually. Before the individual parts are presented, it is important to have a basic understanding of how a simple circuit does work. In the following, this thesis focuses on circuits manufactured in CMOS technology. Electrical currents inside the circuit are seen the technical way i. e., a current flows from the supply voltage V_{dd} to the ground voltage V_{ss}, which also represent the logical one and zero, respectively.

2.4.1 Dynamic Power

Dynamic power dissipation, denoted as P_d, occurs while individual transistors of a circuit are switching. For this thesis, dynamic power dissipation comprises only transistors directly implementing the behaviour i. e., the functional units, multiplexers, and registers of the data path. Other transistors, like the ones implementing the controller, for example are considered separately in Section 2.4.3.

In CMOS technology, a circuit is built using two sub-circuits. The *pull-up network* allows a connection between output signal and the supply voltage, whereas the *pull-down network* allows the connection between the output signal and the ground voltage. It is obvious, that the pull-up and pull-down networks must implement complementary behaviour. Commonly, the pull-up network is implemented using PMOS transistors, whereas NMOS transistors are used to implement the pull-down network. A simple CMOS inverter gate, as the one shown in Figure 2.2, visualises this.

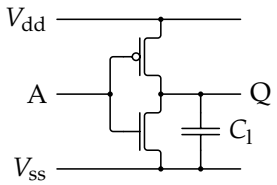


Figure 2.2: Simple inverter gate in CMOS technology — The virtual capacitance C_l is charged via the pull-up network if a logical zero is applied to input A. It is discharged via the pull-down network, if a logical one is applied to A.

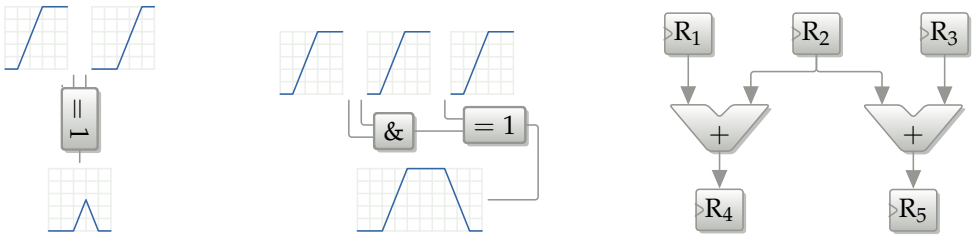
Dynamic power dissipation is three-fold. The first part of the total energy dissipates while the virtual capacitance C_l, connected to the output of the inverter, is charged via the supply voltage V_{dd}. In this case, 50 % of the energy dissipate while the electric charge passes through the PMOS transistor.

Since both P- and NMOS transistors are switching simultaneously i. e., one transistor starts conducting, while the other one stops to conduct, there is a small time window where both transistors are partially conducting. In this time frame, there is a direct connection between supply and ground voltage. This short circuit causes additional power dissipation. This second part of the total dynamic power dissipation becomes an important part, if the supply voltage is up-scaled. It may account up to 20 % of the total power dissipation as shown by Zaccaria et al. [142, p. 27].

The third and final part dissipates if the capacitance is discharged again via V_{ss}. In this case, it is assumed that the remaining energy dissipates while the electrical charge passes through the NMOS transistor. That is, a complete charge and discharge cycle of the capacitance takes two clock cycles. This fact is also reflected by the well-known formula for dynamic power dissipation, shown in Equation (2.1), where C_l is the switched capacitance, α is the switching activity, and V_{dd} is the supply voltage. The clock frequency of the system is given by f_{clk}.

$$P_{load} = \frac{1}{2}\alpha C_l V_{dd}^2 f_{clk} \tag{2.1}$$

These just mentioned effects are all relevant for estimating the dynamic power dissipation of a single transistor or simple logic gates. If estimating power dissipation of a whole circuit, additional parasitic effects occur that are influencing the total amount of energy that dissipates. There exist incomplete or unnecessary transitions and operations, for example. Three of these parasitic effects are shown in Figure 2.3.



(a) Glitch (b) Hazard (c) Parasitic functionality

Figure 2.3: Additional dynamic power effects — A glitch occurs, if two signals arrive subsequently, a hazard occurs, if signals arrive subsequently due to different lengths of the data path, and parasitic functionality is caused by RT components that are active, but whose results are not required.

The first one is called *glitch*. A glitch is an incomplete transition of the output signal of a logic gate, which is caused by asynchronously arriving input signals. That is, the first input signal arrives at the logic gate, causing the output signal to start changing its value. After a certain amount of time, the second signal arrives and the output signal goes back to its initial value. Figure 2.3a visualizes this. Remembering Figure 2.2, this means that the capacitance C_l is not completely charged or discharged, respectively.

A hazard is a complete, but unnecessary transition. This is shown in Figure 2.3b. Hazards occur, if logic gates are chained. This means that input signals have different path lengths. In this case, it might occur that an input signal of a logic gate arrives delayed compared to the other ones, because it is the result of an upstream logic gate. If so, the output signal will perform a complete transition, stay at its new value for a certain amount of time, and will then perform another transition back to its initial value. Of course, this can happen multiple times, depending on the length of the preceding chain of logic gates.

Additional or parasitic activity is not logic level specific. It can also occur at RTL. For instance, a register may serve as input for two operators but only one result is required, depending on the current state of the controller. If the value of that register changes, both operators are

active resulting in corresponding switching activity and thus power dissipation, although only one result is required and thus only the necessary operator should be active. Figure 2.3c on the previous page depicts such behaviour. This parasitic functionality can be avoided by techniques like *operand isolation*, but it comes with a corresponding overhead in terms of area.

2.4.2 Static Power

Besides power dissipation caused by charging and discharging capacitances, there is another type of power dissipation, which is caused by parasitic currents. All in all, there are eight types of so-called *leakage currents*, or simply *leakage* for short, all exemplified in Figure 2.4 for an NMOS transistor.

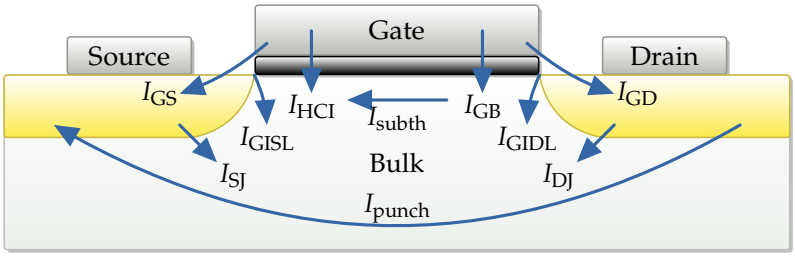


Figure 2.4: Leakage currents in a NMOS transistor — There are various leakage currents in a typical transistor. The type of the transistor i. e., NMOS or PMOS and its current state i. e., locking or conducting, determine which is the dominating current.

It is clear that the actual currents depend on the voltages applied to source, drain, gate, and bulk, respectively. A PMOS transistor has similar leakage currents, but due to the underlying physics they are weighted differently. The currents can be divided in three main classes.

The first class contains the current from source to drain. This *sub-threshold current* I_{subth} flows, even though the transistor should be locking. Due to a very small difference between supply voltage V_{dd} and threshold voltage V_{th} the channel is in a low inversion state. Even a low voltage between source and drain will thus result in this current, which increases with the temperature of the system [104, p. 498]. Source-Drain leakage I_{subth} is the dominating leakage current for technologies between 180 nm and 90 nm. For technologies with a feature size smaller than 90 nm other leakage currents are taking precedence.

The second class contains currents coming from the gate. Overall there are three currents, in particular *gate-source* I_{GS}, *gate-drain* I_{GD}, and *gate-bulk leakage* I_{GB}. Together they are referred as *gate leakage* I_{gate}. In all cases, electrons tunnel potential barriers. Hence, a smaller oxide results in an exponentially larger gate leakage. For technologies between 90 and 65 nm this is the most important leakage. In technologies below 65 nm high-k materials are used, making the problem of gate leakage manageable.

The last class contains junction leakage, flowing through the p-n junction. These currents occur at the source as *source junction leakage* I_{SJ} as well as at the drain as *drain junction leakage* I_{DJ}. However, more important are *gate induced drain leakage* I_{GIDL} and *gate induced*

source leakage I_{GISL}. This is especially true for technologies smaller than 65 nm. Generally these currents are caused by randomly drifting carriers or electron-hole generation at imperfections. In addition, the field between source and gate or drain and gate advances the probability that electrons tunnel through the barrier.

In addition to the eight types of leakage current, there are two additional effects, which typically do not lead to leakage currents, but limit the voltages the transistor can operate at. Both are mentioned for completeness. The first one, is *hot carrier injection*, where accelerated electrons can go from the channel to the gate, causing a current I_{HCI} from gate to the channel. The second effect is a *punch-through* of electrons from drain to source. The exact location of p-n junctions depends on doping and fields inside the transistor. A stronger field brings source and drain regions closer together. If both touch, a short circuit is formed, resulting in a current I_{punch}. To preserve the functionality of the transistor, the supply voltage is chosen in such way that a punch-through does not occur.

Most of the leakage currents just mentioned partly depend on the state of the transistor i. e., if the transistor is conducting or if it is locking. Thus, leakage could be assumed data-dependent [59, 75]. However, Helms et al. have shown that in a larger perspective, leakage is nearly data-independent and that data-dependency decreases with an increasing level of abstraction [76]. That is, in large designs data-dependent leakage currents of individual transistors are averaging out each other. Due to space limitations, only a very brief overview of the topic could be given. A more detailed overview of leakage is given by Narendra and Chandrakasan [103]. An extensive description of leakage currents and especially their estimation is given by Helms [73].

2.4.3 Additional Sources of Power Dissipation

Besides dynamic and static power dissipation, which are directly caused by implementing the intended functionality of the module, there are some additional sources of power dissipation. They are introduced during system synthesis and must also be regarded during power estimation.

The first one contains all structures, required for storing data values, such as registers and memories. In contrast to a typical full-custom hardware module, which has a fairly irregular structure, the structure of memories is very regular. This regular structure can be utilised, enabling a large number of techniques especially for memory power estimation [124].

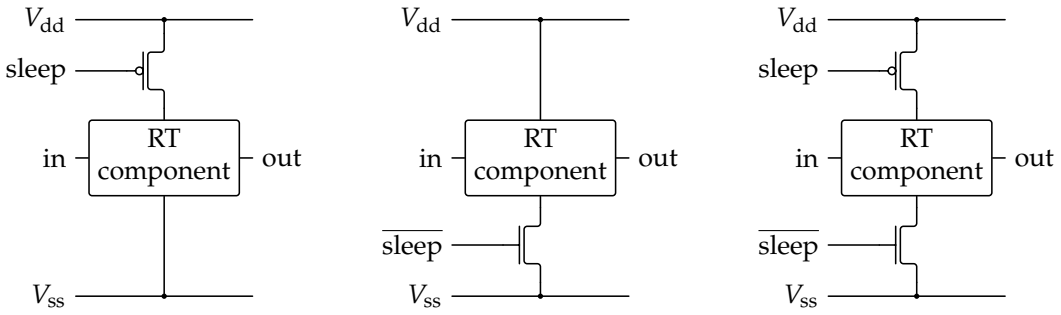
The clock-tree must be regarded, too. It is a set of wires across the overall system, providing the clock signal to all registers and other clock-dependent components that are distributed all over the system. This large network is charged and discharged in each clock cycle, in order to provide the rising and the falling clock edge. Thus, depending on the size of system, a comparatively large capacitance is switched twice in each clock cycle. There exist techniques for reducing the switched capacitance such as clock-gating. These optimisation techniques are a research topic of their own and are not discussed in detail in this thesis. This thesis as well as the used estimation tool PowerOpt assume a non-gated clock signal. However, this feature can be provided easily, if necessary.

For buses the same applies as for the clock-tree. Buses are a set of parallel and comparatively long wires connecting two or more modules or components of the embedded system. Special attention is required by the bus arbiter. This component is arbitrates of resolves concurrent and thus conflicting accesses to the bus. For sophisticated buses like AMBA AHB or IBM CoreConnect, complex bus controllers are required. These implement the bus protocol, including burst-mode, byte enabling, or timeouts. For this thesis a simple and state-less resolution logic is assumed, which is the type of bus arbitration that is also introduced by PowerOpt for inter-process communication etc.

2.4.4 Power Gating

There are several techniques for reducing the power dissipation of a system. Examples are *operand isolation*, or power-optimising binding and allocation like it is done by PowerOpt. Power optimisation techniques at very low levels of abstraction i. e., at electrical level are regarded during characterisation of the logic gates. Power optimisation techniques at RTL are typically applied during synthesis and are thus reflected within the resulting data path. All these techniques are transparent to the design process, proposed in this thesis. That is, their effect is regarded implicitly. One exception to this is *power gating*.

Power gating of individual RT components was introduced for reducing leakage currents while the particular components are not operating. If a component is not required for a certain amount of time, the controller is able to disable the particular component by separating it from the supply or ground voltage, respectively. This is done using so-called *sleep transistors*. Generally speaking, there are three different variants in which power gating can be implemented. These are shown in Figure 2.5.



(a) PMOS gating (b) NMOS gating (c) Double gating

Figure 2.5: Power gating — An RT component is separated from supply or ground voltage, respectively. For this, a special sleep-transistor is used that is typically slower than the transistors, implementing the RT component, but it is able to efficiently conduct larger currents than the usual transistors can.

Power gating allows disconnecting a particular module from the supply and/or ground voltage, respectively. If the module is disconnected from the supply voltage, this technique is called *PMOS power gating* and is shown in Figure 2.5a. If the supply voltage is no longer applied to the module, the module's internal capacitances are discharged over time due to leakage currents.

If the module is disconnected from ground instead of supply voltage, a technique called *NMOS power gating*, which is depicted in Figure 2.5b, is used. In this case the module's internal capacitances a charged instead of discharged.

P- and NMOS power gating can be combined to a technique called *double power gating*, as shown in Figure 2.5c. However, this technique is only rarely used and more an academic approach. It is mentioned here for completeness, only. Each technique can be implemented in various variants e. g., using multiple sleep-transistors in a row or in parallel. For the remainder of this thesis, power gating is not seen as an optimisation technique for individual RT components, but at a larger level. Not individual RT components, but the entire module can be power gated, if its functionality is currently not needed by the system. This is simply done by power gating all RT components that belong to the module.

Of course charging and discharging capacitances requires some time. While it can be assumed that an individual RT component can be enabled within one clock cycle, this is not true for an entire module. Due to limitations of the supply grid, charging all capacitances of an entire module cannot be completed within a single clock cycle. The actual time required for the module to become ready after it had been enabled depends on the module and its internal structure. An extensive description of power gating and its application during high-level synthesis is given by Rosinger [114].

2.5 Simulation Models for Power and Timing Estimation

The previous sections gave a brief overview of the different sources of power dissipation. These are the basics for understanding and developing power estimation techniques. As mentioned in Section 1.2 and shown in Figure 1.6 on page 10, there are various levels of abstraction at which power and timing estimation can be performed.

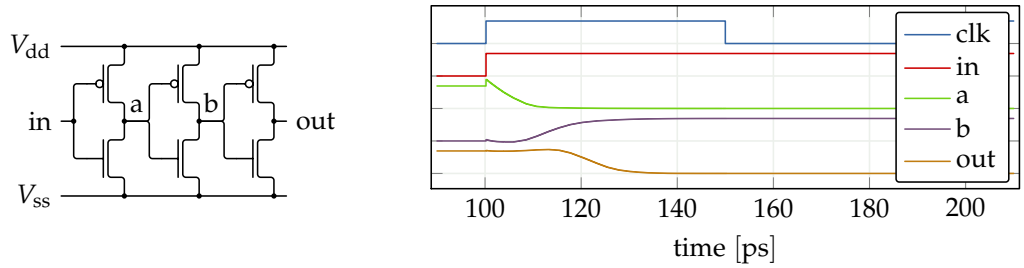
While synthesis lowers the level of abstraction by adding more fine-grained information, power estimation approaches try to go the other way round by applying different abstractions. Starting from a low level of abstraction, where precise information about power and timing are available, more abstract models are generated, providing fast, yet accurate estimation results. The following sections give a succinct overview of different abstraction level at which power estimation is typically performed. It will be outlined which information is available at which level of abstraction and which information is lost during abstraction.

2.5.1 Electrical Level

Power and timing estimation on the electrical level is carried out by simulating each individual transistor of the circuit. This is mostly done using the *simulation program with integrated circuit emphasis (SPICE)* or one of its derivatives like HSPICE or PSpice. Simulation of the circuit is done using a MOSFET transistor model, where the likely most common model is the *Berkeley short-channel IGFET model (BSIM)* [126].

SPICE is widely accepted as reference if no physical measurement is possible, either due to unavailable test equipment or due to the absence of the manufactured chip. Besides a physical measurement, a SPICE simulation can be assumed to give the most precise estimation

results. At this very low level of abstraction all electrical effects, such as partial or incomplete transitions, etc. can be seen and estimated accurately. Figure 2.6 shows the simulation of an inverter chain with three CMOS inverters using HSPICE.



(a) Inverter chain (transistors) (b) SPICE simulation trace

Figure 2.6: SPICE simulation of an CMOS inverter chain — The simulation uses continuous signal values. Simulation is done by solving differential equations.

A recent full-fledged characterisation card, describing all physical properties and characteristics of a CMOS transistor has more than 700 entries [73, p. 96]. The most important ones are obviously width and length of the transistor gate as well as electrical properties like threshold voltage. A comprehensive description of CMOS devices with their structure, parameters and characteristics is given by Rabaey et al. [109, sec. 3.3].

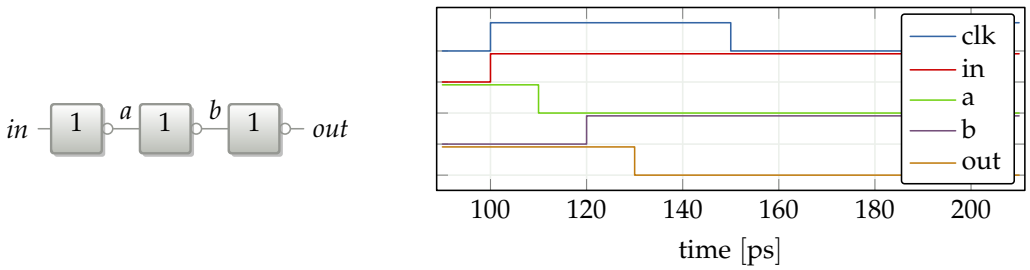
A circuit can be built by instantiating the appropriate transistor models in SPICE. Afterwards a transient simulation of the circuit can be performed. Continuously-valued voltages and currents can be measured at various points of the circuit. Measured values are then used to compute precise switched capacitance, static power dissipation, etc. It is obvious that estimations at such low level of abstraction are very time consuming.

2.5.2 Logic Level

Simulations at the electrical level can be used to characterise complete logic gates. This is done by creating an electrical level model for each logic gate. These circuits are than simulated using SPICE and a characterisation is performed afterwards. The characterised models can then be used during a logic-level simulation. This simulation is performed digitally. That is, all signals have discrete values. Typically, a four-valued semantic is used, which supports a logical zero and one as well as the special values *undefined* and *conflict*. Simulation of effects like glitches or other short impulses depends on the used delay model.

Simulation can be performed using different delay models, each one with its own accuracy and simulation speed. The fastest simulation model is called *Zero Delay*. It is clock based, meaning that all signal values and transitions are evaluated at clock-cycle borders, only. Using this delay model no validation of circuit's timing is possible. Hazards cannot be modelled. Switched capacitance is computed by evaluating signal values before and after the transition. A more advanced delay model is *Unit Delay*, at which all gate have the same delay. This

model allows a rough hazard detection and exact timing can only be estimated rudimentary. Figure 2.7 shows a unit delay simulation of an inverter chain.



(a) Inverter chain (logic gates) (b) Unit delay simulation trace

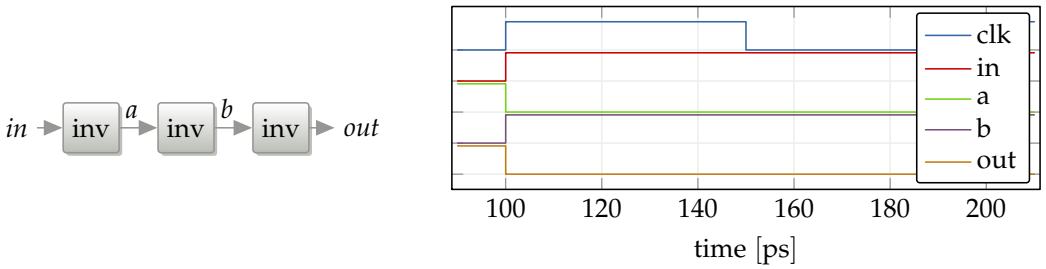
Figure 2.7: Logic-level simulation of an inverter chain — The discrete-valued trace has been obtained using a unit-delay simulation with an unit delay of 10 ps.

The most advanced delay model is *Real Delay*, which tries to determine the exact pin-to-pin delay. This delay depends on various parameters such as transition direction, fan-in and -out etc. These real delays can also be obtained from a standard-cell library, if available. These libraries are typically provided by the technology vendor. The real-delay model uses a discrete event simulation, for which various delay models such as *transport*, *inertial*, or *dynamic delay*, are available. Other types of models, which are pattern independent, are also possible. These include Markov chains or logic tables, for example. Some of them are mentioned in Section 3.1.

2.5.3 Register Transfer Level

The models available at logic level can be further abstracted to RT-level models. Again, this is done by performing an estimation and characterisation at the lower level of abstraction i. e., at logic level. Besides behaviour, main parameters for the generated RT models are technology, bit-width, applied input patterns, and temperature. In a first step, all structural parameters like technology or bit-width are separated from the data dependency. That is, switched capacitance is modelled data dependent and for each clock cycle, whereas static power dissipation is modelled data-independent. Figure 2.8 on the following page shows a typical RT-level simulation trace.

As can be seen, glitches and hazards as well as logic-level optimisations can no longer be modelled when using this approach. Nevertheless, the RT component internal glitches and hazards can be considered during characterisation, depending on the timing model used during logic level simulation. Assuming a compact layout of an RT component, glitches due to the interconnection between the logic gates can be estimated as well. More difficult is the prediction of glitches and hazards due to the interconnection between RT components, since their exact position in the design and thus the length of the interconnect might not be known.



(a) Inverter chain (RTL) (b) RT-level simulation trace

Figure 2.8: RT-level simulation of an inverter chain — The digital simulation at RTL uses the zero-delay transport model. All transitions occur at the rising clock edge.

Generally, it can be stated that a shorter data path between registers helps reducing glitches and hazards, since the output of a register is glitch and hazard free. Therefore, shorter data paths can be estimated more accurately at RTL than longer ones. One way to deal with this issue is to add an overhead, depending on the length of the data path.

During characterisation of an RT component, consideration of all possible input vectors is infeasible. Therefore, data abstraction and reduction must be applied. Input data can be abstracted in terms of Hamming- and signal distance, for example [80]. There are also several approaches available, which help reducing the amount of data to be considered. The most important ones are explained in Section 3.1. If data had been reduced, a lookup interpolation is required during estimation at RTL. Again, there are various ways how this can be done. Prominent techniques are nearest neighbour, linear interpolation, Lagrange interpolation, spline or bilinear interpolation, which are listed with an increasing order of complexity. Internally, PowerOpt uses such lookup tables. It is also possible to use analytical model like regression instead of lookup tables. Some of them are also mentioned in Section 3.1.

2.5.4 Above RT-Level

Abstraction from RTL is done in very different ways. As stated in Section 1.2, no generic encompassing library of power and timing-aware RT components can be provided. This leads to a confusing variety of different estimation approaches. In general, abstraction is done by creating equivalent *transaction-level modelling (TLM)* descriptions, *power state machines (PSMs)*, or by transforming the RT-level description into a *cycle-accurate functional description (CAFD)* using a high-level language such as C or C++. A more detailed overview of available techniques is given in Section 3.2.

All these abstraction techniques have usually in common that they discard information about synthesis artefacts like extra functionality, clock-tree, etc. Also, the more abstract the approach is, the less information about the processed data is considered during estimation. While CAFDs may consider the processed data, TLM-based techniques only consider the type and amount of data that is been processed by or communicated between the individual parts of the system. Finally, techniques like PSMs do not consider the processed data at all. They simulate and estimate the system in terms of abstract states or processing phases. Thus, only the control-flow is considered during simulation and estimation.

2.6 Summary

This chapter gave the basics and fundamentals, the approach proposed in this thesis relies on. After a short introduction of the nomenclature, used in this thesis, it had been shown, how embedded systems are generally defined. An outline of classical methodologies for the specification and the system design processes was given, both with focus on hardware modules. The typical design process, based on a high-level synthesis was presented and typical steps of a high-level synthesis were described. It was also mentioned, which information is available at the different levels of abstraction. Causes of dynamic and static power dissipation including several synthesis artefacts had been introduced. Based on the level of detail several estimation techniques are presented. Each with its own level of accuracy and required computational effort.

Main message of this chapter is that power dissipation is inherently a structural property. That is, power estimation can only be performed, if the physical structure of the design is known. A pure behavioural description of the system is not sufficient. However, in order to meet the requirements of today's system design process a suitable abstraction must be found. This abstraction must allow fast estimations of complex and heterogeneous systems while maintaining the desired level of estimation accuracy.

After having a comprehensive understanding of the hardware design and synthesis process, the causes of dynamic as well as static power dissipation and after knowing some basic power and timing estimation techniques at several levels of abstractions, it becomes possible to asses existing work of the scientific community, regarding power and timing estimation of embedded hardware modules. This assessment with respect to the requirements, identified in Section 1.2, is done in the following chapter.

Chapter 3

State of the Art

Abstract

This chapter gives an overview of existing work regarding power and timing estimation at different levels of abstraction as well as several model abstraction techniques. Power estimation is of interest since the late 1940s, when the first transistor was introduced. Since then methods for designing ICs are a research topic and the researching community has proposed a large number of different concepts and approaches. This chapter gives an overview of existing approaches regarding power estimation as well as high-level model generation. In the following, several different low- and high-level power estimation approaches are presented and discussed, each one representing a typical class of approaches. Drawing on these examples classes, the approaches are compared to each other and assets as well as limitations are shown.

The chapter is divided into two different parts. The first one tackles power estimation. Starting at a low level of abstraction and then reducing the granularity step-by-step, main concepts and techniques for estimating a system's power dissipation are discussed. The second part considers techniques for abstracting RT-level models in such way that a fast simulation of the model becomes possible. Finally, the summary will show that none of the approaches discussed can deal with all challenges, identified in the previous chapter. Individual aspects and functionaries from the presented approaches are picked and it is outlined how they can be combined to build a new and sophisticated characterisation and model generation process.

After knowing the fundamentals from Chapter 2, an overview of existing work can be given. Aim of this chapter is to identify approaches and techniques suitable for dealing with the challenges, mentioned in Section 1.1. Due to the large variety of different approaches and techniques a comprehensive overview of all existing approaches cannot be given. Instead, different classes are identified. Their assets and limitations are then explained using a representative example. First, methods and techniques for characterising and estimating the power dissipation of an embedded system are presented and discussed. These vary from approaches at logic level up to experience-based approaches, purely relying on knowledge databases and the designer's know-how. By discussing the basic idea of the approaches as well as their advantages and disadvantages, suitable techniques and methods can be identified.

The second part of this chapter tackles the field of model abstraction. Like before, basic idea as well as advantages and disadvantages are discussed. The information collected in both parts is then used for identifying existing methods and techniques that can be combined with newly developed strategies for building a design process, which is able to deal with the challenges in today's embedded hardware design process.

3.1 Power Estimation

Chapter 1 shows that power dissipation is a topic of concerns since the introduction of transistor technology. Reliable power estimation is the key for effective power optimisation. Even back in the 1970s and 80s, when NMOS was the only technique used for system design, power dissipation was a serious problem, not least because a logical zero at the gate's output caused a comparatively large electrical current between supply and ground voltage. New techniques were developed to deal with this issue, but until they were available power dissipation had to be considered during design development. This is still true today. Power dissipation is still a concern and had to be addresses during system design.

Various techniques for predicting the system's power dissipation have been proposed and implemented. These techniques use different metrics and inputs for estimating the power dissipation. Some of the metrics can be applied on different levels of abstraction, while others require a very specific input or they are specialised to certain types of designs. For different parts of the design like controller, memory, clock-tree, etc. different estimation techniques are available to come up to the special structure of these parts. Several surveys of the different techniques and methods are available from various authors [15, 17, 94, 111]. Another good and well known survey especially for logic level estimation techniques is given by Najm [101]. A supplementary survey especially for estimation techniques at higher levels of abstraction is given by Landman [86] as well as Macii and Poncino [93].

In order to develop a new power and timing estimation approach that tackles all the problems and challenges mentioned in Section 1.1, an overview and specifically a classification of the existing approaches and techniques is required. Based on such a classification, identifying the techniques which are most suitable for the mentioned challenges becomes possible. Based on and inspired by the existing techniques a new approach can then be developed in a second step. It must be noted that classifying different existing approaches is quite subjective to a large extend. The large variety of approaches makes it impossible to come up to all

different aspects tackled by the individual approaches. Basic techniques can be identified and compared, but no general classification is possible. There are too many details in which the particular techniques differ from each other. Each technique has its own advantages in a certain field of application, while in other fields it might suffer from a lack of applicability.

Generally speaking, there are two main ways of classifying power estimation approaches. First, they can be classified based on the design process. In this case, techniques and approaches are classified based on when and where in the design process a particular technique is applied. A typical classification can identify four different classes:

Top-Down Approaches start with a high-level simulation to generate the inputs for a lower-level power model. They characterise each design individually. Thus, each design has its own power model. However, these models are usually based on existing designs or the experience of the designer. Top-down approaches typically use metrics like switching activity or activation patterns for predicting the power dissipation. These approaches are commonly used in early design phases, where the concrete implementation of the functionality and the technology to be used is not known, yet.

Bottom-Up Approaches perform a low-level characterisation, enabling a high-level simulation to perform the power estimation. For the technology used for implementing the design, a power macro model is built that will be used during estimation. Such a macro model typically represents an RT component like an adder, a multiplier, or the like. The model estimates the power dissipation of the particular component based on the input values, collected during a functional simulation. That is, macro models are typically strongly pattern dependent. In order to use such a bottom-up approach nearly the whole design process must be completed. The implementation of the functionality as well as the technology for implementing the behaviour must be fixed.

Analytical Approaches attempt to correlate power dissipation to measures of design complexity. These approaches estimate power dissipation indirectly and are useful in early design stages. The designer is required to provide several values, such as gate count, switching activity, etc. Since these values typically are not known exactly before the final implementation is available, the designer has to estimate them. This leaves a lot personal judgement and thus yields very inaccurate results. While analytical approaches are error-prone in general, they may give accurate results for parts of the design for which the values are easy to estimate and where the underlying technology nodes are very similar in terms of structure, behaviour, and power dissipation. This is true for memories, clock-trees, or complete FPGAs, for example.

Hybrid Approaches are more a combination of the classes, mentioned above rather than a class of its own. In most cases, a hybrid approach combines a bottom-up and a top-down approach. Starting point is typically a high-level description, when no low-level description is available, yet. A quick synthesis is performed in order to create an initial guess of the target design. The created low-level description contains all parameters that are required for the intended estimation. Once a final implementation of the intended behaviour is available, it is characterised and the obtained values are back-annotated to the high-level description. The annotated high-level description can then be used in a high-level simulation or if the particular behaviour is reused in another design.

This classification seems to be obvious. The identified classes are very general and not bound to a specific level of abstraction. At the same time this is a disadvantage. It is not clear which information is available i. e., which technical details are already known and can be taken into consideration when developing the new estimation process.

A second way for classifying power estimation approaches and techniques bases on the level of granularity at which technical information is available. This type of classification is more technology-related and each class of estimation approaches can be assigned to a specific level of abstraction. In each of these classes in turn, there are different approaches of how the available information is used. The following classification groups the approaches based on their behaviour during estimation and not during characterisation.

3.1.1 Pattern-Based Approaches

One of the oldest and probably the most widely used method for power estimation is macro modelling. For each component of the system a macro model is provided. During a functional simulation of the system, in- and sometimes also output patterns of the component are captured. These patterns are then used by the macro model to compute the component's power dissipation. Using actual data patterns, spatial and temporal correlations as well as non-uniformities of the pattern distribution are taken into account.

In practice, only power dissipation of atomic RT components like functional units, multiplexer, buses, and latches or register is modelled [74]. These components do not have an internal state. Thus, the macro model is nearly time independent. Because dynamic power dissipation P_d directly correlates to the switched capacitance and thus the switching activity, two consecutive pattern pat_{i-1} and pat_i are required by the macro model. In this case, generic power estimation Equation (3.1) can be applied.

$$P_\mathrm{d}^i = f(pat_{i-1}, pat_i) \tag{3.1}$$

Aim of macro modelling is to derive function f itself. This is repeated for each atomic building block of the design. For power estimation, more complex blocks of a design are decomposed into these atomic components [108]. The building blocks belonging to the design and thus the macro models to be used during estimation are typically obtained from functional and/or structural description of the design. At RTL this could be Verilog or *very-high-speed integrated circuit hardware description language (VHDL)*, for example.

As mentioned above, a given building block like an adder is available in different sizes i. e., in different bit widths. In this case, a *soft-macro* is used. Such a macro models the power dissipation with respect to different parameters like the bit-width etc. If the component is also available in different implementations e. g., in a small and in a fast version, different macro models are required—one for each implementation. The same applies, if the component can operate in different *power modes*. In this case, it might be reasonable to provide one macro model for each mode, as it has been proposed by Potlapally et al. [108]. Two general assumptions can be made, as stated by Benini et al. [16] and Bogliolo et al. [21]:

1. In a combinational circuit some input has to switch in order to dissipate power.
2. The presence of switching outputs always corresponds to some internal activity.

Macro Models at Logic Level

A macro model library at logic level using SystemC has been prototypically developed by Xanthos et al. [139]. Their work shows in a very descriptive way the application of a macro model. A design is built from SystemC modules implementing the functionality of a certain logic gate such as `INV`, `NAND2`, or `OR2`. More complex gates are built from these primitives. The approach is also able to handle glitches. A modification of the SystemC kernel was necessary in order to extend the SystemC modules with power estimation capabilities.

Very similar to the just mentioned approach is the concept presented by Klein et al. [82, 83]. An augmented SystemC implementation of the design is created. The instrumented code allows monitoring the signal statistics during a functional simulation. In contrast to the work of Xanthos et al. the concepts allows creating models that work at RTL by overloading the SystemC implementation of the particular RT-level operations. Again, activation is monitored and patterns are captured to obtain a power estimate.

The other way round is done by Llopis and Goossens [90]. Instead of coarse-grain macro models, Llopis and Goossens use simple single-bit operators, so-called *power primitives*. An expansion function is used for expanding the given RT component into a number of these primitives. Different expansion functions can be applied to the RT component, each one providing a different low-level implementation of the component. This can be seen as some kind of a quick-syntheses approach. Considering data patterns at logic level gives very accurate result, including glitches and hazards. If a macro model is provided for each implementation of a logic gate, the accuracy will even increase. Nevertheless, estimating power for each logic gate individually can only be done for a very small number of gates. Even the increasing amount of computation power of today's work-stations is not able to cope with the increasing number of logic gates per design. For today's designs, much faster but still accurate approaches and techniques are required.

Information Theoretic Approach at Logic Level

Marculescu et al. presented an approach that uses information-theoretic data to generate the power model for designs at logic level and for simple RT components [96]. This is done by extending the generic Equation (2.1) on page 21 to consider each gate individually, which yields Equation (3.2).

$$P = \frac{1}{2} f_{\text{clk}} V_{\text{dd}}^2 \sum_{n=1}^{N} (C_{\text{l}n} \alpha_n) \tag{3.2}$$

It is obvious that switching activity α_n per logic gate plays a key role. In order to estimate power dissipation of more complex components on RTL, switching activity is represented by a typical wire inside the RT component. A so-called *effective information scaling factor* is used to reflect the structure and the functionality of the component. In contrast to the approaches mentioned above, not the actual input patterns are applied to the model, but some information about the switching activity at the component's inputs. For each component the output entropy, which is computed from the input entropy, serves as input for subsequent components. Considering an information-theoretic representation of the in- and outputs of an RT component gives a significant speed-up compared to conventional, pattern-based

approaches. Even though no actual data patterns are required for power estimation, the RT component's input switching activity must be known. That is, a functional simulation is still required, even if only at RTL.

Macro Models at Register-Transfer Level

As one of the first research groups, Burch et al. began to consider RT components as a whole instead as the sum of its logic gates [35]. A large number of randomly generated input patterns are applied to the RT component and the resulting power dissipation is measured. Such an approach is called *Monte Carlo approach*. Since each pattern should be considered individually and independent from the previous patterns, estimation is split into a *set-up phase* and a *sampling phase*. The set-up phase initiates the circuit and assures that only power dissipation caused by switching activity, but not caused by the initial loading of capacities is considered. That is, after set-up the power dissipation is a stationary process. Estimation is repeated until a certain level of confidence is reached i. e., a trade-off between the performance and the desired percentage error is possible. Burch et al. also found out that their approach can also be applied to circuits that have a non-normal power distribution e. g., a double-normal distribution, which has two local maxima of the dissipated power. Tailed-normal and chopped-normal distributions can be considered as well. However, it is difficult to determine the correct set-up time. The authors propose summing-up the delays of all logic gates on the longest path, but this will not work for circuits containing feedback-loops.

Bogliolo et al. presented a regression-based approach for creating power macro models for combinational logic blocks [21]. Again, for each RT component in the RTL-library a macro model is generated. Basically, they propose three different types of estimates. The first one performs a static estimation. During a functional simulation switching activity at the in- and outputs of the RT components are monitored. The average switching probability can then be computed, which serves as input to the power model. The second approach is a conventional macro model approach. During a functional simulation of the system, each in- and output pattern is forwarded to the model. Their third approach tries to tune the macro model during a functional RT-level simulation, using a logic level simulation. The approach achieves an error of about 15 %. However, for a cycle-accurate estimation the error becomes larger than 34 %. The approach is very similar to the one presented by Wu et al. [138], which had been proposed two years earlier. Another approach that relies on regression is given by Gupta and Najm [68, 69]. But instead using the input pattern directly, their approach uses the Hamming distance of two consecutive input patterns as input to their model, which helps reducing the amount of data that must be considered. All regression-based approaches have in common, that their accuracy depends on the abstraction of the input values. This allows a trade-off between performance and estimation accuracy. However, considering data patterns, even if abstracted, still requires a lot of computational effort.

Nemani and Najm [105] state that power dissipation is highly related to the area. Since area depends on the implementation of individual RT components, Nemani and Najm propose a technique for deriving the circuit's area from the Boolean function's structure and entropy. This in turn can be used for an optimised implementation of the circuit. Considering the area instead of functionality takes into account the fact that power dissipation highly depends on the number and size of the used transistors that are implementing the behaviour.

Main drawback of the model is that it assumes that inputs are spatial and temporarily independent, which is not true, even for larger designs. Moreover, the approach can only consider combinational macros. Complete clocked sequential systems cannot be considered. With this approach an average error between 30.21 and 33.70 % is achieved, although for about 80 % of the designs, the estimation error is below 25 %.

Reducing the complexity while increasing the model's flexibility can be achieved by abstracting the model's input data, as shown by Jochens et al. [80]. By computing the Hamming distance of two consecutive data patterns with respect to different bit regions, as proposed by Landman and Rabaey [87], the accuracy of the power estimation model can be notably increased. A given word consists of two regions. The first one contains the *sign bits*, while the second one contains bits that are uncorrelated in terms of space and time. With the proposed approach, the estimation error is typically less than 15 to 20 %, while providing sophisticated soft-macros that are parameterised in terms of bit-width. This in turn will reduce size and complexity of the component library.

FSMD at Register-Transfer Level

During logic synthesis, the design is transformed from an RT-level to a logic-level description. Within this step, the boundaries between the RT components are broken and optimisations are applied to all logic gates at once. As a result, power dissipation can no longer be attributed to individual RT components. But typically synthesis artefacts like introduction of multiplexers, wires of different lengths, or simple gates that serve as glue logic are neglected in common approaches at RTL. Bruno et al. presented a methodology that can deal with these artefacts [34]. While achieving a speed-up between 2 and 38 $\times$, the relative error is between 14.20 and 34.10 %. The authors also state that their approach is technology independent. This is achieved by introducing a scaling factor, which is used to adapt the generated power model to a new technology. This scaling factor is computed by selecting a *golden instance* from the library. Since new technologies may comprise completely new designs components, this is not sufficient. While the authors' statement that a RT-level net list does not model the logic-level net list is true in general, sophisticated RT-level estimation tools like PowerOpt consider the technology that was used during syntheses and also take different ways of implementing an RT component into account. Thus, they will give good estimates.

Soft Macros at Register-Transfer Level

With increasing complexity of the designs, using hard-macros had become too inflexible to cope with the new requirements. Parametrically soft-macros were the solution. These macros can be modified using one or more user-defined parameters like bit-width, for instance. Macro models for power estimation had to keep up the pace and also become parametric. Bogliolo et al. propose to create a macro model that relies on the transition probability of the in- and output signals [22]. The model is scalable in terms of bit-width and size of the technology nodes. Using a reference bit-width and technology, a power model that only depends on the signal statistics can be created. This can be done using conventional hard-macro modelling techniques. Using this technique, an error of about 10 % is achieved. Parameterised soft-macros give a lot of flexibility. In particular, it becomes possible to obtain extrapolated assumptions for new and uncharacterised bit-widths or technology feature sizes.

Primitives and Operator Overloading

An approach that replaces conventional RT-level components with ones that are power aware is proposed by Ravi et al. [112]. A library of common RT components is characterised and corresponding power models are created. The original RT-level input model along with the power-aware RT-level component library is then used to create an enhanced RT-level model with power estimation capabilities. During generation of the enhanced model, various optimisations are applied in order to increase its simulation speed. While sampling at RTL is usually done for the complete design, Ravi et al. propose *partitioned sampling*. That is, more estimation effort is spent for components that contribute more to the overall power dissipation. Using partitioned sampling combined with the optimisations during model generation, a speed-up of about 31 $\times$ is achieved.

There is also an approach proposed by Damaševičius and Štuikys that overloads logic and arithmetic operations like `+`, `-`, etc. in SystemC [47]. Same applies to `read` and `write`-operations on SystemC signals. The approach tries to estimate the total power by summing power dissipation for each operation, considering the concrete input vectors. With this, an error of about 6 % is achieved, but simulation time increases by 71 %. Mayor benefit of this approach is its transparency to the developer. Monitoring components and collecting input vectors is done fully transparent to the developer, because of the overloaded operations. Despite the fact that a modified SystemC kernel is required, main disadvantage is that a design given in SystemC typically does not represent the synthesis result. That is, mapping different operation on the same operator and the resulting sequence of input patterns are hard to consider. Also, effects like parasitic functionality cannot be considered at all.

Summary of Pattern-Based Approaches

Most techniques discussed above use the macro-modelling approach. They rely on a library of macro models, containing a model for each available component. During estimation, the models are stimulated using some kind of activity measure. This can be the actual data patterns or some abstracted information like Hamming distance or transition density, as proposed by Najm [100]. Considering the correct in- and output pattern for a certain simulation run, it is possible to obtain very accurate results. Moreover, regression-based approaches at RTL cannot only be applied to the data path, but also to the controller as shown by Benini et al. [19]. Pattern-based approaches in general and macro models in particular are not applicable in all cases. This is because they rely on the assumption that there is a direct correlation between in- and output activity on one hand and the component's power dissipation on the other hand. For more generic and sequential components i. e., components with an internal state, this assumption does not hold. In these cases, power dissipation is time-dependent i. e., it does also depend the previous input patterns.

All these techniques suffer from the same problem. They require actual or some kind of abstract data patterns to be applied i. e., they can only be used after allocation and binding have been performed. They have to reflect the final implementation of a design. In principal it is possible to assume that each operation is mapped to its own operator. In this case, data pattern per operation are still monitored, but it is not feasible to re-construct the correct pattern sequences that occur if two or more operations are mapped to the same operator.

3.1.2 Activation-Based Approaches

Activation-based approaches typically do not consider individual patterns of logic or RT-level components. They consider activations of larger parts of the design. That can be a control step of the FSMD or the activation of a more complex behaviour such as an instruction.

FSMD with Data Patterns

The power estimation approach presented by Zhong et al. is somewhat in between a pattern- and an activation-based approach [144, 145]. As can be seen from Chapters 4 and 5, this approach is similar to the one presented in this thesis. Zhong et al. propose to transform the given design into a CAFD, which is a C/C++ representation of the design's CDFG. Basically, the CAFD is a large `switch`-statement, where each state of the CAFD represents a specific control step. For each RT component, a virtual component with an associated macro model is created in the CAFD. During simulation, each in- and output pattern of a component is passed to the corresponding macro model. At the end of each control step the power of all RT components of the design is collected and accumulated in order to give the total power of the actual clock cycle.

Using a CAFD as the executable model for power estimation gives good results, because these types of models are relatively close to the final implementation at RTL. It is obvious that creating an executable model that monitors every in- and output pattern of each RT component and forwards this information to one out of a large set of macro models yields a very large system model. Since the model is created by a tool and must not be read by a human, this is not a problem. More important is the fact that the macro models have to consider each in- and output of all RT components. This will massively slow down the simulation speed. In order to speed up the simulation, the number of considered data patterns must be reduced, which will lower the accuracy of the approach. Moreover, synthesis artefacts like parasitic functionality cannot be modelled, since it is not possible to determine the pattern of the RT components that are active due to parasitic i. e., unwanted activity.

FSMD with Toggle Count

Reducing the required computational effort can be done by using the more abstract toggle count per state instead of considering every in- and output pattern of each RT component individually. Based on the implementation given as FSMD, a characterisation of the design can be performed. Using statistical methods, the average toggle count per state and unit of the data path is obtained and annotated to the CAFD. During functional simulation of the design, toggle information is collected and used as input for the power model. This approach, which has been proposed by Ahuja et al., therefore requires two different phases [6, 7]. In the *learning phase* characterisation of the implementation model i. e., power characterisation as well as a characterisation of the FSMD model i. e., activity characterisation is performed. In the second phase, the *utilisation phase*, the measured activity is applied to the power model and information about power dissipation is obtained. Using only designs with six to ten states, a speed-up up to $1.57\times$ is achieved, while having a relative error of 0.25 to 3.81 %, compared to an RT-level power estimation.

There is one main disadvantage of this approach. Only a statistical representation of the toggle count for each state is given, assuming that the activity is roughly the same each time a specific state of the FSM is entered. For complex designs this is not true. Most controllers are implemented as Mealy machine. The activity occurring in a specific state therefore also highly depends on the inputs, applied to the FSM. Of course, each Mealy machine can be transformed into a Moore machine, where the activity does only depend on the machine's state, but this transformation possibly causes a state explosion and is therefore not feasible in most cases.

Power State Machines

A modelling technique using so-called *power state machines (PSMs)* was introduced by Benini et al. in 1998 [17]. This power modelling technique at system level requires the power states of the modelled components to be known or to be derived from a model at a lower level of abstraction. In many cases, power states will correspond to functionalities, provided by the component. For a *universal asynchronous receiver/transmitter (UART)* this can be `IDLE`, `SEND`, or `RECEIVE`, for example. If power states are not specified by the component's documentation, the states must be identified and power dissipation in the states needs to be distinguished, which may be a difficult task. Usually, there are only a small number of power states because of the increased complexity and overhead for supporting the power management [18]. Transitions between power states are triggered by events. These events need to be generated by a simulation environment, hence an executable system model is required. In early steps of the design process, executable models are not available. There are concepts available allowing to apply formal verification methods to PSMs in order to evaluate power management policies, like the one presented by Shukla and Gupta [127].

Power states provide an easy way to model the power dissipation of a given design. The modes itself can be identified easily and associated power values can be obtained in various ways, starting from a low-level simulation an characterisation and ending at using the value from the module's data sheet. The functional model must be extended by adding the capability to send the required events. If this is not possible, monitors can be added to the design, monitoring the module's communication protocol and deriving the module's actual power state from it [92, 135]. This technique is discussed below. Originally introduced for estimating power dissipation of components like hard drives etc., PSMs can only be used for designs that have clearly distinguishable modes of operation. Also, their power dissipation must mainly depend on this power state and not on the data that is processed while a certain state is active.

Instruction Based

If the system is built from cores or components communicating with each other, the approach proposed by Givargis et al. [60] can be applied. This approach assumes that the individual cores of a system provide a specific functionality that can be activated by calling a method of the core, implementing the functionality. This functionality or *instruction* as the authors call it, is characterised using a low-level model of the particular core. For a typical UART the authors mention `ENABLE`, `DISABLE`, `RESET`, and `WRITE_BUFFER` as possible instructions, for

example. The created model contains information about the toggle-count of the particular instruction. During a high-level simulation, the toggle-count for all parts of the systems is collected. In a second step this information is used to obtain power dissipation values from a power model. This approach is also applicable for designs, where communication takes place in terms of messages that are passed from one core to another. An error between 11 and 31 % can be achieved while having a speed-up of about 1000 × comparing the C/C++ to the logic level simulation. An approach very similar was presented by Maldari et al. [95]. Their approach is applied to the AMBA AHB, a SoC communication infrastructure. Typical instructions are `read`, `write`, or `idle`. An again, the chosen model depends on the component under observation.

Since performance and accuracy of both approaches depend on the size and the complexity of the defined instructions this approach is highly scalable. Allowing models of different complexity to be used, the approach becomes even more scalable. Even though the developer-based identification of the instructions of a core and the model to be used makes the approach highly scalable it causes a problem. The developer will typically define the instructions based on the functional behaviour of the core. This is already shown by the selection made by the authors themselves in their publication. However, the power model does not necessarily correspond to the behavioural instructions. Another system breakdown might give a much better power model, even though this breakdown is not obvious.

Signal Monitors

A very similar but more generic approach has been published by Vece et al. [135]. Their approach can be used to estimate power dissipation of a SystemC design. The SystemC module under observation is enclosed by a so-called *power estimator*. This estimator manly consists of a simulation engine, also called *Power Kernel* and a set of user defined power states. These states are distinct in terms of the algorithm used for characterisation and the different data that is applied to the power kernel. The developer must also specify a state machine, defining possible transitions between states and the events that cause such state switches. The third part of the approach are modified in- and output signals. These augmented signals allow the power kernel observing the communication of the SystemC module and trigger the state machine. A very similar, but more advanced approach is presented by Lorenz et al. [92], which provides sophisticated techniques for driving the state machine such as time-outs.

All approaches using a wrapper or monitor for examining the component under observation require only a minimum of changes to that component. Only in- and output signals must be augmented in order to monitor their activity. On the other hand, these black-box-like approaches suffer from the fact that the internal state of the component is not known and must be estimated with a low level of confidence. This complicates modelling, especially of complex behaviour.

TLM Based

Dhanwada et al. present an approach similar to the signal monitors, but their approach bases on TLM transactions [49]. The approach assumes that the components of a system provide different functionalities. It monitors the activation of these functionalities in order to obtain a power estimate. Communication patterns are identified at a high-level of abstraction. These patterns in turn are assumed to cause specific patterns of power dissipation. The authors introduce a so-called *hierarchical transaction-level power (HTLP)* structure, which allows different levels of granularity starting from data-phase level up to complex compositions of various transactions. Having a speed-up between 45 and 4460 $\times$, a relative error between 2.60 and 11.19 % compared to a logic level estimation can be achieved. Two very similar approaches are proposed by Ben Atitallah et al. [14] and Lebreton and Vivet [88], respectively. The first one also uses a hierarchical description to model power dissipation. In contrast the approach by Dhanwada et al., they do not model transactions but activities at different levels of granularity, starting from complex instructions down to more fine-grained single operations. The second one uses a concept based on *power phases*. Each phase has associated a specific power dissipation. This value can then be modified based on the component's state i. e., power mode. Like signal monitors, observing the TLM transactions is non-invasive. Since the pattern of power dissipation depends on the communication pattern, a strong correlation between both is required. If the power dissipation depends on the data that is been processed, but which is not part of the transaction e. g., a *direct memory access (DMA)* operation, it might be difficult to derive the power dissipation from the communication protocol.

Summary of Activation-Based Approaches

Abstracting from concrete data patterns that are processed by a component to its activations, a notable simulation and estimation speed-up is achieved. The accuracy of the corresponding approach depends on the level of granularity, at which the activations are considered as well as the data-dependency of the underlying power model. While activations of individual states of an FSM provide relative good estimates, activations of instructions or TLM transactions are quite more inaccurate. They however are preferred, if the component under observation has distinct power modes or power states, wherein each state is homogeneous with respect to its power dissipation. All approaches have in common that they consider the control-flow in some way and thus are at least implicitly data-dependent.

3.1.3 Stochastic-Based Approaches

While information-theoretic approaches at lower levels of abstraction estimate average activity of the components based on the entropy of their in- and output signals [96], higher-level approaches estimate power dissipation based on activation and interaction probability. For generating appropriate input-stimuli, the behaviour of the individual modules and their interaction must be known, respectively.

PSM of Multiple Cores

In 2003 Bergamaschi and Jiang proposed a PSM-based concept where each state of the PSM represents a specific power consumption mode, rather than a physical state of a certain core of the SoC [20]. In their concept, each core implements one of three different PSM classes, depending on the power modes, supported by the particular core. The PSM of each core is triggered by an external *power management unit (PMU)*, selecting the actual power mode or state, respectively. It is also possible that the PSMs directly interact with each other. The compound system is also represented as a PSM. This PSM is created by building the product state machine of all core-PSMs. Power dissipation of the overall system is obtained using a symbolic simulation. Starting from the initial state of the SoC-PSM each reachable state is derived. Specifying invalid states and the input given by the PMU, the set of reachable states can be reduced. While traversing all possible paths of the PSM, minimal and maximal power dissipation is computed. Similar to formal verification techniques, a *power reachability* analysis can be performed this way.

This concept is well suited for early estimation, where no detailed information about the system implementation is available. Simple and generic core-PSMs allow usage of a PSM library, containing pre-characterised power models. Symbolic simulation enables a verification-like assessment of the system's power dissipation. Power constraints like maximal power dissipation can be formally checked. However, pre-defined PSM classes are only suitable for cores with a simple behaviour of the power dissipation e. g., software processors. Full-custom hardware or ASICs have much more complex power states. Introducing more complex PSMs to this approach will result in a very large state space when building the SoC-PSM. Moreover, power estimates don not rely on a given use case. Total energy consumption can only hardly be predicted.

Neural Networks

Veller et al. have patented an approach that uses a neural network for power estimation [136]. The system is assumed to be a set of components using a message- or transaction-based communication infrastructure. The approach provides a protocol library and each message that is recognised during communication is mapped to a specific step in the used communication protocol. Several transactions can be combined to so-called *super transactions*, which represent complex control operations, for example. In a second step, a low-level i. e., logic-level description of a design is enriched with monitors. These monitors are then used to build a power per message/transaction library. This library is used to train the neural network. That is, the neural network represents the high-level power model of the given design.

This approach is suitable for simple designs where transactions are used to initiate certain behaviour. This is even true for complex protocols, where a sequence of transactions causes a particular behaviour regarding power dissipation. Nevertheless, the accuracy of the approach depends on the uniqueness of the transactions. The given approach may fail, if a sequence of transactions is used to configure a hardware accelerator component and where the last message initiates the operation of the component. In this simple but frequently occurring example only one single bit might set the power mode in which the component will operate.

It is obvious that the power dissipation of the component highly depends on this particular bit. If the other bits of the transactions heavily vary e. g., if a lot of data is to be transferred to the component, it will be very difficult for the neural network to learn the meaning of that specific bit, which is responsible for setting the power mode.

Summary of Stochastic-Based Approached

Having probabilities used as stimulation data, statements about the upper and lower bound of the power dissipation are possible. If the probabilities are derived from a set of functional simulations, average power dissipation for the given use cases can be estimated, too. Main benefit of stochastic-based approaches is that they allow a (semi-)formal verification of the design's power dissipation. That is, corner cases can be identified automatically. These are hard to identify using concrete use cases, especially if the corner case origins from an unknown side-effect.

3.1.4 Experience-Based Approaches

The last class of estimation approaches is more a class of guess- and experience-based approaches. All approaches have in common, that estimates highly depend on the experience of the designer. They are often used when existing designs are mapped onto a new technology. In this case, low-level estimates, which can be used for characterisation are not available yet. Estimates from older technologies however are available and can be used to perform an extrapolation.

Spreadsheet-Based approaches

In very early design stages spreadsheet-based approaches are used. Such spreadsheets are usually provided by the technology vendor and offer only very coarse-grained trade-offs. The designer adds custom information like usage, switching activity, or certain architecture parameters. This information is typically based on the experience of the designer. The spreadsheet then calculates the power dissipation, temperature development and so forth. The concrete activity and behaviour of the intended application or design, respectively is not considered. Since influences of optimisation techniques can only hardly be predicted, spreadsheets approaches are mainly used for designs relying on regular structures like FPGAs. An example for such a spreadsheet is the *Xilinx XPower Estimator* [141], which is available even as a smartphone app [140].

Spreadsheets are a well-known tool for estimating the power dissipation of FPGAs. Having the spreadsheet available and having an idea of the activity and size of the design under development, early estimates are possible. These can then be used for selecting the right FPGA for the given application. The quality of the estimates depends on the quality of the power models, embedded in the spreadsheet. Activity and size are hard to predict, even if a behavioural RT-level description of the design exists, since there are too many optimisations that can be performed by the synthesis tool, which have a large effect on the resulting implementation.

Data Sheet and Experience-Based Approaches

If no other information is available, the data sheet of the module can be used as reference. This is most often the case, if the module is provided as black-box-model by a third-party-vendor and if non-invasive approaches like signal monitors or protocol state machines are not possible or too complex to implement. Depending on the granularity of the data sheet, individual power or operation modes are given, for which values regarding the power dissipation are also given. By having a rough understanding in the behaviour of the system and how it interacts with the module, an estimate of the module's power dissipation can be calculated.

Ultimately, if no information is available at all i. e., the design is available as box-and-arrow diagram, only and no decision regarding technology and third-party-vendors had been made, power estimation completely relies on the experience of the developer or some knowledge-data base. In this case, experience from older projects and designs, that where alike the actual one, are used to made some kind of an educated guess. If the technology used for a well-known design is scaled-down for example, this can be used for estimating the resulting power dissipation in a first approximation.

3.1.5 System-Level Power Estimation

System-level power estimation is a topic of its own. It defies the previous classification. While the approaches and techniques mentioned above are specific to one type of embedded modules, system-level power estimation must deal with different types, such as hardware, software, and IP modules. Section 4.1 argues that system-level power estimation must comprise different characterisation technologies in order to cope with the different types of modules. This is reflected in the following sections. It can be seen that some of approaches, presented below combine multiple techniques, mentioned in the previous sections.

Platform-Based Design

Key for enabling system-level power estimation is the utilisation of platform-based design during the design phase. If platform-based design, as proposed by Sangiovanni-Vincentelli [120, 122] is used, the functionality of the system is defined separately from its architecture. This enables an evolution of future hardware generations. A functional refinement is done by mapping the functionality onto existing modules or by developing ASIPs along with dedicated hardware accelerators or ASICs. In platform-based design, individual modules of the system are specified at different levels of abstraction. Multi-abstraction-level simulation is possible by generating TLM wrappers of each of the system's modules.

The refinement and module selection decision are guided by the goals and constraints, specified along with the functional specification. Thus, for each level of abstraction at which a module is specified, a corresponding power model must exist. During the multi-abstraction-level simulation, these models can be used to obtain an estimate of the overall system's power dissipation. It is obvious that the accuracy of the estimate per module depends on the abstraction level, the particular module is specified at. The following approaches deal with these heterogeneous multi-abstraction-level simulations.

Hardware/Software Co-Estimation

Already back in 1998, Fornaciari et al. presented an approach for co-estimating hard- and software modules [52]. A detailed RT-level power model is used for characterising the hardware modules of the design. The model considers components of the data path, the controller, memory, etc. The generated model is weakly pattern dependent i. e., during power estimation only switching probabilities, reflecting the typical behaviour are required. For estimating software power, a simpler model is used. The given code is compiled into a virtual instruction set, making the approach processor independent. Software power dissipation is then computed by assuming a constant power dissipation, if a certain instruction is executed. This information is then abstracted to basic block level. For the first time, this approach allows a co-estimation of hard- and software modules. While a detailed model is used for estimating hardware modules, the software model is kept simple, as the authors point out themselves. It was sufficient for that time, but it does not meet current requirements. Today's software cores are much too complicated to be modelled by this approach.

Power Monitors

The concept of a multi-abstraction-level simulation was already implemented by Bansal et al. in 2005 [12]. The authors argued that adding power estimation capabilities to a high-level simulation slows down the simulation speed by a factor of about 8.50. They tackled this problem by distributing the estimation effort unevenly over space and time. Particularly, the estimation effort is different for different modules and may also vary over time. This is achieved by adding so-called *power monitors* to the system. There exist five different monitor types for the different types of modules i. e., there exists monitors for *central processing units (CPUs)*, buses, memories, and caches as well as for ASICs. Like the signal monitors, proposed by Vece et al. [135] and Lorenz et al. [92], they monitor the in- and output of each module. Each monitor is equipped with different power models, allowing a run-time trade-off between accuracy and performance. For ASICs, models at logic level, at RTL, and at behavioural level with and without sampling applied, are available.

Using different power models during simulation and selecting the appropriate accuracy depending on the component and time, allows a fast simulation while having a precise accuracy, whenever needed. For the two power models at behavioural level, Bansal et al. refer to the work of Zhong et al. [144], which is outlined above. Thus, their approach has the same drawbacks such as that parasitic functionality cannot be considered.

System-Level Power State Machines

For high-level power estimation Streubühr et al. propose a process very similar to the already known PSMs [130]. A SystemC-based design description is developed by extending so-called *virtual processing components*. Each processing component implements a single module of the system. It has three different execution modes, namely `IDLE`, `EXEC`, and `STALL`. A module can also operate at different power modes. Each combination of execution and power mode has a dedicated power dissipation as well as a transfer delay assigned. The later one is used for performance estimation.

With a description at such coarse-grained level of detail, a fast simulation and estimation is possible. Besides detailed drawbacks, for example that a delay, which depends on the data being processed, cannot be modelled directly, only rough estimates can be obtained, since the processed data is not considered at all. Having only three different execution modes available for modelling the power dissipation of a module is not suitable for the complex modules, embedded in modern SoCs.

Lajolo et al. [85] propose a technique for hardware/software co-estimation, which utilises different power estimators. Each estimator is plugged into a system-level simulation master. During simulation, the individual estimators such as RT-level estimators or *instruction set simulators (ISSs)*, are activated whenever needed. The authors propose some speed-up techniques, which reduce the number of activations of the power estimators. For software *macro modelling* is applied, allowing an estimation of complex instructions like operation and assignment instead of estimating each individual micro-code instruction of the processor. For hardware modules *energy and delay caching* is used. For a part of the control flow whose power dissipation has only a small variance i. e., a small data dependency, a single power value can be used as an approximation. Parts of the control-flow with a higher variance must be estimated using the RT-level power estimator. This caching gives an average speed-up of 13 $\times$, while the software macro models provide an average speed-up of about 44.80 $\times$. The average estimation error for total energy dissipation is around 24.30 %.

3.1.6 Summary of Power Estimation Approaches

The previous sections classified various approaches and techniques for power estimation. Figure 3.1 on the following page depicts the coarse classification of the approaches mentioned above. The approaches and techniques are classified based on the level of detail as well as their model of computation during estimation i. e., during usage of the generated model. The x-axis shows the level of detail which is required by the generated model to compute the power dissipation. The y-axis shows the model of computation implemented by the generated model.

Summarising it can be stated that experience-based approaches provide the fastest estimates. Nearly no computation effort is required, but a fairly good understanding of the design under development is required, in order to provide the measures, required by the spread-sheets etc. Stochastic-based approaches benefit from the fact that they do not need a concrete use case. By using probabilities, estimation of upper and lower bound of the power dissipation as well as identification of corner cases is possible. It also allows a formal verification of the design. Activation-based pattern can be used, if they have been created on a characterisation of the underlying RT implementation of the design. The accuracy of the model then highly depends on the granularity i. e., the level of detail the activations are considered at. Pattern-based approaches are slow in terms of simulation speed, since they typically consider the same amount of data as the behavioural model.

The temporal resolution of the presented approaches depends on their level of detail. The more low-level details are considered by the model, the more fine-grained the temporal resolution typically is. At the lowest level of abstraction, tools like SPICE allow a continuous estimation i. e., a suitable sample rate is automatically determined by SPICE and only limited

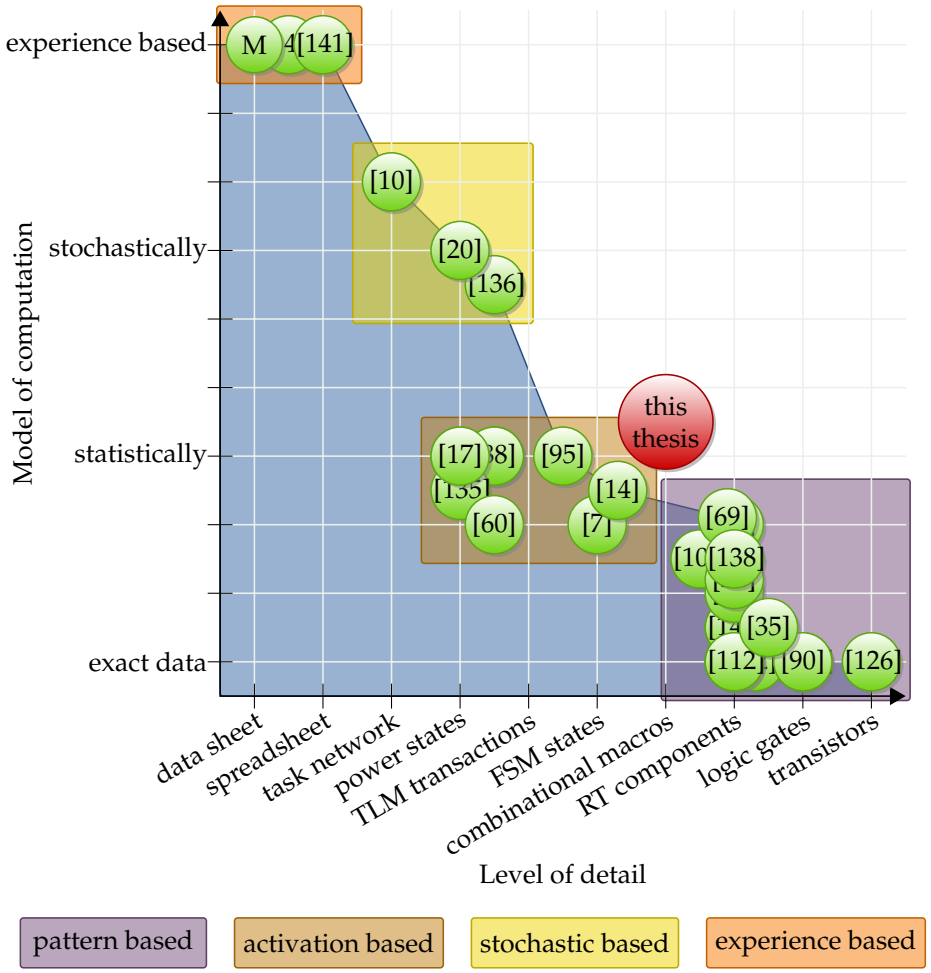


Figure 3.1: Coarse classification of existing power estimation approaches — A higher level of abstraction typically gives faster, while lower levels give more accurate results. A pure simulative model of computation leads to coverage problems, while formal approaches often suffer from infeasibility. Simulation speed is is third dimension, but comparable values are regrettably difficult to obtain from the presented papers and articles. It can be assumed that a lower level of details as well as simpler model of computation increases simulation speed.

by the resolution of the data types used for representing the time. Depending in the used delay model, estimation approaches at logic level provide a temporal resolution higher than a clock cycle i. e., they are able to model glitches, hazards, etc. All estimation approaches at RTL typically provide cycle-accurate power estimates. Above RTL, approaches and their temporal resolution become more diverse. Activation-based approaches that consider states of an FSM still provide cycle-accurate estimates, while approaches considering TLM transactions, instructions, power states, etc. provide a granularity that depends on the length of the particular activation. Approaches, which use spread- or even data sheets have no temporal resolution at all. They only provide a single value such as total energy or average power dissipation.

In order to obtain fast, yet accurate estimates an approach should be located between the pattern- and activation-based approaches. Activations can be easily obtained during behavioural simulation while a reduced level of detail enables faster estimations. However, the chosen level of detail will determine the accuracy of the approach and must therefore be chosen wisely. The implications and requirements that are arising from this knowledge are discussed in Section 3.3.

3.2 RT-Level Abstraction Techniques

After knowing different types and techniques for estimating power dissipation of embedded hardware, the most important abstraction and model generation techniques can be discussed. These are techniques allowing the generation of a high-level model, which in turn allows a fast simulation of the design. The high-level model itself is augmented with non-functional information such as power and timing. A static analysis is difficult, if not impossible due to the dynamic behaviour of the system and its complex interactions [57].

Generally, two types of abstraction techniques can be identified. The first one abstracts the characterised low-level model to a higher-level model, which only models the power dissipation. That is, the high-level model does not contain any information about the functional behaviour. Only the control-flow is taken into account, as far as this is necessary for estimating the power dissipation. Typical representatives of this class are all approaches using only a PSM and obviously all estimation approaches, relying on the developer's experience. Some of these techniques have been discussed in the previous section.

The second class is more advanced. Besides the power dissipation, this class models the functional behaviour. This feature is a key for embedding the generated model into a virtual system prototype, as required by requirement 8. In order to fulfil the requirement, only the second class is considered in this thesis. From Section 2.4, it should be clear that the low-level model must be at least at RTL or even lower, in order to obtain accurate results. Therefore, this section will focus on abstraction techniques, creating a high-level model, based on an RT-level description.

Already back in 1995 Hojati and Brayton proposed a technique for abstracting the behaviour of a data paths [77]. They identified two different abstraction steps that can be performed. *Data path abstraction* reduces the complexity of the data path by removing parts of it or by reducing the number of bits within it. *Control abstraction* however, tries to reduce the state space to be considered. That is, it tries reducing the number of the controller's output values or the number of its states. Both steps must be taken for creating a model with an acceptable complexity, which in turn will result in a reasonable simulation time.

Host-Compiled Software

Model generation for embedded software can be done straight-forward. The software is cross-compiled for the target architecture. A basic block analysis and characterisation is then performed. Collected information is then back-annotated to the source code, which in turn can be compiled for execution on the host system, as proposed by Gerstlauer et al. [57].

For embedded hardware abstraction is more complicated. At RTL, embedded hardware is most often described as set of concurrently running processes that are communicating with each other and their environment. Simulation is typically done using a discrete-event simulation. A direct translation of such a model into a software model is not possible. Instead, fundamental structure and behavioural properties are extracted and a new model is generated.

Boolean Expressions

At RTL or logic level the design can be seen as set of Boolean equations, describing the behaviour. These equations can be extracted and a Boolean network [13] can be generated. One typical purpose of this kind of abstraction is to allow a formal verification of the design. Tools like *Vapor* as proposed by Andraus and Sakallah [8] are typically used for allowing such an automatic verification of the given design. That is, it can be checked, if certain assumptions are fulfilled by the design's low-level implementation. These techniques can also be used, if automatic power verification e. g., using power contracts, as proposed by Nitsche et al. [107] should be performed.

Data Flow, Task Graphs, and State Machines

If only a rough representation of the behaviour is required, process networks, state machines, or data-flow-based models can be used as the most abstract level. These models do not describe the exact behaviour of the design, but model its behaviour in terms of functional tasks, exchanging messages over dedicated or shared channels. These channels can be used for exchanging data as well as for synchronising the tasks. During simulation, events and activations can be monitored.

TLM Transactions

Models at RTL can also be abstracted into equivalent TLM descriptions, as shown by Bombieri et al. [23, 26]. The design is converted into an *extended FSM (EFSM)*, which is basically a conventional FSM with register values from the data path. This state machine is automatically extracted and its equivalence to the RT-level model is checked automatically [24]. The EFSM is then embedded in a TLM slave module, which is triggered from its environment. Execution is then performed in three steps, namely *reading input*, *performing computation*, and *generating output*. A very similar approach is presented by Lorenz et al. [91]. This approach has also been extended for enabling power estimation, by introducing a so-called *protocol state machine* [92]. This technique allows a non-invasive power estimation of TLM modules. Therefore, it can also be used for black-box IP modules. It might happen that the first two phases i. e., reading the input and performing the computation are interleaved. That is, the computation is performed and other input is requested during the computation phase. These types of behaviour cannot be modelled by these approaches.

Software

In contrast to the initially mentioned abstraction, which creates a host-compiled software model from an embedded software module, this section describes approaches for generating a power and timing augmented software model from an RT-level model. If the RT-level model was obtained using a high-level synthesis, this opens the possibility to perform some back annotation. That is, directly enriching the input model with the properties that had been obtained during characterisation. While this way seems obvious for power and timing estimation of software modules, it is not that easy for hardware modules. During the compilation process, a lot of transformations and optimisations are applied to the input model. Regarding high-level synthesis it becomes even more difficult, since the synthesis also introduces resource sharing, parallelism, etc. It is thus very hard, not to say infeasible, to identify which artefact at RTL is caused by which entity or instruction in the input model.

Krishnaiah et al. stated that behavioural models with algorithmic descriptions are ten times faster than those with architectural details [84]. The authors give three advises for accelerating SystemC simulations: first, minimise architectural detail; second, minimise the number of concurrent processes, and finally maximise the usage of native C/C++ data types. These recommendations are followed by most approaches.

Back in 2000, Greaves presented a tool called *VTOC*, which was able to convert a synthesisable Verilog model into an ANSI C model [61]. In a first step, the Verilog hierarchy is flattened, until a single module, containing the complete behaviour of the Verilog module, is available. For each assignment type like blocking, non-blocking, etc., a corresponding C construct is selected. Dependencies between the assignments are resolved by a static schedule and intermediate variables are introduced, whenever needed. Using this approach a speed-up between 15.74 and 105.00 $\times$, depending on the granularity of the input model, can be achieved. The approach has been successfully applied to an industrial five million logic gategate design [129].

Bombieri et al. follow a very similar approach, but spent some significant effort for generating an optimised C/C++ model. Almost all structural information is discarded, while preserving the functionality of the module. In a *loop rolling* transformation, multiple instance if the same entity are removed and replaced by a loop, implementing the same behaviour. The generated model cannot only be used for a high-level simulation, but also for re-synthesis to RTL. Experiments showed, that area optimisation and re-synthesis steps allow an area improvement by a factor of 3 and a performance improvement by a factor of 2.50, compared to the original RT-level model.

A very similar approach is taken by the well-known tool *Verilator*. The open source tool was introduced in 1994 and creates a cycle-accurate behavioural C/C++ model from a given synthesisable Verilog model. Verilator resolves the module hierarchy i. e., it resolves all modules until only one top-level module is left. This is very similar to a synthesis step. Verilator also simplifies the four-valued logic that is supported by Verilog to a two-valued logic that is used in the generated model. By applying a zero-delay simulation model, the generated C++ model can be simulated about 100 times faster than the original input model, which typically has a more fine-grained event-driven simulation semantic.

A way for creating *Systems Modeling Language (SysML)*, which is an extension of *Unified Modeling Language (UML)*, from RT-level models is presented by Bombieri et al. [27]. Based on their existing work [25], where the authors presented a way for generating embedded software from RT-level modules, they propose a process that allows generation of C/C++, SystemC, or Java models by previously generating SysML.

All these approaches, which are generating some kind of a software model from the RT-level model have in common that the generated model in terms of lines of code, is significantly larger than the original input model. Zhong et al. showed that a CAFD abstracted from an RT-level description, is some kind of a three-address-code. This kind of code can be easily optimised by the compiler. Therefore, no significant increase in simulation time is observable, when executing the generated binary [145].

3.2.1 Summary of Abstraction Techniques

Having the challenges identified in Section 1.2 and the existing approaches just discussed in mind, an abstraction from RTL to a power and timing augmented high-level model can be developed. As mentioned above, performing a back-annotation i. e., augmenting the input model with non-functional properties is not possible. Thus, a new model must be generated, containing a behavioural description of the data path as well as all properties obtained during characterisation. In order to allow detailed estimations, the generated model should be a CAFD, which is enriched with power and timing information. By using C/C++ as language for the generated model, a compiled simulation is possible, which is expected to provide a reasonable speed-up.

Host-compiled software cannot be used. It is only applicable for embedded software. Boolean expressions are a quite powerful tool. However, they are better suited for formal verification and estimation of upper and lower bounds of the power dissipation, than for the exact power estimation with respect to a given use case. If a fast simulation is required, task graphs or state machines can be used. However, since they lack of detailed information about the

behaviour, they cannot be used for a behavioural simulation of the given design. Approaches using TLM wrappers or the like are well suited for black-box IP modules, because they do not require any or only a minimal change to the existing model. However, these types of approaches suffer from a lack of accuracy, if the component has an internal state. Creating a software model from the given module has several benefits. By abstracting from architectural details, simulation speed can be improved without changing the accuracy or the timing of the behavioural simulation. The generated software model can also easily be augmented with power information, still considering all architectural artefacts that had been removed by the behavioural abstraction step. In other words, the behavioural simulation considers only necessary parts of the design, while the power estimation also considers all artefacts introduced by the high-level synthesis. This will allow a fast but still accurate simulation and estimation of the design.

3.3 Summary

This chapter presented existing work regarding power estimation and characterisation as well as several approaches and techniques for model abstraction. As can be seen from the large number of different approaches, power estimation can be performed in very different ways. Accurate estimation is typically performed at RTL. Even though low-level approaches at, or even below RTL will provide accurate estimation results, they are very time consuming. Therefore, they cannot be used for estimating such complex systems, like the ones considered in this thesis. Early or high-level estimation approaches however, are available but inherently cannot consider the actual implementation of the behaviour. Therefore, they lack of the required accuracy.

Simulating and estimating such complex systems in a reasonable amount of time, while preserving the desired accuracy of the estimation results, requires an appropriate level of abstraction. Estimation must be performed fast but accuracy requirements must still be met. None of the presented approaches and technologies is able to fulfil all the challenges, stated in Section 1.1.

Considering these challenges and the more detailed requirements mentioned in Chapter 4, selected approaches can be endorsed and modified to facilitate the development of a new and sophisticated approach for fast high-level power and timing estimation with respect to low-level properties. Using a high-level synthesis, a high-level input model can be transformed into an RT-level description. The low-level description is characterised with respect to as many physical properties as needed. Using this information, an abstraction can be performed. The behaviour is described in terms of combinational macros, which are at the next level of abstraction above RTL. The control-flow can be easily represented as an FSM, which allows cycle-accurate estimates. Together with the macros, the FSM can be used to build a CAFD that is augmented with accurate power and timing information. Using C/C++ as output model language, a compiled simulation becomes feasible, which is expected to provide a significant speed-up compared to the RT-level simulation and which allows the model to be embedded into a virtual system prototype.

Summarising the existing work and rating the individual approaches and techniques, a new characterisation and model generation process can be outlined. Such an ideal design process would be four-fold:

1. The process starts at a very high level of abstraction, where there is only a rough idea about the design's implementation. Of course, required functionality is known and might be represented as box-and-arrow diagram, but hardware/software partitioning has not been performed, yet. Despite the functionality, some ideas about existing components and modules that should be reused or adapted are also known. At this first stage in the design process, high-level approaches can be used to support the design decisions especially with respect to the platform selection and the hardware/software partitioning.
2. This first step is followed by a newly developed second design step. This level of abstraction can be referred to as *platform level*. Functionality to be implemented as well as the underlying platform is known. There is still a large degree of freedom regarding module selection, structure of the interconnect, or implementation and optimisation of full-custom hardware components, for example. At this level, a framework is required to guide the developer through the design process and provide different Pareto-optimal solutions to the developer in order to support the design space exploration.
3. The third step is performed on a very low-level of abstraction. Most decisions have been made and design space exploration is nearly completed. During this step, focus is on optimising the implementation of the selected modules and platform artefacts. Depending on the required level of accuracy, some parts of the design can be estimated at a low level of abstraction, but this is done for each part individually. The surrounding environment i. e., the system the module is part of, serves as test bench and provides the stimuli for characterising the particular module.
4. Finally, information obtained during characterisation is used to create a high-level model of the module. Besides the behaviour this model contains information about non-functional properties like power and timing. The model itself can then be embedded into a virtual system prototype, allowing an estimation of the module with respect to the behaviour of the overall systems and by applying a large number of complex use cases.

The detailed requirements to such a characterisation and model generation process are identified in the following chapter. The chapter also shows how such a process can be designed and which problems must be addressed.

Chapter 4

Power Estimation & Characterisation

Abstract

This chapter presents the proposed techniques for automatically identifying and characterising combinational macros within a given full-custom hardware module, as well as considering as many synthesis artefacts as possible. Based on the RT-level data path and its corresponding controller, both created by the high-level synthesis, a characterisation of the module is performed. Combinational macros are automatically identified and augmented with accurate power and timing information. Together with information about non-functional properties like clock or controller power, they can be used to create an executable, power and timing aware high-level model

The chapter starts with an overview of the overall characterisation process, the proposed flow is part of. After outlining a typical high-level synthesis of full-custom hardware modules, the structure of the generated RT-level description is explained. This description is the input model to the proposed characterisation process. A detailed description of the identification process for combinational macros as well as their characterisation is given. Synthesis artefacts like static power dissipation or clock and controller power are also taken into consideration.

THE previous chapters made clear that power dissipation is still an important issue in today's system design process. It outlined the basics that are required for creating a new, sophisticated and versatile power and timing estimation process. The first question to be answered when setting up such a new power and timing estimation process is the question for the causes of power dissipation, since these must be tackled by the estimation process. In Chapter 2 this question is answered in a technical way by explaining static and dynamic power dissipation. But the effects described there are only the physical cause i. e., the system's physical reaction to an external event. These effects are not the initial cause. Remembering the multimedia-player mentioned in Chapter 1, the initial root cause for power dissipation is the user, using the device or more precisely the system's stimulus given by the user or the system's environment, respectively.

Such a stimulus is referred to as *use case*. A use case is a high-level description of how the user or the system's environment interacts with the device. In case of the multimedia player the user can play music files, show images, or play videos, for instance. Each use case itself can be divided into sub-cases. A music file, for example can be encoded in stereo or mono. If the multimedia player has two audio-decoder queues, one for the left and one for the right channel, one of the queues can be switched off, if a mono-encoded file is played. Yet, these subdivisions are still not sufficient enough. The music files can be encoded with different bit rates or can even have complete different file formats. Interestingly the same applies to playing videos or even to showing images. Compressing file formats like JPEG, for example use different channels, one for each colour and like music files, they also support different compression rates. Even the decoding algorithms are very similar. Most of them rely on *fast Fourier transforms (FFTs)* or *discrete cosine transforms (DCTs)* and thus can share respective hardware co-processors. Power estimation at a high-level of abstraction however, cannot regard all the different behaviours i. e., bit rates, channels, file formats, etc.

While power estimation approaches at a high level of abstraction might be suitable for power budgeting, they cannot be used if a detailed analysis is required. Power estimation at a high level of abstraction will always give very inaccurate estimates, as already shown in Section 3.1. The reason for that is very simple. For high-level estimation approaches it is hard to consider implementation artefacts of the specified behaviour. There are too many implementation details and correlations between the individual parts of the design that must be taken into consideration, requiring a very large set of use cases that has to cover very little details of the actual implementation. If the system shall be broken down into such small and fine-grained use cases, it is obviously infeasible for the designer to cover or even to identify each one of them.

A closer look at the fine-grained use cases reveals that they will be inherently activated by the input file and have to cover each possible control flow. That is, that power dissipation directly depends on the control flow, resulting from the input data of the system. In order to provide accurate estimates, it is mandatory to consider the concrete control flow as it is enforced by the input data. Since the control flow is directly given by the implementation of the system, it can be analysed automatically. Analysing the control flow also adds the benefit that the design, once characterised with a small representative set of use cases, can be estimated using a large variety of different use cases. This leads to the conclusion that characterisation must be performed on the actual implementation of the design, rather than on the use cases.

Automated analysis of the control flow allows a far more detailed look at the behaviour, as this could be done manually by a designer, who is trying to provide suitable use cases. This is true for all types of modules. No matter whether it is a hardware, a software or an IP module. As stated earlier, power dissipation of the overall system depends on the external stimulus as well as the internal interaction of the different modules of the system. That is, all different types of modules must be considered holistically.

In order to provide a flexible power estimation model, an accurate and versatile characterisation is required. Since the model for power and timing estimation will be generated only once, but is used several times, a higher effort for characterisation and model generation is acceptable, as long as the simulation of the generated model is fast. However, as stated in Section 1.2, the characterisation must be performed for each system individually. Thus, an extreme long characterisation, which might be acceptable for the characterisation of an RT-level component library, is still not applicable. A good trade-off between characterisation effort and estimation accuracy must be found.

As seen in the previous chapter, a lot of power estimation approaches at various levels of abstraction exist. Simplistically, it can be said that low-level approaches provide accurate results, but require too much computation time. High-level approaches however, allow fast simulations but give inadequate estimates. It is therefore a necessary compromise which combines the advantages of both, the quick characterisation and the accurate estimation. In expansion to the challenges mentioned in Section 1.2, the following more precise requirements to a fast, but sill accurate high-level power and timing estimation process can be identified:

1. Even though this thesis will focus on a high-level estimation process for embedded hardware modules, the proposed characterisation and estimation process must allow the consideration of larger and heterogeneous systems, the hardware module is possibly part of.

2. Since the behaviour and thus power dissipation of the hardware module depends on the system the module is part of as well as the way it interacts with the other modules of the systems, the manner in which the module is used within the system must be regarded, too.

3. Dynamic as well as static power dissipation are inherently structural properties. Therefore, the new estimation process must consider artefacts introduced by the high-level synthesis. Therefore, not the behavioural description must be considered, but the structural RT-level description, generated by the high-level synthesis tool must be used as input model for the characterisation process.

4. The characterisation process should only be based on structural properties like switched capacitance, which are fixed after synthesis. These properties should be separated from physical parameters like supply voltage or clock frequency, which might vary over time or between different system instances during design space exploration.

5. Since especially dynamic power dissipation is data dependent, its characterisation and estimation must be done with respect to the data that is processed by the module. Characterisation must therefore be performed with data and workloads that are typically processed by the module.

6. In order to provide fast, yet accurate estimates, a high-level model must be generated containing as much low-level information as necessary for giving accurate estimates, while still allowing a fast simulation and estimation.

7. In order to be as flexible as possible, the proposed process must allow an estimation of the module's power dissipation and timing behaviour under consideration of various use cases and scenarios.

8. For supplementing requirement 1, it must be possible to embed the generated power and timing model into a larger and heterogeneous virtual system prototype, allowing an assessment of the overall system's metrics.

4.1 The COMPLEX Design Space Exploration Process

Enabling such a holistic approach that does not only allow estimating digital hardware modules, but other parts of the system as well, leads to some additional requirements. A typical embedded system can be split in three different types of components. There are the aforementioned digital, often full-custom ASICs, there is software, running on some kind of embedded processor, and there are IP modules, available from third-party vendors. Due to the diversity of all these types, each one requires its own estimation techniques and tools. Although this thesis focuses on full-custom digital hardware modules, it must be seen in a larger context, well knowing that the proposed estimation process for digital hardware modules must be usable together with the estimation processes for the other module types.

The presented approach is part of a larger estimation framework that allows a fast and accurate power and timing estimation of the overall system, which is built from all component types. The system estimation process was developed by an international consortium during the EU FP7 Project COMPLEX [1]. An overview of the proposed system estimation process is given in several publications [63, 65, 66], each one with its own focus on a specific aspect of the estimation process. By considering complex and heterogeneous systems, the COMPLEX project inherently fulfils requirement 1. The proposed process is outlined in Figure 4.1.

The COMPLEX estimation process starts with an executable description of the system behaviour given as C/C++ or SystemC source code. The design is verified by simulation using a set of use cases. After the correct behaviour of the system has been proven, the system is partitioned. This can be done automatically, but is typically done manually, since automatic hard-/software partition is an ambitious task of its own. In COMPLEX, the hard-/software separation is performed by the tool SMOG, allowing a user-constrained semi-automatic hard-/software separation. It also generates a test bench for the separated module. Input stimuli of the separated module are monitored during a functional simulation of the overall system and serve as input to the test bench, used for simulating the separated module individually. This allows a self-contained simulation and characterisation of the module while still considering the overall system behaviour and module interaction. Having a hard-/software separation tool available that is able to automatically generate a test bench with respect to the overall system behaviour allows a characterisation of an individual system module with respect the overall system's behaviour. With this, requirement 2 is fulfilled.

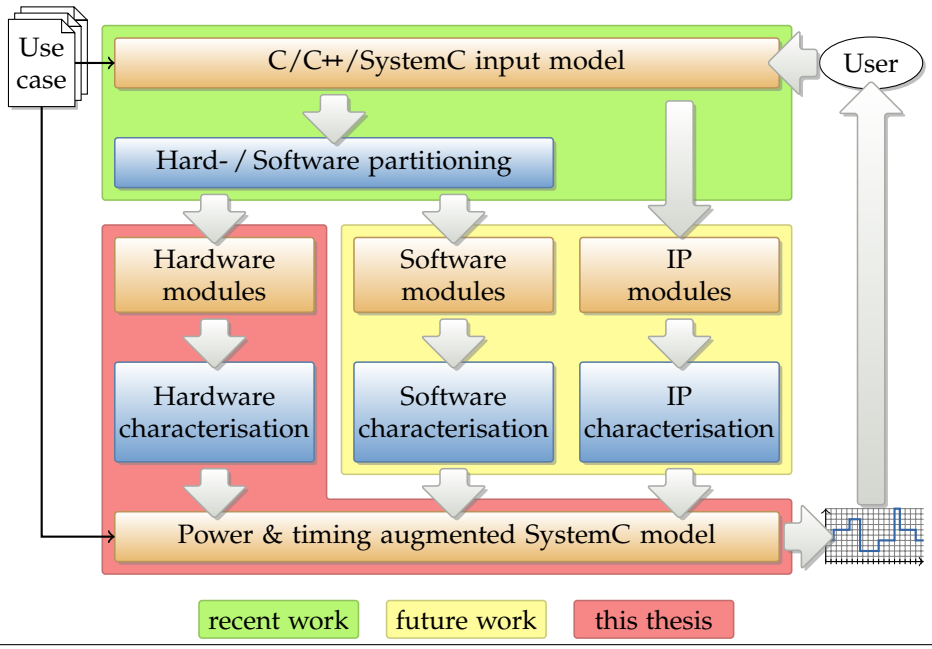


Figure 4.1: Design process, as proposed by the project COMPLEX — A complex design containing full-custom hardware, software, and IP modules is split into individual parts. Each module is characterised using the appropriate tool. After characterisation, a virtual system is created that contains augmented versions of the characterised modules. Estimates obtained during simulation of the virtual system are used during the next iteration step of the design space exploration.

For each module type, an individual characterisation is performed, using different techniques for hardware [78], software [32], and black-box IP modules [91]. All characterisation processes have in common that they create a behavioural description of the particular module. This description is augmented with estimated power and timing information. An executable model of the overall system is generated that allows a fast and precise power and timing estimation of the entire system. This self-simulating model enables a functional simulation of the system while simultaneously estimating power dissipation and exact timing. This is done using the same use cases as has been used for simulating the pure behavioural description of the system. The results of the simulation, given as traces, for example, are then used to evaluate the system and the chosen implementation.

4.2 Power Estimation Process for Digital Hardware

While power and timing estimation of software and IP modules is described by Brandolese and Fornaciari [32] and Lorenz et al. [91], respectively, this thesis focuses on power and timing estimation of full-custom digital hardware modules. Some of the methods and procedures described here as well as in Chapter 5, have previously been published [63–67, 78].

4.2.1 Basic Idea

In Section 3.1 it had been stated, that an RT-level estimation is time-consuming, because it considers individual RT components. High-level estimation however lacks of the desired accuracy. The presented approach combines the benefits of both abstractions levels by automatically creating a fast high-level model that considers as much low-level information as needed for providing accurate estimation results.

The basic idea of the approach is to consider RT components that are jointly active at once. In analogy to a basic block known from software development, such a set of atomically executed operations is called *hardware basic block*. By considering such a set of jointly active RT components at once, a significant simulation speed-up is expected. This is reinforced by the fact that certain RT components do not need to be included in the behavioural simulation but during characterisation, only.

The hardware basic blocks are automatically identified and characterised. Besides the basic blocks, other design parts such as clock-tree, controller, etc. must be considered, too. During characterisation, concrete data patterns will be used. This allows modelling data-dependencies implicitly and therefore will provide reliable estimates.

A high-level synthesis is used to refine the initially pure behavioural description given in C/C++ and SystemC into an RT-level description. This description along with representative data patterns is then used for characterising the design. From all information obtained during design analysis and characterisation, an executable C++-based high-level model is generated. This high-level model is a behavioural virtual prototype of the intended design. Besides the functional behaviour, the prototype also provides accurate information about the design's power dissipation and timing behaviour. It can then be used to perform fast, yet accurate estimations considering a large number of complex and holistic use cases. Figure 4.2 shows

how the proposed flow can be mapped onto the well-known Y-Chart, proposed by Gajski and Kuhn [53].

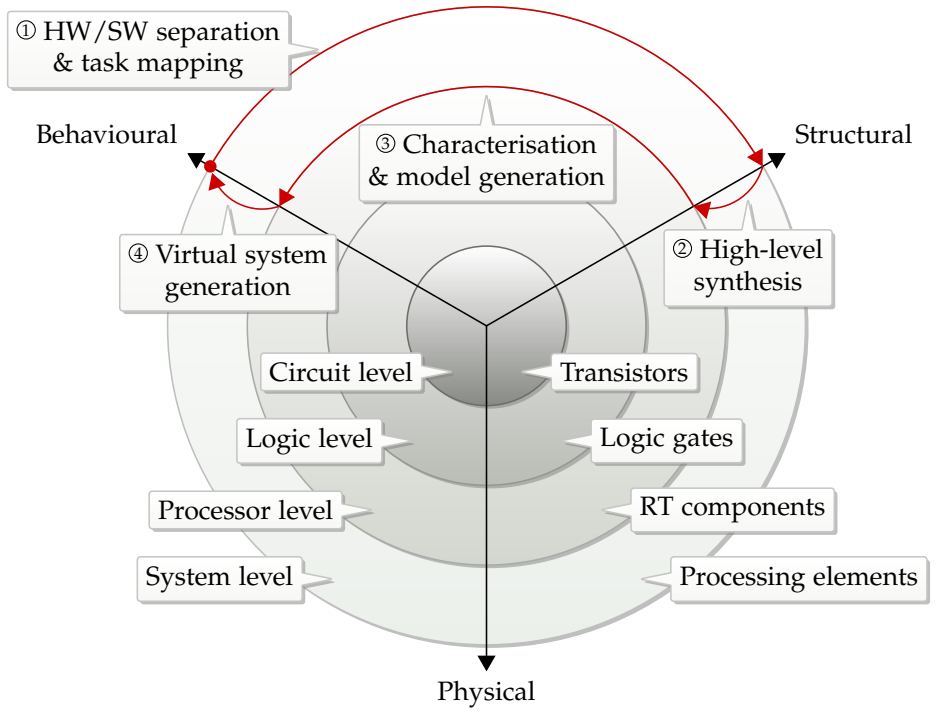


Figure 4.2: Mapping of the proposed process onto the Y-Chart — The process starts with pure behavioural description given in C/C++/SystemC. Hardware/software separation and task mapping are performed, both adding structural information to behavioural description. In a second step, a high-level synthesis refines the hardware modules onto structural RTL. During characterisation and model generation, which is the third step, information like dynamic and static power are added to the cycle-accurate behavioural model. During the final virtual system generation step all augmented sub-models are re-assembled to create the power and timing aware overall system model.

4.2.2 Estimation Process Outline

As already mentioned in the previous section, the overall system is given as pure behavioural description in C/C++ or SystemC. By performing a user-guided hardware/software separation followed by a task mapping, structural information is added to the model. After this step, the processing unit of each task is known. That is, the particular processor for software tasks or the hardware accelerator for hardware tasks, for example. The modules to be implemented as hardware accelerators are refined onto RTL. This is done automatically in the second step by a high-level synthesis tool. After synthesis, a functional description at structural RTL

is available. The RT-level description is characterised and a power and timing augmented virtual prototype of the module is generated during the third step. In the fourth and final step, all virtual prototypes of all modules are re-assembled to build the power and timing aware high-level model of the entire system. Besides the functional behaviour, this virtual system prototype contains structural information like switched capacitances, static power dissipation, or area.

The proposed estimation and characterisation process for digital hardware modules mainly consists of steps two and three i. e., high-level synthesis and augmented model generation. This process is depicted in Figure 4.3. It bases on a conventional high-level synthesis using the power-optimising synthesis tool PowerOpt. Therefore, the modules of the design that should be implemented as full-custom hardware must be given as a C/C++ or SystemC description, respectively. In any case the input code must follow the restrictions given by the synthesis tools, since not all valid C/C++/SystemC constructs can be synthesised. Using this behavioural description, a high-level synthesis is performed. The high-level synthesis performed by PowerOpt consist of all typical synthesis steps already mentioned in Section 2.3 and depicted in Figure 2.1 on page 18.

Based on the given behavioural description a CDFG, containing all information about the functional behaviour is generated. By performing scheduling, allocation, and binding, the CDFG is transformed into a structural RT-level description.

The RT-level description consists of so-called *processes*. Each process in turn consists of a controller and a corresponding RT data path. All processes operate in parallel. Inter-process communication takes place using a two-way hand-shake protocol. Data is exchanged between processes using data channels like shared registers, FIFO-queues, or memories. A design may have several instances of the same process. All instances perform the same behaviour, but each instance must be estimated and characterised individually, in order to consider different data dependencies of the individual process instances.

In a conventional high-level synthesis process, the RT data path is generated in hardware description languages, like VHDL or Verilog. These can then be used as input for the back-end synthesis process. In the proposed process however, the generated RT-level description is used to generate a power and timing augmented high-level model. The characterisation process of a hardware module is shown in detail in Figure 4.4 on page 64.

During characterisation, two types of aspects must be considered: *functional* and *non-functional* aspects. Functional aspects are directly related to the functional behaviour of the module, such as dynamic power dissipation due to system activity or timing. Non-functional aspects comprise all synthesis artefacts. These artefacts include static power dissipation and clock-tree power, for example. By considering functional as well as non-functional aspects during the characterisation process, requirement 3 from the beginning of this chapter is satisfied.

The characterised metrics are used to create a power and timing augmented C/C++ model of the hardware module. Annotating information about non-functional aspects to the behavioural description is done on the basis of basic blocks, hence the name *block annotated C++ (BAC++)*. This virtual prototype of the hardware module can then be used to build the power and timing aware virtual prototype of the overall system.

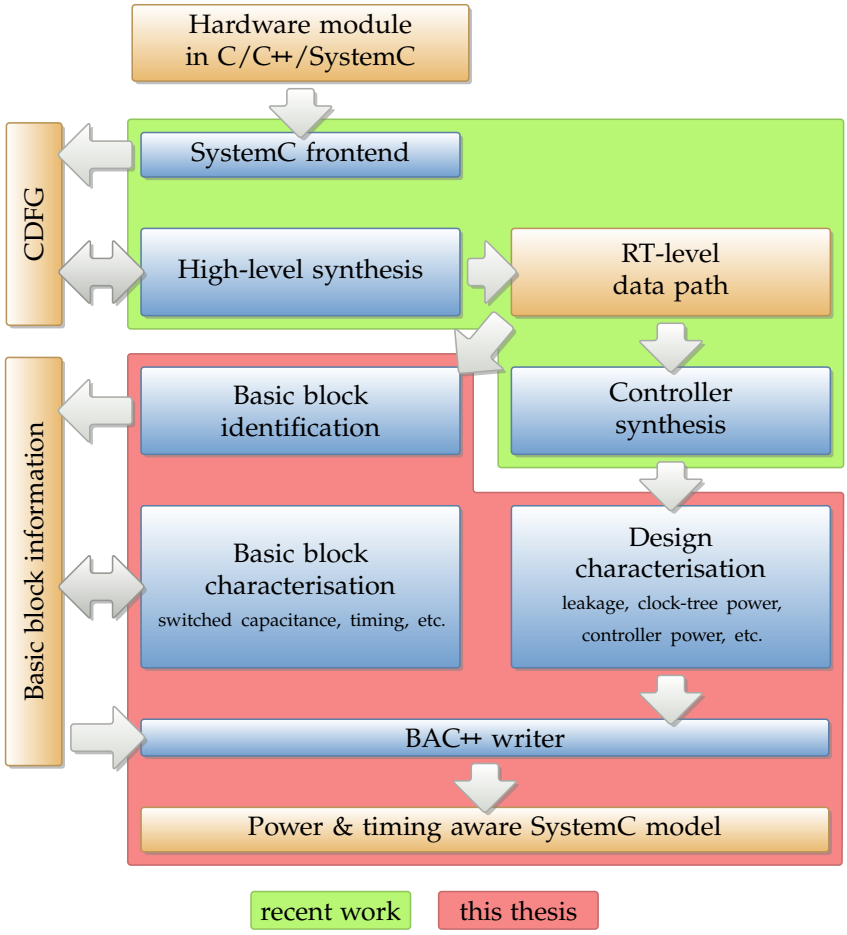


Figure 4.3: Synthesis-based characterisation process — The proposed characterisation process depends on a typical high-level synthesis, as performed by PowerOpt. A behavioural high-level description is transformed into an RT-level description by applying loop-unrolling, scheduling, allocation, binding, etc. Based on the synthesis result, hardware basic blocks are identified and characterised and non-functional properties are characterised as well. Finally, a power and timing augmented version of the module is created.

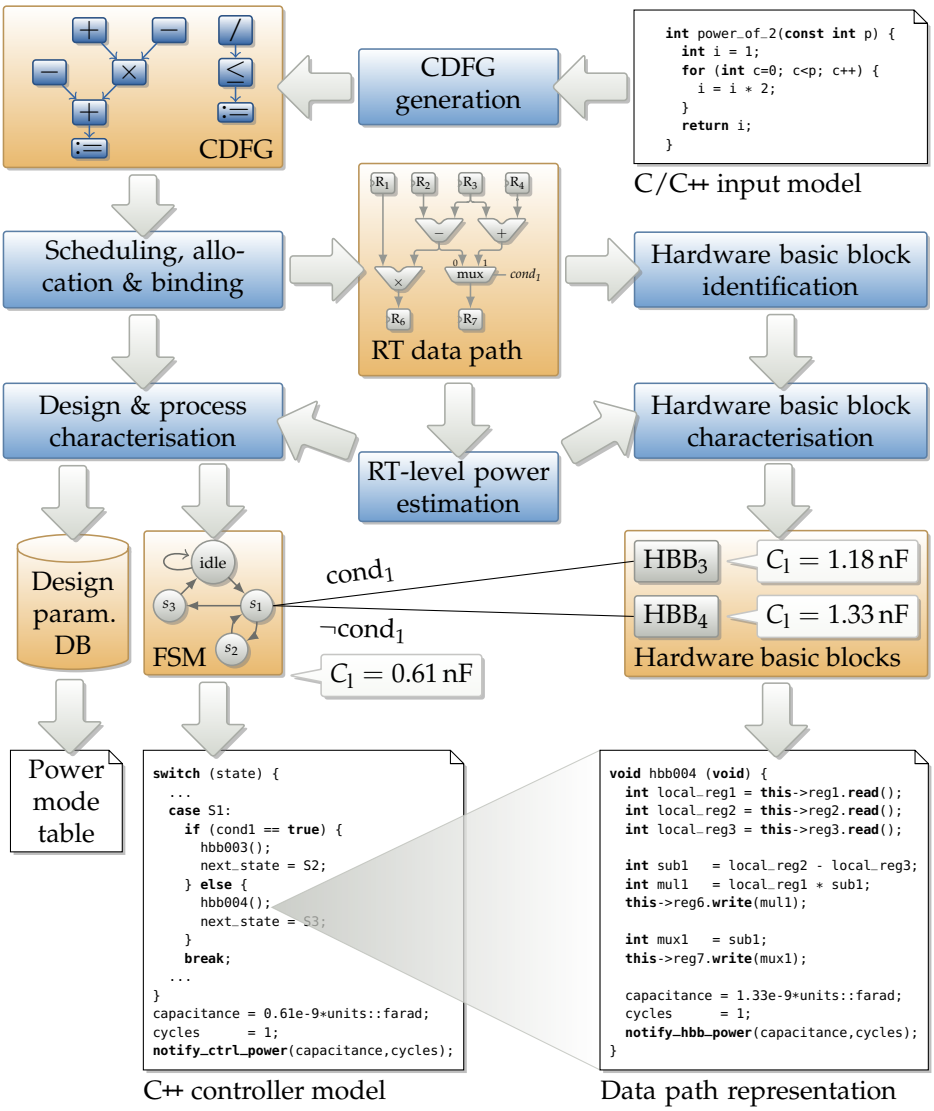


Figure 4.4: Module's behaviour characterisation process — A controller and its corresponding RT data path are generated by a conventional high-level synthesis. An RT-level power estimation is performed on both. Hardware basic blocks are identified within the data path and characterisation in terms of switched capacitance is performed. Same applies to the controller. The controller's FSM is transformed in to a large `switch`-statement, whereas the hardware basic blocks are implemented as C++-methods. Both are augmented with the previously characterised power and timing information. General design parameters like equivalent conductance, average power dissipation, or different supported supply voltages and clock frequencies are used for creating the power mode table, which specifies different modes the module can operate at.

4.2.3 Classification of the Proposed Process

The proposed approach can be classified with respect to the existing work, mentioned in Chapter 3. Like many of the approaches mentioned above, the proposed approach uses an RT-level description as input model, since it provides the appropriate level of detail, required for accurate power and timing characterisation. Considering the RT-level characterisation approaches above, it can be seen that the approach just outlined is more precise than the one presented by Ahuja et al. [6, 7], which considers only the toggle count during a given state of the FSM. However, compared to the approach presented by Zhong et al. [144, 145] it is less precise, since not individual RT components and their input patterns are considered, but combinational macros. Individual RT components are considered during characterisation, only. Moreover, the approach presented in this thesis uses a more detailed RT-level power model for characterisation, which considers all components of the data path as well as interconnect, memories, etc. In doing so, the proposed approach surpasses most of the drawbacks of the approach presented by Zhong et al.

During model generation, the presented approach follows the advices given by Krishnaiah et al. [84]. It tries to reduce the architectural detail as much as possible, while maintaining all information required for a proper behavioural simulation and accurate power estimation. It also tries to use native C/C++ data types in the generated high-level model whenever possible, in order to speed up the simulation. Like most RTL-to-C/C++ abstraction tools [28, 61] the proposed approach will statically schedule the RT-level statements inside the identified combinational macros. In contrast to the work of Bombieri et al. [28], no optimisation in terms of re-scheduling, register reduction, etc. will be done, in order to maintain an accurate power and timing estimate of the characterised module.

The TLM wrapper, used for integration of the generated high-level model into the virtual system prototype is similar to the TLM wrappers proposed by Bombieri et al. [23, 26]. One difference is that the approach, proposed in this thesis uses a CAFD instead of an EFSM, which, however, is not very different. The most important difference is that in contrast to the abstraction technique described by Bombieri et al., the approach presented in this thesis provides cycle-accurate power information.

4.3 Input Model

Before describing the proposed power and timing characterisation and estimation process in detail, the input model to the process is specified. As pointed out earlier, the generated RT-level description consists of a set of parallel running processes. Each of these processes is estimated and characterised individually. The following sections describe the models used for representing a single process i. e., the controller and its corresponding RT data path. Since the following sections of this chapter all refer to the RT level, the term *data path* refers to the RT data path hereafter. Describing the input model in a formal way helps adopting the proposed techniques to other synthesis and estimation tools than PowerOpt.

4.3.1 FSM of the Controller

The controller, controlling the data path's behaviour is an FSM F. It corresponds to the Mealy machine model, which has been developed by Mealy for the synthesis of sequential circuits [98]. It is defined as follows:

$$F = (\Sigma, \Gamma, S, s_0, \delta, \omega)$$

The elements of the sextuple are defined as:

$$\begin{aligned} \Sigma &: \text{Input alphabet} \\ \Gamma &: \text{Output alphabet} \\ S \neq \emptyset &: \text{Finite set of states} \\ s_0 \in S &: \text{Initial state} \\ \delta : S \times \Sigma \mapsto S &: \text{Transition function} \\ \omega : S \times \Sigma \mapsto \Gamma &: \text{Output function} \end{aligned}$$

The state machine's in- and output symbols can be split into individual bits. Each bit represents a specific in- or output control signal of the controller, respectively:

$$\sigma \in \Sigma : \{0,1\}^m \qquad \text{and} \qquad \gamma \in \Gamma : \{0,1\}^n$$

The in- and output signals connect the controller's FSM to the data path and other processes of the module. The controller's output signals are used to control the data path. That is, enabling registers, selecting active multiplexer inputs, and performing the hand-shake with other processes. Inputs, however, are used to access register values, immediate computation results, and are also used by the hand-shake protocol. Figure 4.5 gives overview of the controller's structure how it interacts with its environment. There may be also direct connections between the data path and the other processes. These data and address signals are used by the data path for accessing shared data channels.

4.3.2 RT Data Path

The data path G, defined by Equation (4.1) is a directed graph. It consists of RT components, represented by the vertices V of the graph and connections lying in-between, corresponding to the edges E of the graph. The data path is controlled by the FSM F, whose output alphabet Γ consists of all valid combinations of control signal values. On the other hand, some signals of the data path serve as input for the FSM and thus form the input alphabet Σ.

$$G = (V, E) \qquad \text{with} \qquad E \subseteq (V \times V) \tag{4.1}$$

The set V of RT components can be split into subsets, depending on the type of the individual components:

$$V = V_{\mathrm{R}} \cup V_{\mathrm{M}} \cup V_{\mathrm{O}} \cup V_{\mathrm{C}}$$

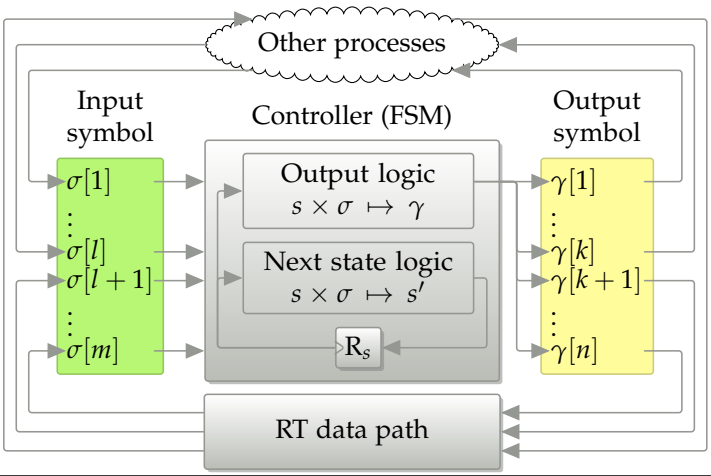


Figure 4.5: Controller interaction — The controller is a FSM. The input symbol σ is built from registers and intermediate results of the data path as well as from the handshake signals of the other processes. Based on the input symbol and current state s, the output symbol γ and the next state s' are computed by the output and next state logic, respectively. The output symbol can be split into register-enable and mux-select as well as handshake signals. The first two control the data path and the latter are used for inter-process communication.

The individual sets are defined as:

$$V_\mathrm{R}: \text{Registers of the data path}$$
$$V_\mathrm{M}: \text{Multiplexers of the data path}$$
$$V_\mathrm{O}: \text{Operators of the data path}$$
$$V_\mathrm{C}: \text{Constants of the data path}$$

Each RT component $v \in V$ has in- and outputs, used for interconnecting two consecutive components. Each input can have exactly one connection. An output can have multiple connections. Therefore, each component has only one output. The general statement that a component has one output and possible multiple inputs has some exceptions. Constants do not have any inputs. Registers and multiplexer however, also have control signals, allowing to control the component's behaviour. That is, enabling the register or selecting the multiplexer's desired input, respectively.

An edge (v_1, v_2) represents a directed connection between two RT components v_1 and v_2, where the data output of v_1 is connected to one of the data inputs of v_2. An auxiliary function shown in Equation (4.2) on the next page allows traversing through the data path. It specifies an ordered tuple of components that have a direct or an indirect connection between each other i. e., there exists a path between the two components. Such a path may contain cycles, but it is required that all cycles in the data path must contain at least one register, as shown

in Equation (4.3).

$$\text{path}(v_1, v_2) = (w_1, w_2, \ldots, w_n) \mid w_1 = v_1, w_n = v_2, (w_i, w_{i+1}) \in E \tag{4.2}$$

$$\exists v_i (v_i \in \text{path}(v_1, v_1) \wedge v_i \in V_\text{R}) \tag{4.3}$$

This is a typical requirement for clocked sequential systems. It also assures that no delta cycles must be considered during characterisation of the data path. Nevertheless, there might be loops between the data path and the controller as well as between different processes. This issue will be discussed in Section 5.3. Due to timing requirements and in order to prevent resonant circuits, most synthesis tools do not implement unregistered loops in the data path. Thus, this requirement does not restrict typical designs. The limitations caused by it can therefore be neglected.

4.4 Functional Properties

As pointed out earlier, estimation and characterisation of functional properties is two-fold. First, so-called *hardware basic blocks* are identified within the data path. A hardware basic block is a set of RT components jointly active within a specific control step i. e., a specific clock cycle. A hardware basic block can thus be considered as a combinational macro, describing all operations performed during that control step. The concept of a hardware basic block is similar to a basic block, known from compiler construction [62, sec. 9.1.2] and software estimation [33]. While a software basic block is a temporal abstraction, covering instructions from multiple cycles, a hardware basic block is a spatial abstraction, covering multiple operations, executed in parallel. Second, an estimation of each hardware basic block's power dissipation is performed. Estimation results are then used to characterise each hardware basic block. Finally, some optimisations can be performed on the characterised hardware basic blocks, in order to speed-up their execution.

Performing characterisation based on hardware basic blocks provides an appropriate level of granularity. It allows a cycle-accurate estimation of the power dissipation as well as accurate timing prediction, while providing a lot optimisation potential to the compiler, as shown in Section 5.4.1. Since a hardware basic block comprises several RT components, it might occupy a large area of the chip. Evaluation shows that up to 24 % of the total chip area can be activated by a single hardware basic block. While considering a large number of components at once gives a significant increase in simulation performance, it reduces the spatial resolution of the estimated power dissipation. In other words, the amount of energy dissipating as well as the point in time it dissipates can be estimated accurately, but not the location on the chip. Unfortunately, this information is required for identifying hot-spots, for instance. However, Sander et al. have shown that distribution of heat can be considered at a much coarse-grained level in terms of spatial and temporal resolution [119]. Put simply, the chip is not becoming much warmer in a single clock cycle.

4.4.1 Preliminary Considerations

Before giving a detailed description of the identification and characterisation process, some preliminary consideration must be made. It is important to understand how the controller creates the output symbol and how data is processed within the data path in order to understand how unique hardware basic blocks can be identified. Reducing size and complexity of the power and timing augmented high-level model, generated from the characterised hardware basic blocks is an important goal. Thus, it is required to reduce the number and complexity of the identified hardware basic blocks in the first place. The following paragraphs give an insight of the controller's and data path's internal behaviour. This basic knowledge is then used for developing a hardware basic block identification technique.

Behaviour of the Controller's Output and Next-State Logic

In order to provide a hardware basic block identification process, it is required to understand how the RT components that are active during the execution of a hardware basic block are activated by the controller. That is, the interaction between the controller and the data path must be understood. The set of active components is defined by the output signals of the controller, which in turn depend on the controller's current state and input signals, applied at the same time. Figure 4.6 depicts how the values of the output signals are computed by the controller.

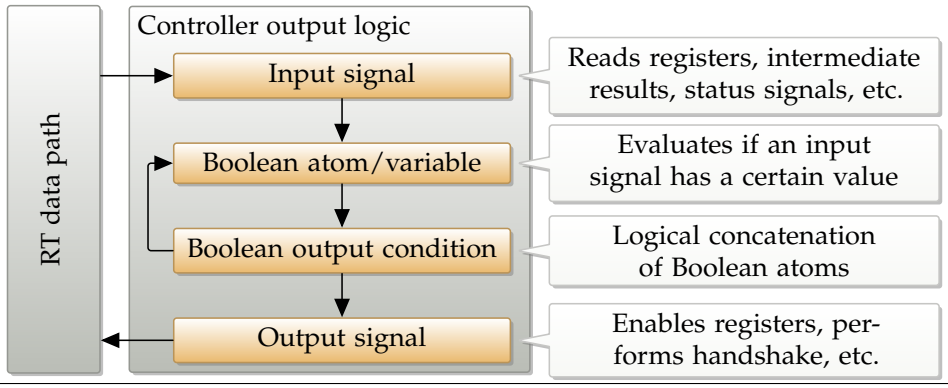


Figure 4.6: Controller output logic — Input signals are read from the data path and are used as Boolean atoms within the output conditions. An output condition is a logical concatenation of multiple atoms. An output condition can also be an atom itself. If an output condition evaluates to `true`, the output signal gets the value assigned that is associated to the condition. This in turn can be a Boolean condition again.

For each state of the controller's FSM a set of so-called *output conditions* determines if a specific output symbol i. e., a specific pattern of control signals is applied to the data path. These output conditions also determine the next state of the FSM. That is, the output conditions represent the controller's output and next-state logic or the FSM's output and transition function, respectively. An output condition can perform assignments to several signals at the same time. If an output condition evaluates to `true`, the value, associated to the condition

is assigned to the corresponding control signals. This value in turn can also be a Boolean condition. In this case, the value assigned to the signal arises from a conjunction of both conditions.

An output condition is a Boolean expression very similar to the formulas known from *satisfiability modulo theories (SMT)* problems. It is built from AND and OR concatenation of SMT predicates. In contrast to a conventional Boolean predicate, the SMT predicates make statements about non-Boolean-valued variables. Using a Boolean abstraction, the SMT predicates can be considered as Boolean atoms. During simulation, it is possible to evaluate the statement of the predicate and thus determine the truth-value of the Boolean atom. An output condition may also be an atom itself. This is, its logical name is used within another output condition.

The variable assignment for each Boolean atom or SMT predicate, respectively is obtained from the input symbol i. e., the controller's input signals. That is, the predicates make statements about the values of the signal assigned to the particular predicate. In their entirety, the input signals to make a statement about the state of the data path.

An input signal is used for accessing register values and intermediate results of the data path. Handshake signals, used for communicating with other processes and data channels are also used as input. An example pseudo-code, showing the correlation of the individual stages of the output logic is given in Listing 4.1.

```
   //Obtain value of Boolean atoms from data path (SMT predicate evaluation)
   boolean Sel_Reg_1    := (Reg_1==5);
3  boolean Sel_Reg_2    := (Reg_2==true);
   boolean Sel_Assign_1 := (Assign_1==false);

6  //Compute output conditions from Boolean atoms
   boolean ctrlflow_1 := (Sel_Reg_1==true);
   boolean ctrlflow_2 := (Sel_Reg_1==false)    && (Sel_Reg2==true);
9  boolean ctrlflow_3 := (Sel_Assign_1==false) || (ctrlflow_2);

   //Assign values to output/control signals
12 MuxSelect_Mux_4 := ctrlflow_1;
   RegEnable_Reg_4 := ctrlflow_2;
   if (ctrlflow_3) then
15    RegEnable_Reg_5 := true;
   else
      RegEnable_Reg_5 := false;
18 endif
```

Listing 4.1: Pseudo-code implementation of the output logic — Input signals are accessed. Results of the SMT predicate evaluation are assigned to Boolean atoms. These are used to evaluated the output conditions, which in turn set the output signals.

Operation of the Data Path

In an RT data path, operations are not only required for providing registers' inputs, but also for obtaining values used for communicating with the surrounding system i. e., other processes of the module. In this case, values are not stored in registers, but are written to data ports, connecting the process with the surrounding processes and data channels. While the activation of a Register can be obtained from its enable signal, the activation of a data port can be obtained from the value of the associated handshake signal. In the following, registers and ports can be considered to be equal. Thus, in the next sections the term *register* referrers to both, data ports and registers.

If a register takes a new value, this value appears at the register's output at the beginning of the next clock cycle i. e., control step. From the output of the register the signal value propagates trough the data path, activating several RT components, which perform different transformations i. e., operations on the signal. The signal may also be split or duplicated. Signal propagation stops at registers, since they will propagate the signal in the cycle after the next one. Hence, the set of active RT components and thus the hardware basic block, which describes the operation of the data path within one control step, is enclosed by registers. Timing of the system is given by the propagation delay of the longest path between two consecutive registers. Signal transformations and updating the values stored in the registers cause dynamic power dissipation, as explained in Section 2.4.1. Dynamic power dissipation is therefore caused by register outputs that change at a rising or falling clock edge, respectively, depending on if the registers-enable signals are active-high or active-low.

Top-Down versus Bottom-Up Identification Approach

Following a straight forward approach, identifying a hardware basic block requires knowledge of which registers are updated at the end of one control step, which in turn will cause RT components to be active in the next control step. An RT component is considered to be active if it propagates the updated output signal of a register. The active components are determined by the given data path i. e., the connections between the RT components. This is especially true for the multiplexers of the data path and the values of their select-signals in the second control step.

The proposed estimation approach uses a static analysis of the process's controller and data path. This requires characterising each possible data flow in advance and then using the pre-characterised properties during high-level estimation. For the straight forward approach, the data path is analysed top-down along the data flow. Using this approach, a large set of possible combinations of control signal values must be considered. The total number of possible combinations of signal values is shown in the first part of Equation (4.4) on the following page. It consists of all possible combinations of register enables signals in state s_i denoted by reg_ena(s_i) and the possible combinations of mux-select signals in *all* subsequent states s_{i+1}, denoted by mux_sel(s_{i+1}) Handling such a large number of possible combinations will lead to a very large number of hardware basic blocks that must be characterised. This large number of basic blocks cannot be identified and characterised in an acceptable amount of time. It will also lead to very complex high-level model in the second place. Therefore, the straight forward approach is not feasible. A more sophisticated technique is necessary.

Assuming that the results of the signal transformations are required for the operation of the design and are thus stored in registers, another approach can be used. In this approach the hardware basic block is defined by the registers whose values are updated at the end of the control step. The hardware basic block then consists of all components that are required for providing the registers' inputs. Since the values of the register-enable and mux-select signals are applied in the same control step and due to the fact that the output of the source registers does not change during that control step, only this single one control step must be considered. The second part of Equation (4.4) shows that this drastically reduces the number of possible combinations of control signal values that must be considered. In this approach power dissipation should not be referred to as *caused by changing register outputs*, but as *required for providing registers' inputs*.

Using the bottom-up approach has the disadvantages that it is assumed that source registers take a new value exactly one clock cycle before their value is required. In other words, it is assumed that the value of a register is used in the clock cycle directly after the register has taken its new value. This is obviously not always true. It might occur that a register value is updated, but that the updated value is required several cycles later. In this case the caused power dissipation is assumed to occur later, which in turn blurs the estimation result. This effect and its influence are discussed in more detail in Section 6.5.5.

Equation (4.4) compares the number of possible combinations of signal values and therefore hardware basic blocks that must be identified and characterised when using the straight forward and the proposed approach, respectively. Since register-enable and mux-select signal are set in the same control step, they may share some signals. This again will reduce the number of possible signal value combinations that must be considered, as shown in the third part of Equation (4.4).

$$\underbrace{\sum_{\forall s_{i+1}} \left(2^{|\text{reg\_ena}(s_i)|+|\text{mux\_sel}(s_{i+1})|}\right)}_{\text{Top-Down}} \gg \underbrace{2^{|\text{reg\_ena}(s_i)|+|\text{mux\_sel}(s_i)|}}_{\text{Bottom-Up (worst case)}} \geq \underbrace{2^{|\text{reg\_ena}(s_i)\cup\text{mux\_sel}(s_i)|}}_{\text{Bottom-Up (general case)}} \tag{4.4}$$

Unique Hardware Basic Block Identifier

After having defined that register-enable and mux-select signals from the same control step are used for hardware basic block identification, a unique hardware basic block identifier is required next. Since a hardware basic block is assumed to be responsible for providing register inputs, the first idea is to use the enable signals of the registers that will store the computed values at the end of the clock cycle as the identifier for the hardware basic blocks. Actually there are some drawbacks, when using the register-enable signals, only. First, register enable signals might by ambiguous. Algorithm 4.1 depicts this. Register R_1 is enabled, regardless of the condition. This is, knowing that register R_1 is enabled is not sufficient enough for knowing which computation must be performed by the activated hardware basic block.

Second, there are a lot possible combinations of the register-enable signals, but not all combinations are used, as shown in Algorithm 4.2. In the example, only two out of 16 combinations must be considered, since register R_1 and R_2 as well as R_3 and R_4 are enabled jointly. That is, the values of a large number of output signals may be obtained from a

smaller number of output conditions. Even though registers R_1 and R_2 might share the same register-enable signal, this is not always the case. Both registers might be used individually by other hardware basic blocks requiring a single register-enable signal for both of them. Obviously the same is true for registers R_3 and R_4.

```
if condition then
    R1 ← R2 + R3;
else
    R1 ← R4 − R5;
end if
```

Algorithm 4.1: Assignment to same register — Using register-enable signal may be ambiguous. In both branches R_1 is enabled, even though different computations are performed.

```
if condition then
    R1 ← R5 + R6;
    R2 ← R6 + R7;
else
    R3 ← R5 − R7;
    R4 ← R6 + R8;
end if
```

Algorithm 4.2: Multiple register assignments — Considering the four register-enable signals, $2^4 = 16$ combinations are possible, although only two will occur.

The first issue concerning the ambiguity can be solved by considering register-enable as well as mux-select signals. Then, the hardware basic block is not only defined by the register-enable signal, but also by the mux-select signal selecting the correct operator's output. But again, there are lot combinations of signal values possible, even more than when only considering register-enable signals. Once more, some combinations of output signal values are used, while others will never occur. It is apparent that some knowledge about data path' behaviour intended by the controller is required.

As mentioned earlier and shown in Figure 4.6 on page 69, the values of the control signals are computed by evaluating the controller's output conditions. The controller's output conditions in turn perform an evaluation of the input signals, which provide the assignments to the Boolean variables. An output condition of a signal can be built from multiple Boolean atoms. It is required that the total number of Boolean variables that must be taken into account when identifying a hardware basic block is smaller than the number of output signals driven by the controller. Otherwise using output signals as identifier would be more efficient. Evaluation shows that the number of Boolean atoms used inside the output conditions is significantly smaller than the number out output signals, driven by the controller. This is partly because most of the atoms are used in several output conditions at the same time.

The behaviour of the data path and thus each hardware basic block can therefore be identified by a combination of the controller's state s and a set of variable assignments a to the Boolean atoms of the output conditions. The considered variable assignments must cover all variable assignments, possibly caused by the controller's input signals, required for evaluating the output conditions. That is, a hardware basic block describes the behaviour of the data path caused by the unique output symbol generated by the output conditions, if the given variable assignment is applied. An example identifier is shown in Equation (4.5). For a better understanding and readability, the Boolean atoms have been replaced with the SMT

predicates, they are mapped to. For this example, the hardware basic block, associated to the identifier is active, if the controller's FSM is in state s_1, register R_1 has the value true, while register R_5 does not have value 0 and finally, comparator C_9 must not give the intermediate result `true`.

$$(s, a) = (s_1, \{(R_1 = \text{true}) = \text{true}, (R_5 = 0) = \text{false}, (C_9 = \text{true}) = \text{false}\}) \tag{4.5}$$

4.4.2 Hardware Basic Block Identification

For each combination of state and variable assignment, a hardware basic block is identified. Using state and assignment, the controller's output symbol i. e., the values of the controller's output control signal can be computed. Starting from the control signals, enabled registers are identified. These so-called *target registers* define the end of the hardware basic block. All RT components that are required for providing the target registers' inputs are part of the hardware basic block. Which components are required for the particular register's input is also defined by which inputs of the data path's multiplexers are selected during the current control step. Only components connected to the selected input must be considered. The beginning of the hardware basic block is defined by the registers that provide the input to the active RT components. These registers are called *source registers*. Source and target registers must not necessarily be a disjoint sets i. e., a source register can also be a target register and vice versa.

Providing the registers' inputs may cause some parasitic functionality. Since this parasitic functionality also causes power dissipation, these components must also be considered while estimating power dissipation of the hardware basic block. Components causing parasitic functionality are identified by searching the data path for components whose inputs are provided by the source registers but which do not belong to the components required for providing the target registers' inputs. In other words, they get new input values, but their results are not needed and thus discarded.

Figure 4.7 shows exemplary the hardware basic block that results, if the controller performs the following operations in the current control step:

$$R_6 \leftarrow R_1 \cdot (R_2 - R_3)$$
$$R_7 \leftarrow R_2 - R_3$$

In the particular control step, the controller enables registers R_6 and R_7. These two registers are the target register of the hardware basic block and serve as starting point for the identification of active RT components. Traversing the data path upwards contrary to the data flow, it can be seen that the multiplication and the multiplexer are active. Registers R_1 and R_2 serve as input to the multiplication and thus are the first identified source registers. In the particular control step, the controller also configures the multiplexer to select input 0. Thus, only the data path connected to that input is considered. In this example the subtraction must be considered. Registers R_2 and R_3 serve as input, said register R_2 is already known to be a

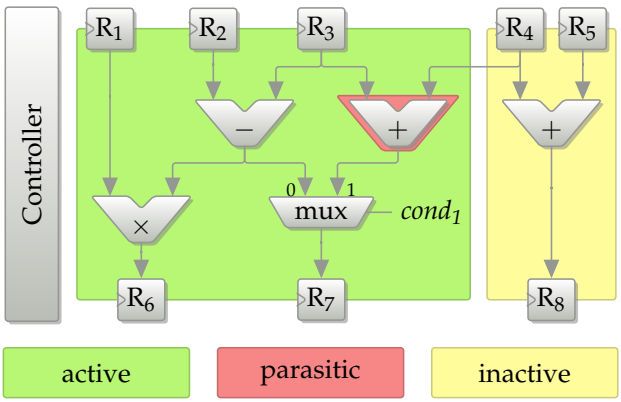


Figure 4.7: Definition of a hardware basic block — A hardware basic block is a set of RT components that is required for providing the enabled registers' inputs. It may also contain components that are active due to parasitic functionality. Registers define the beginning and the end of the hardware basic block, respectively.

source register. Again it must be noted that all source registers are assumed to provide new values, even though this is not always the case. Since register R_3 is a source register and may change its output the adder may also become active, even though the result of the operation is discarded. Thus, the adder belongs to the set of RT components active due to parasitic functionality. Since register R_8 is not enabled in the particular control step, the data path providing its input is considered to be inactive.

Formally, a hardware basic block H is a sub-graph of the data path G. It contains all RT components that are active while the FSM of the controller is in a specific state s and the input symbol σ is applied to the FSM. The input symbol creates a specific variable assignment a, causing the controller's output conditions i. e., output logic to apply output symbol γ to the data path. The output symbol defines which register-enable signals are set and which inputs are selected by the multiplexers of the data path.

Since the output symbol applied to the data path and thus the hardware basic block depends on the current state s and variable assignment a of the FSM, the basic block can be identified by (s, a). Equation (4.6) defines a hardware basic block formally.

$$H^{(s,a)} = \left(V^{(s,a)}, E^{(s,a)}\right) \subseteq G = (V, E) \tag{4.6}$$

The elements of the tuple are defined as:

$V^{(s,a)}$: RT components, active during (s, a)

$E^{(s,a)}$: Connections between the active RT components

In order to define different sets of RT components inside a hardware basic block Equation (4.7) on the next page provides useful specialisation of the above mentioned path-function from Equation (4.2) on page 68. It requires that all valid paths within a hardware basic block must

start and end at a register, but do not pass one. This assures that all paths belonging to the hardware basic block can be computed in one clock cycle.

$$\text{path}_H(v_1, v_2) = \text{path}(v_1, v_2) \colon \forall v_i (v_i \in \text{path}(v_1, v_2) \setminus \{v_1, v_2\} \implies v_i \notin V_\text{R}) \tag{4.7}$$

During execution, not all possible paths inside the data path of a hardware basic block are active. A multiplexer for example does only forward one of its inputs. That is, only components connected to the selected input belong to an active path. Components connected to the inputs that are not selected by the multiplexer must not be considered. Equations (4.8), (4.9), and (4.10) allow determining whether a specific path is active, or not.

$$\text{input}(v_1, v_2) \mapsto \mathbb{Z} \colon \begin{cases} -1, \text{ if } (v_1, v_2) \notin E \\ \text{Index of the input port of } v_2 \\ \text{to which } v_1 \text{ is connected, else} \end{cases} \tag{4.8}$$

$$\text{select}^{(s,a)}(v) \mapsto \mathbb{N}_0 \colon \begin{cases} \text{Index of the input port of } v \text{ that} \\ \text{is selected during } (s, a) \end{cases} \tag{4.9}$$

Equation (4.8) allows identifying the input port of the multiplexer to which a given component is connected. Equation (4.9) however, determines the input that is selected by the multiplexer if the controller is in state s and assignment a is given by the input symbol. Having these functions defined, it is possible to define Equation (4.10), which determines whether a specific edge, connected to one of the multiplexer's inputs is active, or not.

$$\text{active}^{(s,a)}(v_1, v_2) \mid (v_1, v_2) \in E \colon \begin{cases} \text{true}, & \text{if } v_2 \notin V_\text{M} \\ \text{true}, & \text{if } v_2 \in V_\text{M} \text{ and} \\ & \text{input}(v_1, v_2) = \text{select}^{(s,a)}(v_2) \\ \text{false}, & \text{else} \end{cases} \tag{4.10}$$

If the target component is not a multiplexer, all incoming edges are always active. This is the case for adders, multipliers, or comparators, for example. An incoming edge of a multiplexer however, is only active if it is connected to the selected input.

Using the definition from Equation (4.10), it is possible to define an *active path*. An active path is a path that contains only components that are pairwise connected via active edges. That is, if two components on the path are connected via an edge, it must be an active one.

$$\begin{aligned} &\text{activepath}(v_1, v_2) = \text{path}_H(v_1, v_2) \mid v_1, v_2 \in V \colon \\ &\qquad \forall w_1, w_2 \Big(w_1, w_2 \in \text{path}_H(v_1, v_2) \wedge (w_1, w_2) \in E \iff \text{active}^{(s,a)}(w_1, w_2) \Big) \end{aligned} \tag{4.11}$$

Having all these auxiliary functions defined, it is possible to define different sets of components that are all belonging to a specific hardware basic block. The components $V^{(s,a)}$, which are active while a hardware basic block is active, can be split into source registers $V_\text{SR}^{(s,a)}$ providing the input, target registers $V_\text{TR}^{(s,a)}$ storing new values at the end of the current control

step, active components $V_{\mathrm{A}}^{(s,a)}$ performing the arithmetic operations of the hardware basic block, and some additional nodes $V_{\mathrm{E}}^{(s,a)}$, which are active due to parasitic functionality. All are shown in Equation (4.12).

$$V^{(s,a)} = V_{\mathrm{SR}}^{(s,a)} \cup V_{\mathrm{TR}}^{(s,a)} \cup V_{\mathrm{A}}^{(s,a)} \cup V_{\mathrm{E}}^{(s,a)} \tag{4.12}$$

The individual sets of components are defined as follows: The set of target registers i. e., the set of registers storing new values at the end of the current control step is defined by Equation (4.13). For the example hardware basic block from Figure 4.7 on page 75 $V_{\mathrm{TR}}^{(s,a)}$ contains registers R_6 and R_7.

$$V_{\mathrm{TR}}^{(s,a)}\text{: Registers enabled during } (s,a) \tag{4.13}$$

Equation (4.14) defines the set of active components $V_{\mathrm{A}}^{(s,a)}$. These are all the components that are on an active path, which ends at one of the target registers, but are not a register themselves. Starting from registers R_6 and R_7 in the example above, the set of active components contains the multiplier, the multiplexer, and the subtractor. The multiplier and subtractor are on the active paths that end at register R_6. For the given example, the multiplexer selects input number 0. That is, the active and thus possible paths must pass the subtractor. All paths through the adder are not active. Thus, the RT components on these paths do not belong to the set of active components.

$$V_{\mathrm{A}}^{(s,a)} = \mathrm{activepath}(v_1, v_2) \setminus V_{\mathrm{R}} \mid v_1 \in V, v_2 \in V_{\mathrm{TR}}^{(s,a)} \tag{4.14}$$

The set of source registers providing the input to the hardware basic block are defined by Equation (4.15). The set contains all registers having an active path to at least one of the target registers. Applying this equation to the example will result in a set, containing R_1, R_2, and R_3. For all these registers there exists a path, ending at one of the target registers from set $V_{\mathrm{TR}}^{(s,a)}$. There is also a path from R_4 to R_7, but since the multiplexer does not select input number one, this one is not active. Therefore, register R_4 does not belong to the set of source registers.

$$V_{\mathrm{SR}}^{(s,a)} = \mathrm{activepath}(v_1, v_2) \setminus \left(V_{\mathrm{TR}}^{(s,a)} \cup V_{\mathrm{A}}^{(s,a)} \right) \mid v_1 \in V_{\mathrm{R}}, v_2 \in V_{\mathrm{TR}}^{(s,a)} \tag{4.15}$$

Finally, the set of components active due to parasitic functionality comprises all components that have an active path, starting at one of the source registers, but are neither a target register nor an active component. This set is defined by Equation (4.16). For the example used here, this means that the adder and subtractor as well as the multiplier and the multiplexer lie on a path starting at one of the source registers $V_{\mathrm{SR}}^{(s,a)}$. However, only components that are neither member of the active components nor are registers are considered. Thus, the only remaining component is the adder.

$$V_{\mathrm{E}}^{(s,a)} = \mathrm{activepath}(v_1, v_2) \setminus \left(V_{\mathrm{R}} \cup V_{\mathrm{A}}^{(s,a)} \right) \mid v_1 \in V_{\mathrm{SR}}^{(s,a)}, v_2 \in V \tag{4.16}$$

Roughly speaking, active components $V_{\mathrm{A}}^{(s,a)}$ perform the operations, whose results are stored in the target registers $V_{\mathrm{TR}}^{(s,a)}$. That means that there exists a path from at least one of the target registers to the active component. The source registers $V_{\mathrm{SR}}^{(s,a)}$ provide the input for the active components. That is, there exists a connection from the target registers to the source registers via the active components. Thus, the active components are enclosed by source and target registers, respectively. A register can be source and target register at the same time. Some components $V_{\mathrm{E}}^{(s,a)}$ are also active due to activity induced by the active components and the source registers. In this case, both serve as input for these additional components. The results of these components are neither required nor used. Hence, their unwanted activity is called parasitic functionality. An algorithmic description of the hardware basic block identification can be found in Algorithm A.1.

4.4.3 Handling the State Explosion

For reasonable designs, creating a hardware basic block for each possible output symbol i. e., all possible assignments to the Boolean output conditions in a specific state of controller's FSM is infeasible. For n Boolean-valued output signals 2^n hardware basic blocks must be characterised, in order to achieve a full coverage. Knowing that typical designs can have more than 100 output signals, yielding more than $2^{100} \approx 1.27 \times 10^{30}$ hardware basic blocks, it is obvious that the number of hardware basic blocks must be reduced.

First idea is to use a *Moore machine* instead of a *Mealy machine* for describing the controller's FSM. In contrast to a Mealy machine, where the output symbol is generated based on the machine's actual state and applied input symbol, the output symbol of Moore machine depends on the machine's state, only. Mealy and Moore machines have the same cardinality and can be converted into each other [71, p. 25]. Transforming the Mealy machine into a Moore machine is done by creating a state in the Moore machine for each combination of state and input symbol of the Mealy machine. Again, for n Boolean-valued input signals and m states in the Mealy machine $2^n \times m$ states are created for the Moore machine.

This effect is also known as *state explosion* and prohibits transformation of the Mealy machine into a Moore machine for reasonable complex designs. Because of the exponential behaviour, less considered output signals lead to significantly less hardware basic blocks to characterise. The number of considered output signals can be reduced in various ways. The high efficiency of the techniques presented in the following, in reducing the number of actually identified basic blocks to only a minimal fraction of the originally estimated number, is shown in Section 6.1.1.

Consider Only Control Signals

As can be seen in Figure 4.5 on page 67, not all output signals of the controller are connected to the data path. Some signals are required for performing the hand-shake protocol with other processes or are used for accessing data channels like shared registers and memories. It is obvious that these signals must not be considered during hardware basic block identification. For the given example, only signals $\gamma[k+1]$ to $\gamma[n]$ must be taken into account. Exceptions are

signals that are used for enabling external data channels. These signals must be considered in the same way as register-enable signals.

Consider Only Relevant Signals/Conditions

Not all signals i. e., bits of the output symbol are required for performing the data path's operation. In a specific state, only a subset of the control signals is driven with values depending on the output conditions. All other signals are driven with their default values. For register-enable signals that is, that the register is disabled and for mux-select signals the default input is selected.

These signals are considered to have no influence – neither on the behaviour nor on the power dissipation in the particular state. During basic block identification, signals driven with their default values can be ignored. Thus, only control signals driven in the given state are considered to be relevant and only these are used during identification and characterisation.

Predicate Reduction

After knowing the signals to be considered during hardware basic block identification, the output conditions, required for generating that signals are also known. These output conditions typically share some predicates and thus Boolean atoms. As mentioned earlier, for all possible Boolean assignments to the atoms, a hardware basic block must be identified. That is, the number of hardware basic blocks highly depends on the number of predicates that must be considered.

For Boolean-valued input signals and thus Boolean-valued predicates a simplification can be made. If the set of SMT predicates contains predicates $(\sigma[i] = \text{true})$ as well as $(\sigma[i] = \text{false})$, for each of them a Boolean atom is introduced during Boolean abstraction, as shown in Equation (4.17).

$$(\sigma[i] = \text{true}) \mapsto a \qquad \text{and} \qquad (\sigma[i] = \text{false}) \mapsto b \tag{4.17}$$

For these two Boolean atoms, a total amount of four Boolean assignments is possible. However, some statements are redundant, unnecessarily increasing the identification effort. This can be prevented by considering the predicate $\sigma[i] = \text{false}$ to be $\sigma[i] = \text{true}$ and invert the associated Boolean assignment. That is, assignments to the second Boolean atom can be replaced by assignments to the first one. In doing so, the second Boolean atom can be discarded. The transformation is shown exemplary in Equations (4.18) and (4.19). Using this simplification, the number of possible Boolean assignments that must be considered during hardware basic block identification is halved for each Boolean-valued predicated that can be discarded.

$$b = \text{true} \;\Leftrightarrow\; (\sigma[i] = \text{false}) = \text{true} \;\Leftrightarrow\; (\sigma[i] = \text{true}) = \text{false} \;\Leftrightarrow\; a = \text{false} \tag{4.18}$$

$$b = \text{false} \;\Leftrightarrow\; (\sigma[i] = \text{false}) = \text{false} \;\Leftrightarrow\; (\sigma[i] = \text{true}) = \text{true} \;\Leftrightarrow\; a = \text{true} \tag{4.19}$$

Zero-Strength Hardware Basic Blocks

Zero-strength hardware basic blocks occur, if no behaviour needs to be executed for the given assignment to the output conditions. That is, the hardware basic block does not contain any active RT components. In particular, it does not contain any activated target registers. This is the case, if the controller is performing the hand-shake protocol, for example.

If the controller only performs the hand-shake protocol, no register enable signal is driven. This case may also occur, if the given Boolean assignment set does not activate any target registers, as shown in Table 4.1. In this case, no RT component is assumed to be active and thus no power dissipation of the data path must be considered. Since zero-strength hardware basic blocks do not perform any operation, they are ignored during basic block identification.

Remember Required Assignments

Even if several signals are driven by the controller in a specific state, not all signals are required for identifying the associated basic block. This is due to dependencies between the signals. Figure 4.8 shows a simple data path with hierarchical organised multiplexers.

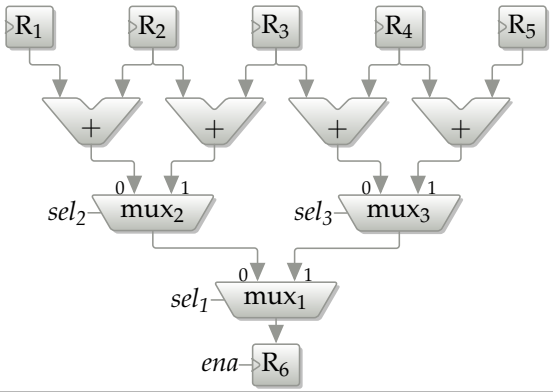


Figure 4.8: Required assignment identification — The evaluated mux-select signals and thus the required Boolean assignments may depend on other mux-select signals and are thus known after hardware basic block identification, only. The value of the mux-select signal of mux_2 is only required, if mux_1 selects input 0, whereas the select signal of mux_3 must only be evaluated, if mux_1 selects input 1.

The value of the mux-select signal `sel1` determines whether mux-select signal `sel2` or `sel3` is required for identifying the corresponding hardware basic block. Table 4.1 lists all hardware basic blocks that could possibly identified for the given data path.

The table shows that there are some zero-strength hardware basic blocks. It is obvious that there is no operation performed, if the register is not enabled. It is also visible, that the operation of the data path does not necessarily depend on all mux-select signals. As mentioned, the required mux-select signals depend on the value of mux-select signal `sel1`. Ignored signal values are denoted in grey in the table. Unfortunately, the required signals are only available after identification of the hardware basic block.

#	ena	sel1	sel2	sel3	operation	note
1	0	0	0	0	–	zero-strength
⋮	⋮	⋮	⋮	⋮	⋮	⋮
8	0	1	1	1	–	zero-strength
9	1	0	0	0	$R_6 = R_1 + R_2$	new one
10	1	0	0	1	$R_6 = R_1 + R_2$	copy of 9
11	1	0	1	0	$R_6 = R_2 + R_3$	new one
12	1	0	1	1	$R_6 = R_2 + R_3$	copy of 11
13	1	1	0	0	$R_6 = R_3 + R_4$	new one
14	1	1	0	1	$R_6 = R_4 + R_5$	new one
15	1	1	1	0	$R_6 = R_3 + R_4$	copy of 13
16	1	1	1	1	$R_6 = R_4 + R_5$	copy of 14

Table 4.1: Required assignments — Boolean assignments that are not considered during hardware basic block identification are printed in grey. Hardware basic blocks one to eight must not be identified, since no target register is activated at all.

For each identified hardware basic block, its required control signals are known. Before identifying a hardware basic block for a new Boolean assignment, all driven signal values are obtained by evaluating all relevant output conditions. If there already exists a hardware basic block for a subset of the signals with the given values, the new hardware basic block will be a copy of the existing one. In this case, no identification must be performed for the given Boolean assignment.

Discard Duplicate Hardware Basic Blocks

Due to the structure and complexity of the output conditions it might occur that different Boolean assignments, which are applied to the output conditions, activate the same behaviour of the data path. In this case, only one copy of the hardware basic block must be characterised and used during model generation. The equivalence of two hardware basic blocks is given, if they enable the same target registers and configure the multiplexers in the same way. In this case, both hardware basic blocks have the same active paths and thus perform the same behaviour.

$$\begin{aligned} H_a = H_b \Leftrightarrow \Big(& \forall_{r \in V_{\mathrm{TR}}^{(s,a)}} \Big(r \in V_{\mathrm{TR}}^{(s,a)}(H_a) \Leftrightarrow r \in V_{\mathrm{TR}}^{(s,a)}(H_b) \Big) \wedge \\ & \forall_{v \in V_{\mathrm{M}}} (v \in V_{\mathrm{M}}(H_a) \Leftrightarrow v \in V_{\mathrm{M}}(H_b)) \wedge \\ & \forall_{v \in V_{\mathrm{M}}} (v \in V_{\mathrm{M}}(H_a) \Rightarrow (\mathrm{select}(H_a, v) = \mathrm{select}(H_b, v))) \Big) \end{aligned} \tag{4.20}$$

After a new hardware basic block has been identified, it is checked, if another hardware basic block performing the same behaviour is already known. Is this the case, the Boolean assignment set, enabling the second hardware basic block is added as enabling assignments set to the first one and the second basic block is discarded. Since the characterisation should be performed considering data-dependencies implicitly, only hardware basic blocks from the same state are checked for equivalence. Hardware basic blocks from other state might

have the same behaviour in principal, but since they are activated in a different context, their data-dependent power dissipation might be different. Thus, hardware basic blocks from different states cannot be considered to be equal, even if they perform the same behaviour.

4.4.4 Special Cases

There are some special hardware basic blocks that must be identified or that might occur during hardware basic block identification, which deserve special attention.

Assignment Hardware Basic Blocks

In order to obtain the assignments to the Boolean atoms, which in turn will compute the controller's output symbol, some information from the data path is required. This are register values, for example. But not only values stored in registers are required. Some intermediate results might be required, too. Such intermediate results check, if a specific register holds a specific value or if an operator has computed a specific result, for example. To be able to evaluate the output conditions, these intermediate values must be known. In contrast to a register, whose value is available all the time, an intermediate result must be recomputed, each time its value is required.

The points in the data path at which the controller accesses these intermediate results are called *assignments*. They are not to be confused with the Boolean assignments to the Boolean expressions, required for evaluating the controller's output conditions and thus the controller's output symbol.

Each state of the controller has some *assignment hardware basic blocks* associated, computing all controller inputs required for evaluating all relevant output conditions of that particular state. Figure 4.9 shows how assignment and conventional hardware basic blocks are related. Since the input to the controller might depend on different mux-select values, it is possible that a state has multiple assignment hardware basic blocks associated.

Invalid Hardware Basic Blocks

Invalid hardware basic blocks are generated, if the basic block identification tries to find a hardware basic block for a Boolean assignment to the output conditions that cannot occur. Figure 4.10 shows this relationship.

The basic block identification process presented above, tries to identify a hardware basic block for each possible Boolean assignment to both output conditions `c1` and `c2`. The analysis does not consider how the inputs `input1` and `input2` are obtained, since this is part of the data path. It can thus not determine that the assignment (`input1=true`, `input2=true`) is not possible and therefore can never occur. Identifying all valid input values of all input signals is an ambitious task, requiring a sophisticated and very time consuming analysis of the data path. This is not possible during characterisation. Because it is not possible to determine all possibly occurring input values and thus all possible Boolean assignments, a hardware basic block for each assignment is generated.

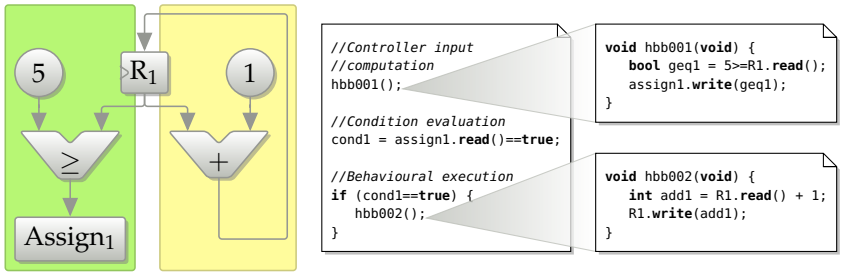
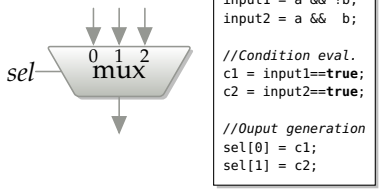


Figure 4.9: Assignment hardware basic block — Assignment hardware basic blocks are required for obtaining intermediate values that are necessary for determining the control flow. That is, they are required for evaluating the controller's output conditions. In contrast to a register, whose value is available all the time, an intermediate result must be recomputed, each time its source registers is updated or some of its required multiplexers selects a different input port.



condition		mux	required input value	
c1	c2	sel	a	b
0	0	0	false	false OR true
0	1	1	true	false
1	0	2	true	true
1	1	3	true	false AND true

Required input values

Figure 4.10: Emergence of invalid hardware basic blocks — Hardware basic block identification tries to identify a hardware basic block for all possible Boolean assignments to output conditions `c1` and `c2`. But it cannot determine that the combination of input values yielding Boolean assignment (`input1=true`, `input2=true`) is not possible and thus will never occur. Identifying a hardware basic block for that assignment is not feasible, since the multiplexer does not have an input 3. The hardware basic block for that assignment is considered to be invalid and therefore is discarded.

During basic block identification for the Boolean assignment (`input1=true`, `input2=true`) it is not possible to determine the correct input for the multiplexer, since it does not have four inputs. The hardware basic block is thus invalid and is discarded.

It might occur that a hardware basic block can be identified for an invalid assignment i. e., the error is not detected during identification. In this case the hardware basic block is generated appropriately during model generation, but the generated hardware basic block will never be executed during simulation. Thus, identifying invalid hardware basic blocks does slow down the characterisation process but has no effect on the correctness or the accuracy of the generated model.

4.4.5 Hardware Basic Block Characterisation

After all hardware basic blocks had been identified, their power dissipation and timing can be characterised. In order to be able to consider different power modes i. e., different combinations of supply voltage and clock frequency, using dissipated energy as characterising property is not sufficient. Instead, a more structure-related metric is chosen. It separates parameters like supply voltage, which may vary during simulation, from architectural properties like capacitance, which are fixed after synthesis. This separation approach is shown in Equation (4.21).

$$\begin{aligned} E_\mathrm{d} &= f(H, V_\mathrm{dd}) \\ &\approx f_1(H) \cdot f_2(V_\mathrm{dd}) \\ &= C_\mathrm{l}(H) \cdot V_\mathrm{dd}^2 \end{aligned} \tag{4.21}$$

Terms $f_1(H)$ and $f_2(V_\mathrm{dd})$ are independent from each other. It must be noted that $f_1(H)$ is a linear factor and $f_2(V_\mathrm{dd})$ is the same for all processes. This allows a hardware basic block H to be characterised by the capacitance C_l it switches when activated.

The second property needed for representing a hardware basic block is the number of clock cycles required for its execution. Since the basic blocks are generated from an internal cycle-accurate model, their timing behaviour can also be characterised with the same accuracy. The hardware basic blocks that are identified as described in Section 4.4.2 require exactly one clock cycle to perform their operations. However, hardware basic blocks covering multiple cycles are conceivable, as shown in Section 7.1. Having cycle-accurate hardware basic blocks available, the temporal resolution of the generated model will be t_clk i. e., one clock period.

Simple Capacitance Characterisation

As just mentioned, hardware basic block characterisation is done in terms of switched capacitance and clock cycle count. Also mentioned was the fact, that each hardware basic block requires exactly one clock cycle to be computed, since it is a spatial abstraction. Characterisation is done as follows.

The power dissipation of each RT component is given by its RT-level power model. Various techniques for estimating power dissipation at RTL are outlined in Section 3.1. Different

RT-level power models can be used for characterisation. Currently, the PowerOpt-internal model is used, allowing a comparison of the results of the proposed approach to the estimates of PowerOpt. This comparison is done in Section 6.2. The PowerOpt-internal power model computes dissipated dynamic energy E_d with respect to the RT component's input patterns. These are obtained from a functional simulation. Equation (4.22) shows how the average switched capacitance is obtained using the RT-level model. By characterising a hardware basic block using its switched capacitance, requirement 4 from the beginning of this chapter is satisfied.

$$C_\mathrm{l}(H) = \frac{1}{V_\mathrm{dd}^2} \sum_{i=1}^{\#_\mathrm{comp}(H)} \frac{1}{\#_\mathrm{pat}(v) - 1} \sum_{j=2}^{\#_\mathrm{pat}(v_i)} E_\mathrm{d}\left(v_i, pat_{j-1}, pat_j\right) \tag{4.22}$$

The number of RT components in the given hardware basic block H is denoted by $\#_\mathrm{comp}(H)$. The number of input patterns that had been applied to RT component $v_i \in H$ is given by $\#_\mathrm{pat}(v_i)$. The RT-level power model then computes the dynamic energy dissipating, if the pattern at t_{j-1} and t_j are applied to v_i consecutively.

Since the actual data patterns are used, the characterisation is implicitly data dependent, satisfying requirement 5 from this chapter's introduction. Considering the data dependency requires a representative data set to be used for characterisation. So far, a hardware basic block describes the data dependency related to the control flow, but not the data-dependent activity for each individual RT component.

Obtaining the Number of Activations Knowing the timestamps, a given RT component is active is key to the simple characterisation process. As mentioned, the pattern lists are used for obtaining average switched capacitance per activation of an RT component. Actually, the concrete pattern lists are not known. Instead, a value change sequence for each RT component is known. This sequence is loaded from a *value change dump (VCD)* file, generated by the RT-level simulation tool during power estimation. The sequence consists of timestamp/value-tuples, each one denoting the point in time, a specific signal changes its value and the new value it has from that time on.

If an operator performs the same operation twice i. e., the same data patterns are applied twice in succession, this cannot be seen from the value change sequence list. In this case, the second activation is missed, since no value change had occurred. If the patterns are missing on the sequence, less activations are assumed than actually occurred. This causes the average power dissipation per activation to be higher, in the second place.

In order to cope with this problem, an algorithm has been developed that is able to obtain the exact number of times, a given RT component is active. The algorithm assumes an RT component to be activated either if its output value is stored in a register or if the component is generating an output due to parasitic functionality.

The times the output values of an RT component are stored can be obtained by starting from the RT component and traversing the data path downwards until all target registers of that component had been found. The times the RT component is used is then the joint set of all times, all target registers are enabled. Timestamps are considered with respect to the select values of all affected multiplexers. That is, only timestamps at which the multiplexers forward

the output value from the RT component are taken into account. For the example data path shown in Figure 4.11, this is done using Equation (4.23).

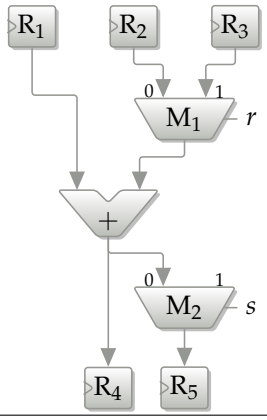


Figure 4.11: Obtaining times an RT component is active

$$t_{\text{used}} = t_{\text{enable}}(R_4) \cup (t_{\text{enable}}(R_5) \cap t_{s=0}(M_2)) \tag{4.23}$$

$$t_{\text{output}} = t_{\text{enable}+t_{\text{clk}}}(R_1) \cup \left(t_{\text{enable}+t_{\text{clk}}}(R_2) \cap t_{r=0}(M_1)\right) \cup \left(t_{\text{enable}+t_{\text{clk}}}(R_3) \cap t_{r=1}(M_1)\right) \tag{4.24}$$

$$t_{\text{active}} = t_{\text{used}} \cup t_{\text{output}} \tag{4.25}$$

$$\#_{\text{activations}} = |t_{\text{active}}| \tag{4.26}$$

However, this does not take parasitic functionality into account. That is, an RT component might be active more often than its output value is required. In order to consider this too, the data path is traversed upwards, until all source registers are found. Again, this is done with respect to the affected multiplexer's select values. At all times at which new values from the source registers are forwarded to the RT component, it is assumed to generate an output value and thus to be active. A source register provides its new output value exactly one clock cycle after it has been enabled. For the given example, this set of timestamps is obtained by using Equation (4.24).

The set of timestamps the RT component is active then is the joint set of used and output timestamps, as stated by Equation (4.25). There is typically a large overlap between both sets. A special case and exception are multiplexers, which select the default channel when not used. They become active significantly more often, than their results are used. This is especially true if there is a lot of activity on the default input, while the results are discarded after they passed the multiplexer. Having the correct number of activations available from Equation (4.26), Equation (4.22) on the preceding page can be modified to obtain the correct value, as done by Equation (4.27).

$$C_{\text{l}}(H) = \frac{1}{V_{\text{dd}}^2} \sum_{i=1}^{\#_{\text{comp}}(H)} \frac{1}{\#_{\text{activations}}(v)} \sum_{j=2}^{\#_{\text{pat}}(v_i)} E_{\text{d}}\left(v_i, pat_{j-1}, pat_j\right) \tag{4.27}$$

Scaling Factor for Operators and Multiplexers Even though Equations (4.23) to (4.26) provide a good way for obtaining the correct number of activations, the result will not be correct in all cases. Especially if communication between processes takes place or if special resolution logic e. g., for arbitrating memory access from multiple processes, is inserted into the data path determining if and when an input port is written is not always possible. In this

case, the assumed number of activations is obtained from the value-change-sequence. This will lead to an overestimation of the dissipated energy, as described above.

Another obstacle arises from the simulation of assignment hardware basic blocks. As mentioned earlier, these are required for computing intermediate results, required as inputs to the controller's SMT predicates of the output conditions. Section 5.4.2 explains that the assignment hardware basic blocks are computed during delta cycles and therefore are possibly computed multiple times within the simulation of one clock cycle. Furthermore, there typically exists a comparatively large overlap between the assignment hardware basic blocks and the behavioural basic block, executed in the same cycle.

To avoid double-counting of the capacitance switched by RT components belonging to an assignment as well as to a behavioural basic block, only the later one is taken into consideration. Regardless of the overlap there might be some RT components only belonging to the assignment hardware basic block. Such an example is given in Figure 4.9 on page 83. Not considering such components during power estimation will lead to an underestimation.

A scaling factor k_C^s is introduced for coping with the over- and underestimation just mentioned. It is applied to the switched capacitance of operators and multiplexers, respectively. Registers must not be taken into account since the number of their activations can be obtained exactly. They also do not belong to any assignment hardware basic block. This scaling factor can be easily obtained by performing an estimation using BAC++ with a scaling factor of 1.00 with the results from the estimation using PowerOpt and computing the relation between both. This is shown in Equation (4.28). Of course, this is done with respect to operator and multiplexer power, only. A scaling factor $k_C^s < 1$ typically indicates designs with a lot inter-process communication, while a factor $k_C^s > 1$ is an indicator for a complex output- and next-state logic. This is also visible from the detailed evaluation results in Appendix D.

$$k_C^s = \frac{C_1^{\text{mux\&op, PowerOpt}}}{C_1^{\text{mux\&op, BAC++ (simple)}}} \tag{4.28}$$

Advanced Capacitance Characterisation

Evaluating the simple characterisation approach, as done in Section 6.2.1, shows that the simple characterisation is not effective enough i. e., will give a relatively high estimation error. This is especially true for the error-over-time i. e., the relative error per clock cycle. This is because the simple characterisation considers *all* data patterns that are applied to the RT component while characterising the hardware basic block, the component belongs to. Since the high-level synthesis may use resource-sharing and thus bind several operations to a single RT component, the data patterns of the particular component may belong to different hardware basic blocks. Different hardware basic blocks in turn may use the shared RT component differently. This results in a varying dynamic power dissipation, depending on the context i. e., the basic block the component is used in. In other words, the power dissipation of a given RT component at a given time depends on the context of the operation.

Thus, a more sophisticated characterisation approach considers only the data patterns belonging to the particular hardware basic block under consideration. An *unbinding step* separates the pattern list of the RT component. In other words, a separate pattern list for each hardware

basic block the RT components belongs to, is created. This is shown in Figure 4.12. These pattern lists are then used during basic block characterisation.

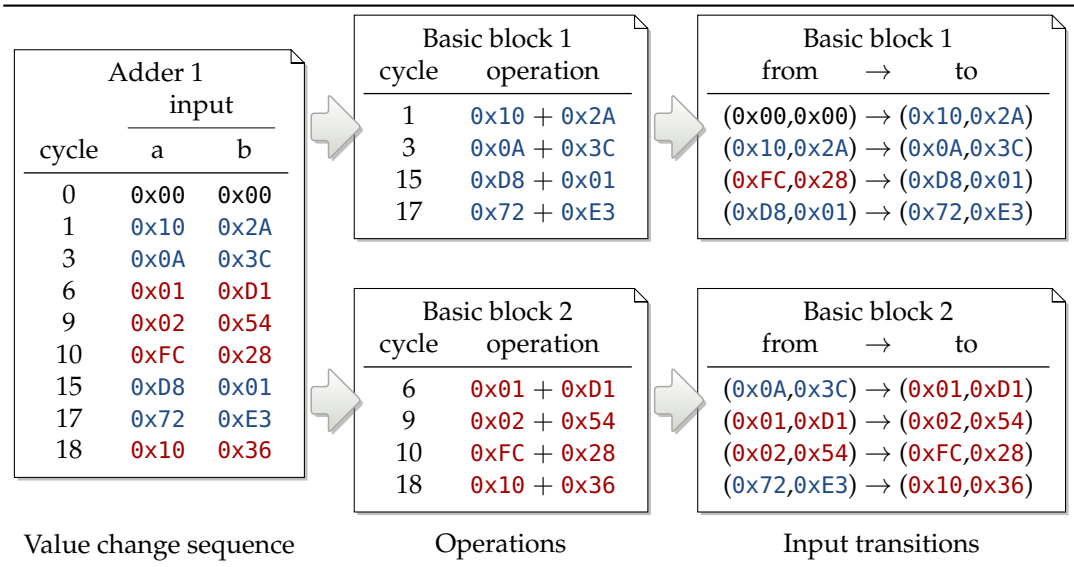


Figure 4.12: Unbinding of operations — The value change sequence of a given RT component must be split in order to assign individual operations to the corresponding hardware basic blocks. During characterisation of the basic blocks, input transitions are taken into account. The current pattern always belongs to the actual basic block. However, the previous patter must be taken from the value change sequence, since it might belong to another hardware basic block. For better readability, data patterns belonging to the same basic block are encoded in the same colour. Cycle zero denotes the initial pattern applied to the component, if the simulation starts.

Equation (4.27) from the simple characterisation approach on page 86 can be modified to consider only input patterns belonging to the actual hardware basic block. A delta-function shown in Equation (4.29) determines whether a data pattern must be considered, or not.

$$\delta(H,t) \mapsto \{0,1\}: \begin{cases} 1, & \text{if } t \in \text{activetimes}(H) \\ 0, & \text{else} \end{cases} \tag{4.29}$$

Since the RT-level power estimation does not consider individual data patterns but transitions between them, a pair of data patterns must be considered. Such a pair from the pattern list is only considered, if the second one belongs to the hardware basic block. This reflects the fact that the input of the RT component switches from pattern pat_{j-1} to pat_j during execution of the particular hardware basic block. It is important to note, that both patterns do not necessarily belong to the same basic block. On the contrary – it is very likely that both patterns belong to different basic blocks. This is depicted in the right tables of Figure 4.12.

The modified characterisation is given by Equation (4.30). Again, characterising a hardware basic block using its switched capacitance satisfies requirement 4, which has been identified

at the beginning of this chapter.

$$C_{\mathrm{l}}(H) = \frac{1}{V_{\mathrm{dd}}^2} \sum_{i=1}^{\#_{\mathrm{comp}}(H)} \frac{1}{\#_{\mathrm{activetimes}}(H)} \sum_{j=2}^{\#_{\mathrm{pat}}(v_i)} \delta\big(H, t\big(pat_j\big)\big) E_{\mathrm{d}}\big(v_i, pat_{j-1}, pat_j\big) \tag{4.30}$$

Obtaining the Number of Activations As for the simple characterisation approach, the number of activations must be known. But in contrast to the simple characterisation, not activations of individual RT components but activations of the complete hardware basic block must be considered.

By analysing the controller's FSM and its corresponding data path along with the pattern lists obtained from a functional simulation, it becomes possible to determine the timestamps a particular hardware basic block was active. This step requires a lot of computational effort. However, the more precise this identification is, the more accurate the power estimation results will be. In contrast, with a less precise timestamp identification, power estimation results will converge to the simple characterisation approach, which considers all timestamps during basic block characterisation.

A hardware basic block is assumed to be active at a specific timestamp, if all of its target registers have their enable-signals set at that timestamp. At the same timestamp, all other registers of the process must not be enabled, as defined by Equation (4.31). Related mux-select signals can also be taken into account during identification of active timestamps of a hardware basic blocks. In this case it is checked, whether all mux-select signals have the value as during identification of the basic block. This will increase the accuracy of the characterisation for cases where two distinct basic blocks have the same set of target registers, but activate different parts of the data path. Experiments have shown that the improvement in estimation accuracy is not worth the additional effort during characterisation.

$$active(H,t) \Leftrightarrow \forall_{r \in V_{\mathrm{R}}} \Big(\Big(r \in V_{\mathrm{TR}}^{(s,a)} \Leftrightarrow r \text{ is enabled} \Big) \wedge \Big(r \notin V_{\mathrm{TR}}^{(s,a)} \Leftrightarrow r \text{ is not enabled} \Big) \Big) \tag{4.31}$$

Scaling Factor for Operators and Multiplexers Like for the simple characterisation approach, the number of activations cannot always be obtained correctly. This is caused by incomplete value change sequences for data ports, for example. In rare cases it might also occur that two distinct hardware basic block have the same set of target registers, but different mux-select configurations. Since the active times of a basic block are determined based on the target registers only, it is assumed that a basic block is more often active than it really is. In contrast to the simple estimation approach, where a wrong number of activations yields an under- or overestimation, for the advanced estimation approach it yields a fuzzy result. If more activations than actually occurred are assumed, data patterns, belonging to different hardware basic blocks are taken into consideration, too. These additionally considered patterns reduce the accuracy of the estimation. This error must be corrected.

Therefore, a scaling factor k_C^{adv} is introduced, allowing to adjust the power estimation. Again, this factor is computed by comparing the estimation results of BAC++ with the ones obtained from an estimation using PowerOpt after the design had been characterised. This is shown in Equation (4.32) on the next page. Of course, this scaling factor is only applied to the switched

capacitance estimated for multiplexers and functional operators. Switched capacitances of registers etc. are left unchanged.

$$k_C^{\text{adv}} = \frac{C_l^{\text{mux\&op, PowerOpt}}}{C_l^{\text{mux\&op, BAC++ (advanced)}}} \tag{4.32}$$

Contingency Option It might occur that for certain hardware basic blocks no active time-stamps and thus no data patterns are available at all. That is, the particular basic block is never activated by the use case, employed during characterisation. If no data patterns for the specific basic block are available, the advanced characterisation cannot be performed. In this case, the simple characterisation approach is used as contingency option.

4.4.6 Data Channels

Another source of dynamic power dissipation that must be considered are data channels, such as memories or shared registers. In this thesis only a very basic characterisation is used, since main focus is on the afore mentioned hardware basic blocks. For *memories* it is assumed that activation takes place whenever the memory block is enabled. That is, at times when the chip-enable signal is low. In the current implementation, no difference between read and write accesses is made. Equation (4.33) assumes that the same capacitance is switched, if the memory is accessed, independent from the type of access. This is a sufficient approximation, if it is assumed that the relation between read and write accesses is the same for all use cases. Like before, total dissipated energy is obtained from the RT-level power model.

$$C_l(M) = \frac{1}{V_{dd}^2} \frac{1}{\#_{\text{activations}}(M)} E_d^{\text{total}}(M) \tag{4.33}$$

For *shared registers* the characterisation is straight forward. A shared register is assumed to be written, if its enable-signal is set by one of the processes sharing the register. Like for the simple hardware basic block characterisation approach, it is assumed that each write-access to the register switches the same capacitance, as shown in Equation (4.34). Again, total dissipated energy is provided by the RT-level power model.

$$C_l(R_{\text{shared}}) = \frac{1}{V_{dd}^2} \frac{1}{\#_{\text{activations}}(R_{\text{shared}})} E_d^{\text{total}}(R_{\text{shared}}) \tag{4.34}$$

4.5 Non-Functional Properties

Non-functional properties are not directly related to the functionality and the behaviour of the design. Instead, they are introduced during high-level synthesis, in order to implement the design. In other words, they are a necessary overhead that is required for implementing the intended behaviour.

While dynamic power dissipation of the data path is characterised per hardware basic block, dynamic power dissipation due to non-functional properties is characterised per process. This is especially true for all sources of power dissipation that are active in each clock cycle, including clock and controller power. Again, physical parameters like supply voltage are separated from structural properties like capacitance. This is shown in Equation (4.35).

$$\begin{aligned} E_{\mathrm{d}} &= f(p, V_{\mathrm{dd}}) \\ &\approx f_1(p) \cdot f_2(V_{\mathrm{dd}}) \\ &= C_{\mathrm{l}}(p) \cdot V_{\mathrm{dd}}^2 \end{aligned} \tag{4.35}$$

The total capacitance that is switched by a process p in each clock cycle is built from various sources, as shown in Equation (4.36). Each one of them is described in more detail in one of the following sections.

$$C_{\mathrm{l}}(p) = C_{\mathrm{l}}(ctrl) + C_{\mathrm{l}}(clk) + C_{\mathrm{l}}(net) \tag{4.36}$$

Another important non-functional property is static power dissipation, whose characterisation is done in terms of a conductance instead of switched capacitance. Finally, so-called idle-power must be characterised, allowing to perform an estimation of the module's power dissipation during idle-phases. That is, if no behavioural simulation of the module takes place. These sources of power dissipation also discussed in the following sections.

4.5.1 Controller Power

Besides RT components directly related to the functional behaviour of the hardware module, there are components required for controlling the behaviour i. e., for controlling the RT data path of the module. As can be seen in Figure 4.5 on page 67, the *controller* consists of a state register and output as well next-state logic, which are active in each clock cycle.

Even though the controller is relatively small compared to the data path [145], it must be regarded during power estimation. Due to its small influence on the total power dissipation, its dynamic power dissipation can be assumed to be the same in each clock cycle. It thus can be averaged over all cycles, yielding Equation (4.37). Total energy dissipated by the controller is given by the RT-level power model. Keeping individual activities for each state could be easily included in the model, but experiments have shown that an overall average is acceptable. More details on this are given in Section 6.2.

$$C_{\mathrm{l}}(ctrl) = \frac{1}{V_{\mathrm{dd}}^2 \cdot \#_{\mathrm{cyc}}} E_{\mathrm{d}}^{\mathrm{total}}(ctrl) \tag{4.37}$$

4.5.2 Clock Power

Probably the most active part of a hardware module is the clock. The clock or its distribution net also called the clock-tree to be more precise, switches every clock cycle. Even worse, the clock-tree performs two complete transitions within one single clock cycle. That is, at a rising and at a falling clock edge. Assuming that no power saving techniques like clock gating are

applied, the clock's power dissipation is obviously very regular and is thus the same for all cycles. Clock power is characterised using Equation (4.38), in which total energy dissipation of the clock net is provided by the RT-level power model. Even though clock gating is not directly supported by PowerOpt, it supports clock-gating-aware RT components such as registers. However, their modified power dissipation is part of their particular RT-level power model and thus must not be considered separately.

$$C_{\mathrm{l}}(clk) = \frac{1}{V_{\mathrm{dd}}^2 \cdot \#_{\mathrm{cyc}}} E_{\mathrm{d}}^{\mathrm{total}}(clk) \tag{4.38}$$

4.5.3 Interconnect Power

While PowerOpt internally provides a cycle-accurate model for power dissipation due to switched interconnect capacitances, it uses the all-cycles average value during power trace generation. In order to allow the power & timing model creating a power trace, which is comparable to the power trace directly obtained from PowerOpt, the interconnect power is also characterised using the average value, as shown in Equation (4.39). If a more precise power-over-time information regarding the interconnect power is required, this can be implemented easily.

$$C_{\mathrm{l}}(net) = \frac{1}{V_{\mathrm{dd}}^2 \cdot \#_{\mathrm{cyc}}} E_{\mathrm{d}}^{\mathrm{total}}(net) \tag{4.39}$$

4.5.4 Static Power

As stated in Section 2.4.2, static power dissipation can be considered to be data-independent for larger parts of the design. The processes created during high-level synthesis can be considered to be such large parts. As shown in Appendix C, a typical process comprises several hundred RT components with thousands of transistors.

If static power dissipation is considered to be data independent, it can be considered to be a synthesis artefact, depending on the semiconductor technology used for synthesis. Common estimation approaches use transistor count or gate-equivalents for modelling static power, since they give a good first approximation if no detailed information is available. However, in this thesis static power dissipation is characterised by the internal equivalent conductance G, which is a complex function of voltage and temperature, as well as synthesis and process parameters [73]. Thus, a nominal value, obtained during characterisation is used for modelling the equivalent conductance. This nominal value can be scaled appropriately during estimation, as described in Section 5.6.

Characterisation of static power dissipation is done analogously to the characterisation of hardware basic blocks, even though it is more complex [73]. Again, structural properties like conductance can be separated from parameters like supply voltage. But a closer look reveals that conductance itself also depends on the applied supply voltage or temperature, for example. This fact is regarded by the dimensionless function $g(V_{\mathrm{dd}}, T)$, which scales the

nominal conductance G with respect to the aforementioned parameters.

$$\begin{aligned} P_\mathrm{l} &= f(p, V_\mathrm{dd}, T) \\ &\approx f_1(p) \cdot f_2(V_\mathrm{dd}) \cdot g(V_\mathrm{dd}, T) \\ &= G(p) \cdot V_\mathrm{dd}^2 \cdot g(V_\mathrm{dd}, T) \end{aligned} \tag{4.40}$$

The conductance of a given process p is the sum of its local conductance and the conductance of all its sub-processes, as shown in Equation (4.41).

$$G(p) = G_\mathrm{local}(p) + \sum_{i=1}^{\#_\mathrm{children}(p)} G(\mathrm{child}_i(p)) \tag{4.41}$$

The local conductance itself is the sum of the conductance of all RT components v_i belonging to the given process p and can be computed using Equation (4.42). The static power dissipation of a single RT component is available from the PowerOpt internal RT-level power models.

$$G_\mathrm{local}(p) = \frac{1}{V_\mathrm{dd}^2} \sum_{i=1}^{\#_\mathrm{comp}(p)} P_\mathrm{l}(v_i) \tag{4.42}$$

4.5.5 Idle Power

A special case must be considered, if the module under consideration is embedded in a larger system. In this case, it may occur that the module is idle for a longer time e. g., between two calls or activations of the module. In this case, not all idle cycles will be simulated individually. On contrary, there is no functional simulation at all, as explained in Section 5.3. In order to still obtain the power dissipation of the module, a so-called *idle power* is provided. The idle power of the module is computed once, if the module enters its idle state. The trace generation then assumes that this power will be correct until a new power value is set. This is the case if the next activation of the module occurs or if a new supply voltage is applied to the module.

During idle-phases of the module, each of the included controllers waits for its surrounding environment to perform the hand-shake protocol. That is, for each process, the state register keeps its value, the output logic does not activate the data path, and the next state logic only observes the input signals, required for performing the hand-shake protocol. Summarising, the data path can be considered to be dormant and the output- and next-state-logic are virtually inactive. The only remaining source of dynamic power dissipation is the clock-tree. However, static power dissipation must be considered as usual during idle-phases.

Average switched capacitance during idle-phases of the process is characterises very similar to the static power of the modules, as described in Section 4.5.4. It can be obtained by Equation (4.43), shown below.

$$C_\mathrm{l}^\mathrm{idle}(p) = C_\mathrm{l}(clk) + \sum_{i=1}^{\#_\mathrm{children}(p)} C_\mathrm{l}^\mathrm{idle}(\mathrm{child}_i(p)) \tag{4.43}$$

4.6 Power Modes

It is possible that a hardware module can operate in different modes, so-called *power modes*. A power mode is a set of system properties. In particular it is a tuple, containing the actual supply voltage and clock frequency, the module will operate at while the particular power mode is active. Using so-called *power mode transitions*, it is possible to switch from one power mode to another one. Due to its strong effects on the module's behaviour, a power mode transition can only take place in specific phases of the module's behaviour. For full-custom hardware such a transition can take place if the module has finished its actual task i. e., if the module becomes idle.

Power mode transitions are controlled, or requested to be more precise, by the global power management. That is, a system-wide power manager decides in which power mode a specific module has to operate in. This decision is based on a so-called *power management policy*. During synthesis of the hardware module, a so-called *power mode table* is derived, which contains all power modes that are supported by the module, as well as all allowed power mode transitions. This table can then be used for deriving a suitable power management policy for the overall system.

As described in more detail in Section 5.7, a global power manager requests the local power manager to perform a power mode transition. Such a power mode transition is a change of supply voltage and clock frequency, where the former is of particular interest during characterisation. Changing the supply voltage to a higher or lower electrical potential causes the capacitances inside the module to partially charge and discharge, respectively. Of course, charging and discharging capacitances requires some time. Thus, a power mode transition introduces a penalty in terms of power and delay, which must be considered during estimation and functional simulation.

A special case of a power mode is the application of power gating, which is described in Section 2.4.4. This technique allows switching off a hardware module completely. By disconnecting the design from the supply voltage, static power can be reduced. If a module is power gated, its internal state i. e., the content of its registers and memories, is lost typically. Special state retention registers can be used to prevent this. While the module is power gated, all internal capacitances are discharged slowly, due to leakage currents. If the module is re-activated after being power gated, all discharged capacitances must be charged again and all internal signals must reach a steady state. Thus, a power mode transition from an off- to an on-mode dissipates significantly more energy than a transition between two on-modes.

4.6.1 Power Mode Identification

The supply voltage of a SoC is often defined externally, sometimes even external to the chip. From a constant supply voltage source, a DC/DC-converter creates the desired voltages. The delay for switching the supply voltage is typically in the order of tens of microseconds. Recent research has developed methods for applying *dynamic voltage/frequency scaling (DVFS)* using on chip voltage regulators, allowing the supply voltage to be scaled in the order of tens of nanoseconds [51]. The clock frequency is modified in a very similar way. An oscillator,

providing a given frequency is also typically located externally to the chip. Using clock dividers and multipliers, the desired clock frequency can be generated.

In the design process of the overall chip, available supply voltages are limited by the technology used for synthesis. Available clock frequencies can be defined later, since clock divider and multiplier can be adapted easily. Thus, power modes are identified by specifying the desired supply voltage. Afterwards, PowerOpt's voltage-dependent delay model can be used for obtaining possible frequencies in dependence of the critical path in the generated data path. A power mode is then defined by the given supply voltage and the next lower clock frequency provided by the clock divider/multiplier, which is next to the frequency identified by PowerOpt.

If such a differentiated characterisation is not possible, the *α-power law*, introduced by Sakurai and Newton [118] can be used. The α-power-law models the drain current of MOSFETs which in turn affects the maximal frequency a circuit can operate at. It shows that the delay of a circuit is proportional to a function of supply and threshold voltage [117], as shown in Equation (4.44).

$$\frac{1}{f_{\mathrm{clk}}} \propto \frac{V_{\mathrm{dd}}}{(V_{\mathrm{dd}} - V_{\mathrm{th}})^{\alpha}} \tag{4.44}$$

Knowing this and having a reference point $(V_{\mathrm{ref}}, f_{\mathrm{ref}})$ available from the high-level synthesis, it is possible to obtain new power modes in a first approximation. The required α can be calculated from the technology information. The maximal possible frequency for a given supply voltage V_{dd} can be estimated using Equation (4.45). Result is a new possible power mode $(V_{\mathrm{dd}}, f_{\mathrm{clk}})$.

$$\frac{1}{f_{\mathrm{clk}}} = \frac{(V_{\mathrm{ref}} - V_{\mathrm{th}})^{\alpha}}{f_{\mathrm{ref}} V_{\mathrm{ref}}} \cdot \frac{V_{\mathrm{dd}}}{(V_{\mathrm{dd}} - V_{\mathrm{th}})^{\alpha}} \tag{4.45}$$

Reference voltage and frequency V_{ref} and f_{ref}, respectively have been used during characterisation. Parameters V_{th} and especially α are technology dependent and can be obtained if technology data is available. For recent technologies α seems to converge at 1.3. Generally speaking it can be said that the α-power law is suitable at the working point of a given technology, but not at the borders of the typical parameter interval.

4.6.2 Power Mode Characterisation

As mentioned above, a power mode is characterised by its combination of supply voltage and clock frequency. These two values are suitable for enabling a DVFS-aware functional simulation. However, in order to derive power management policies for the global power manager, some more information is needed. For deriving good power management policies, average dynamic as well as average static power dissipation must be known. These two values can be easily obtained from PowerOpt by performing an RT-level power estimation using the supply voltage and clock frequency, specified by the particular power mode.

4.6.3 Power Mode Transition Characterisation

A characterisation of power mode transitions cannot be done directly using PowerOpt, since it does not provide the required power models. However, characterisation can be done using the models presented by Rosinger [114]. The following sections describe how a characterisation of power mode transitions can be performed.

A power mode transition is performed by applying techniques known from DVFS. An example for such a transition on a system with two different supply voltages is given by Cheng and Baas [39]. First, the design is stalled, in order to save all intermediate results i. e., the registers' content. Second, the power supply is shut off, followed by a short delay. Third, the power supply is switched over and finally the stall is released. In order to simplify the power manager, it is typically assumed that all possible power mode transitions require the same amount of time. The presented power mode model however, is able to cope with different transition times. More details about the power required for enabling and disabling the supply voltage can be found in the work of Rosinger et al. [116].

If the supply voltage is changed for a larger part of the design, the required energy can be obtained by summing up the values for all RT components that belong to the particular area. Such an area is also called a *power island*. For the approach, presented in this thesis, the complete module belongs to a single power island i. e., the supply voltage and clock frequency is the same for all RT components. Basically, there are different types of transitions that can be performed by the power island.

Transitions between On-Modes

The simplest case is a power mode transition between two on-states. During a transition between two on-modes, the internal state of the RT component or even of the entire hardware module remains stable. Charged capacitances must be partially discharged or charged, depending on whether the new power mode has a higher or lower supply voltage. Discharged capacitances remain unaffected. No glitches or hazards occurs during the mode transition.

Transitions between Off-Modes

Transitions between off-modes i. e., between different power-saving or sleep-modes, are very similar to power mode transitions between on-modes. Again, the internal state of the module remains stable, since all capacitances remain unchanged. When performing a transition from a lower to a higher supply voltage, the capacitances are slightly loaded, due to the higher potential difference. This is true, even for a PMOS power gating transistor due to leakage currents, passing through the gate transistor. The same happens vice versa, if a power mode transition to a lower supply voltage is performed. However, more important than the partial charging and discharging of capacitances is the fact that the leakage currents of the module will increase due to the higher supply voltage.

Transitions between Off- and On-Modes

Most interesting are power mode transitions between an off- and an on-mode and vice versa. In the first case the module is enabled. Therefore, not only internal capacitances must be charged, the internal state of the module must also become stable. During the stabilising phase, a lot of glitches and hazards may occur inside the module, causing additional power dissipation. Moreover, the stable internal state that must be reached, depends on the applied input vector, making an exact estimation even more complex. During a transition from an on to an off state, the module is disabled. In this case, all capacitances are partially discharged and the state is lost.

Transition Costs

A transition between power modes causes a penalty in terms of delay and additional power dissipation. For simple RT components, it can be assumed that the power mode transition is performed within one clock cycle. This is also true, if a transition from an off- to an on-mode is performed, as shown by Rosinger [114, sec. 4.2.5]. The delay can be approximated using Equation (4.46). In the equation, τ_{RC} denotes the RC time constant. This is the time required for charging capacitance C through resistor R to about 63.20 % of the difference between initial and final value. The capacitance is assumed to be fully charged after $5\tau_{RC}$, when it is charged to about 99.30 %.

$$\tau_{\text{trans}} \approx 5\tau_{RC} < \frac{1}{f_{\text{clk}}} \qquad \text{with} \qquad \tau_{RC} = RC \tag{4.46}$$

For single RT components, the limiting factor is the power-gating transistor, which separates the RT component from the supply and ground voltage, respectively. For larger structures, the supply grid becomes the limiting factor. If a lot of RT components are enabled at the same time, the supply grid cannot deliver the required electrical current, resulting in an IR-drop. This in turn causes the timing penalty to increase heavily. It can increase from 1 to 5 ns to several milliseconds. Since the delay highly depends on design properties that are not available before a low-level placement and supply grid routing has been performed, any statement at a high level of abstraction is only a first approximation. Equation (4.47) can be used for obtaining the energy, required for performing the power mode transition.

$$E_{\text{trans}} = \frac{1}{2} C_{\text{l}} \left| V_{\text{to}}^2 - V_{\text{from}}^2 \right| k_{\text{trans}} \tag{4.47}$$

The scaling factor k_{trans} is component dependent and reflects the fact, that there is some internal toggling of the individual gates, until a stable state is reached. This cannot be approximated using area or the like. This k_{trans} depends on the component's type, bit width, as well as initial and target voltage.

4.7 Summary

This chapter has shown a new power and timing characterisation approach for digital hardware modules. It briefly described the project COMPLEX, which allows a design space exploration for complex and heterogeneous embedded systems. With this, the design space exploration process developed by the project COMPLEX fulfils requirement 1, from the beginning of this chapter. Based on the COMPLEX design space exploration process, an estimation process for embedded digital hardware modules has been developed, allowing an estimation and characterisation of hardware modules embedded in a complex and heterogeneous system. By considering the entire system as test bench for the hardware module requirement 2 is fulfilled. Module's estimation is performed on an RT-level data path, which is obtained from a high-level synthesis. The characterisation is based on combinational macros, automatically identified in the RT data path.

Each identified and characterised macro or hardware basic block is a spatial abstraction by describing a set of RT components, jointly active at once. Several methods for handling the occurring state explosion are given. These range from the reduction of the control signals that must be considered to discarding hardware basic blocks whose behaviour is already implemented by another hardware basic block. Characterisation of the hardware basic blocks is done in terms of structural properties, as required by requirement 4. Physical parameters like supply voltage are separated, allowing the executable model, described in the following chapter, to be used in different scenarios. Synthesis artefacts such as parasitic functionality are also considered, fulfilling requirement 3.

Two different characterisation techniques are shown, both considering data dependencies implicitly, as required by requirement 5. Non-functional properties like power dissipation due to clock or controller activity are also included in the model. The aforementioned separation of structural properties and physical parameters allows the model to regard several power modes and also considers transitions between the individual modes. Special attention was given to power mode transitions from off- to on-states.

Summarising, it can be stated that this chapter has shown a way for raising the level of abstraction at which accurate power and timing estimates can be made, by almost an entire level. The presented hardware basic blocks encapsulate a large number of RT components, promising a large speed-up during simulation. Despite the expected gain in simulation speed, the characterisation based on structural properties and with respect to implicit data-dependencies is still providing accurate results.

With the results of the identification and characterisation process, described in this chapter, it becomes possible to generate an executable, power and timing aware high-level simulation model. This model and its generation are described in the following chapter.

Chapter 5

Executable Model Generation

Abstract

This chapter explains how the characterised information is used to create an executable virtual prototype of the hardware module. This prototype is used to estimate the overall system's behaviour in terms of power and timing, by performing a simulation of the virtual prototype. The prototype consists of augmented versions of its contained modules. Each module is provided with additional information about power and timing, obtained during characterisation. Along with a functional simulation of the overall system, activity information is collected and power and timing is estimated.

The following chapter presents in detail how the virtual prototype for a full-custom hardware module is generated. First, it outlines the structure of the generated virtual prototype and how the individual parts interact with each other. It then gives a detailed insight in the implementation of the functional, the power mode as well as the power & timing model. It shows how these models are combined for obtaining accurate estimates. Towards the end, the chapter also explains how the virtual prototype of the hardware module is connected to the overall system prototype.

AFTER the characterisation of the design under consideration has been performed, a power and timing aware behavioural model can be generated. It utilises the characterised values in order to provide fast, yet accurate estimates. This satisfies requirement 6 of the requirements to a new characterisation and estimation process, as identified in the introduction of Chapter 4.

For gaining a notable simulation speed-up, the simulation semantics must be simplified. Simulation details must be abstracted in a way that improves simulation speed, but retains the desired accuracy of the estimation. Some abstraction techniques are mentioned in Section 3.2. Generally, there are different types of abstraction that can be applied [48]. It is also possible to apply a combination of them. Even though the author's original intention was their application on software systems, they also can be applied to models, describing hardware systems [123]. Abstraction by *translation* evaluates the input model and translates it into a new output model. This is the most familiar technique in hardware design. Abstraction by *extension* utilises facilities from the lower level to build the higher one. A prominent example is SystemC, which extends the C++ standard, allowing to model hardware modules. Finally, there is *interpretation* e. g., in the form of firmware or microcode. It can be used for software modules, for example. For modelling hardware modules, it might be best compared with VHDL or Verilog, which are not executed directly, but interpreted by the simulator. However, there are also some tools which compile VHDL and Verilog before performing the simulation.

As mentioned in Section 4.3, the behavioural input model of the proposed characterisation process is a C/C++ model, which in turn is a sequential description of the behaviour. This model can be executed on a software processor i. e., the host machine. During high-level synthesis, this model is transformed into a description consisting of hierarchical organised and concurrently executed processes. Each process in turn consists of a data path and an associated controller. During scheduling, allocation, and binding phases of the synthesis process, individual operations of the input model are mapped to hardware operators.

In order to optimise power and utilisation of the hardware resources, several operations are mapped onto the same operator. Different optimisation strategies can be used to minimise the number of RT components, required for implementing the design. These massive transformations and optimisations prohibit a simple back-annotation of the characterised hardware properties to the C/C++ input model. It is necessary to create a new high-level description of the syntheses result. This new description must represent the functionality and all syntheses artefacts, required by an accurate power and timing estimation.

In order to allow a co-simulation with the original test bench as well as the surrounding system i. e., the virtual system prototype, the output model preferably should be a C/C++ model. As mentioned above, augmentation takes place in terms of basic block, hence the name *block annotated C++ (BAC++)*. Such a model allows a fast compiled simulation with power and timing estimation capabilities. Since the original test bench can be utilised, the generated model can be used with different input stimuli, and not only the ones it had been characterised with. This fulfils requirement 7 of the requirements identified at the beginning of the previous chapter.

5.1 Simulation Models

Different simulation models can be used for implementing the BAC++ model. Basically, three different types can be identified [143, chap. 3]. *Differential equation models* are the most powerful models. The actual state of the system is specified as a set of state variables. Using derivative functions, which specify the rate at which these variables change, the system's state can be computed for any point in time. For computing the state of the system for t_{i+1} from t_i without knowing what happens between those two points in time so-called *solver* or numerical integration methods are required. This type of modelling is mostly used for modelling continuous systems.

Using *discrete time models*, simulation is performed step-by-step, where each step covers a well-defined but maybe abstract amount of time. An example for such a model is an FSM. In each simulation step, a transition between two states takes place. The transition and possible output words are defined by corresponding functions. An example of such a model is given in Section 4.3.1. This type of modelling is well-suited for clocked systems.

Modelling digital systems is most often done using *discrete event models*. A system consists of different components, each one issuing events. These events can be internal i. e., do not affect other components, or they can be external. Internal events are triggered and scheduled by the component itself. External events however, are caused by other components. Therefore, the time they occur cannot be predicted. All events that occur are scheduled and stored in an event queue. This list is processed event by event. Processing an event might cause other events, which in turn are scheduled and stored in the event queue. Simulation is completed, if no events are left in the event queue or if a specified timeout is reached.

While differential equation models are suitable for continuous simulations such as Simulink, for example, they are not suited for modelling digital hardware at logic- or RT-level. Discrete time models seem to be the most obvious solution. Due to the discrete nature of the systems discussed here, they are preferable. The design's controllers are implemented as FSMs and can therefore be modelled easily using a discrete time approach. Communication between processes however is performed asynchronously and might iterate several times during a single timestamp. Therefore, discrete time modes are not powerful enough.

The solution is using a discrete event model approach. This type of model is capable of modelling the process-internal synchronous as well as the process-external asynchronous behaviour accurately. However, discrete event models typically use a sophisticated and therefore complex event queue, providing different delay models to signal assignments, such as inertial or transport delay. Considering the model, which is generated during synthesis, such a sophisticated event queue is not required and can be replaced by a simpler one. It can be assumed that events can occur only in the next delta or clock cycle, respectively. Moreover, it can be assumed that each process causes at least one event per clock cycle. This event is the computation of the FSM's next state. Regarding the hardware basic blocks it can be guaranteed that a static scheduling of their operations is possible.

Summarising, the generated model will be a light-weight discrete event simulation, allowing asynchronous communication between processes and statically scheduled behaviour inside the identified hardware basic blocks.

5.2 Structure of the Generated Model

The hardware implementation of the characterised module is a hierarchical structure of concurrently running processes. These are implementing the behaviour of the initial system description. Each process consists of an FSM and a corresponding data path. A process may also contain one or more sub-processes. Inter-process communication is performed by internal signals, which are connecting the sub-processes with each other as well as with the controller of the process itself.

During creation of the corresponding BAC++ model, this structure is maintained. Additionally, the generated BAC++ module contains models, required for estimating the power and timing behaviour of the module. The structure of the generated model is shown in Figure 5.1.

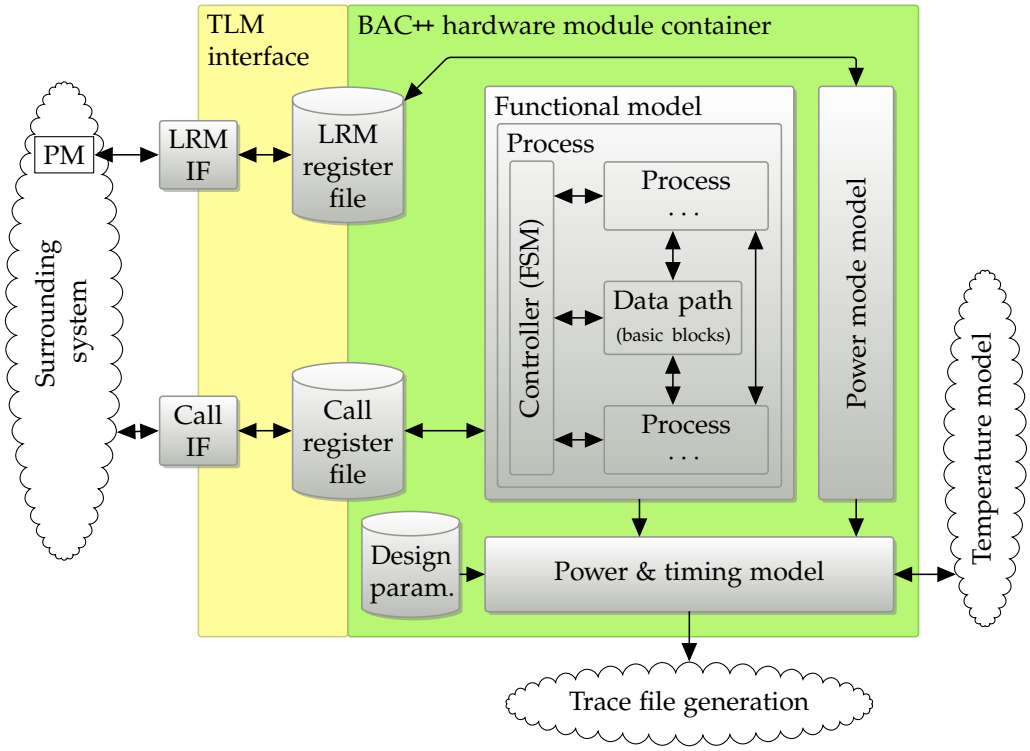


Figure 5.1: Structure of a generated BAC++ module — The created BAC++ HW module contains the functional and power mode model as well as the power & timing model. The functional model itself consists of processes containing a RT data path and the corresponding controller. It may also contain sub-processes. The created module is connected to the TLM interface using shared register files.

The structure of the generated BAC++ model consists of three main parts. The *functional model* contains the behaviour of the module in terms of previously identified hardware basic blocks, whose characterisation is described in Section 4.4. These are enriched with power and timing information, obtained during characterisation. Execution of the hardware basic blocks is managed by a controller, created from the process' FSM. The *power mode model* implements the

available power modes that had been introduced in Section 4.6 and the transitions between these modes. Power mode transitions can be requested from an external power manager, for example. The *power & timing model* considers information from the functional model, the actual power mode, and also non-functional design artefacts of the module, as described in Section 4.5. It combines this information in order to obtain the metrics required during design space exploration.

Besides the three models just mentioned, two register files are provided for accessing the module. The first one, the *local resource management (LRM)* register file is used for accessing the module's power management i. e., for requesting power modes. The second one is used for emulating the function call of the C/C++ input model, the generated hardware module was originally derived from. Both register interfaces can be wrapped by a TLM interface, allowing the module to be optionally used in a TLM-based virtual system prototype. The TLM wrapper basically provides a TLM-interface for accessing these two register files.

If the generated virtual prototype is embedded in a virtual system prototype, the BAC++ model must be synchronised with the overall system. Synchronisation with the surrounding system is performed by the BAC++ module container. Even though the BAC++ model is cycle accurate, it performs a temporal decoupled simulation. During execution of the functional behaviour, local simulation time is ahead of SystemC's simulation time. Moreover, simulation of the BAC++ module cannot be interrupted until it has completed its actual behaviour, which is known as *run-to-completion* scheduling. The local time of the BAC++ module is an offset to the global simulation time. At the end of the behavioural execution, local and SystemC time are synchronised. This is done by resetting the local time of the BAC++ model.

An eventually pending power mode transition is performed afterwards. Also, the trace-generation back-end is notified that the module is now idle. Even though no behavioural simulation of the module takes place, its idle-power is visible within the overall system's power trace. All four parts of the generated BAC++ representation will be explained in more detail in the following sections, but the model of computation is introduced first.

5.3 Model of Computation

The generated BAC++ model provides a compiled simulation. That is, it has to be compiled in order to be executed. The simulation performed by the model is cycle accurate. As stated, the generated model consists of a set of hierarchical structured processes. All processes are executed simultaneously. Execution of a process or the design, respectively, can be split into three phases. This is shown in Figure 5.2 on the next page.

In the first phase *initialisation & power management*, the generated design can be considered to be controlled externally. In other words, there exists no internal control flow. This phase can only be entered if the module is idle. The phase is entered directly before the simulation starts in order to perform an initialisation of the module. All registers are set to their default values and signals for inter-process communication are set to valid values. This phase is also entered during transitions between power modes. In this case the module cannot operate, since it is controlled by the power management.

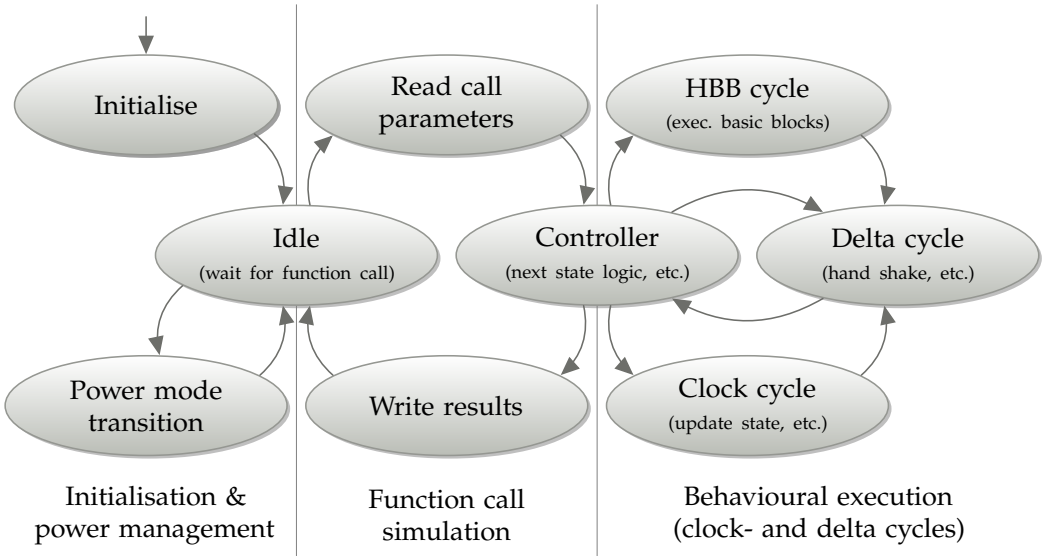


Figure 5.2: Main simulation loop — The main simulation loop consists of three main phases. In the first phase the module is initialised and power mode transitions are performed. The second phase simulates the function call. That is, it passes the call parameters and the return values to and from the module, respectively. The third phase performs the actual behaviour by triggering all concurrently running state machines of the module.

The second phase performs the *simulation of the function call*. After initialisation, the module is idle until it is been activated i. e., it is called using the call-interface. If the module is activated, call parameters are moved from the function-call interface into the data path of the module. In particular, the parameters are moved into the corresponding data channels like shared registers for simple call parameters or memories for arrays or user defined data types like structs. After all parameters have been provided to the data path, the handshake protocol is performed accordingly.

In the third phase, the *behavioural execution* is performed. This means, that the module's processes perform their control steps, until computation is complete. After the behavioural execution phase had been completed, computed results are moved to the function-call interface and the operation is finished by completing the handshake protocol. After the complete function call has been completed, the module is idle again and power management can be performed, for example.

5.3.1 Behavioural Execution

After all call parameters have been provided to the functional model, execution of the modules behaviour can be performed. As mentioned, all processes are executed simultaneously and cycle by cycle. That is, the FSMs of the processes are triggered in the same way. Generally, there are two differed types of simulation steps.

First, there are synchronous steps. This is, all the behaviour directly related to a rising clock edge. Computation of register values is done only once per cycle, for example. Data exchange between processes takes place by means of registered interfaces. That is, exchanged data is also only read and written on a rising clock edge. This is especially true for data channels like shared registers or memories. Again, exchanged data is computed only once per cycle. Therefore, this simulation step is called *HBB cycle*. The ordinary hardware basic blocks described in Section 4.4.2 are activated only once per simulation of a complete clock cycle. At the end of their execution, each hardware basic block notifies the power & timing model about the capacitance that had been switched during execution. During an HBB cycle, the process' controller computes the next state and also notifies the power & timing model about switched capacitance.

Besides the ordinary hardware basic blocks there is some behaviour that must be executed asynchronously, like the handshake protocol, for example. In order to perform the handshake as well as for computing the next state in the HBB cycle, the controller's conditions must be evaluated. These conditions do not depend solely on synchronous memory components like registers. They also depend on the asynchronously performed handshake protocol as well as some intermediate results. These intermediate results or assignments, as stated in Section 4.4.4, are directly obtained from the data path. Hence, some asynchronous computation must be performed. This is done during so-called *delta cycles*. Since the computation of a delta cycle of one process may affect the input of the other processes of the module, delta cycles must be performed until the module reaches a steady state. Due to the simple handshake protocol, the steady state is reached after approximately two to three delta cycles, depending on whether the processes are in a handshake phase, or not. Co-operation of asynchronous handshake protocol and synchronously performed behavioural simulation is shown in Figure 5.3 on the following page.

There is a third type of simulation step, called *clock cycle*. This simulation step marks the end of the currently simulated clock cycle. Hence its name. During this step, the internal time reference in terms of cycle count is increased and the current states of all the controllers' FSMs are set to the ones computed in the HBB cycle.

5.3.2 Inter-Process Communication

While the behaviour of each process is implemented as a clocked sequential system and thus uses a synchronous model of computation, the handshake protocol is performed asynchronously. That is, inter-process communication is performed asynchronously while behaviour and data exchange is performed synchronously.

Asynchronous behaviour is performed by means of a very simple discrete event simulation. In such a simulation, writing a value to some kind of channel causes an event to occur. In other words, the new value is not available instantaneously, but is scheduled using a possibly timed event-queue. The new value becomes visible, if the particular event is processed by the simulation engine. All event-queues are processed one after another and the behaviour, sensitive to the channel is triggered. During execution of the behaviour new events may be triggered, which again will be scheduled using the event-queues. Simulation of all delta cycles is completed if no more events are left in the queues.

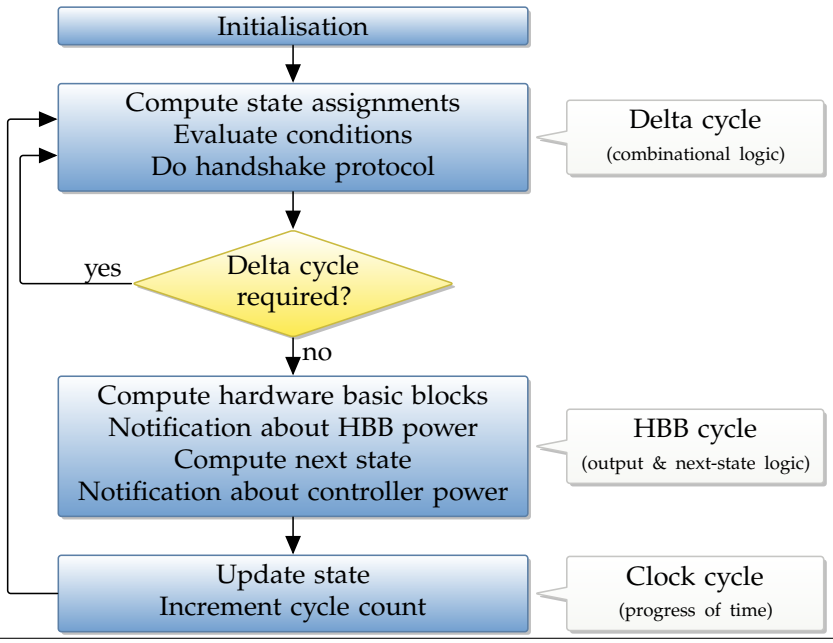


Figure 5.3: Behavioural execution — Behavioural execution is controlled by the generated controller. Delta cycles perform the handshake and compute combinational operations, HBB cycles compute hardware basic blocks, and clock cycles update the controllers' states and compute the progress of time.

Timeless behaviour of the generated high-level model is performed during these delta cycles. As detailed in Section 5.4.3, signals are used for communication. These signals provide a very simple event-queue that can hold exactly one event of type *value changed*. Using this simplified technique, no sophisticated event-queue like in VHDL or Verilog is required.

Each time the signal gets a value assigned that is different from its actual one, the event is raised. If there are events left to be processed, a delta cycle is executed. First, all signals take their new values and their events are reset. Hereafter, the timeless behaviour of the processes is simulated. Specifically, the controller's state machine's read and write handshake signals etc. The handshake protocol used for inter-process communication is a very simple protocol, requiring only a small number of delta cycles, which in turn will fasten the simulation. After all delta cycles have been performed, simulation continues with the synchronous behaviour, as shown in Figure 5.3.

5.3.3 Progress of Time

Module internal progress of time is correct by construction. This is because a cycle-accurate model is been created, based on the also cycle-accurate RT-level description of the module, which was generated during high-level synthesis. Process-local time is considered in terms of cycle count. After all delta and the single HBB cycle had been performed, local time of the process is increased, by increasing the cycle count by one. The process' internal cycle count is used as reference timestamp sent to the power & timing model. This uses the timestamp the compute the actual elapsed time in seconds, based on the applied clock frequency.

Progress of time must be considered in two ways. At first, it must be considered during trace generation. As mentioned in Section 5.6.5, any information that is added to the trace contains information about time in terms of a timestamp given in seconds. The timestamp is an absolute time information that is obtained by adding the local time offset to the global SystemC time.

Secondly, time must be considered, if the generated BAC++ model returns simulation control to the SystemC kernel. In order to further increase simulation speed, the behaviour implemented by the generated BAC++ model is not immediately synchronised with the actual simulation time of SystemC. Instead, synchronisation is only performed, if it is required by the behaviour. Local BAC++ and global SystemC time are synchronised at communication borders i. e., if an external channel is accessed. Behaviour that has been obtained from a C/C++ input specification is always executed to completion, also known as *run-to-completion*. In other words, an activation of the generated BAC++ module represents the particular function call from the input model.

Synchronisation behaviour is implemented by the created BAC++ module container. It is transparent to the functional model. In order to run ahead of the actual SystemC simulation time, additionally elapsed time is stored by the module container. The actual time inside the BAC++ module is obtained by adding the additional elapsed time to the actual simulation time of the SystemC simulation kernel. This way it is possible to obtain exact timestamps during execution of the functional behaviour, as it is required for generating timestamp/value-tuples that should be traced. After behavioural execution has finished, the simulation kernel is notified about the elapsed time and the local time is reset.

5.4 Functional Model

The functional model contains the behavioural description of the module as well as all power and timing information directly related to its behaviour. Its structure is directly derived from the structure of the RT-level description, which was generated during high-level synthesis.

The model is generated from hardware basic blocks, identified during characterisation. Each basic block contains a small part of the behaviour as well as metrics for power and timing estimation, directly depending on its behaviour. During simulation, different basic blocks are executed, depending on the actual control flow, which in turn was triggered by the applied input stimuli. Each time a hardware basic block is executed, it notifies the power & timing model about the switched capacitance and the amount of clock cycles that where required for its execution.

During BAC++ model generation, a C++-class hierarchy is created, representing all processes with their controller and RT data path as well as their sub-processes. Each controller is implemented as a `switch` statement, emulating the controller's FSM and thus the control flow. The behaviour of the data path is represented by the identified hardware basic blocks. Each one is implemented as a C++ member method of the class, which implements the particular process the hardware basic block belongs to.

In complex designs, which contain sophisticated and concurrent running behaviours, these processes are arranged in a hierarchical structure. This means that a process may contain

sub-processes. The process itself and its sub-processes can communicate with each other. The created process classes are connected with each other during instantiation using references to shared variables. Not shown in Figure 5.1 on page 102, but part of the generated model are special communication channels, such as shared registers or memories. These channels allow a data exchange between the individual processes.

The implemented model of computation uses a simple discrete-event simulation for inter-process communication. Thus, all operations on signals are executed on current signal values. After obtaining all new values, the values of the signals are updated. For hardware basic blocks a more simple approach is feasible. Because a hardware basic block is enclosed by registers and may not contain any loops, it can be statically scheduled. Thus, no delta cycles are required for simulating a basic block.

5.4.1 Hardware Basic Block Implementation

Most important part during generation of the executable model is the code generation for implementing the hardware basic blocks. The previous chapter has defined a hardware basic block as combinational macro, representing a certain functionality that is activated in the data path, if a specific output symbol is applied by the controller. While the RT-level model allows a parallel execution of the behaviour, the C++ implementation of the basic block requires the behaviour to be executed sequentially. Therefore, all operations belonging to the basic block must be statically scheduled. During operation scheduling, all inter-operation dependencies and lifetime of registers values must be regarded.

Operations belonging to a hardware basic block are implemented using local variables. Therefore, the result of an operation is only available within the method, implementing the hardware basic block. However, registers and other RT components that are unique to a single process are created as process-local variables, accessible from all hardware basic blocks and the process' controller. An example implementation of a hardware basic block is shown in Listing 5.1. It depicts the implementation of the hardware basic block from Figure 4.7 on page 75.

Code generation is done in three major steps. First, a local copy of all required source registers is created. This is done by the lines 3 to 5. Creating a local copy of the required register values allows the registers to be read and written in the same hardware basic block without creating a race condition. Second, new register values are computed by the lines 8 to 13. The code created for value computation is generated using a bottom-up depth-search in the data path. That is, before creating a certain operation, its required inputs must be available. Therefore, the inputs are generated first. This is repeated until the required input contains only source registers, since they can be assumed to be available.

An important property that must be regarded during code generation is the bit-with of the operator. The RT-level description is bit-accurate i. e., registers, operators, etc. have the required bit-width. Unfortunately, C/C++ does not natively provide arbitrary bit-widths. Depending on the compiler's target architecture only types with a certain bit-width are available. These types are used, whenever possible. However, especially bit-widths which are not a multiple of a byte are not available. In order to maintain the correct behaviour during overflows etc., bit-masking and shifting operations are added whenever necessary.

```
void hbb0001(void) {
   // Create local copy of all source registers
   int local_r1 = this->r1.read();
   int local_r2 = this->r2.read();
   int local_r3 = this->r3.read();

   // Compute new values of registers r6 & r7
   int sub1 = local_r2 - local_r3;
   int mul1 = local_r1 * sub1;
   this->r6.write(mul1);

   int mux1 = sub1;
   this->r7.write(mux1);

   // Notify power & timing model
   this->send_notification(hbb_notification(
      "hbb0001",            // sender name
      7.362e-13*si::farads, // functional units & muxes
      0.000e-00*si::farads, // interconnect
      1.182e-12*si::farads, // registers
      1                     // duration [cycles]
   ));
   return;
}
```

Listing 5.1: Abridged BAC++ implementation of a simple hardware basic block — The code consists of three main parts. The first section is responsible for creating a local copy of all source registers, while the second part performs the actual computation. The third part finally notifies the power & timing model about dissipated power and elapsed clock cycle.

In the last step, after all new values have been computed, the power & timing model can be notified about dissipated power and elapsed clock cycles. Lines 16 to 22 show how such a notification is send. It contains the values obtained during characterisation of the hardware basic block. Even though the implementation of the hardware basic block only contains the required computations, synthesis artefacts like parasitic functionality are part of the characterised values for dynamic power dissipation.

The generated code is some kind of three-address code, well known from compiler construction. In other words, each line contains a single and simple operation. Typically this is an operation, two operands and an assignment of the result. Generating the three-address code in the way described above, may cause successive, simple assignments to occur. Such a sequence of simple assignments occurs during code generation for multiplexers, for instance. An example is shown in line 12 of Listing 5.1. Fortunately, these simple assignments can be easily optimised by the compiler. Zhong et al., which use a very similar approach for generating C/C++ code, have shown that converting C-code into three-address C-code does not lead to an observable increased execution time [145]. This is because most compilers internally use some kind of a three-address representation as well.

5.4.2 Controller Implementation

The controller of each process is implemented as one large `switch` statement, where each `case` section represents a specific state of the controller's FSM. In each state, the controller evaluates the SMT predicates, which are required for computing the controller's next state or the active hardware basic block, and behaves accordingly. The behaviour within a state is two-fold and reflects the different execution cycles, introduced in Section 5.3.1. A sample implementation of a single state of the controller's FSM is shown in Listing 5.2.

```
case S1: {
   switch (cycle_type) {
      case DELTA_CYCLE:
         if (IOST_fsync_syncRight_rdy_r==true && SEL_Assign_1==false) {
            IOEN_mem_in_readWritePort_0_CEN0.write(0);
         }
         if (true) {
            IOEN_fsync_syncRight_ack_r.write(1);
         }
         if (true) {
            this->hbb0001();
         }
         break; //DELTA_CYCLE

      case HBB_CYCLE:
         if(!IOST_fsync_syncRight_rdy_r){
            this->state.write(S1);
         }else if (true) {
            this->state.write(S2);
         }
         if (IOST_fsync_syncRight_rdy_r==true && SEL_Assign_1==false) {
            this->hbb0002();
         }
         break; //HBB_CYCLE
   }//switch cycle_type
} break; //S1
```

Listing 5.2: Sample BAC++ implementation of a single state of the controller's FSM — During a delta cycle, handshake with other modules like memories or the surrounding processes is performed. Intermediate results, required for evaluation the Boolean conditions are also computed. In a HBB cycle, the next state is computed and the behavioural hardware basic blocks are activated.

In the listing, lines 3 to 13 embrace the implementation of a delta cycle. The controller evaluates the SMT predicates and determines its behaviour during the delta cycle. Register values are directly available, but intermediate results, required by the controller must be computed during the delta cycle, as mentioned in Section 4.4.1. First, the controller performs the handshake protocol. In the example, line 5 shows the handshake with a memory, while line 8 performs the handshake with the surrounding processes. Second, line 11 activates the assignment hardware basic block `hbb0001`, which is responsible for computing

the intermediate results like the value of `Assign_1`. Since predicate `SEL_Assign_1` depends on the value of `Assign_1`, which might change during execution of `hbb0001`, the predicates must be re-evaluated. Thus, another delta cycle is required, if a value is changed by an assignment hardware basic block. This phase is repeated by the simulation kernel, until no more delta cycles are required. This is also shown in Figure 5.3 on page 106.

After completing all delta cycles, the actual behaviour of the data path can be simulated, which is done in lines 15 to 24. After the handshake has been completed and all intermediate results are available, the next state of the controller's FSM is computed in lines 16 to 20. The next state is computed before the hardware basic block is activated, because it might depend on the register values, which in turn might be changed during execution of the basic block. Computing the next state before executing the behaviour of the data path allows using a single-valued register logic instead of a costly current-/next-value logic for each register. After the next state has been computed, behavioural hardware basic blocks are activated. In Listing 5.2 this is done in lines 21 to 23. Exactly one hardware basic block per process is executed, since exactly one of the conditions will evaluate to `true`. In the example, it can bee seen that `hbb0002` and the memory handshake depend on the same condition. Therefore, it can be assumed, that the memory is used by the basic block.

In the final behavioural execution phase, which is not shown in Listing 5.2, the clock cycle is performed. Since this phase is the same for all states of the controller's FSM, it is done directly after the `switch` statement. During a clock cycle, the local simulation time i. e., the local cycle count is increased. After the clock cycle had been performed, the behaviour, described in Section 5.3.1 as well as in this section, is repeated, until all data has been processed.

5.4.3 Implementation of Inter-Process Communication

There are two different ways for implementing inter-process communication. First, there are simple *signals*, which can be directly connected to the processes. These are typically used for implementing the handshake protocol. Data exchange however, is done using so-called *data channels*. This second way can be subdivided into different data channels. In this thesis memories and shared registers are supported.

In the generated model, inter-process communication is performed by means of signals, which are implemented as shared variables. Connecting the processes with the signals is done in the constructor's initialisation list of the parent process i. e., the class that instantiates the processes to be connected. Inside a process, the external signal is represented as a reference to the object representing the signal. If a signal is passed to sub-processes, again this is done as a reference to the actual object. This way, the interconnection between processes is checked during compile time. No special elaboration phase is necessary.

All signals provide a common interface for reading and writing values. Since the actual internal behaviour is hidden, they can be considered as abstract channels. The channel can then be accessed using simple function-calls. Using the proposed channel approach it is not possible to detect competitive write operations from multiple clients. It is even not possible to determine, if a certain client is a channel reader, a writer, or both. Again, this simple structure of the inter-process communication improves the simulation speed. Since it can be assumed that the generated model is correct by construction, the drawback is negligible.

For data exchange between processes data channels are used. Like processes, they are part of the design hierarchy. A connection between processes and data channels is made by means of the aforementioned signals. Most prominent data channels are shared registers and memories. A *shared register* is basically an ordinary register. Main difference is that the shared register might be enabled by several processes. While the register's value can be read all the time, write accesses are handled during clock cycles, only. As usual, a register can only hold a single value. For exchange of larger sets of data, *memories* are used. The provided template for instantiating memories allows multiple read and write as well as read-only ports. During initialisation of the simulation, an image file, generated during characterisation is loaded into the memory. Accesses to the memory are handled during clock cycles, as expected. In contrast to process-local data structures, data channels can be accessed by multiple processes. In order to provide a conflict-free access to the channel, a resolution-logic is provided, whenever necessary. Since the resolution logic is pure combinational, it is evaluated during delta cycles and thus is transparent to the processes, trying to access the channel.

5.5 Power Mode Model

The *power mode model* contains information about the module's available power modes and the allowed transitions between them. The information that is contained within the power mode table is described in detail in Section 4.6. Also, an example table is given in Section B.3. The power mode table is generated during characterisation and loaded during the initialisation phase of the virtual prototype simulation. As mentioned earlier, a power mode is a combination of applied supply voltage and clock frequency. Power mode transitions allow switching from one power mode to another one, although not all transitions that are possible are actually allowed.

Besides the static information about available power modes and transitions between them, the power mode model contains information about the power mode, which is actually selected by the global power manager during simulation of the virtual prototype. Changing the power mode will result in a different execution time and power dissipation of the functional model.

Power mode transitions can be requested using the LRM-interface. It is a simple register-interface consisting of three registers. The first two registers contain the ID of the actual and the requested power mode, respectively. The third register is a status register and determines if a power mode transition is requested and pending, or if an error had occurred. For example, if an unknown power mode is requested, or if the requested transition is not allowed. Figure 5.4 shows the state machine of the power mode model.

If a power mode transition is requested by the global power manager, the power mode model performs the requested transition as soon as possible. As an implementation artefact, these transitions are only performed, if the behavioural code releases the control of the simulation. Power mode transitions can only be performed between two activations of the module i. e., only if the module is idle. The execution of behavioural code is not interrupted by a power mode transition request. As mentioned, the behavioural simulation has a *run-to-completion* semantic. Due to the fact that typical systems do not perform a power mode transition while

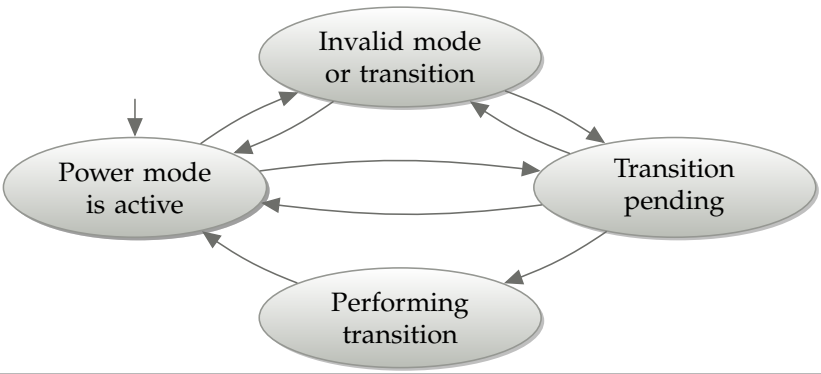


Figure 5.4: FSM of the power mode model — Initially, a valid power mode is active. A power mode transition can be requested at any time. If the module is idle during the request, the transition takes place immediately. Otherwise the power mode transition is pending. If the requested power mode is not known or if the requested transition is not allowed, the state register of the LRM-interface is set accordingly. A pending or invalid transition can be revoked at any time. If the module becomes idle and a power mode transition is pending, the transition is performed.

a certain component is active, this disadvantage is acceptable. A description of the register interface is given in Section 5.7.2.

If a power mode transition is performed, the power & timing model gets notified about changed supply voltage and clock frequency. It is also notified about penalties caused by the transition. Another important task of the power mode model is providing power information for intervals, in which the particular BAC++ module has no activity. That is, if the module is in a power saving mode like `sleep`, or `power_gated`. The same is true if the module is not used i. e., if it is idle.

5.6 Power and Timing Model

Purpose of the power & timing model is to compute the required metrics. That is, power dissipation in terms of watts and timing in terms of seconds. These are then used for generating the power trace and for performing synchronisation with the virtual system prototype, if necessary.

The power & timing model is configured with the static design metrics that cannot change during runtime. These values are design-depended constants of the power model. The first one is the *apparent internal conductance G* of the design. It describes the virtual internal conductance of the hardware module. This constant is mainly used for computing the static power dissipation of the module in dependence of the applied supply voltage. The second parameter is *average capacitance switched during idle-phases* C_1^{idle}, which is used to provide a power value during idle and sleep-phases. It mainly describes the capacitance that is switched by the clock-tree and the controller, while the module is in its idle state. There is also information about default voltage and clock frequency, in case that no power mode

table is provided. Basically, power estimation is done by using Equations (5.1) and (5.2), respectively.

$$P_{\mathrm{d}} = \frac{1}{2} C_{\mathrm{l}} V_{\mathrm{dd}}^2 f_{\mathrm{clk}} \tag{5.1}$$

$$P_{\mathrm{l}} = V_{\mathrm{dd}}^2 G \tag{5.2}$$

Since several notifications from different sources are received during a single clock cycle and since a more fine-grained power trace is required, an approach more sophisticated than Equations (5.1) and (5.2) is used. During simulation of the virtual prototype, the power & timing model collects the information, sent from the functional and the power mode model. As mentioned in Section 5.4, activated hardware basic blocks, processes, and data channels as well as the power mode model send notifications to the power & timing model. This is done each time power dissipates or if supply voltage and clock frequency have changed. The following sections give more details on the particular models.

5.6.1 Dynamic Power Dissipation

Remembering the separation from Equation (4.21) on page 84, switched capacitance can be accumulated for all activated hardware basic blocks of all processes of the module. Using the *observer design pattern*, a notification is send from each activated hardware basic block to the power & timing model. The same is true for the controller of the module, with the difference that the controller is active in each clock cycle. Switched capacitance of a hardware basic block $C_{\mathrm{l}}(H)$ as well as switched capacitance of a process $C_{\mathrm{l}}(p)$ is built from more fine-grained values, using Equations (5.3) and (5.4).

$$C_{\mathrm{l}}(H) = C_{\mathrm{l}}^{\mathrm{reg}}(H) + C_{\mathrm{l}}^{\mathrm{op}}(H) \tag{5.3}$$

$$C_{\mathrm{l}}(p) = C_{\mathrm{l}}^{\mathrm{ctrl}}(p) + C_{\mathrm{l}}^{\mathrm{clk}}(p) + C_{\mathrm{l}}^{\mathrm{net}}(p) \tag{5.4}$$

Switched capacitance C_{l} of each hardware basic block H_i and switched capacity of each process' controller p_j is summed up. It is then multiplied by the square of the supply voltage V_{dd}, giving Equation (5.5). It computes total dynamic energy at the end of each clock cycle. That is, after the behaviour of this control step has been executed.

$$E_{\mathrm{d}}^{\mathrm{total}} = \left(\sum_{i=1}^{\#_{\mathrm{H}}} C_{\mathrm{l}}(H_i) + \sum_{j=1}^{\#_{\mathrm{proc}}} C_{\mathrm{l}}(p_j) \right) V_{\mathrm{dd}}^2 \tag{5.5}$$

5.6.2 Static Power Dissipation

While dynamic power estimation must be performed in every single clock cycle, static power dissipation must be estimated after power mode transitions, only. As mentioned in Section 4.5.4, static power depends on the equivalent conductance of the module i. e., its top-level process $M = p^{\mathrm{toplevel}}$, which is fixed for a given module. The equivalent conductance

is obtained using Equation (4.41) on page 93. Besides the equivalent conductance, static power dissipation also depends on the applied supply voltage, given by the actual power mode.

Using the separation from Equation (4.42) on page 93 and assuming that supply voltage V_{dd} and temperature T is the same for all processes of the module M, static power can be computed by Equation (5.6).

$$P_{\mathrm{l}}^{\mathrm{total}} = G(M)V_{\mathrm{dd}}^2 g(V_{\mathrm{dd}}, T) \tag{5.6}$$

The term $g(V_{\mathrm{dd}}, T)$ is used to scale the module's conductance with respect to the applied supply voltage and temperature. Usually, three mesh-points, typically located at 20, 70, and 120 °C, are used for providing very low-level and temperature-dependent .LIB files. These in turn are used to generate the temperature depended *RT component data base (CDB)*-library, used by PowerOpt. Knowing this, syntheses and static power characterisation can be performed for these three mesh points.

5.6.3 Total Power Dissipation

The functional model gives switched capacitance, which in turn is used to compute dissipated energy. This is because time is modelled in terms of abstract clock cycles, only. Power dissipation can then be obtained by using information about the clock frequency, provided by the actual power mode. Total power dissipation can then be computed by adding the static power, as done in Equation (5.7).

$$P_{\mathrm{total}} = E_{\mathrm{d}}^{\mathrm{total}} f_{\mathrm{clk}} + P_{\mathrm{l}}^{\mathrm{total}} \tag{5.7}$$

A special case occurs during idle-phases. In this case, no dynamic power dissipation must be considered, since the module is not active. However, even during idle-phases the module is not completely inactive, as mentioned in Section 4.5.5. The clock-tree and the controller are still active, for example. The capacitance that is switched during idle-phases was characterised separately. Total power dissipation during idle-phases can be computed using Equation (5.8).

$$P_{\mathrm{total}}^{\mathrm{idle}} = C_{\mathrm{l}}^{\mathrm{idle}} V_{\mathrm{dd}}^2 f_{\mathrm{clk}} + P_{\mathrm{l}}^{\mathrm{total}} \tag{5.8}$$

5.6.4 Delay

The delay of each process p is obtained using Equation (5.9). In the current model of computation as described above, all processes are virtually executed concurrently. Moreover, each process is activated once in a clock cycle, yielding $\#_{\mathrm{cyc}}(p) = 1$. In other words, the delay of all process is the same. That is, $\tau(M) = \tau(p_i)$ for all processes p_i belonging to module M. However, since the clock frequency f_{clk} is defined by the actual power mode, the actual delay of the processes may vary over time, depending on which power modes are selected during simulation.

$$\tau(p) = \frac{\#_{\mathrm{cyc}}(p)}{f_{\mathrm{clk}}} \tag{5.9}$$

5.6.5 Trace Generation

Finally, the power & timing model generates a cycle-accurate power-over-time trace. Besides the trace of total power, the power & timing model is capable of creating more fine-grained traces. It is able to trace dissipated energy as well as dynamic power dissipation for each kind of resource like registers, operators, and multiplexers as well as the controller, clock-tree, and interconnect. This allows a more detailed analysis of the system's power dissipation than was originally possible with PowerOpt. Also, the power & timing model can create a detailed event log, containing all notifications sent by the functional model i. e., sent by all hardware basic blocks, processes etc.

Even though a fine-grained power trace is desirable, it might become very large. This is especially true for larger designs with several processes which are simulated using large sets of stimuli data. For such long simulation runs, which are typically used as basis for temperature or degradation analysis, a cycle-accurate power trace is not required. In these cases, a reduced trace is typically sufficient enough.

In order to provide traces with a reduced amount of data, the power & timing model is able to perform a sampling on the received data. That is, received data is collected and averaged over multiple clock cycles. Besides the reduced amount of data, sampling allows smoothing the generated trace. In a typical design, there is a big difference in the power that dissipates in two consecutive clock cycles. Displaying such uncompressed data yields confusing figures. Smoothing the trace tackles this problem. Sampling has been used during generating of some figures in Appendix D, for instance.

If sampling is applied, the temporal resolution of the generated power trace is reduced to nt_{clk}, where n denotes the width of the sampling window and t_{clk} is the period of the clock. Depending on the number of clock cycles required for completing the computation, the last sample might not be complete. In this case, the sampling window is shortened, in order to create the entry for the trace file. Timing estimation i. e., the estimated delay for the operation is still cycle accurate.

The power & timing model supports two different back-ends for generating the trace. The first one is used, if the virtual prototype is simulated stand-alone. In other words, simulation of the module is performed without the surrounding system. The same test bench as has been used for characterisation, is also used for estimation. Of course, a different set of stimuli data was used. If the virtual prototype is simulated stand-alone, all notifications from the functional model belonging to the same clock cycle are collected by the power & timing model. If the actual sampling period ends, this back-end creates a *comma separated values (CSV)*-file containing a single line for each traced sample period. Each line contains information about total energy, total power as well as energy per resource type etc.

If the BAC++ model is simulated within a simulation of the virtual system prototype, a more sophisticated back-end is required. This back-end must be capable to cope with the decoupled simulation approach, described in Section 5.3.3. In this case, the timestamps from the notifications, received during behavioural simulation are an offset to the global system time. That is, the timestamp must be adjusted. This is done by adding the actual system time, which is the point in time, the virtual prototype of the hardware module has been activated by the surrounding system.

The values are then forwarded to the tracing back-end, developed during the project COMPLEX. This back-end uses the SystemC build-in tracing features. The COMPLEX back-end utilises so-called *timed value streams*. Each entry to these streams is generated as a tuple containing corresponding power and timing information. There are two different types of tuples. The first one contains an energy value and a duration. This type is generated by the functional model, since the execution of each hardware basic block is well-defined in terms of dissipated power and execution time. The second tuple is caused by setting the power mode. It contains a power value and a timestamp, from which on the power value is valid. If a certain power mode is activated, the time when the particular mode becomes active is known, but it is not possible to make any statement about how long this mode will be active.

The same type of entry is used for modelling idle-phases of the module. If the module is idle i. e., between two consecutive activations, no behavioural simulation of the module is performed. Therefore, no notifications are sent to the observer. As mentioned in Section 4.5.5, the pre-characterised idle-power is used to model the module's power dissipation during such phases. More details on this type of tracing back-end can be found in the corresponding COMPLEX deliverable [133, sec. 3].

Besides the aforementioned power traces it is possible to have the virtual prototype to generate additional information. First, it is possible to generate an *event-log*. It contains all notifications and events like *call start/end*, *power mode transitions*, etc. in the order they are received by the power & timing model. This log provides a deep insight into the course of the simulation. For each event detailed information is available. This includes the origin of the notification, a local and global timestamp, at which the event that had occurred, or switched capacitance per resource type, if applicable. Second, statistical information can be generated by the back-end. Besides total energy per resource type as well as static power dissipation, statistics contain information about how often a certain hardware basic block or process has been activated. Its contribution to the total energy dissipation is also visible.

5.7 Virtual Prototype Interfaces

For accessing the generated virtual prototype two different interfaces are provided. The first one is the *function-call interface*, which allows the activation of the behavioural simulation. By passing the appropriate function-call parameters to the prototype its behavioural simulation can be initiated. After simulation has been completed, the same interface can be used for accessing the computed results.

The second interface is the *power management interface*, allowing selection of the desired power mode. Using this interface power modes can be requested and information about the actual power mode as well as status information can be obtained.

These two interfaces can be used directly by the test bench, which is the typical procedure, if a stand-alone simulation is performed. However, for embedding the prototype in a virtual system prototype, a *TLM-interface* is provided. It wraps the aforementioned function-call and power management interfaces and can be accessed using standardised TLM transactions. The following sections describe the interfaces in more detail.

5.7.1 Function-Call Interface

The function-call interface is used for activating the behavioural simulation of the module. As remarked in Chapter 4, the hardware module is generated using a high-level synthesis. Input to the synthesis is pure-functional C/C++ model that is then transformed into an RT-level description of its hardware implementation. Exactly one method in the input model must be declared as top-level module. The generated hardware module is thus derived from a conventional C/C++ function call.

Since the characterised module is represented as a function call within the original source code, this function call is reproduced within the BAC++ model. Therefore, a structure containing the in- and output parameters as well as the optional return value of the function call is created. The function-call interface thus represents the signature of the C/C++ model. Of course, this type of interface must be generated for each module of the system individually. Its basic structure is shown in Table 5.1.

The layout of the structure depends on the signature of the function call, the hardware component was generated from, since it provides a re-implementation of the original function call. Each in- and output parameter of the function's signature is accessible using this interface. In addition to the function's parameters some control registers are also added to the interface. These allow invoking the behaviour and provide status information. This structure is then written and read by the TLM interface as well as by the functional model.

Each BAC++ module implements a method `call(call_param_reg_map_type &params)`, which simply forwards the function call and its parameters to the module's functional model. This member function then implements the actual behaviour of the module i. e., it reads the input values and writes the output values from and to the referenced structure, respectively.

A function call is performed by calling the `call` method of the BAC++ module. A reference to the function-call register file is passed as parameter to the method. The method then copies the function call parameters from the function-call interface into the generated process hierarchy of the design, as described in Section 5.3 and depicted by the central part of Figure 5.2 on page 104. It then calls a method `execute`, which executes the behaviour of the module, as depicted by the right part of Figure 5.2. The `execute` method performs the behavioural simulation. Remembering Figure 5.2, the method `call` implements the emulation of the function call and the method `execute` implements the behavioural simulation.

After completion of the functional behaviour, the resulting values are stored into the function-call parameter structure. Again, member methods for performing the handshake protocol are created by the BAC++ code generation, since a deeper knowledge of the generated model is required in order to set the values correctly. More details about the internal structure of the functional model and the handshake protocol are presented in Section 4.4.

function-call parameter 1
⋮
function-call parameter n
return value

Table 5.1: Function-call register map

desired power mode		
recent power mode		
valid	reserved	status

Table 5.2: Power management register map

5.7.2 Power Management Interface

The power management interface is used by the global power manager of the virtual system prototype for requesting power modes. The interface does also provide information about the actual power mode as well as some status information. Its underlying state machine is shown in Figure 5.4 on page 113. While the function-call interface must be generated for each module individually, this interface is the same for all power-managed modules of the virtual system prototype.

Like the register file representing the parameters of the original function call, this register file represents the interface to the power management i. e., to the power mode model. The interface is directly controlled by the power mode model. After receiving a TLM call that writes to the register file, the power mode model is notified about the change. The model then sets the status register appropriately. If a new power mode is requested, the corresponding power mode transition is performed as soon as the module becomes idle. After the transition, the power & timing model is notified about the new power mode and the penalty, caused by the transition. The register file is updated accordingly to the new power mode. The structure of the interface is shown in Table 5.2.

5.7.3 TLM Wrapper

In order to connect the generated virtual prototype of the module to the virtual prototype of the overall system, a *TLM-interface* is available. This interface is basically a wrapper, providing TLM-based access to the function-call as well as the power management interface. The wrapper provides an interface for activating the module's functionality from within a TLM simulation environment. The TLM interface satisfies requirement 8, since it allows the module to be embedded in a virtual system prototype. More details about the TLM interface can be found in the corresponding COMPLEX deliverable [134, sec. 4].

5.8 Provided BAC++ Library

A generic class library has been implemented, providing base classes for processes and other structural components like registers, memories, resolution logic, etc. The library also provides the observer, the power mode as well as the power & timing model. Using this library it is easily possible to instantiate the hierarchical structure of the design and to implement its functionality. During compilation of and linking against the library, various settings are provided allowing an optimisation of the generated executable. The executable can be optimized for simulation speed or for a more detailed estimation, for example. Other settings allowing detailed debugging or profiling are also provided.

Most important part of the library is the process template. It provides functionality for registering the process' registers and signals as well as functionality for automatically updating their values. It also implements the infrastructure required by the notification pattern for the communication with the power & timing model. Each process instance has to instantiate its sub-processes and to implement its hardware basic blocks as well as its FSM, only.

The template of the functional model instantiates the top-level process as well as the power mode and power & timing model. It contains the complete implementation of the simulation semantic as described in Section 5.3 and provides the interface to the power mode model. Calls to the function-call interface are transparently forwarded to the top-level process.

Besides the aforementioned templates and classes, the BAC++ library contains various base classes and utilities that can be used by the functional model. This includes templates for instantiating data channels like shared registers or memories, for example. The later one can be specialised by defining the bit width of the address and data ports. The memory can be initialised using a memory image file, which has been generated during high-level synthesis. Templates for signals and variables are implemented, providing a generic `read/write` interface. This way, a delta-cycle-logic, as required by the discrete event simulation kernel, is transparently usable. If ports have different data types at the outer and inner side of the process they belong to, data conversion can be performed automatically.

Finally, the library contains all constructs for sending and receiving notifications in a convenient way. That is, notifications can be instantiated easily by the hardware basic block and processes of the module. The correct timestamp is added and all registered observers are notified automatically.

5.9 New Workflow

Using the approach presented in this and the previous chapter, a new workflow arises. It is depicted in Figure 5.5. This workflow smoothly integrates into the existing workflow. High-level synthesis, in this case using PowerOpt, is carried out as usual, but with the additional usage of the newly implemented hardware basic block identification, characterisation, and BAC++ model generation command. As can be seen from the figure, all required files for compiling and simulating the virtual prototype of the design under consideration are generated automatically. The virtual prototype can be compiled and linked against the BAC++ library easily, using a pre-defined Makefile structure.

By simulating the executable virtual prototype, a power-over-time trace as well as some statistic information is generated. It is now possible to evaluate the design with a large number of different and complex use cases and to obtain almost RT-level accurate results without the need of a time-consuming RT-level simulation.

Along with the extension of PowerOpt, a script has been developed that is able to automate the design space exploration. The script can automatically perform a number of steps in the design space exploration i. e., synthesise a design, characterise it and create the corresponding BAC++ model. This can be automatically repeated for different designs and synthesis settings. The script itself is controlled by a configuration file. An example of such a configuration is shown in Listing B.4 on page 167.

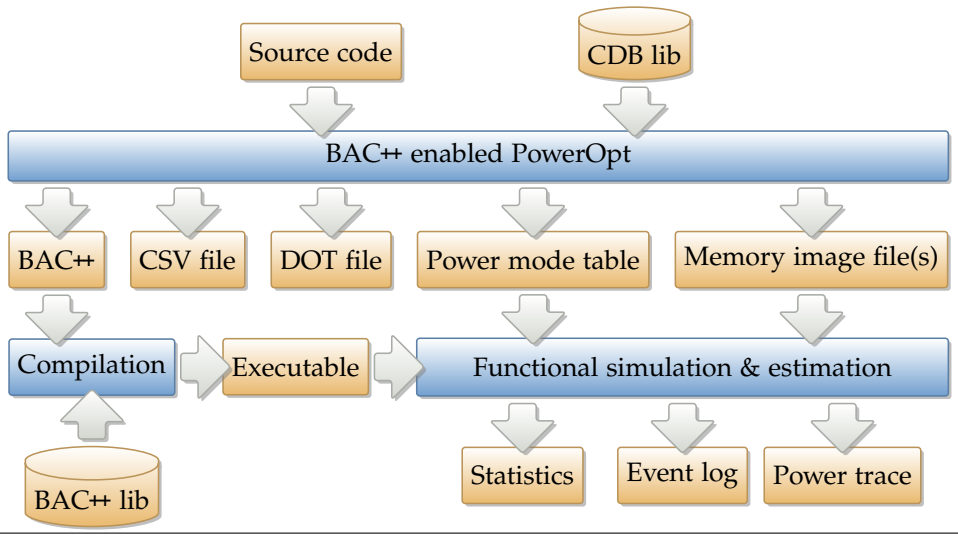


Figure 5.5: New workflow — New features have been integrated seamlessly into PowerOpt. The previous workflow remains virtually unchanged. By additionally executing the new PowerOpt command, the BAC++ model is created automatically. It can be compiled easily and along with the power mode table as well as the memory image files, a simulation and estimation of the hardware module or even the entire system can be carried out. Resulting statistics together with the created power trace can be used for evaluating the system's implementation.

5.10 Summary

This chapter has described how the identified and characterised hardware basic blocks, introduced in Chapter 4, can be used to create a power and timing aware virtual prototype of the hardware module. This fulfils requirement 6 as defined at the beginning of Chapter 4. The structure of the generated BAC++ model consists of four main parts. At first, it contains a functional model, representing the module's behaviour. It also contains behaviour-depended information about switched capacitance as well as non-functional artefacts like equivalent conductance. Second, the structure contains a power mode model. It selects the actual clock frequency and supply voltage. As third part there is a power & timing model computing the desired metrics from the information provided by the functional and the power mode model. In order to comply with requirement 7, an interface is generated, mimicking the original function-call interface of the module's initial behavioural description. This interface allows the virtual prototype to be executed with a set of different use cases. An optional forth part is the TLM wrapper, which is used for connecting the generated model to the virtual system prototype. This is done in compliance with requirement 8.

Along with Chapter 4, this chapter has described a way for generating a power and timing aware high-level simulation model from an FSMD, generated during high-level synthesis. Compared to most of the existing approaches mentioned in Chapter 3, the approach presented in this thesis can be considered as a grey-box approach. Full information about the low-level description is available, but the input model must be left unchanged. Concerning the overall

approach, a very similar concept can be applied to software as well as black-box IP modules. A description of power estimation of black-box IP module using a protocol state machine is given by Lorenz et al. [92].

Having the complete estimation process including hardware basic block identification, characterisation, and model generation available, it can be evaluated. This is done in the following chapter.

Chapter 6

Evaluation

Abstract
This chapter evaluates the proposed design characterisation and model generation approach. Evaluation is done using a set of eleven academic and industrial use cases. Individual steps of the proposed process are evaluated independently, before the overall process is assessed. The evaluation will show that the proposed concept of hardware basic blocks provides a good granularity. Large basic blocks allow a significant speed-up in simulation and estimation speed. A large variation in dissipated energy per basic block provides accurate estimates, since a large variation of the power dissipation over time can be assumed. The approach, presented in this thesis, provides an estimation speed-up of about $160\times$*, while having an error-per-cycle of less than* $15\,\%$ *for most simulated clock cycles. If a small sampling window is applied, this error can be reduced to about* $6.93\,\%$*.*

After a brief introduction, evaluation is done in four steps. First, the hardware basic block identification process is evaluated. This comprises the efficiency and quality of the proposed identification technique as well as a rating of the identified hardware basic blocks. In a second step, the accuracy of the simple as well as the advanced hardware basic block characterisation is shown. The evaluation of the characterisation is followed by a short assessment of the power mode support. The fourth step evaluates the complexity and performance of the generated high-level models. Finally, the overall power and timing estimation process is evaluated and its advantages over the conventional estimation process are explicated. At the very end of this chapter, known problems and issues of the proposed characterisation and estimation approach are recapitulated and a summary is given.

EVALUATION is performed using academic examples, standardised benchmarks as well as industrial designs. Overall, the evaluation is carried out on eleven example designs. For comparability all input models were synthesised using the same technology and synthesis settings. These are shown in detail in Table C.2 on page 170. Characterisation and evaluation was done using different sets of input stimuli.

For a semi-automatic evaluation, a script has been developed allowing the automatic configuration, synthesis, characterisation, and BAC++ model generation for each design. The script is controlled, using a simple configuration file. An example configuration of the script is given in Section B.4. The generated models have been compiled using different settings in order to perform regression, performance, and accuracy measures. For example, regression was deactivated during measuring the performance of the generated models. During accuracy measurement however, regression was enabled.

As just mentioned, different types of examples are used during evaluation. Besides some basic and advanced academic examples, standardised benchmarks are used, allowing a comparison of the proposed approach with existing ones. For evaluating RT-level power estimation approaches, standardised benchmarks like ISCAS-85 or ITC-99 are used typically. For the purpose of this thesis, these benchmarks are unsuited since they only contain small designs. The approach, presented in this thesis, tackles larger and more complex designs. Thus, two benchmarks from the *CHStone* benchmark suite where chosen. The benchmark suite was presented by Hara et al. [70] and is freely available on the internet [5]. Additionally, a benchmark from the COMPLEX project was chosen, showing the applicability of the proposed techniques to industrial-strength designs.

The first class of examples contains simple academic use cases. These use cases implement typical arithmetical algorithms, including the computation of a faculty, a square root, or factorise a given number into its prime factors. This set also contains algorithms for performing a *fast discrete cosine transform (FDCT)* or the heapsort sorting algorithm. The second class consists of more advanced examples. It contains algorithms such as *advanced encryption standard (AES)* encoding, MP3 decoding, a wavelet transformation or the *linear predictive coding (LPC)* known from the *global system for mobile communications (GSM)* standard. The industrial-strength benchmark consists of a hardware accelerator, implementing an FFT. It is embedded in a complex and heterogeneous audio/video surveillance system. The accelerator is available in seven different configurations, which allows performing a design space exploration of the overall system. However, in this work only the configuration with a fixed FFT-depth of 256 was used. Detailed descriptions of all benchmarks and their structural properties as well as the settings used for their synthesis are given in Appendix C. Detailed evaluation results for each design can be found in Appendix D.

Comparing the achieved results to a low-level i. e., logic-level estimation is difficult and very time consuming. Thus, it cannot be done for all designs that had been used for the comparison with PowerOpt. Also, using low-level tools, it is not possible to perform such long simulations. The computational effort that is required for a logic-level simulation and estimation is several orders of magnitude larger than the effort required by an estimation using the BAC++ model. Same applies to the generated trace files. Nevertheless, a comparison for Design VII with an shortened simulation was performed. Results are shown and discussed in Section D.11.

6.1 Hardware Basic Block Identification

Evaluation of the hardware basic block identification is two-fold. First, the identification process itself is evaluated. This is done by comparing the upper bound of the number of existing hardware basic blocks to the number of actually analysed and generated basic blocks. In a second step, the identified hardware basic blocks themselves are assessed. The number of RT components per basic block gives an indication of the size and complexity of the later generated high-level model. In addition, the activated area is an indication for the energy dissipating, if the particular basic block is activated.

6.1.1 Identification Process

The first evaluation assesses the hardware basic block identification process. Since each process of a design is characterised individually, all processes are evaluated separately. Thus, a total of 15 processes are used for evaluating the basic block identification process.

Three metrics are used for evaluating the efficiency of the proposed identification process. The *number of analysed hardware basic blocks* gives an indication of the performance of the identification process. The less hardware basic blocks must be identified, the faster the identification process is. As mentioned in Section 4.4.3, it is not necessary to create code for each identified hardware basic block in the generated model. Zero-strength or invalid basic blocks can be discarded, for example. For hardware basic blocks that are not discarded, code must be created during model generation. Thus, the *number of generated basic blocks* gives a first indication of size and complexity of the generated high-level model. The last metric comprises the *number of used hardware basic blocks*. This is the most important metric, since it states the quality of the identification process. The usage denotes the relation between the number of generated basic blocks and the number of basic blocks that are actually activated during simulation. The metric can be used for determining the overhead of the generated model in terms of unnecessarily generated basic blocks. The higher the usage is, the better is the quality of the identification for that particular process.

It is important to note that the use case, which is used for obtaining the number of used hardware basic blocks does not necessarily provide a full coverage of the control-flow. That is, more basic blocks might be required for providing a full-coverage than are actually activated by the use case. Anyhow, the relation between the number of generated and the number of used hardware basic blocks still gives a good first hint. A detailed evaluation of the complexity and efficiency of the generated model is also given in Section 6.4. All three metrics for all processes can be seen in Table 6.1 on the following page.

In a straight-forward approach, one hardware basic block must be identified for each output symbol of a process. It can be seen from Table 6.1 that the proposed identification algorithm drastically reduces the number of basic blocks that must be analysed. Only a minimal fraction of the basic blocks that are generally possible are actually analysed. It can also be seen that there is no direct relation between the number of states or the number of output symbols and the number of basic blocks that must be analysed. This is because the number of basic blocks depends on the way the data path is used by the controller i. e., it depends on the complexity of the output logic in each state of the controller's FSM.

	process		hardware basic blocks			
	states	out. sym.	anal.	gen.	used	usage [%]
Design I	4	5.12×10^{2}	21	9	9	100.00
Design II	18	2.10×10^{6}	66	21	12	57.14
Design III	8	3.28×10^{4}	26	15	14	93.33
Design IV	12	1.48×10^{20}	13	11	11	100.00
Design V	29	1.13×10^{15}	110	38	27	71.05
Design VI	107	1.18×10^{21}	176	118	106	98.83
Design VII	110	2.53×10^{176}	111	109	109	100.00
Design VIII	95	1.30×10^{33}	213	135	101	74.81
Design IX (top-level)	10	6.40×10^{1}	12	2	2	100.00
Design IX (autocorrelation)	160	1.36×10^{39}	1248	141	85	60.28
Design IX (quantisation)	60	3.25×10^{32}	215	134	46	34.33
Design IX (coefficients)	75	1.36×10^{39}	11 127	2169	67	3.09
Design IX (transformation)	5	1.64×10^{4}	40	17	9	52.94
Design X	65	4.21×10^{65}	9843	1751	93	5.31
Design XI	23	1.58×10^{29}	—	—	—	—
Average						67.94

Table 6.1: Number of hardware basic blocks — There is an incredible large number of output symbols possible for each process. The majority of hardware basic blocks that are possible in principle must not be identified. For some of the identified basic blocks no code must be generated, since they are zero-strength or invalid basic blocks. Not all hardware basic blocks for which code has been generated are actually executed during simulation of the BAC++ model. The usage denotes the relation between used and generated basic blocks.

Not for all basic blocks that are analysed, corresponding code must be generated in the BAC++ model. Invalid as well as zero-strength hardware basic blocks can be discarded once they had been identified as such. However, their analysis still requires some computation time and thus slows down the identification process. Zero-strength hardware basic blocks are comparatively simple to identify, since no data path must be analysed at all. Identifying invalid basic blocks is more time-consuming, as its invalidity might be not detected until analysis of all active paths of the particular hardware basic block is completed. Again, the number of discarded hardware basic blocks highly depends on the way the data path is used by the controller and cannot be derived from the number of states or the number of output symbols of a process.

Even from the greatly reduced number of generated hardware basic blocks, not all of them are actually used during simulation of the BAC++ model. This has multiple reasons. First, the identification process might identify hardware basic blocks that can never be activated, since the activating condition of a particular basic block can never evaluate to `true`. Second, it might be the case that the use case that is used for simulation does not provide a full-coverage of the data path and thus might not activate all hardware basic blocks required for a complete replication of the control-flow and data path. Having an average usage of about 70 %, it is safe to say that nearly all hardware basic blocks, which are generated are actually used during simulation.

Design XI deserves special attention. As can be seen from Table 6.1, the basic block identification was not possible for this particular design. The number of hardware basic blocks that must be identified amounted to several millions, which is not manageable by the presented approach. This is true, even if the previously mentioned optimisation approaches are applied. A deeper inspection of the design reveals that the state, which is responsible for decoding the instruction that is actually executed by the *microprocessor without interlocked pipeline stages (MIPS)* processor has a very complex output logic. In that single state, 178 control signals are driven, which depend on 5202 output conditions, each one considering three Boolean atoms.

6.1.2 Identified Hardware Basic Blocks

The second step for evaluating the hardware basic block identification process considers the basic blocks itself. The histogram in Figure 6.1 on the following page shows the distribution of RT components per hardware basic block i. e., how many basic blocks contain how many RT components. It is important to note, that the number of RT components relates to the number of operations that are required for simulating the design's behaviour. RT components active due to parasitic functionality are not considered in the figure. It must also be noted that designs with more basic blocks contribute more to the figure than designs with less basic blocks. Generally speaking, the more RT components a hardware basic block contains, the more complex the generated model will be. But since all RT components are considered at once, the high-level estimation will also be faster. Because a hardware basic block is implemented as a single C++ method, as described in Section 5.4.1, basic blocks comprising more operations provide a better optimisation potential to the compiler. A more detailed analysis of model efficiency is given in Section 6.4.3.

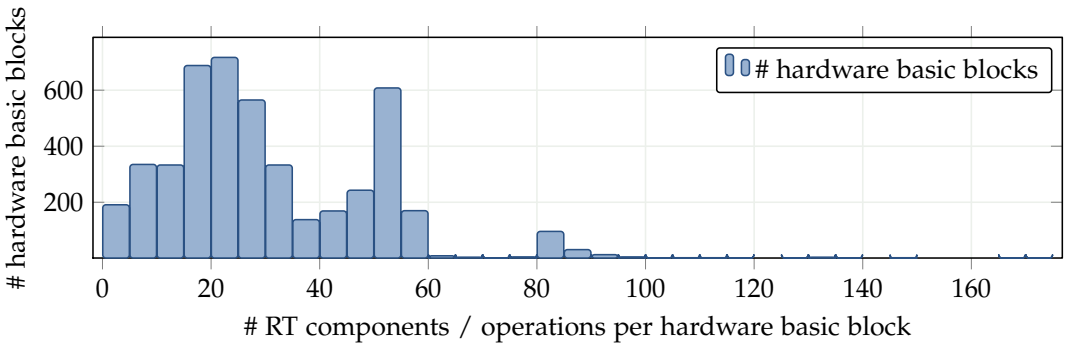


Figure 6.1: Histogram of RT components per hardware basic block — A hardware basic block typically contains 10 to 60 RT components, which are required for a behavioural simulation. Some hardware basic blocks comprise up to 170 RT components.

As just mentioned, the histogram in Figure 6.1 allows a statement about the expected complexity of the generated model. For statements about estimation granularity and therefore estimation accuracy, the histogram is not suitable, since it contains zero-strength operations such as assignments or constant shifts by a power of two. These zero-strength operations can be easily implemented in hardware and have only a minor or even no effect on the power dissipation. Also an important factor during power estimation is the kind and bit width of an operation. An adder with a small bit width will dissipate significantly less energy than a multiplier with a larger bit width. Therefore, another metric must be chosen for making statements about the estimation granularity.

Figure 6.2 shows the distribution of area, activated per hardware basic block. It can be assumed that a larger active area will result in a larger amount of energy dissipating during activation of the basic block. Since all designs are synthesised using the same technology and synthesis settings, activated area can be compared among the individual designs. There is large variation between the areas, activated by a single hardware basic block. Thus, hardware basic blocks provide a good level of granularity. They will therefore provide accurate estimates as shown in the following Section 6.2.

Summarising it can be stated that hardware basic blocks provide a good level of granularity. Large hardware basic blocks provide a good speed-up, since many operations are considered at once. More details on this are discussed in Section 6.6. Variation between the hardware basic blocks in terms of activated area provides a good level of accuracy, as shown in Sections 6.2.1 and 6.2.2, respectively.

6.2 Design Characterisation

This section focuses on the characterisation of the identified hardware basic blocks and other design artefacts like the clock-tree or the controller. It evaluates how well the generated traces that are obtained from a BAC++-based simulation fit the original trace, obtained from an estimation using PowerOpt. Evaluation is done with respect to the absolute power and

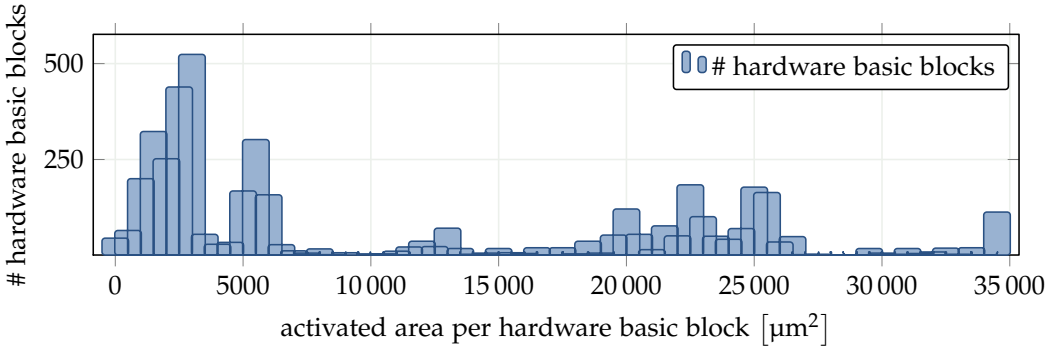


Figure 6.2: Histogram of active area per hardware basic block — There is a large variation in activated area between the hardware basic blocks and thus in the energy dissipated by a single basic block. Some hardware basic blocks activate up to 140 000 µm², which is about 24 % of the total area of that particular design. For a better readability of the figure, the right outermost values have been cropped.

energy values, respectively as well as for the shape of the generated traces. Both hardware basic block characterisation processes are evaluated individually, allowing a comparison of both approaches. In a third part, the characterisation of other non-functional properties like clock-tree or controller power is evaluated.

Evaluation is done using the examples explained in detail in Appendix C. First, a high-level synthesis is performed in order to create a specific design implementation. This implementation is then characterised using the power measures obtained by PowerOpt. A different set of input stimuli is then used to perform an estimation of the design. This estimation is done twice. First, using PowerOpt and second using the generated BAC++ model. These two traces and estimation results are then compared to each other. This complete procedure again is done twice. First, using the simple characterisation process and then using the advanced one. Non-functional properties like the clock-tree or the controller will give same results, since they are characterised equally. Dynamic power dissipation due to execution of the functional behaviour however, will give different estimates. More details of the evaluation conditions and results for individual designs can be found in Appendix D. It is worth mentioning that simulation of the BAC++ model automatically performs a regression by comparing the computational result obtained by the BAC++ model with the result obtained from the C/C++ input model. This assures the functional equivalence of the input, the RT-level, and the BAC++ model.

In order to allow a comparison with existing approaches like the ones mentioned in Chapter 3 and those that will be published in the future, standard error measures are used for evaluation. Most papers and articles, discussed in Chapter 3, provide only relative error measures of the total power dissipation per use case, even though the used power models provide more fine-grained information. Only a minority of the papers and articles provide a cycle-per-cycle error measure, making a deep comparison of the proposed approach with the existing ones difficult. Anyhow, evaluating the accuracy of the approach presented in this thesis is done using five different error measures, covering errors for total as well as cycle-by-cycle power dissipation. The measures can be split into two different classes.

Absolute error measures allow a comparison of the traces generated by BAC++ with the traces obtained from PowerOpt with respect to the particular design. That is, they have the same values range and unit as the traces themselves. They play a major role when planning the power supply for a given design, because the planning must allow for a safety margin. While allowing statements about a particular design, these types of error measures cannot be compared among different designs.

Relative error measures however, allow a comparison of the presented characterisation processes for the different designs, used for evaluation. The relative measures allow a statement of how well the presented approaches behave for the individual designs. This information can then be used to identify certain types of classes of designs where the characterisation is satisfying or erroneous, respectively. For the purpose of evaluating the characterisation techniques presented in this thesis relative measures are better suited.

For all error measures explained below, it is assumed that a trace x contains N samples. Variable x_i denotes a value obtained from the BAC++ model while $\hat{x}_i$ denotes a value obtained from the estimation using PowerOpt. All error measures are given in order of increasing complexity.

The most simple error measure is the plain *relative error (*RE*)*. As the name suggests, it gives the relative difference between two traces, where all elements of each trace have been summed up. Positive and negative errors per clock cycle will average out each other. This is apparent from the measure's definition, shown in Equation (6.1). Of course, this type of error measure is only valid for traces of time-invariant units such as energy in contrast to power dissipation. In other words, this measure can be used for comparing total values like total energy dissipation. It is used for evaluating the characterisation of non-functional properties like clock-tree or controller power, for example. These properties have comparatively simple characterisation models. Therefore, a more detailed analysis is not required.

$$\mathrm{RE}(x) = \frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)}{\sum_{i=1}^{N}(\hat{x}_i)} \tag{6.1}$$

More complex and more meaningful than the RE is the *mean absolute percentage error (*MAPE*)*. Its definition is shown in Equation (6.2). It is the average relative error per clock cycle. By computing the relative percentage error before averaging, all absolute errors are equally weighted. This is the basic measure for comparing both characterisation processes. It gives a good statement about the quality of the characterisation processes since it is a comparable measure of the per-cycle-accuracy of the generated traces.

$$\mathrm{MAPE}(x) = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{x_i - \hat{x}_i}{\hat{x}_i}\right| \tag{6.2}$$

The *root mean square error (*RMS*)* measure, as shown in Equation (6.3), introduces a weighting to the measurement. By squaring the error per cycle, large errors are weighted more than smaller ones. The general assumption when using this error measure is that small and medium errors are caused by minor and thus ignorable effects, while larger errors are a better indicator for the general quality of the approach. For the hardware basic blocks introduced in

this thesis, RMS weights the error of basic blocks dissipating a lot of energy and thus having higher absolute errors higher than the error of hardware basic blocks dissipating only a small amount of energy. By squaring and then extracting the root again, unit and values range are kept and also only absolute values are used. Moreover, assuming a Gaussian distribution of the errors, the RMS can draw conclusions about the standard deviation.

$$\mathrm{RMS}(x) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2} \tag{6.3}$$

Like RMS, the *relative root mean square error (*RRMS*)* weights larger errors more than smaller ones. This measure, shown in Equation (6.4) allows comparing different errors from different approached and designs independent from the values ranges of the particular design.

$$\mathrm{RRMS}(x) = \frac{\sqrt{N\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}}{\sum_{i=1}^{N}|\hat{x}_i|} \tag{6.4}$$

The final error measure is the *coefficient of multiple determination (*R^2*)*. This measure, shown in Equation (6.5), is a special case differing from the ones mentioned above. It is a measure of how good the BAC++ curve fits the trace, obtained from PowerOpt. In other words, it is a measure how similar the shapes of the traces are. It can be seen as percentage of how well the BAC++ trace models the variation of data in the trace obtained from PowerOpt. The higher the value is, the better the traces fit. It is important to note that meaning of R^2 depends on the variation of the reference data i. e., the values obtained from PowerOpt. If the reference data has only a very limited variation, a large and thus good value for R^2 will be achieved, even though the model is not correct.

$$\mathrm{R}^2(x) = 1 - \frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{\sum_{i=1}^{N}\left(x_i - \frac{1}{N}\sum_{j=1}^{N}(\hat{x}_j)\right)^2} \tag{6.5}$$

All error measures mentioned above consider each individual clock cycle while computing the measure. Since the power estimation presented in this thesis allows very long and extensive simulation runs, a very large number of clock cycles can be simulated. The generated traces will thus be very long. Since for such long simulations typically no cycle-by-cycle analysis is performed, the presented approach allows sampling. This is described in detail in Section 6.2.4. During sampling, several cycles are considered at once, reducing the total amount of data in the traces. If multiple clock cycles are considered together, individual error might average out each other. This in turn will reduce the error of the estimation. In other words, if sampling is suitable during design evaluation, a lower estimation error can be assumed. The following evaluation assumes that no sampling is applied at all. In other words, the figures and tables below show the worst-case. However, for the simple as well as for the advanced characterisation process, the influence of sampling along with more detailed tables and figures considering the different power traces and error measures for all individual designs are given in Appendix D.

6.2.1 Simple Characterisation

The error measures obtained using the simple characterisation process are shown in Table 6.2. All measures are obtained with no sampling applied. This provides the highest temporal resolution, but will also give the maximal error for each error measure. It shows that the relative error for total energy dissipation is negligible for all designs. This is because for total energy dissipation all hardware basic blocks' individual errors are evening out each other. All other measures show a variation between the individual designs. Using the simple characterisation approach, an average MAPE of 22.98 % can be achieved.

	energy trace	power trace				
	RE [%]	mean [µW]	RMS [µW]	RRMS [%]	MAPE [%]	R^2 [%]
Design I	0.23	227.27	30.00	11.20	7.60	89.43
Design II	0.34	24.03	20.00	78.69	36.06	15.32
Design III	0.02	281.27	30.00	11.40	9.50	89.46
Design IV	0.14	1712.65	140.00	8.38	5.53	95.36
Design V	0.33	257.15	70.00	27.04	23.38	39.58
Design VI	0.49	314.55	110.00	33.42	37.29	18.59
Design VII	0.18	5003.72	970.00	19.43	16.65	87.58
Design VIII	0.96	412.20	200.00	47.50	46.66	40.86
Design IX	2.92	832.83	190.00	22.90	21.27	52.02
Design X	0.21	885.45	240.00	27.34	25.90	24.30
Average	0.58	995.11	200.00	28.73	22.98	55.25

Table 6.2: Error evaluation (simple characterisation)

Using a single number for making a statement about the accuracy of the simple estimation approach is not sufficient. Figure 6.3 shows how the relative error is distributed over all simulated clock cycles of all examples. This representation has the advantage that hardware basic blocks that are activated more often are weighted more than basic blocks that are activated rarely. Such a histogram is built for each design individually. The individual error-per-cycle distribution for each example can be found in Appendix D. In a second step, all these individual histograms are merged to create Figure 6.3. During merging, all evaluation designs a weighted equally. In can be seen in the figure that about 22.97 % of all simulated clock cycles have a relative error of less than 5 %. As many as 46.79 % of all cycles have a relative error that is less than 15 %.

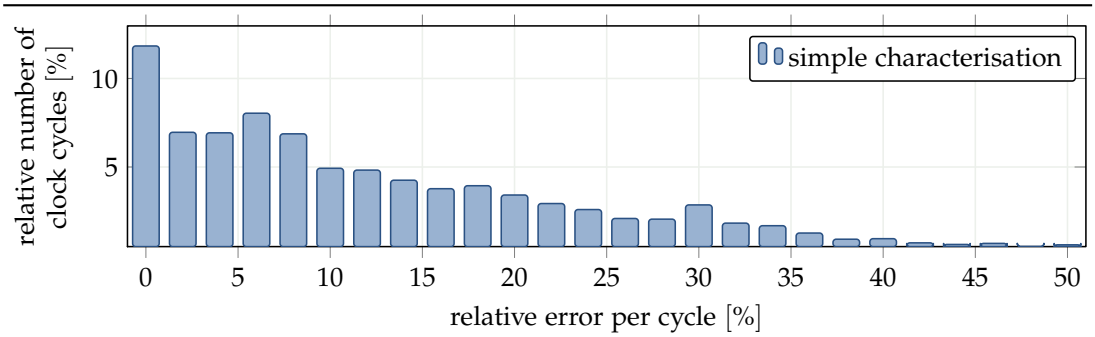


Figure 6.3: Distribution of the relative error per clock cycle (simple characterisation)

6.2.2 Advanced Characterisation

The advanced characterisation process has been evaluated using the same data sets for characterisation and evaluation as has been used for evaluating the simple characterisation process. Table 6.3 shows the error measures for all evaluation designs.

	energy trace	power trace				
	RE [%]	mean [µW]	RMS [µW]	RRMS [%]	MAPE [%]	R^2 [%]
Design I	0.10	227.27	20.00	8.87	5.46	93.36
Design II	4.22	24.03	20.00	98.65	50.10	-33.10
Design III	0.01	281.27	30.00	9.31	7.14	92.97
Design IV	0.14	1712.65	110.00	6.26	4.22	97.41
Design V	2.35	257.15	70.00	27.37	22.81	38.07
Design VI	1.58	314.55	100.00	31.56	34.63	27.38
Design VII	0.06	5003.72	250.00	5.05	3.73	99.16
Design VIII	0.94	412.20	140.00	35.04	19.98	67.81
Design IX	4.52	832.83	180.00	21.48	14.79	57.80
Design X	0.31	885.45	210.00	23.39	19.74	44.57
Average	1.42	995.11	113.00	26.70	18.26	58.54

Table 6.3: Error evaluation (advanced characterisation)

Results are very similar to the ones from the simple characterisation, but accuracy has been improved for almost all designs, with an average MAPE of about 18.26 %. Only exception is Design II, where an increased error can be observed. This is mainly due to the comparatively large scaling factor. More details on the particularities of Design II are given in Section D.2. If Design II is not considered, the advanced characterisation provides an average MAPE of about 14.72 % and an R^2 of about 68.72 %.

Again, a histogram showing the average error per clock cycle of all designs has been generated. It is depicted in Figure 6.4. Comparing Figure 6.4 to Figure 6.3 shows the advantage of the advanced characterisation process over the simple one. About 31.45 % of all simulated clock cycles have a relative error of less than 5 % and about 58.07 % of all cycles have a relative error of less than 15 %.

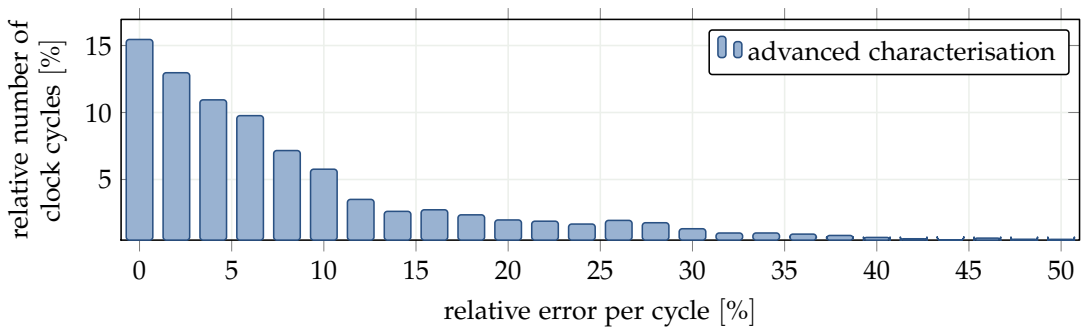


Figure 6.4: Distribution of the relative error per clock cycle (advanced characterisation)

6.2.3 Non-Functional Properties

Although non-functional properties like clock-tree, static power dissipation, etc. have already been considered during evaluation of the power traces, they can be evaluated separately. Due to their simple underlying estimation model, only total energy must be considered. Furthermore, besides the breakdown for total energy, PowerOpt does not provided a detailed cycle-by-cycle evaluation of the non-functional properties.

The detailed evaluation in Tables D.1 to D.10 shows that characterisation of non-functional properties like clock-tree or controller power is quiet accurate. This is because of the simple model used in PowerOpt, which can be easily rebuilt by the BAC++ model.

Regarding the interconnect, there is a small difference between the estimates obtained from PowerOpt and BAC++, respectively. PowerOpt internally uses a data-dependent power model for estimating the interconnect. The estimated value however, is averaged over all clock cycles in PowerOpt's estimate. During design characterisation, only this average value is taken into consideration and the value is assumed to occur in each clock cycle. Using a different data set during evaluation than during characterisation might cause the average power value to be different.

Considering static power dissipation, a large variation of the relative error between 0.00 and 6.47 % can be noticed. Since static power contributes only a small fraction to the overall power dissipation. Therefore the error as well as its variation can be neglected.

6.2.4 Summary

The previous sections showed that the simple as well as the advanced characterisation process provide good estimates, wherein the advanced characterisation is superior to the simple one. This advantage in accuracy comes with a higher computational effort during characterisation, as discussed in Section 6.6.

Tables 6.2 and 6.3 on the previous pages show a high variation of the error measures between the individual designs. This is caused by the structure of the designs. Especially for the smaller designs it might occur that a single side-effect becomes very dominating. Such an example is Design II, whose modulo-operation dominates its power dissipation. Also, the design comprises some special cases like small hardware basic blocks or multi-cycling, which are discussed in detail in Sections 6.5.4 and 6.5.5, respectively.

Characterisation of non-functional properties like clock-tree, static power dissipation, etc. is the same for the simple as well as the advanced characterisation process. The breakdown for total energy in Tables D.1 to D.10 shows that the error is negligible for all types of non-functional properties.

More interesting is the estimation of dynamic power dissipation. Evaluation shows that both approaches provide good estimates for registers as well as functional units. The simple characterisation has a slight advantage over the advanced one, regarding total energy dissipation. This is due to the characterisation technique, which uses average switched capacitance. The advanced characterisation process however, takes the context of the activation i. e., the hardware basic block into consideration. This will slightly increase the error for total energy dissipation, but significantly improves the cycle-by-cycle estimation accuracy.

Most effort has been spent for evaluating the power traces. Different error measures have been applied, allowing a good assessment of the achieved accuracy. Tables 6.2 and 6.3 show the error measures, obtained from the power traces. It can be seen that an acceptable error can be achieved as well as that the shape of the trace can be modelled accurately. Again, the advanced characterisation process has an advantage over the simple one.

On closer viewing, it can be assumed that complex designs with larger hardware basic blocks provide a smaller error, since more RT components per basic block will even out better. It can be seen in Figure 6.5 that this is not true for all designs. As described in Section 6.2.1, the error per cycle might depend on the actual execution phase of the simulated design. Applying a sampling window while within such a phase does not reduce the error significantly. This is particular evident for Design IX, for example, whose power trace is depicted in Figure D.25 on page 194.

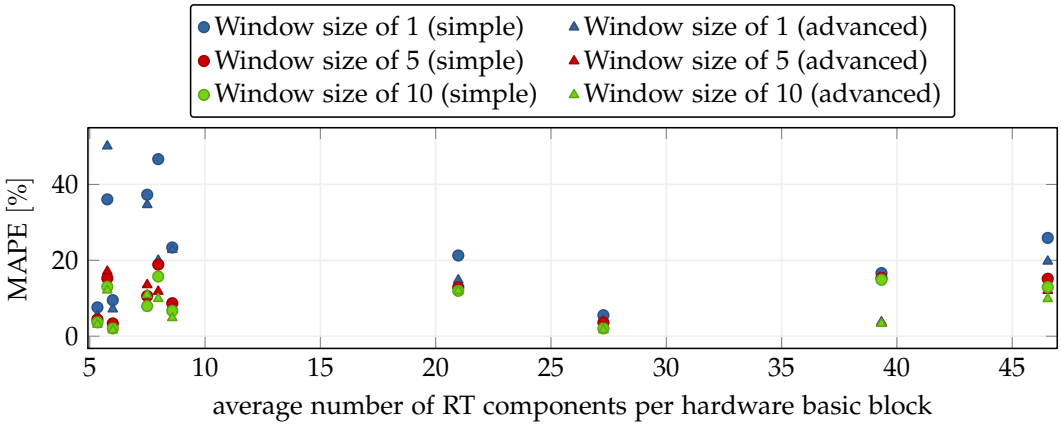
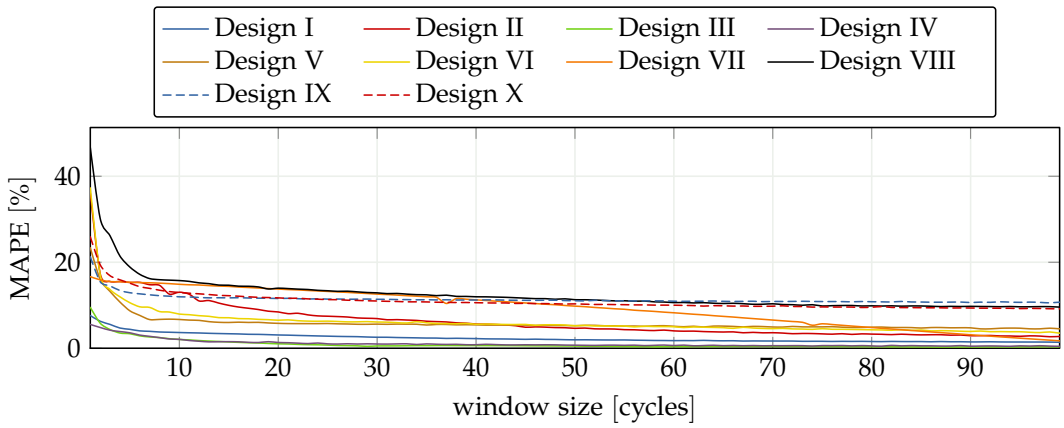


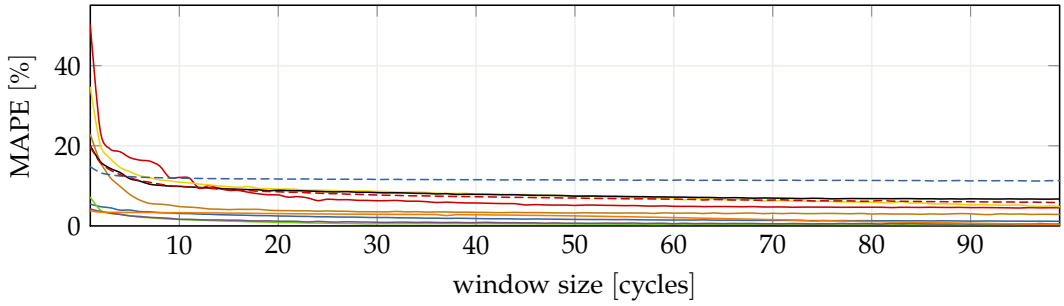
Figure 6.5: Average error per sample vs. average basic block size — The marks are organised in columns. Each column represents one design. The average relative error per cycle is given for different sampling window sizes (colour-encoded) and for both characterisation processes (shape-encoded). It can be seen that the error decreases with increasing window size and that the advanced characterisation has a smaller error. There is no negative correlation between error and the number of RT components per basic block.

This effect is reinforced by the fact that the power, which is dissipating during execution of a hardware basic block, does not only depend on the operations that are performed in that basic block, but also on the patterns that had been applied to the RT components in the previous clock cycle and thus by a different basic block. This can be seen from the right tables of Figure 4.12 on page 88.

The estimation error can be reduced by reducing the temporal resolution i. e., by applying sampling. The average error per sample depends on the size of the sampling window, as shown in Figure 6.6 on the next page. The figure shows that for most designs an acceptable error is achieved with a sampling window size of seven to ten clock cycles. Table 6.8 on page 149 shows that the speed-up achieved by the proposed approach allows very long simulations i. e., a considerable large number of clock cycles can be simulated. In this case, applying a sampling window is sufficient and also required for reducing the amount of generated data.



(a) Simple characterisation



(b) Advanced characterisation

Figure 6.6: Relation between average error per sample and sampling window size — The average relative error per sample depends on the width of the sampling window. The error decreases with larger window size. For most designs, an acceptable error is achieved with a window size of about seven to ten clock cycles.

From Figure 6.6 it is visible that the error can be reduced significantly by applying a sampling window. Table 6.4 shows the error measures known from Tables 6.2 and 6.3 but with having a sampling window of ten clock cycles applied.

Comparing the error measures obtained using the simple and the advanced characterisation process, it can be seen that the advanced characterisation provides better i. e., more accurate estimates than the simple one. Table 6.5 shows a detailed comparison of both approaches for all error measures. The improvement, with an exception for R^2, has been computed using Equation (6.6).

$$\text{improvement}(\text{measure}) = 1 - \frac{\text{measure}_{\text{advanced}}}{\text{measure}_{\text{simple}}} \tag{6.6}$$

	simple characterisation			advanced characterisation		
	MAPE [%]	RRMS [%]	R^2 [%]	MAPE [%]	RRMS [%]	R^2 [%]
Design I	3.65	4.51	82.12	3.11	3.73	87.81
Design II	13.03	18.35	22.70	12.11	17.95	26.07
Design III	2.05	2.53	93.95	1.63	2.07	95.93
Design IV	2.06	2.65	92.41	1.66	2.21	94.72
Design V	6.69	8.37	17.82	4.84	6.21	54.71
Design VI	7.94	9.58	59.34	10.93	12.67	28.92
Design VII	14.87	16.99	89.42	3.32	4.19	99.36
Design VIII	15.74	18.64	21.04	9.89	13.60	57.93
Design IX	11.96	13.50	66.61	11.94	15.49	56.02
Design X	12.95	14.11	40.58	9.85	11.75	58.79
Average	9.09	10.92	58.60	6.93	8.99	66.02

Table 6.4: Error evaluation (with a sampling window of ten clock cycles applied)

	accuracy improvement [%]				
	RE	MAPE	RMS	RRMS	R^2
Design I	58.04	28.13	33.33	20.74	4.39
Design II	-1149.33	-38.93	0.00	-25.37	-316.07
Design III	31.62	24.81	0.00	18.35	3.93
Design IV	-0.01	23.80	21.43	25.29	2.15
Design V	-608.22	2.44	0.00	-1.25	-3.83
Design VI	-223.80	7.13	9.09	5.55	47.29
Design VII	68.43	77.57	74.23	74.00	13.22
Design VIII	1.94	57.18	30.00	26.22	65.95
Design IX	-55.07	30.46	5.26	6.22	11.12
Design X	-44.60	23.79	12.50	14.43	83.43
Average	-144.49	20.55	43.50	7.06	5.96

Table 6.5: Accuracy improvement — The table shows the relative improvement gained when using the advanced characterisation technique instead of the simple one. All values are given as relative improvement and not in percent-points. The large decline of the accuracy for the relative error is explained by the very small values of the errors. Thus, even very little absolute differences will result in large relative variations. Most important columns are MAPE and R^2.

6.3 Power Modes

Quantitative evaluation of the provided power mode model is difficult, since such a feature is not supported by PowerOpt. Figure 6.7 shows a power trace for different power modes for the simple and the advanced characterisation, respectively. Power mode one is defined as ($V_{\mathrm{dd}} = 1.00\,\mathrm{V}, f_{\mathrm{clk}} = 100\,\mathrm{MHz}$), while the second power mode is defined by ($V_{\mathrm{dd}} = 0.90\,\mathrm{V}$, $f_{\mathrm{clk}} = 75\,\mathrm{MHz}$). Having the first one used during synthesis, it can serve as reference point for the α-power law, introduced in Section 4.6.1. Equation (4.45) on page 95 then gives the second power mode. It can be seen in the figure that the power mode transition takes place at around 1.20 µs. It lasts for about 125 ns and causes a power dissipation of 2.50 mW during that time. The corresponding power mode table can be found in Listing B.3.

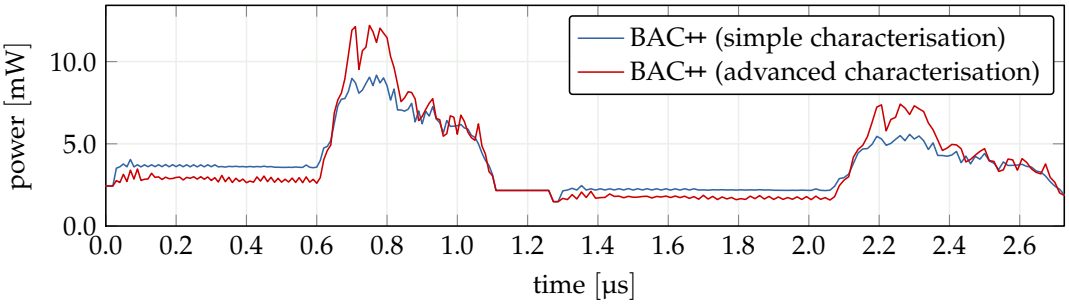


Figure 6.7: Power trace with respect to different power modes

6.4 Generated Model

Evaluation of the generated BAC++ models consists of two steps. First, the complexity of the models is assessed. In a second step the efficiency is evaluated. Before evaluating the models, their functional correctness must be shown.

6.4.1 Functional Correctness of the Generated Model

The structure and behaviour of the hardware basic blocks is derived from the RT data path. The static scheduling is done using a back-tracking algorithm, shown in Algorithm A.2. Along with the requirement that the data path must be cycle-free this yields a correct implementation. Since each hardware basic block describes the behaviour of exactly one single clock cycle, its timing is correct by construction.

The same applies to the design's controller. The BAC++ model generation directly derives the controller's cycle-accurate FSM from the RT-level description. Hence, the generated model's timing behaviour is correct by construction. The behaviour of the FSM is defined by the state and output transitions functions. Again, these are derived from the data path and thus correct by construction. During basic block identification various optimisations are performed, reducing the total number of basic blocks to be created. In order to assure that the generated

model is still functionally correct, an automatic regression test has been performed during simulation of the BAC++ model. The results of each run of the BAC++ model are compared to the results, obtained from the original C/C++ input model. If the regression test is passed successfully, the BAC++ model as well as the RT-level model can be assumed to be functionally correct.

6.4.2 Complexity of the Generated Model

Comparing the complexity of the generated BAC++ model to the complexity of the C/C++ input model gives an indication of the quality of the generated model. In other words, it allows a first impression of the expected compile and simulation time of the created BAC++ model. Additionally, it allows a statement about the expected readability of the generated code.

The generated BAC++ model is a representation of all controllers and their particular data paths, belonging to the design. Besides the behaviour, this representation also contains structural information as well as information about power and timing. This additional information has been added during synthesis and characterisation, respectively. Because information is added and a more fine-grained description of the behaviour is generated, it is expected that the generated model will be notably more complex than the original C/C++ input model.

Since the input as well as the generated BAC++ model are software-based models, metrics for measuring the software complexity can be utilised for determining the complexity of both models. Besides *lines of code*, there exists no common technique for measuring the complexity of a given software. There are several works on this subject, which all have a different focus. Most techniques aim at determining the readability and maintainability of the code [36] or how error-prone the implementation is [72]. Research shows that developers more often judge the complexity of a given software based on the data- instead of the control-flow [81]. Since most information which has been added by the model generation presented in this thesis, refers to the control-flow, a suitable measure must be found.

Since the BAC++ model is automatically generated from an RT-level description of a synthesised hardware module, it will be difficult to understand in any case. This is because the input model, generated by the syntheses tool is hard to understand in the first place. On the other hand, the generated model is not to be read by a developer. Instead, it is an intermediate description, resulting while creating the virtual system prototype. An adequate metric for measuring the complexity will therefore focus on the control-flow of the generated model, allowing statements on the complexity of the module's simulation.

In this thesis *McCabe's Cyclomatic Complexity* $\mathrm{v}(G)$ is used as metric for the models' complexity. It was introduced by McCabe [97] back in 1976 and directly measures the number of independent paths through the source code. Even though it is not free from critics [125], it is an often cited reference and thus allows comparing the evaluation results from this thesis with existing results from other approaches.

The measure allows statements about the control-flow within the given source code i. e., executable. Thus, it is a metric for the complexity of the CDFG. It is important to note that

this complexity refers to the complete model, which was generated, and not only to the modelled hardware module. The metric can be seen as an indicator for the performance of the simulation and the overall process, discussed in Section 6.6.

The measure $v(G)$ is defined in Equation (6.7), where E is the number of edges in the *control flow graph (CFG)*, V is the number of vertices, and P is the number of independent graphs i. e., procedures and methods.

$$v(G) = E - V + 2P \tag{6.7}$$

Table 6.6 shows the lines of code required for implementing the particular model as well as the model's complexity. The test bench is included in all counts. It is important to note that the Verilog and BAC++ model are automatically generated by PowerOpt.

	C/C++ input			Verilog	BAC++		
	LoC	NoM	$v(G)$	LoC	LoC	NoM	$v(G)$
Design I	70	1	6	917	770	14	89
Design II	96	1	9	1843	1430	17	230
Design III	97	1	8	1193	983	14	110
Design IV	199	1	16	3134	2211	28	156
Design V	133	1	15	2931	2512	27	322
Design VI	395	1	57	6492	6925	35	849
Design VII	583	1	15	25 240	11 907	17	737
Design VIII	273	1	37	21 714	6751	30	800
Design IX	589	1	75	16 360	99 238	56	31 998
Design X	624	1	73	13 236	126 118	42	19 791
Design XI	378	1	50	17 133	—	—	—

Table 6.6: Code complexity comparison — Lines of code (LoC) denote the number of non-blank, non-comment lines in the model. The number of modules (NoM) denotes the number of non-trivial modules such as classes. McCabe's Cyclomatic Complexity $v(G)$ describes the complexity of the model's control flow.

Comparing the code size and complexity with the number of identified and especially generated hardware basic blocks from Table 6.1 on page 126, it can be seen that there is a relation between the number of generated basic blocks and complexity of the BAC++ model. It can be seen that $v(G)$ is correlated to the number of states and the number of generated hardware basic blocks as well as other structural elements like memories and shared registers, for example.

Regarding the readability of the design's controllers it can be stated that the RT-level and BAC++ description are both equal easy to understand. Even if a different syntax is used, in both cases a controller is implemented by one large `switch` statement, as depicted in Listing 5.2 on page 110. Remembering Table 6.1 on page 126, it can be seen that there might be a very large number of hardware basic blocks per design. But each one is very simple in its structure as can be seen in Listing 5.1 on page 109. All basic blocks are implemented in terms of three-address-code. Each hardware basic block contains a large set of operations belonging together. The operations represent the complete computation in that particular clock cycle starting at the source register and ending at the target register. The way data passes through

the multiplexers is transparent i. e., the particular mux-select values had been resolved. In summary it can be said that the behaviour of a hardware basic block and thus the design, can be understood far more easily as it would be possible, if only a structural RT-level description in VHDL or Verilog would have been available.

6.4.3 Efficiency of the Generated Model

In Section 4.2.1 it had been argued that considering several RT components at once gains a speed-up during simulation. In the presented approach, all operations taking place in a certain clock cycle are modelled by one hardware basic block. Due to static scheduling and resolved mux-select values, some of the operations belonging to a basic block can be optimized or even removed by the compiler. An example for such an optimisation is given in line 12 of Listing 5.1 on page 109, where the simple assignment can be removed. Moreover, only RT components required by the behaviour must be considered during simulation. Components active due to parasitic functionality must be considered during characterisation and estimation but not during behavioural simulation. They are considered in the generated notification about switched capacitance, but not in the behavioural description of the basic block. Table 6.7 shows the number of RT components that is considered per basic block. It must be noted that Table 6.7 considers all generated hardware basic blocks and not only the activated ones.

		#RT components per hardware basic block					
		#simulated components			#estimated components		
	#HBBs	min	max	mean	min	max	mean
Design I	9	2	12	5.56	3	13	8.78
Design II	21	1	17	6.33	2	21	11.05
Design III	15	1	14	6.07	6	14	13.93
Design IV	11	11	47	28.00	82	94	96.55
Design V	38	2	28	9.63	24	29	28.03
Design VI	118	3	32	8.97	3	124	69.25
Design VII	109	3	170	41.09	3	862	526.35
Design VIII	135	2	23	8.43	3	115	78.36
Design IX	2463	2	68	21.14	2	92	75.33
Design X	1751	2	97	49.63	58	718	620.97

Table 6.7: Number of RT components considered at once — During execution of a hardware basic block several RT components are considered at once. Components active due to parasitic functionality are considered during characterisation, but must not be simulated when using the BAC++ approach. Only their switched capacitance is considered.

The speed-up that can be achieved with the generated models is discussed in detail in Section 6.6. Figure 6.8 on the next page however, shows the relation between the size of a hardware basic block in terms of considered RT components and the achieved speed-up. It can be seen from the figure that there is a relation between number of RT components considered at once during an activation of the hardware basic block and the simulation speed-up. However, especially the right outermost two marks of the figure show that there

are also other influences having an significant effect on the speed-up. Even though both designs consider roughly the same number of RT components per hardware basic block they provide a significantly different speed-up.

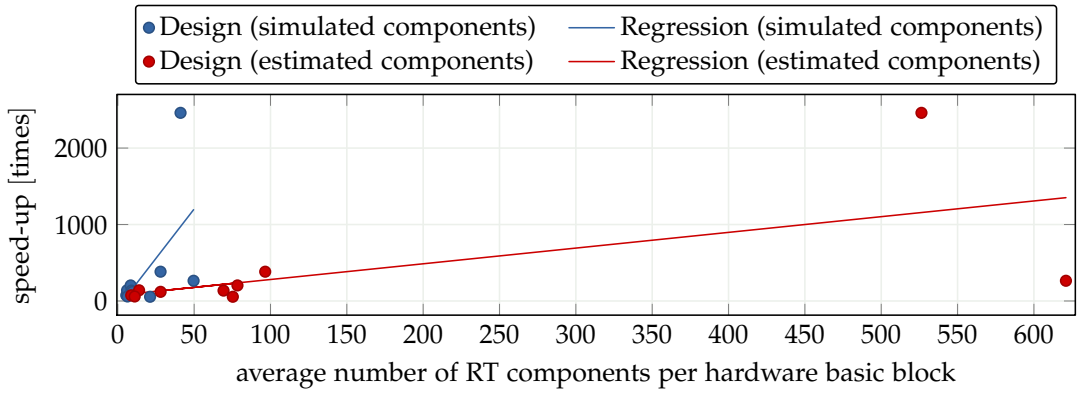


Figure 6.8: Average number of RT components vs. speed-up — The achieved speed-up depends on the size of the hardware basic blocks. The larger the basic blocks are, the larger the speed-up of the simulation will be. There are two reasons for this. First, more operations in a single hardware basic block provide better optimisation potential to the compiler. Second, large basic blocks reduce the relative overhead of the simulation kernel per simulation step.

The penultimate mark relates to Design VII, while the right outermost mark belongs to Design X. Both designs widely differ in terms of their control logic. DESIGN VII has a simple control logic with very simple output conditions. In other words, the decision which hardware basic block is to activate can be made fairly fast. Design X on the other hand has significantly more complex output conditions. Assuming that the generated hardware basic blocks are evenly distributed over all states of the controller's FSM, in each state of Design X the controller must choose one out of 27 basic blocks, which is to activate. In comparison, Design VII as exactly one basic block per state and therefore obviously a significantly simpler activation logic. A deeper investigation of Design X reveals that the hardware basic blocks are not evenly distributed. Instead, only a couple of states have complex output conditions. In these states however, the simulation logic has to choose one out of 256 basic blocks, requiring correspondingly complex output conditions, which in turn slow down the simulation.

6.5 Discussion of Known Problems and Issues

The sections above show that the level of accuracy as well as the simulation speed that can be achieved, depend on the structure and the BAC++ model of the estimated design. Different design properties have an influence on the accuracy of the obtained estimates. This is especially true for the error per cycle i. e., if no sampling is applied. The following sections describe in detail the most important design characteristics and artefacts, having an influence on the accuracy and simulation speed of the generated model.

6.5.1 Micro Controlling

Micro controlling denotes cases in which the controller performs control operations at a very fine-grained level. That is, operations are not chained directly, but a lot of control structures such as multiplexers are inserted into the data path. By adding a lot of this control logic to the data path, the complexity of the controller is increased, since it must control more components of the data path. Since the number of hardware basic blocks that must be identified and characterised highly depends in the controller's output conditions, a more complex output logic will yield a notable larger and more complex BAC++ model. In the following, this effect is discussed using the example of saturating arithmetic operations, which will cause micro controlling if not implemented correctly.

If a saturating arithmetic is used in the input model, it is important to implement this arithmetic in a branch-free manner. If branches are used like in the implementation in Listing B.1 on page 164, synthesis creates a very complex data path, like the one shown in Figure 6.9. Each operation is surrounded by a set of multiplexers, implementing the different branches of the saturation. That is, the controller is responsible for performing a micro-controlling i. e., control the saturation for each individual operation. If a large number of such saturating operations take place in parallel, it is obvious that the controller and its output logic are becoming very complex. Thus, a very large number of hardware basic blocks must be created to cover all possible control-flows – one basic block for each valid branch evaluation of all parallel operations.

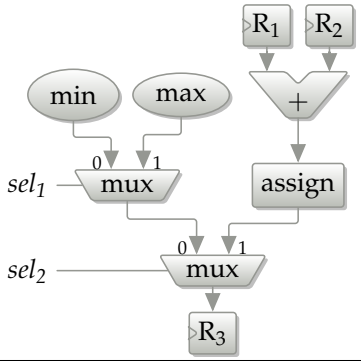


Figure 6.9: Operation saturation with branches

$$
\begin{aligned}
\#_H(n) &= \sum_{i=1}^{n} \binom{n}{i} 2^{2i} \\
&= \sum_{i=1}^{n} \binom{n}{i} 4^i \\
&= \sum_{i=0}^{n} \binom{n}{i} 4^i 1^{n-i} - 1 \\
&= (4+1)^n - 1 \\
&= 5^n - 1
\end{aligned}
\tag{6.8}
$$

For n operations that are possibly executed in parallel, Equation (6.8) gives the number of hardware basic blocks that must be analysed. The binomial coefficient $\binom{n}{i}$ gives the number of possibilities to activate i out of n operations, while 2^{2i} denotes the number of possible combinations of mux-select signals for the i active operations.

This exponential state explosion is an inherent problem of the static analysis and cannot be solved, since all hardware basic blocks must be identified for archiving a full coverage of the control flow. During basic block identification, only three possible combinations of mux-select signals remain per active operation, reducing the total complexity to $O(4^n - 1)$. These are valid hardware basic blocks. Hence, they must thus be identified, characterised, and become part of the generated BAC++ model.

There are two solutions to this problem. The first one relieves the controller from micro-controlling the saturating operation. In this case, a technology must be used for synthesis that provides saturating operators. These operators perform the saturation internally. Therefore, no controller interaction is required. A lot of research focuses on the problem of building saturating operators and their influence on the *signal-to-noise ratio (SNR)* of the processed data [40].

The second solution uses an implementation of saturation logic that is better suited to be implemented in hardware. An example implementation of the *branch-free saturation* is shown in Listing B.2 on page 165. This implementation reduces the micro-control overhead for the controller by providing an implementation using less conditional statements, yielding less complex output conditions.

6.5.2 Missing Information During Characterisation

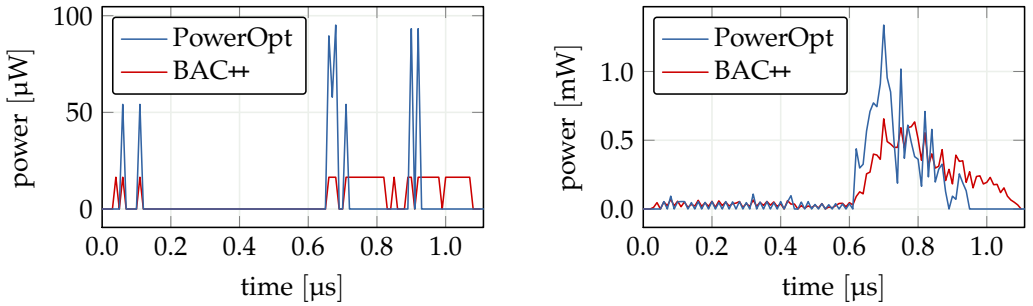
For determining the active times of a certain RT component, considering only value change sequences is not sufficient. This issue is discussed in detail in Section 4.4.5. An exact knowledge of a component's active times requires the active times of its in- and possibly its outputs to be known. It might occur that the number of activations of a certain RT operator cannot be estimated correctly. This case can occur at data in- and output ports, whose active time might not always be available in PowerOpt. If so, the number of activations must be obtained from the value change sequence. As mentioned in Section 4.4.5, if a port or any other RT component is activated twice, but with exactly the same pattern, this is not visible from the value change sequence. In this case, less activations are assumed than had actually occurred.

If the number of activations of a RT component is underestimated, this causes the average switched capacitance per activation and thus average dynamic power dissipation of that particular component to be overestimated. This is visible in the detailed evaluation in Tables D.1 to D.10 on pages 178 to 196.

6.5.3 Fuzzy Estimates

The errors shown in Sections 6.2.1 and 6.2.2 origin from the way the characterisation is performed. This is especially true for the simple characterisation process. The simple characterisation process uses average switched capacitance per activation for characterising a hardware basic block. Because an RT component might be used differently in different hardware basic blocks, the estimated power dissipation becomes fuzzy. Therefore, the dissipating power is overestimated in phases with a low utilisation, while in phases with high utilisation the power dissipation is underestimated. This effect is shown in Figure 6.10.

It can be seen in Figure 6.10a that switched capacitance and thus dissipating power is averaged over all activations of the particular RT component. If more components are considered at the same time, this effect is evening out, but not completely. This is shown in Figure 6.10b. This particularly applies to designs in which the components are used in different phases, such as Design VIII or Design IX. For the later one whose power trace is depicted in Figure D.25 on page 194 this effect yields a very fuzzy power trace.



(a) Single register

(b) All registers

Figure 6.10: Register power trace — As shown in Figure 6.10a, a certain register of Design VII is activated 36 times. The power, dissipating per activation, depends on the data stored in the register. This can be seen in the PowerOpt trace. Using BAC++, the register's total dynamic power dissipation is evenly distributed over all activations. If all 83 registers of the design are considered at once, power dissipation per clock cycle is averaging out over all registers. Yet, this is not complete, as shown by Figure 6.10b. The error is caused by the fact that registers are used differently by different hardware basic blocks.

While this effect does not influence the error for estimation of the total power dissipation, it has an impact on the average error per cycle. This can be seen in Table 6.2 on page 132, where designs which are sharing RT components over different phases have a higher error per cycle than designs where RT components perform a specific operation, only and where no or only a little resource sharing is introduced during synthesis.

The same problem occurs when using the advanced characterisation process. Again, the values are averaged. But in contrast to the simple approach, this is done on a far more fine-grainer level. Switched capacitance is averaged with respect to the basic blocks, the particular RT component belongs to. This notably increases the accuracy as discussed in Section 6.2.4.

6.5.4 Small Hardware Basic Blocks

Small hardware basic blocks are hard to estimate, because the assumption that different power dissipation due to data-dependencies is averaged out in a hardware basic block is no longer true. In a small basic block, its total power dissipation depends only on a small number of RT components and their data-dependency. This is especially true, if the basic block contains a single dominating RT component, like it is the case for Design II. Although even dynamic power dissipation of larger basic blocks does not even out completely as shown in Figure 6.10 and discussed Section 6.2.4, smaller basic blocks tend to have a larger per-cycle-error than larger ones. This is especially true for the simple characterisation process.

Regarding simulation speed, it is obvious that smaller hardware basic blocks provide fewer possibilities for optimisation to the compiler. For smaller basic blocks, the advantage of the optimisation is not enough for compensating the overhead introduced by the simulation kernel. Thus, designs where small basic blocks make up the majority, have a smaller simulation speed-up than designs containing a large number of larger hardware basic blocks.

6.5.5 Multi-Cycle Operations

Time-consuming operations like multiplication, division or the modulo-operation might not always be performed in one single clock cycle. In order to meet the required timing and clock frequency of the design, their operation may span over multiple clock cycles in the schedule. Hence the name *multi-cycle operation*. Such a multi-cycle operation starts in one cycle, but its result is available some clock cycles later.

During RT-level power estimation, model abstraction, and power characterisation, the dynamic power dissipation of such a multi-cycle operation is assigned to one specific clock cycle. In PowerOpt the power dissipation is assigned to the cycle, were the operation starts. For BAC++ however, it is assigned to the cycle, the operation's result is consumed. This is because of the bottom-up basic block identification process. All three cases are visualised in Figure 6.11.

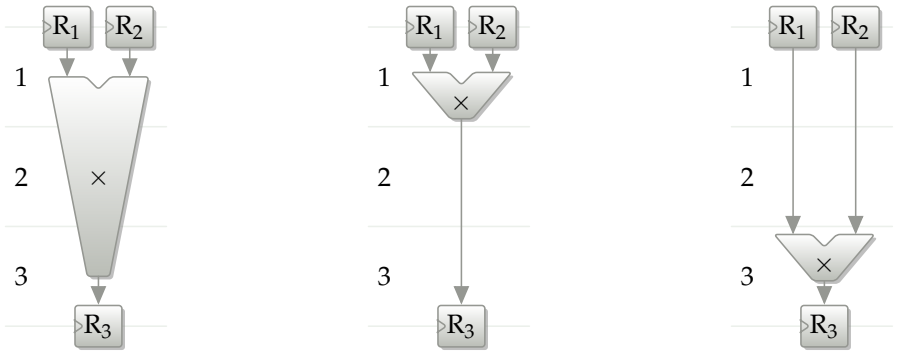


Figure 6.11: Dynamic power dissipation with respect to multi-cycle operations — An operation might be scheduled over multiple cycles in order to met the design's timing constraints as shown in Figure 6.11a. During power estimation, PowerOpt considers the operation i. e., its power dissipation to take place in the first cycle, the operation is active, as depicted in Figure 6.11b. Due to the bottom-up identification process, BAC++ considers the operation to take place in the last cycle, as shown in Figure 6.11c.

This yields two problems. The first one occurs during characterisation. The simple characterisation process is able to cope with multi-cycle operations, since all operations are taken into account, regardless when the individual operations take place. Regarding the advanced characterisation process this however, is no longer true. During the advanced characterisation process, only data patterns or pattern transitions to be precise, belonging to the timestamps the hardware basic block was active, are taken into account. That is, an operation is assumed to take place if its result is consumed. For multi-cycle operations this means that the power dissipation is assumed to take place at the end of operation. For the given example this means, that the pattern at cycle three are taken into account, even though the relevant transition occurs in cycle one. This transition is not taken into account during characterisation, which leads to an underestimation of the component's power dissipation, since the power dissipating in cycle one is not considered.

A very similar problem arises during simulation of the generated model. During simulation of the BAC++ model, the power of an operation is assumed to dissipate, if its result is consumed. For the example in Figure 6.11 this is cycle three, again. During power estimation using PowerOpt however, the power is assumed to dissipate in the cycle the component becomes active. That is cycle one for the given example. This causes different power traces to be obtained from a simulation using BAC++ and PowerOpt, respectively, if the updated value of a register is not used immediately. Again, estimation of total energy dissipation is correct, but power that is dissipating in one clock cycle is considered in another cycle. By applying sampling, this error becomes less important since multiple clock cycles are considered at once. Anyhow, if the multi-cycle operation is covered by different samples i. e., starts in one sample, but ends in another one, the problem persists.

6.6 Summary

This chapter has evaluated the proposed techniques and methodologies for automatically identifying and characterising combinational macros and the subsequent high-level model generation. Starting with the identification process, design characterisation and model generation, up to the entire design process, the efficiency and accuracy of the presented techniques and methods have been shown.

Despite some inherent problems, enumerated in Section 6.5, the identification process drastically reduces the number of combinational macros that must be identified. For Design VII the number of hardware basic blocks that has to be identified and generated within the BAC++ model could be reduced from theoretical 253×10^{174} down to 109. Even for designs with more complex output conditions and thus more valid basic blocks like Design IX, the number of basic blocks could be reduced from 5.71×10^{45} down to 22 158. The high coverage of the generated hardware basic blocks of about 67.94 % shows the efficiency of the proposed identification technique. A hardware basic block typically comprises 10 to 60 RT components, which must be functionally simulated in order to simulate the module's functional behaviour. Regarding power estimation, up to 862 components are considered at once, giving a significant simulation speed-up compared to a conventional RT-level simulation.

Both, the simple as well as the advanced characterisation process have been evaluated, too. A total of five different error measures were used for the evaluation. Both characterisation processes provide good results, with the first one is better suited for the estimation of the total energy and the second one is more suitable for the estimation of the power dissipation per clock cycle or sample, respectively. This is also evident from Figure 6.12 on the next page, which is a combination of Figures 6.3 and 6.4. It can be seen, that the advanced characterisation process simulates more clock cycles with a smaller relative error than the simple one. Simplified, the higher bars are on the left side of the figure, the more accurate the characterisation process is.

The average error of both characterisation techniques depends on the structure and complexity of the design being characterised. It can be stated that the average error per clock cycle is between 4.22 and 34.63 % for reasonable designs that had been characterised, using the advanced characterisation process. This error can be notably reduced for all designs and both characterisation processes by applying a sampling window during trace generation. That is,

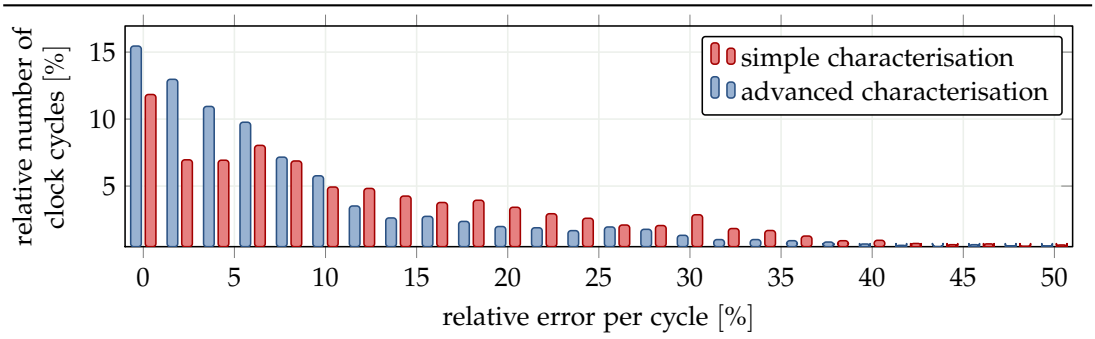


Figure 6.12: Histogram of the relative error per clock cycle

by considering several cycles at once. For long simulation runs, which become possible with the presented estimation approach, this even might be necessary in many cases in order to keep the amount of trace data manageable. With a comparatively small sampling window of seven to ten clock cycles, the average error per cycle can be reduced significantly to around 6.93 %. This effect is also visible regarding the RRMS, which can be reduced from 26.70 % down to 8.99 % on average. Both characterisation processes, but especially the advanced one, are able to reproduce the shape of the power trace very close.

As anticipated, the generated BAC++ model is notably more complex than the C/C++ input model. This is due to the structural and behavioural information that had been added during high-level synthesis. Additionally, power and timing information is also considered by the generated BAC++ model. The majority of the complexity, added to the model, origins from the static analysis of the design and the hardware basic blocks that must be generated for each possible behaviour of the data path. However, the generated model must be seen as an intermediate representation, which becomes part of the virtual system prototype. Even though the BAC++ model is not intended to be read by a developer, its behaviour is simpler to understand compared to the structural RT-level description, generated during synthesis. This is because a hardware basic block describes the complete behaviour in a specific clock cycle, with having all mux-select assignments etc. resolved. Thus, the BAC++ representation is more a behavioural description with added structural information than a structural description with added behavioural information like Verilog or VHDL.

In Section 4.4 it had been claimed that the proposed BAC++ model gains a significant simulation speed-up while still providing sufficiently accurate results. The proposed characterisation and model generation steps require an additional amount of computational effort, but the additional time spent is more than balanced out by the achieved simulation speed-up. This is true especially since characterisation is performed only once, while the simulation is carried out several times. For the examples used for evaluation, an average speed-up of 389.59 $\times$ could be archived, with some examples having a speed-up of up to 2460.71 $\times$. If this outlier is neglected, the average speed-up is still at a reasonable 158.81 $\times$. This more than compensates for the effort spent during design characterisation and BAC++ model generation. Table 6.8 gives more details on the computational effort of the individual steps of the proposed approach as well as the gained simulation speed-up.

	PowerOpt		BAC++				
				charc. and gen.			
	syn.	estim.	ident.	simple	advanced	estim.	speed-up
Design I	3.74	7.73	1.54	145.54	110.30	0.104	74.36
Design II	3.94	7.87	1.52	46.50	46.87	0.152	59.00
Design III	15.59	22.94	4.88	454.01	334.66	0.164	139.90
Design IV	43.45	3.76	0.19	0.83	45.57	0.004	383.03
Design V	24.83	8.62	1.83	67.33	139.38	0.072	119.76
Design VI	656.68	27.71	99.16	186.80	2471.00	0.204	135.84
Design VII	5405.44	72.57	137.65	61.60	1418.19	0.012	2460.71
Design VIII	415.05	26.05	132.28	38.52	1238.30	0.128	203.50
Design IX	485.56	22.79	544.57	28.58	477.16	0.412	55.31
Design X	2019.64	139.65	10 286.10	1399.50	156 542.00	0.528	264.48
Design XI	5.56	5.55	—	—	—	—	—
Average							389.59

Table 6.8: Computational effort — Time required for performing a given step in the estimation process i. e., synthesis and estimation using PowerOpt, hardware basic block identification, characterisation and model generation for both characterisation processes, and finally the estimation using the generated high-level model. All values, except for the speed-up are given in seconds.

As already stated in Section 5.9, the proposed process integrates seamlessly into the conventional synthesis and estimation process using PowerOpt. The tool is used as usual with an additional command to be executed. This command will then issue the automatic basic block identification, design characterisation, and BAC++ model generation. For simulating the virtual prototype, only minor changes to the C/C++ input model are required. The modifications are limited to emulation of the function call. Even if PowerOpt was used for demonstration, the general functionality and applicability of the proposed techniques and methods has been shown.

Chapter 7

Conclusion

Abstract

This chapter recapitulates the presented approach for a fast, yet accurate power and timing estimation of full-custom hardware modules, embedded in a heterogeneous system. It will give an overlook over the claims that had been made at the beginning of this thesis and it compares them to what have actually been achieved by the proposed approach. Evaluation results are put in connection with the proposed characterisation and model generation process.

This chapter is two-fold. First, the identification and characterisation techniques as well as the model generation are summarised. The second part then gives an outlook. It describes future work and possible improvements of the presented approach.

In this thesis, an approach for fast, yet accurate power and timing estimation has been presented. Based on an RT-level description, which has been obtained from a high-level synthesis, a power and timing augmented high-level model is generated. After a comparatively high characterisation effort, the presented process enables a qualified assessment of the given system. It guides the design space exploration and helps making relevant decisions, such as platform selection, hardware/software partitioning, task allocation, definition of communication structures, power management policies, etc.

The newly developed characterisation and model generation approach starts with an automatic identification of combinational macros or so-called hardware basic blocks, fulfilling claim 1 from Section 1.3. These basic blocks are statically analysed based on the controller's output and next-state logic and describe the behaviour of the data path in a particular controller state and while a specific input word is applied.

Static analysis is prone to a state explosion. It has been shown that using the straight forward approach, for one of the evaluated designs up to 253×10^{174} hardware basic blocks are required in order to achieve a full coverage of the control flow. By applying the developed techniques, this number can be drastically reduced. For the designs studied during evaluation, only a few hundred basic blocks on average are sufficient. For the just mentioned design, for example, only 109 hardware basic blocks have been generated. Put simply, it can be said that the number of hardware basic blocks per process has been reduced from $\mathrm{O}\left(2^{\#\,\text{controller ouput signls}}\right)$ to about $\mathrm{O}(n\,\#_{\,\text{states}})$. During simulation of the generated BAC++ model, about 70 % of the generated basic blocks are activated. The significant reduction of identified hardware basic blocks as well as the high usage of the generated basic blocks shows the efficiency of the proposed identification and model generation process.

For the characterisation of the identified basic blocks' dynamic power dissipation two different methods have been developed in compliance with claim 2. The first one provides a faster, while the second one provides a more accurate characterisation, regarding the cycle-by-cycle error. Other design artefacts such as clock-tree or controller power are also taken into consideration, fulfilling claim 3. As stated by claim 4, characterisation is done on structural properties, only. After characterisation, various information is available at different levels of granularity. This is visible from Figure 7.1.

Evaluation shows, that an acceptable estimation error can be achieved with the presented approach. The error regarding total energy dissipation is sufficiently small for all designs. Non-functional properties such as clock-tree and controller power, or static power dissipation can be estimated with a negligible error, which is typically around 1 %. The generated power traces provide a relative error of less than 15 % for most simulated clock cycles. For both characterisation processes, this estimation error can be significantly reduced by applying a sampling window during power estimation. That is, each clock cycle is not considered individually, but some cycles are considered at once. A sufficient error of about 6.93 % can be achieved with a sampling window with a width ten clock cycles. Since millions of clock cycles are simulated, applying sampling is acceptable, not to say necessary, in order to reduce the amount of generated data to a manageable size. For estimating the design's thermal behaviour as well as degradation effects, even larger sampling windows up to hundreds or even thousands of cycles are suitable.

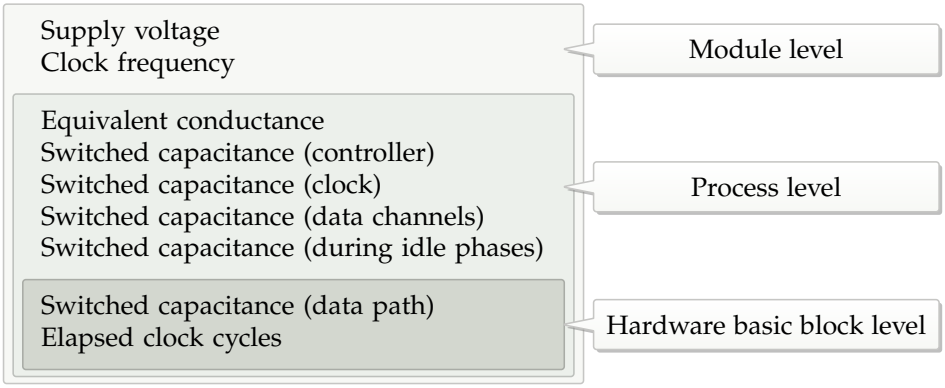


Figure 7.1: Spatial granularity of design properties — Different types of information are available at different levels of spatial granularity.

The identified hardware basic blocks along with the characterised design properties are used to generate an executable high-level model, which is augmented with power and timing information. This fulfils claim 5. The generated BAC++ model contains the description of the design's functional behaviour as well as characterised design properties such as switched capacitance. Only minor changes to the test bench are required in order to replace the original input model with the generated one. Due to the efficiency of the basic block identification process, the generated model has an acceptable size and complexity.

A C++ library is provided that contains all base classes for an easy generation of the virtual prototype. The library also contains the simulation kernel and tracing facilities. Using this library, the generated model can be easily compiled. Statically scheduled hardware basic blocks, which contain a large number of RT components i. e., a large number of operations, can be readily optimised by the compiler, yielding a simulation and estimation speed-up of about 160 $\times$, compared to a conventional RT-level estimation. Also a TLM-interface is provided, allowing the virtual prototype to be embedded into a virtual system prototype, as it is required by claim 6.

The proposed approach has been integrated seamlessly into the power optimising synthesis tool PowerOpt. Even though the presented approach relies on PowerOpt, all assumptions made and techniques presented in this thesis, can be easily adapted to other synthesis tools and design processes as well. The generated RT data path may be different, but the basic properties such as an FSM-based controller with an output and next-state logic, as well as the hierarchical structured processes with inter-process communication, can be found in almost all designs that are generated by a high-level synthesis.

Considering the complete power and timing estimation process as it has been proposed in this thesis, it can be stated that the process allows a fast and efficient simulation and estimation of embedded full-custom hardware module. Using the approach, it is possible to simulate millions or even billions of clock cycles within a few minutes while obtaining nearly RT-level accurate power and timing estimates. With this approach the highest abstraction level, at which power and timing can be predicted accurately, is raised by almost one entire level. All claims, enumerated in Section 1.3, have been addressed and met. The same is true for all requirements identified in Chapter 4.

7.1 Outlook

The presented work must be considered as a *proof of concept*. This is especially true for the implementation of the proposed methods. Although the approach has been implemented with a view to his future expandability and re-usability, a number of implementation improvements possible. An analysis of the of the implementation using different performance measuring tools reveals that at lot computation effort is required for reading, writing, and updating signals i. e., for managing the inter-process communication. In order to provide a homogeneous interface for accessing different types of signals and channels a corresponding class hierarchy has been developed. This polymorphic approach this leads to virtual function calls. These in turn require an additional access to the *vtable*, which selects the desired implementation of the method during run-time. A similar issue occurs while driving the process' FSM. A performance optimised implementation will provide a notable simulation speed-up.

Besides the aforementioned implementation details, improvements of the presented approach are possible. This thesis shows the general applicability of the proposed methods and techniques, but leaves some room for a number of improvements regarding identification and characterisation of the hardware basic blocks as well as generation and simulation of the BAC++ model. Some of these enhancements are listed below. They are ordered along the proposed process.

Improved Basic Block Identification

The basic block identification process can be extended to identify output conditions that can never evaluate to `true`, since the required inputs can never occur. That is, the basic block identification process considers the data path as an additional input. This will further reduce the number of hardware basic blocks. However, identifying invalid or impossible input words is an ambitious task, since it requires a deep knowledge of the design's underlying behaviour.

Improved Characterisation

The characterisation processes can also be improved. This is especially true for the comparatively simple characterisation of shared registers and memories. Currently, it is assumed that an access to the memory requires always the same amount of energy. There should be at least a distinction between read and write accesses. The same is true for all types of abstract data channels. Such a feature can be easily implemented, as soon as PowerOpt provides the necessary RT-level power models. The implementation requires only minor changes of the existing library. Only the notifications that are sent must be adapted. The remaining simulation kernel can be left unchanged. Existing BAC++ models can still be used, even if they then cannot benefit from the enhancement.

The currently applied characterisation of the interconnect can be enhanced, too. Currently, the interconnect is estimated by averaging its dynamic power dissipation over all clock cycles, because the interconnect power estimation is done by PowerOpt in this way. Since pattern

lists are available for each signal of the interconnect, it is possible to estimate the interconnect power with respect to the data patterns that are applied, if a certain hardware basic block is activated. This will be an important improvement, since the interconnect power may contribute to a great extend to the overall power dissipation, as can be seen from the detailed energy breakdowns shown in Appendix C.

Improved Spatial Resolution

The spatial resolution is an important factor, if the power trace is an input for a thermal estimation of the design. Currently, the spatial resolution of the estimated power dissipation relates to the size of the hardware basic block. The location of the basic block is defined by the location of the RT components it is built of. While this may be suitable for smaller hardware basic blocks, for large basic block covering a larger chip area this might be insufficient. Evaluation showed that up to 24 % of the total chip are can be activated by a single hardware basic block. In order to cope with this issue, the spatial resolution of the estimation must be improved. This can be done by subdividing the chip area into so-called *tiles*. During characterisation of a hardware basic block, the power dissipation per tile must be estimated and characterised. The most suitable size of the tiles must be determined for this purpose, in order to provide a good trade-off between accuracy and additionally required effort.

Minimisation of Output Conditions

For some designs, a large number of hardware basic blocks is been created. As mention above, this leads to correspondingly complex output conditions in the generated model. The conditions are given in *disjunctive normal form (DNF)*. This description of the Boolean expression is optimized by the compiler during compile time. Especially for larger compilation units this may take some time. Since the model may be compiled several times, the DNF optimisation should already be done during model generation. Well known algorithms like *Quine-McCluskey* can be used for optimising Boolean expressions. Such an algorithm can be implemented or one of the freely available libraries can be adopted and used during the model generation process.

Considering Integer Variables in Assignments

Regarding the output conditions, a second optimisation is possible. As mentioned, each assignment to a SMT predicate is represented as a single Boolean atom. For simple Boolean assignments this is a suitable approach. For assignments using the value of a register holding an integer variable this might lead to a large number of atoms – one for each possible integer assignment. It is obvious that it is not possible that multiple atoms, which all refer to the same integer assignment, can evaluate to `true` at the same time. In this case it may be useful to directly compare the value of the register, reducing the total number of Boolean atoms that must be considered during optimisation and evaluation of the generated output conditions.

Reuse of Behavioural Code

Also regarding the hardware basic blocks an optimisation is possible. A basic block is generated for each possible behaviour of the data path. However, two different basic blocks might share a subset of their operations. In this case the same code is generated for common operations of the basic blocks, increasing the size of the model and extending the compile time. This issue can be tackled by outsourcing the shared code into a separate method. This method is then used by each one of the two hardware basic blocks. In the final stage, each possible computation of a new register value is outsourced into its own method. However, information about switched capacitance is still associated to a particular hardware basic block.

Even though the size in terms of lines of code of the generated BAC++ model is reduced, some drawbacks might be introduced. During compilation, the compiler has two options. First, the compiler can inline the methods. This seems useful, since each method contains only a small amount of instructions. In this case, the compiler can use the same optimisation potential as in the original implementation. However, the resulting executable will also have the same size as it would have had, when the original implementation had been used.

The compiler can also choose to not inline the methods. In this case the executable will be notable smaller, but the compiler has less optimisation potential available. Also, intermediate results cannot be shared among the computation of different registers and race conditions may occur. By not inlining the method, function calls must be inserted, prohibiting an extended optimisation and moreover increasing the simulation time. The effect on code size and simulation time must be evaluated before a statement about the efficiency of this optimisation can be made.

Multi-Cycle Basic Blocks

Some computation may be too complex to be performed in one clock cycle. In this case the computation covers multiple cycles i. e., multiple states of the FSM. These states are then executed one after another, without any conditional branches. In this case, so-called *multi-cycle basic blocks* can be introduced. These cover multiple states of the FSM. The causal relationship is defined by the *computation tree logic (CTL)* Equation (7.1).

$$(s_i, a_i) \Leftrightarrow \mathrm{X}(s_j, a_j) \tag{7.1}$$

Statements about the assignments a are difficult, since they depend on the input symbol of the FSM. The input symbol in turn, might be unpredictable, since it depends on an external input. Even input from the data path is hard to predict due to data-dependencies. However, one corner case, shown in Equation (7.2), can be considered. In this case the output conditions are trivial, so that no evaluation of assignments is necessary.

$$(s_i, \mathrm{true}) \Leftrightarrow \mathrm{X}(s_j, \mathrm{true}) \tag{7.2}$$

The simplified relation $s_i \Leftrightarrow \mathrm{X}\, s_j$ can be directly obtained from the controller's FSM. In this case, state s_i has exactly one outgoing transition, leading to state s_j. Moreover, this transition

is the only incoming transition of s_j. Both states must activate exactly one hardware basic block. By combining multiple basic blocks, even more RT components are considered at once, providing an even better simulation performance. Even though the multi-cycle basic blocks will speed-up the simulation they will decrease temporal resolution of the estimate. The trade-off between accuracy and performance must be evaluated.

Hardware Basic Block Correlation

The dynamic power dissipation of a hardware basic block does not only depend on the data patterns that are applied in the current clock cycle, but also on the data patterns that were previously applied. This is because the RT-level power models rely on pattern transitions. In order to regard this effect during power estimation, correlations between hardware basic blocks can be considered. That is, the actual power dissipation of a basic block depends on the basic block(s) that were previously active. The additional effort required during characterisation depends on how many previous hardware basic blocks i. e., how many passed clock cycles are taken into consideration. Also, the more correlation must be considered, the more difficult it is to provide a use case with a full coverage of all possible correlations. Again, a good trade-off between characterisation effort and accuracy must be found.

7.2 Last But Not Least

As mentioned in the very beginning of this thesis, the history of semi-conductor technology is full of obstacles and challenges. Until now, almost all of them have been solved or they had become part of today's research objectives. This thesis also tries to make a small contribution for solving today's problems. By supplementing fast high-level power and timing estimation with nearly RT-level accurate results, problems such as thermal behaviour or degradation effects can be regarded during the design process. However, there are new challenges on the horizon.

For over 66 years, shrinking the technology's node size improved the performance of the systems. Transistors had become faster and more of them could be placed in the same chip area. As predicted, within the next 20 to 30 years, feature size will reach about 5 nm. At this point, a physical barrier is reached, which cannot be passed without developing completely new technologies. Quantum computing is one example for such an emerging technology. With these new technologies been developed right now, new challenges for power estimation will arise. With a new type of computation, new ways for estimating the power dissipation are required. The researching community will find new approaches and solutions for the arising problems and challenges. And over the years, with the new challenges becoming more and more prominent, this work will lose its relevance and quit the scene for the next generation of graduate students, ready to advance science with rapid strides. May they be successful. □

Appendix A

Algorithms

A.1 Hardware Basic Block Identification

input: States S of the controller's FSM
input: RT data path G
output: A list of identified hardware basic blocks H
foreach *state* $s \in S$ **do**
  foreach $a \in$ *all possible assignments* A *of state* s **do**
    $H^{(s,a)}$:= new hardware basic block for s and a;
    $V_{\mathsf{TR}}^{(s,a)}$:= all registers of G that take new values if a is applied in state s;
    $V := V_{\mathsf{TR}}^{(s,a)}$;
    while *elements left in* V **do**
      v := first element of V;
      S := all source components of v;
      foreach *source component* $s \in S$ **do**
        if $s \in \tilde{V}$ **then** continue;
        if s *is a register* **then**
          add s to $V_{\mathsf{SR}}^{(s,a)}$ of $H^{(s,a)}$;
        else if s *is a multiplexer* **then**
          add s to $V_{\mathsf{A}}^{(s,a)}$ of $H^{(s,a)}$;
          add the components at the active input of s to V;
        else
          add s to $V_{\mathsf{A}}^{(s,a)}$ of $H^{(s,a)}$;
          add the components at the inputs of s to V;
        end if
      end foreach
      add v to $\tilde{V}$;
      delete v from V;
    end while
    add $H^{(s,a)}$ to H;
  end foreach
end foreach
return H

Algorithm A.1: Hardware basic block identification

A hardware basic block is defined by a state s of the controller's FSM and the Boolean assignment a that activate the basic block. For each combination (s, a) of state and assignment a new basic block $H^{(s,a)}$ is created. Starting from the target registers $V_{\mathsf{TR}}^{(s,a)}$ the data path is traversed upwards, contrary to the data flow, and each passed component is added to the appropriate component set. Traversing through the actual path stops, if a source register is found. Identifying components that are active due to parasitic functionality is done the same way, but the data path is traversed down wards, starting at the source registers $V_{\mathsf{SR}}^{(s,a)}$. This is not shown in Algorithm A.1.

A.2 Basic Block Code Generation

input: A set H of hardware basic blocks to create code for
foreach $h \in H$ **do**
  clear P;
  create local copy of all $V_{\mathsf{TR}}^{(s,a)} \in h$;
  foreach $t \in V_{\mathsf{TR}}^{(s,a)}$ **do**
    `create(`*input of* t, P`);`
    write assignment $t :=$ input of t;
  end foreach
  write power notification for h;
end foreach

Algorithm A.2: Code generation for hardware basic blocks

Since a register can be source as well as target register at the same time, a local copy of all source registers is made and used as input for the computations. For each target register, the input is created using the recursive function `create`. A corresponding assignment to the target register is created, too. After the code for all behavioural computations has been created, code for notifying the power & timing model is generated.

input: A set T of inputs to create
input: A set P of already available inputs
foreach *input* $t \in T$ **do**
  if $t \in P$ **then return**;
  if $t \in V_{\mathsf{R}}$ **then return**;
  if $t \in V_{\mathsf{M}}$ **then**
    `create(`*selected input of* t, P`);`
    write assignment $t :=$ selected input;
  end if
  if $t \in V_{\mathsf{O}}$ **then**
    `create(`*all inputs of* t, P`);`
    write assignment $t :=$ `operation(`t, *all inputs of* t`);`
  end if
  add t to P;
end foreach

Algorithm A.3: Recursive creation of required input values

The given inputs are created using a *depth-first search* algorithm. Before a certain input is created, its own inputs are generated. Registers must not be created, since their values are always available. Each intermediate result is created only once and reused, if necessary.

Appendix B

Listings

B.1 Saturating Arithmetic (with branches)

```
#define MIN (-32768) /*0x8000*/
#define MAX (+32767) /*0x7FFF*/

int16_t add(int16_t a, int16_t b) {
  int16_t result = a+b;
  if (((a^b) & MIN)==0) {
    if (((result^a) & MIN)!=0) {
      if (a<0) {
         result = MIN;
      } else {
         result = MAX;
      }
    }
  }
  return result;
}

int16_t sub(int16_t a, int16_t b) {
  int16_t result = a-b;
  if (((a^b) &  MIN)!=0) {
    if (((result^a) & MIN)!=0) {
      if (a<0) {
         result = MIN;
      } else {
         result = MAX;
      }
    }
  }
  return result;
}
```

Listing B.1: Saturation with branches

A simple implementation of the saturation arithmetic uses three conditional statements per operation. Implementing this algorithms in hardware will cause the controller to perform micro controlling. That is, the controller will check if the actual computation has caused an overflow. Based on the direction of the overflow the controller will select the appropriate saturation value. This will lead to an unnecessary complex controller and a large number of possible hardware basic blocks in the second place.

B.2 Saturating Arithmetic (without branches)

```
#define MIN (-32768) /*0x8000*/
#define MAX (+32767) /*0x7FFF*/

int16_t add(int16_t a, int16_t b) {
  int16_t result = a+b;
  if (((((uint16_t)a>>15)^((uint16_t)b>>15))==0) &&  /* same signs? */
      ( ((uint16_t)a>>15)^((uint16_t)result>>15))) { /* overflow?   */
    result = (int16_t)((uint16_t)MAX + ((uint16_t)a>>15));
  }
  return result;
}

uint16_t sub(uint16_t a, uint16_t b) {
  int16_t result = a-b;
  if (((uint16_t)a>>15)^((uint16_t)b>>15) &&      /* different signs? */
      ((uint16_t)a>>15)^((uint16_t)result>>15)) { /* overflow?        */
    result = (int16_t)((uint16_t)MAX + ((uint16_t)a>>15));
  }
  return result;
}
```

Listing B.2: Saturation without branches

For a more advanced implementation of the saturation logic, shift and masking operators are used. This reduces the number of required conditional statements to one. This algorithm can be easily and more important efficiently implemented in hardware, resulting in a less complex output logic and thus significant less hardware basic blocks.

B.3 Example Power Mode Table

```
<?xml version="1.0" encoding="UTF-8"?>
<power_mode_table>
  <power_mode>
    <parameters>
      <parameter name="ID"              value="0"   unit="no"/>
      <parameter name="clock_frequency" value="100" unit="MHz"/>
      <parameter name="supply_voltage"  value="1.0" unit="V"/>
      <parameter name="avg_dyn_power"   value="0.0" unit="W"/>
      <parameter name="avg_leakage"     value="0.0" unit="W"/>
    </parameters>
    <power_mode_transitions>
      <pm_trans>
       <parameter name="pm_id"           value="1"           unit="no"/>
       <parameter name="switching_time"  value="0.000000150" unit="S"/>
       <parameter name="switching_power" value="0.002500"    unit="W"/>
      </pm_trans>
    </power_mode_transitions>
  </power_mode>
  <power_mode>
    <parameters>
      <parameter name="ID"              value="1"   unit="no"/>
      <parameter name="clock_frequency" value="75"  unit="MHz"/>
      <parameter name="supply_voltage"  value="0.9" unit="V"/>
      <parameter name="avg_dyn_power"   value="0.0" unit="W"/>
      <parameter name="avg_leakage"     value="0.0" unit="W"/>
    </parameters>
    <power_mode_transitions>
      <pm_trans>
       <parameter name="pm_id"           value="0"           unit="no"/>
       <parameter name="switching_time"  value="0.000000250" unit="S"/>
       <parameter name="switching_power" value="0.001250"    unit="W"/>
      </pm_trans>
    </power_mode_transitions>
  </power_mode>
</power_mode_table>
```

Listing B.3: Example Power Mode Table — Example XML file describing the power mode table used during trace generation for Figure 6.7 on page 138.

A power mode table is a conventional XML-file. Each power mode is specified in terms of an identifier as well as its associated supply voltage and clock frequency. Values for average dynamic power and average leakage are not required by the power mode model, but can be used for statically deriving power management policies, for example. Each power mode has also a list of possible power mode transition assigned, which are used to create the power mode state machine.

B.4 Simulation Script Configuration

```
#
# Configuration file for the COMPLEX DCT32 example
#
[GLOBAL]
CDB_path= $POWEROPT_HOME/cdblib
report_path=../simulation_reports
continue_mode=new
technologies=065nm
frequency_range=100 100 0
voltage_range=1.0 1.0 0
temperature_range=50.0 50.0 0

[DESIGN]
build_tag=dct32
src_path=../examples/complex_dct32/src
files=dct32.cc main.cc
build_cmd=exec make clean; exec make
run_cmd=./stimulus
mapped_mems=Design.dct32.this in Design.dct32.this out
optimisation=power
synth_effort=low
estim_effort=high
```

Listing B.4: Simulation script configuration example

The script, which performs the design space exploration is configured using a conventional INI-file. The file consists of several sections. The `GLOBAL` section defines all values that apply to all designs that are part of the design space exploration. Each design is specified in a `DESIGN` section, which can overwrite the settings from the `GLOBAL` section. Advanced settings for frequency, voltage, and temperature ranges allow to specify the range boundaries as well as the step size. This will ease the specification of possible design space alternatives. The script also allows for interrupting and continuing the design space exploration.

Appendix C

Tools, Designs and Settings Used for Evaluation

For simulating, estimating and evaluating the designs, different tools were used. Table C.1 lists them ordered along the presented estimation and evaluation process. All computations were performed on an AMD Opteron with four CPUs providing a total of 16 processor cores. The machine has 128 GB of RAM and was running a 64 bit Debian Linux 6 (Squeeze), which bases on an Linux kernel with version 3.2. Even though the machine provides a lot of processing cores, most tools, but especially the ones used for simulation, characterisation and estimation, are single threaded. Thus, only one processor core was used at a time.

Tool	Version
SMOG	1.0
PowerOpt	2010.2.1_Alpha (incl. BAC++ extension)
DSE script	18 (2013-08-13)
HSPICE	A-2008.03-SP1 32-BIT
GNU compiler collection	4.4.5 (Debian 4.4.5-8)
MATLAB	R2012a (7.14.0.739)
Design Compiler (dc_shell)	F-2011.09-SP1
Questa Advanced Simulator	10.1c 64bit
Power Compiler (dc_shell)	F-2011.09-SP1

Table C.1: Used Tools

The in-house tool SMOG was used to separate the hardware modules from the overall system. It also creates a corresponding test bench and stimuli data with respect to overall system behaviour. The BAC++-enabled version of PowerOpt was then used for performing the high-level synthesis as well as for generating the power and timing augmented BAC++ model. For automating the evaluation process, the in-house design space exploration script was used. For the exemplary traces at electrical level, HSPICE was used. The generated BAC++ model has been compiled using a recent GCC version. Evaluation and computation of the error measures for the obtained estimation results and traces was done using MATLAB. Logic-level synthesis, simulation and estimation were done using Design Compiler and the Questa Simulator, respectively.

In order to provide comparable evaluation results, all designs used for evaluation were synthesised using the same set of parameters. The default register type, provided by the technology library is used for all registers of the design. For C/C++ arrays with less than 256 elements, each one is mapped onto its own register. However, if such an array is used by multiple processes, it is mapped to a shared memory. The states of all FSMs of the designs are encoded in a binary manner. The same is true for multiplexers. An advanced chaining algorithm is used for chaining operations. The length of the critical path is determined automatically by PowerOpt. The same is true for the minimal and maximal number instantiated resources. Registers and operators must have a minimal bit width of four, in order to be considered for resource sharing. It is more efficient to have two small components than creating the complex multiplexers and control logic. The scheduling algorithm tries to smooth the schedule. That is, the algorithm tries to evenly distribute the operations to all control steps. The generic PowerOpt settings are given in Table C.2.

PowerOpt setting	value
default register type	reg
array scalarisation size	256
FSM encoding	binary
multiplexer encoding	binary
chaining	advanced
critical path length	auto
minimum resources	auto
maximum resources	auto
shared register min. bit width	4
shared operator min. bit width	4
optimize schedule	smooth

Table C.2: Default PowerOpt settings used for all designs

The clock frequency was set to the next maximal usual value allowing a valid scheduling of the design. All designs were implemented using an industrial 65 nm technology. In order to obtain comparable results, voltage and temperature were set to 1 V and 50 °C for all designs, with the voltage depending on the chosen technology. Power was chosen as optimisation goal for all designs, whenever possible. With optimisation goal power, PowerOpt performs a data-dependent high-level synthesis. Since different data-sets were used for characterisation and evaluation, they had to be chosen in a way that PowerOpt creates exactly the same controller and data path for both data-sets. For highly data-dependent designs this is hard to achieve. In these cases area was set as optimisation goal, assuring that a data-independent synthesis was performed. Finally, the syntheses effort was set to a value allowing a synthesis in a reasonable amount of time. The number of RT components also includes zero-strength components, which were optimized and eventually removed during logic and layout synthesis low-level syntheses steps. Nevertheless, they had to be considered during characterisation and estimation. The following designs have been used for evaluating the proposed approach. The designs are listed in order with increasing complexity.

Design I – Faculty Computes the faculty of a given number. This is the simplest example. It simply consists of a loop, multiplying the actual faculty with the current value of the counter.

characteristic	value
technology	65 nm
f_{clk}	250 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	4
# RT components	15

Table C.3: Design I characteristics

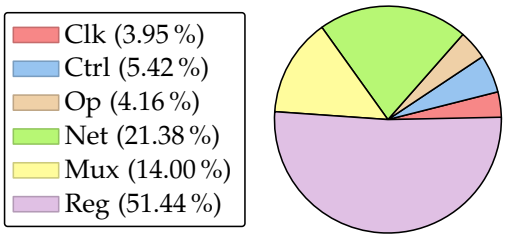


Figure C.1: Design I energy breakdown

Design II – Prime factorisation A simple algorithm for finding the prime factors of a given number. The algorithm is also known as *the Sieve of Eratosthenes*. It consists of nested loops and some arithmetical operations. Most important operation is the modulus operation, since it is comparatively complex the implement this operation in hardware.

characteristic	value
technology	65 nm
f_{clk}	25 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	18
# RT components	42

Table C.4: Design II characteristics

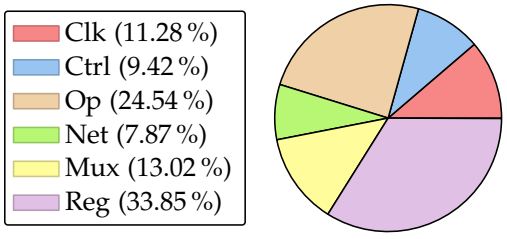


Figure C.2: Design II energy breakdown

Design III – Square root A simple algorithm for computing the square root of a given number. It is an implementation of a binary version of the longhand algorithm.

characteristic	value
technology	65 nm
f_{clk}	250 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	8
# RT components	31

Table C.5: Design III characteristics

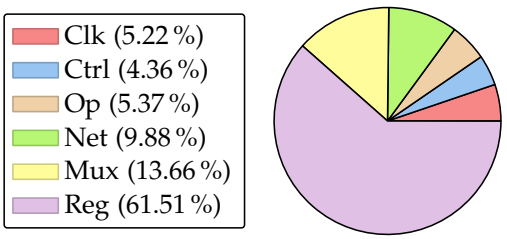


Figure C.3: Design III energy breakdown

Design IV – Simple FDCT This simple one-dimensional, 8-sample FDCT is an example that originally ships with PowerOpt. It consists of several simple arithmetic operations, with a very simple control flow. All parameters are passed directly to the function, no memory is used.

characteristic	value
technology	65 nm
f_{clk}	125 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	12
# RT components	152

Table C.6: Design IV characteristics

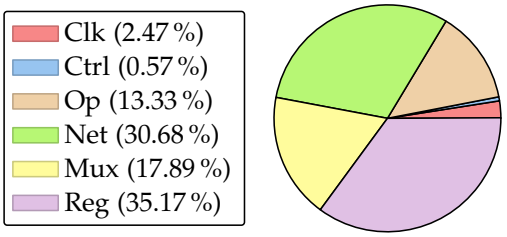


Figure C.4: Design IV energy breakdown

Design V – Heapsort Heapsort [41, sec. 6.4] is an efficient *in-place* sorting algorithm, requiring only a small amount of extra storage. The use case applies the algorithm to random-sized arrays, filled with randomised values.

characteristic	value
technology	65 nm
f_{clk}	125 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	29
# RT components	86

Table C.7: Design V characteristics

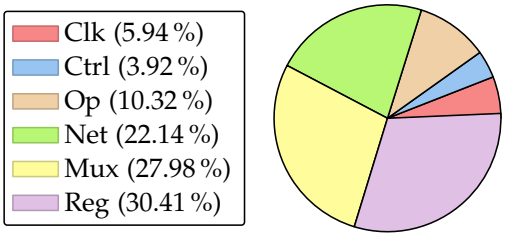


Figure C.5: Design V energy breakdown

Design VI – Simple AES encoder AES is a technique for the encryption of electronic data. It works on four-times-four arrays of data. Typical operations on the array are substitutions, shifts, and XOR-operations. The selected implementation is not optimized for a hardware implementation, but follows the reference implementation as described by Daemen and Rijmen [46, sec. 3]. The required substitution-box as well as its reverse counterpart are implemented as memory-based look-up tables. This allows an easy change of the tables, but prohibits parallel access to the boxes. The synthesis tool has thus less opportunities for an optimisation. The same applies for the round key, which is located in a memory external to the design.

characteristic	value
technology	65 nm
f_{clk}	125 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	107
# RT components	143

Table C.8: Design VI characteristics

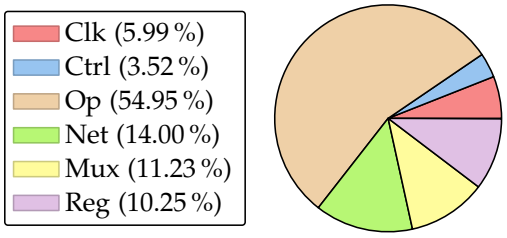


Figure C.6: Design VI energy breakdown

Design VII – DCT of an MP3 media player The design implements a hardware MP3 player. The MP3 player is based on the MAD MP3 encoder library [132]. For acceleration, the DCT was replaced with a hardware accelerator, representing the hardware module under test. The DCT has a fixed depth. Thus, the resulting hardware implementation consists of a very simple control flow. The synthesised behaviour mainly consists of two phases. The first one loads values from the function-call interface into the DCT, while the second phase performs the computation.

characteristic	value
technology	65 nm
f_{clk}	100 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	low
# processes	1
# states	110
# RT components	915

Table C.9: Design VII characteristics

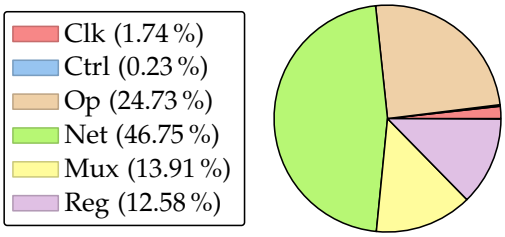


Figure C.7: Design VII energy breakdown

Design VIII – Wavelet transform The wavelet algorithm consists of several nested loops. Data is stored in a number of small internal memories. These are small enough to be implemented as register files instead of separate memory blocks. As can be seen, this design mainly processes large amounts of data with a comparatively small control overhead.

characteristic	value
technology	65 nm
f_{clk}	50 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	95
# RT components	205

Table C.10: Design VIII characteristics

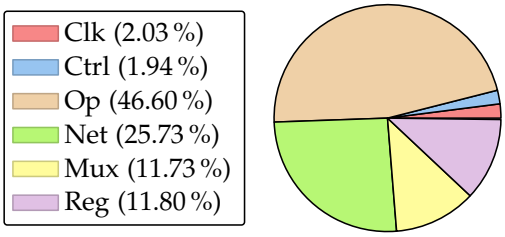


Figure C.8: Design VIII energy breakdown

Design IX – GSM LPC algorithm A LPC algorithm that is part of a GSM implementation. The design consists of a number of sub-processes, such as autocorrelation, quantisation, coding etc. The design has five different processes: The *main* process, activating and synchronising the other processes; the *autocorrelation*; the computation of *reflection coefficients*; the *transformation to Log area ratios*; and finally the *quantization and coding* process. All processes are activated subsequently. The process, responsible for the autocorrelation is the dominant process and contributes about 80.50 % of the total energy dissipation. Slightly changes were made to the code by replacing some of the code, implementing the saturation of operations. Namely functions `gsm_add`, `gsm_mult`, and `gsm_mult_r` have been modified to support branchless saturation. A detailed description of why these modifications where necessary is given in Section 6.5.1.

characteristic	value
technology	65 nm
f_{clk}	125 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	5
# states (toplevel)	10
# states (autocorr.)	74
# states (quantisation)	35
# states (coefficients)	46
# states (transformation)	5
# RT components	812

Table C.11: Design IX characteristics

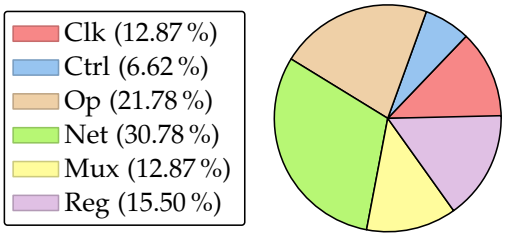


Figure C.9: Design IX energy breakdown

Design X – FFT of an audio/video surveillance system This design is similar to Design VII. It implements a FFT, which is part of an audio/video surveillance system, and is used for detecting unusual noises. The implementation utilises a branch-free saturation logic. However, due to the large number of concurrently executed operations and the resulting complex controller, there is a large number of hardware basic blocks.

characteristic	value
technology	65 nm
f_{clk}	125 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	65
# RT components	748

Table C.12: Design X characteristics

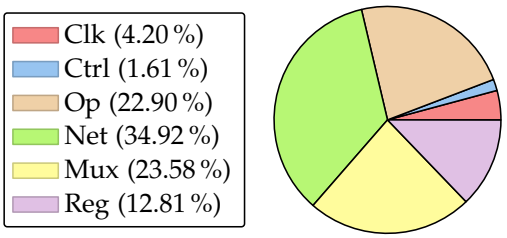


Figure C.10: Design X energy breakdown

Design XI – MIPS processor core A MIPS processor core. The implementation consists of a set of large and nested switch statements. Each case contains only a simple instruction. A given simple application is used as test case. In a single state 421 output signals are driven by numerous output conditions, which in turn rely on even more SMT predicates. These conditions are used to decode the instruction, fetched from the instruction memory.

characteristic	value
technology	65 nm
f_{clk}	100 MHz
V_{dd}	1 V
T	50 °C
opt. goal	power
synth. effort	high
# processes	1
# states	19
# RT components	235

Table C.13: Design XI characteristics

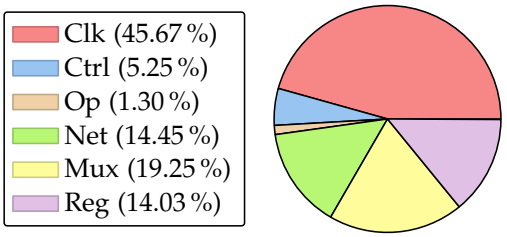


Figure C.11: Design XI energy breakdown

Appendix D

Detailed Evaluation Data

Evaluation of the simple as well as the advanced characterisation approach is done in the same way. A small data set has been used for characterising the design, while a larger data set was used for evaluating the approach. Besides the overview over the evaluation that was given in Section 6.2, this appendix provides more detailed information about the evaluation results for each individual design.

For each design listed below, three figures and one table are given. The first figure shows a section of the power traces obtained from simulations using PowerOpt and BAC++, using the simple as well as the advanced characterisation approach. Sampling or averaging has been applied to some of the designs in order to provide a better readability of the particular trace. If sampling or averaging has been applied, the used window size is mentioned in the description of the particular design's evaluation. The values from the table however, are created using no sampling or averaging.

The respective table shows total dissipated energy and the corresponding relative error. Both are classified by resource type. It is important to note, that the total energy obtained from PowerOpt also includes the energy that dissipated during test bench initialisation. BAC++ does not require this initialisation phase, yielding an extra error. Thus, the actual error for total energy dissipation will be slightly lower than the one shown in the table.

The second figure shows a distribution of the relative error per clock cycle. In other words, it can be seen how many clock cycles of the simulation have a certain relative error.

The third figure shows the development of the MAPE, the RMS as well as the R^2, if sampling is applied during simulation. In other word the figure shows how sampling can be used to reduce the particular error and to improve the fitting of the trace, respectively.

D.1 Detailed Evaluation of Design I

Evaluation was performed by simulating 5000 runs i. e., computations of the faculty for randomly generated numbers. Different seeds have been used for characterisation and evaluation. On average, about six RT components are simulated per hardware basic block, while at the same time about nine components are estimated. A factor of about $k_C^s \approx 1.25$ was used for adjusting the switched capacitances for operators and multiplexers for the simple characterisation approach. The advanced characterisation approach however, requires a scaling factor of $k_C^{adv} \approx 1.19$. The section of the power trace shown in Figure D.1 shows eight activations of the hardware module. The trace is cycle-accurate i. e., neither sampling nor averaging were applied.

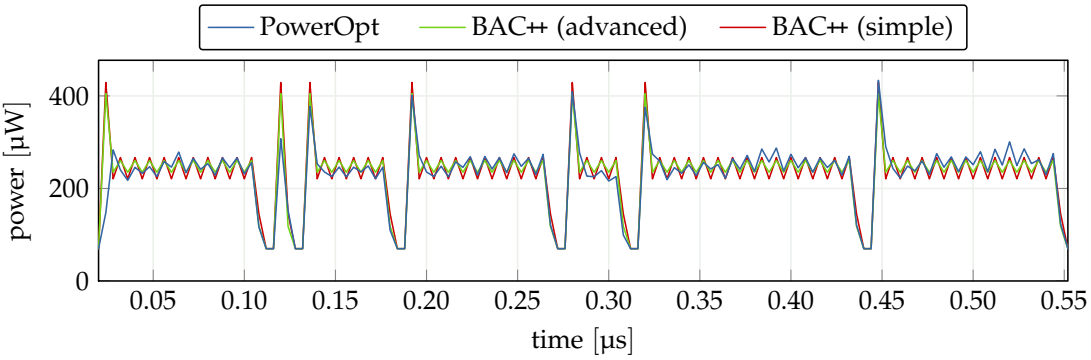


Figure D.1: Power trace for Design I

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	2.92×10^{-9}	2.92×10^{-9}	0.0002	2.92×10^{-9}	0.0002
Ctrl	4.01×10^{-9}	4.01×10^{-9}	-0.0141	4.01×10^{-9}	-0.0141
Op	13.54×10^{-9}	13.55×10^{-9}	0.1010	13.45×10^{-9}	-0.6384
Net	15.81×10^{-9}	15.86×10^{-9}	0.2778	15.86×10^{-9}	0.2778
Reg	38.18×10^{-9}	38.20×10^{-9}	0.0406	38.20×10^{-9}	0.0377
Leak	101.85×10^{-12}	101.85×10^{-12}	0.0047	101.85×10^{-12}	0.0047
Total	74.47×10^{-9}	74.65×10^{-9}	0.2328	74.55×10^{-9}	0.0968

Table D.1: Energy dissipation error per resource for Design I

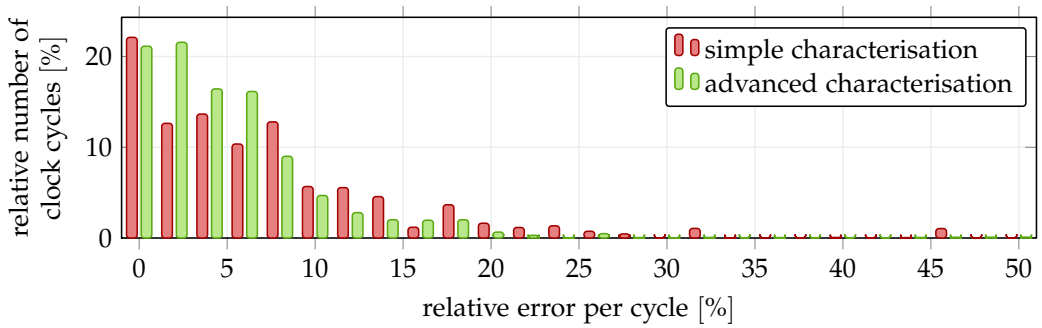


Figure D.2: Distribution of the relative error per cycle for Design I

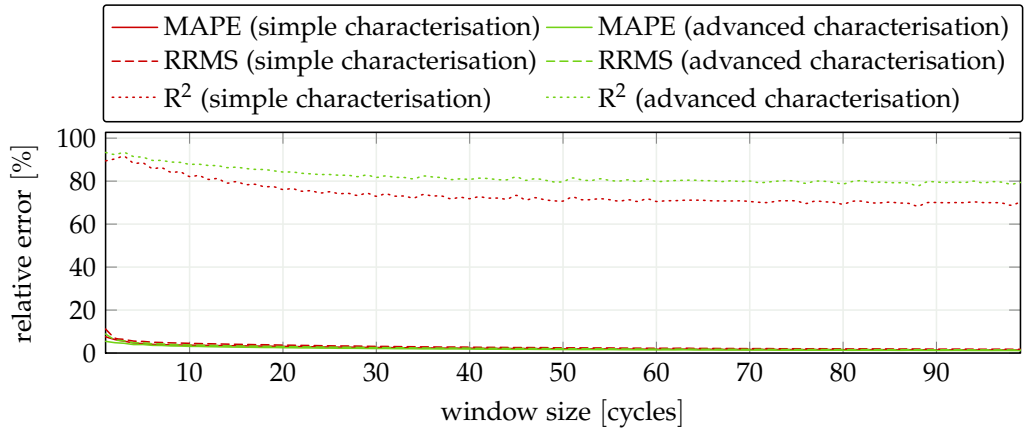


Figure D.3: Average relative error vs. sampling window size for Design I

D.2 Detailed Evaluation of Design II

Characterisation and evaluation were carried out by performing ten prime factorisations, whereas different seeds are used during random number generation. During simulation, about six RT components are simulated per basic block, while elven components are estimated on average.

Even though the design consists of 42 RT components in total, the modulo operation is the domination one. Both, in terms of area as well as power dissipation. As can be seen from its definition in Equation (D.1), the modulo operation consists of a division, a multiplication, and an addition. The first two are complex sub-circuits themselves, which results in a large circuit with a long critical path. For comparison, a small 32 bit modulo-operator's delay is about 31.94 times the delay of a small 32 bit adder, when implemented in the technology used in this thesis.

$$(a \bmod b) \equiv a - b\left\lfloor \frac{a}{b} \right\rfloor \tag{D.1}$$

In order to achieve the desired timing, the modulo operation typically becomes a multi-cycle operation. In other words, it may span over multiple clock cycles. In this design, four states are inserted in order to achieve timing constraint, even though the clock frequency is set to 25 MHz. The modulo operator is also used within an assignment hardware basic block. This results in a large scaling factor of $k_C^s \approx 2.01$, which in turn will make the error worse, since *all* functional power values are scaled. For the advanced characterisation approach the scaling factor is even worse with $k_C^{adv} \approx 5.53$. This is caused by the problem regarding estimation of multi-cycle operations, as mentioned in Section 6.5.5. The modulo operation cannot be assigned to the correct clock cycle, leading to an under-estimation of the operator. This, along with the fact that the modulo operation is the dominating one, causes this high correction factor.

The obtained power trace was 559 984 clock cycles long. A section of the power trace is shown in Figure D.4. Neither sampling nor sliding-window averaging had been applied to the plotted trace.

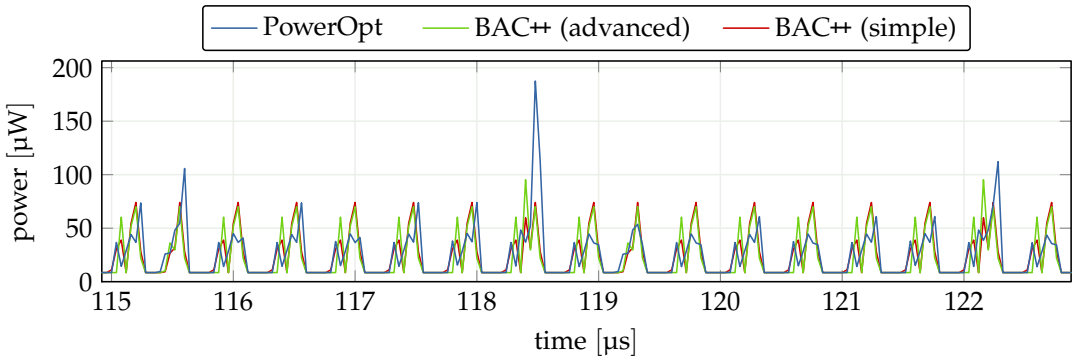


Figure D.4: Power trace for Design II

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	8.73×10^{-9}	8.72×10^{-9}	-0.0557	8.72×10^{-9}	-0.0557
Ctrl	7.28×10^{-9}	7.28×10^{-9}	-0.0556	7.28×10^{-9}	-0.0556
Op	29.40×10^{-9}	29.20×10^{-9}	-0.6817	29.15×10^{-9}	-0.8678
Net	6.19×10^{-9}	6.08×10^{-9}	-1.6728	6.08×10^{-9}	-1.6728
Reg	26.27×10^{-9}	26.29×10^{-9}	0.0814	22.99×10^{-9}	-12.4683
Leak	8.33×10^{-9}	8.38×10^{-9}	0.5773	8.38×10^{-9}	0.5773
Total	86.37×10^{-9}	85.95×10^{-9}	-0.4859	82.60×10^{-9}	-4.3652

Table D.2: Energy dissipation error per resource for Design II

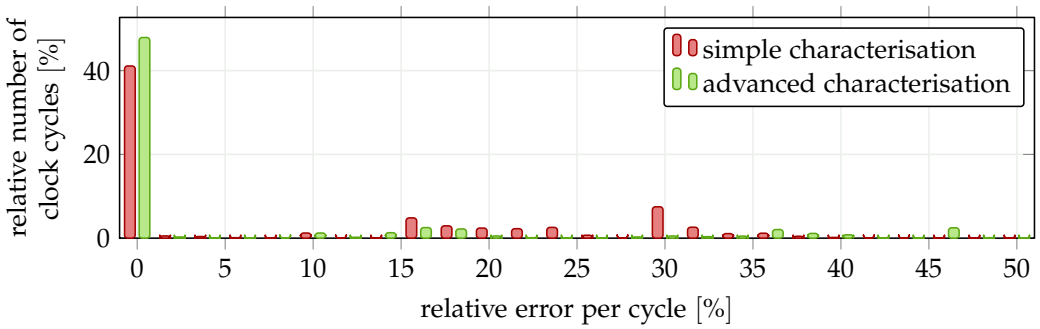


Figure D.5: Distribution of the relative error per cycle for Design III

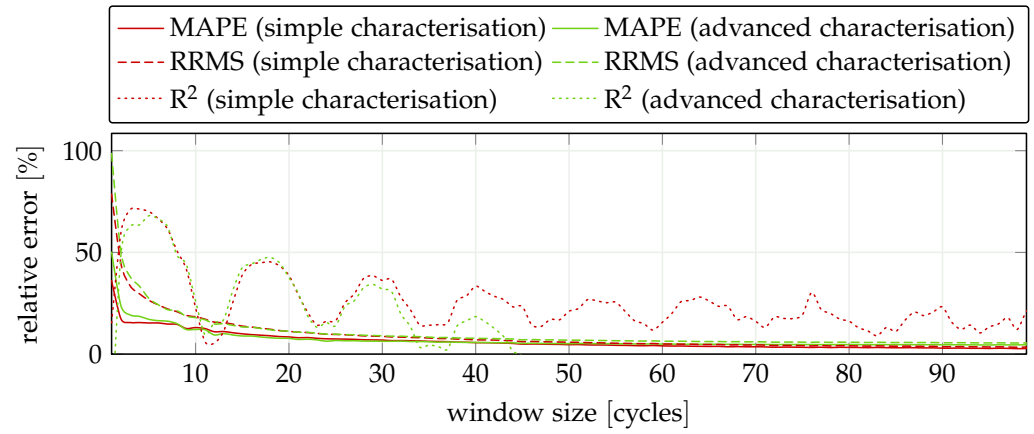


Figure D.6: Average relative error vs. sampling window size for Design II

D.3 Detailed Evaluation of Design III

In total, 5000 computations of the square root, which were performed in a total of 144 774 simulated clock cycles, have been used for evaluation. Again, numbers have been generated randomly, using different seeds for characterisation and evaluation, respectively. The generated BAC++ model contains only a small number of hardware basic blocks, simulating about six and estimating about 14 RT components on average. However, a notable amount of hardware basic blocks is required for providing the controller's input. This results in comparatively large scaling factors, especially for the advanced characterisation.

For the simple characterisation technique, a scaling factor of $k_C^s \approx 1.28$ has been used for adjusting the capacitance, switched by functional operators and multiplexers. A factor of $k_C^{adv} \approx 1.64$ was used for the advanced characterisation approach. Five computations of the square root are shown in Figure D.7.

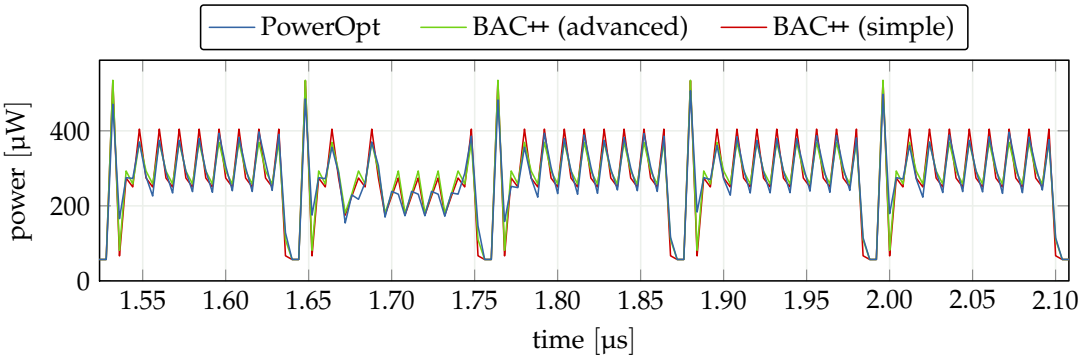


Figure D.7: Power trace for Design III

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	9.04×10^{-9}	9.04×10^{-9}	0.0002	9.04×10^{-9}	0.0002
Ctrl	7.09×10^{-9}	7.09×10^{-9}	0.0002	7.09×10^{-9}	0.0002
Op	30.96×10^{-9}	30.94×10^{-9}	-0.0720	30.96×10^{-9}	-0.0232
Net	16.84×10^{-9}	16.81×10^{-9}	-0.1576	16.81×10^{-9}	-0.1576
Reg	98.86×10^{-9}	98.88×10^{-9}	0.0189	98.87×10^{-9}	0.0133
Leak	92.21×10^{-12}	92.21×10^{-12}	0.0030	92.21×10^{-12}	0.0030
Total	162.89×10^{-9}	162.85×10^{-9}	-0.0194	162.86×10^{-9}	-0.0132

Table D.3: Energy dissipation error per resource for Design III

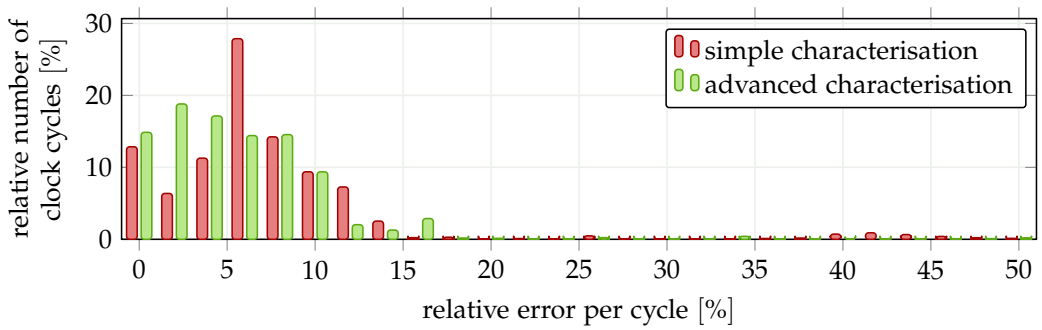


Figure D.8: Distribution of the relative error per cycle for Design III

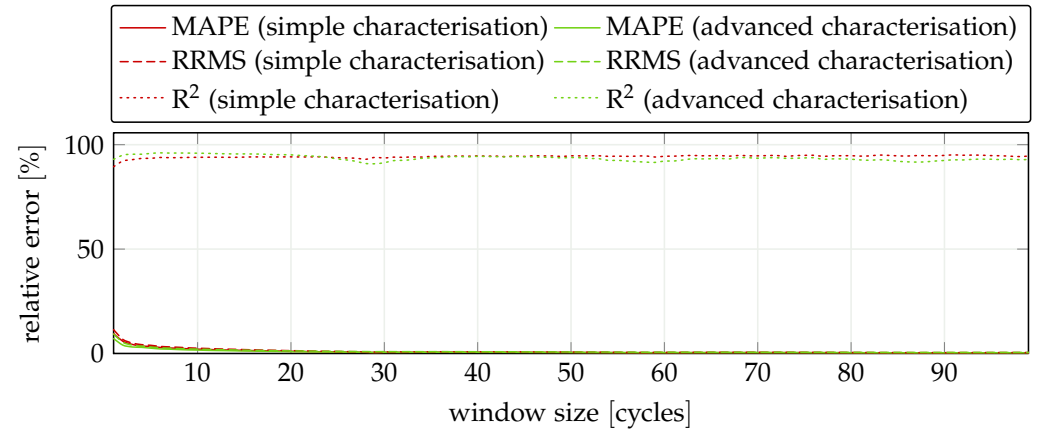


Figure D.9: Average relative error vs. sampling window size for Design III

D.4 Detailed Evaluation of Design IV

This design has been evaluated by performing 257 FDCT transformations. These required 3341 clock cycles in total. Due to the very simple control flow no assignment hardware basic blocks were required. The controller's FSM is also quite simple, with almost all output conditions have the trivial form `true`. A generated average hardware basic block comprises about 28 RT components that must be simulated, but is able to estimate about 97 RT components.

A scaling factor of $k_C^s \approx 0.98$ was used for adjusting operators' and multiplexers' switched capacitance during simple characterisation. Scaling factor for advanced characterisation is $k_C^{adv} \approx 1.13$. The cycle-accurate power trace in Figure D.10 shows ten iterations of the algorithm.

The striking shape of the R^2 is caused by the fact, that a typical run of the algorithm requires about eleven clock cycles. If the sampling window passes the borders of the individual activations, differ notably. Additionally, the power trace becomes less pronounced with a larger sampling window. As mentioned in Section 6.2, the measure R^2 is unsuited for uniform traces.

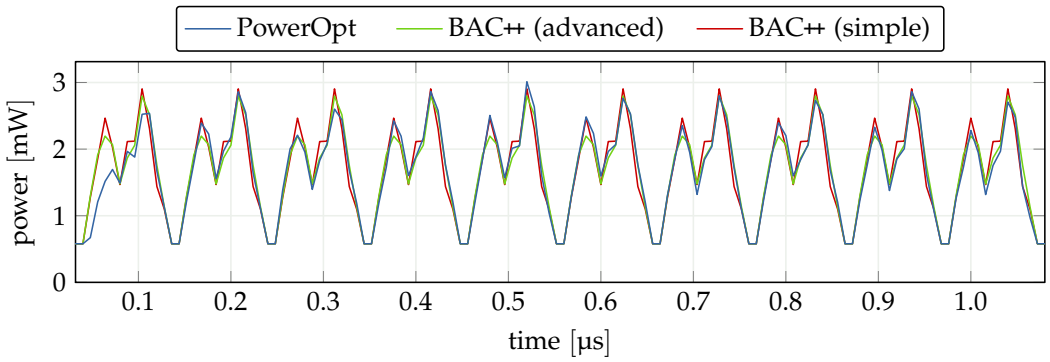


Figure D.10: Power trace for Design IV

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	941.06×10^{-12}	941.06×10^{-12}	0.0004	941.06×10^{-12}	0.0004
Ctrl	217.91×10^{-12}	217.91×10^{-12}	0.0002	217.91×10^{-12}	0.0002
Op	14.57×10^{-9}	14.53×10^{-9}	-0.2320	14.53×10^{-9}	-0.2320
Net	14.28×10^{-9}	14.27×10^{-9}	-0.0850	14.27×10^{-9}	-0.0850
Reg	15.72×10^{-9}	15.70×10^{-9}	-0.1111	15.70×10^{-9}	-0.1111
Leak	36.02×10^{-12}	36.06×10^{-12}	0.1202	36.06×10^{-12}	0.1202
Total	45.78×10^{-9}	45.70×10^{-9}	-0.1791	45.70×10^{-9}	-0.1791

Table D.4: Energy dissipation error per resource for Design IV

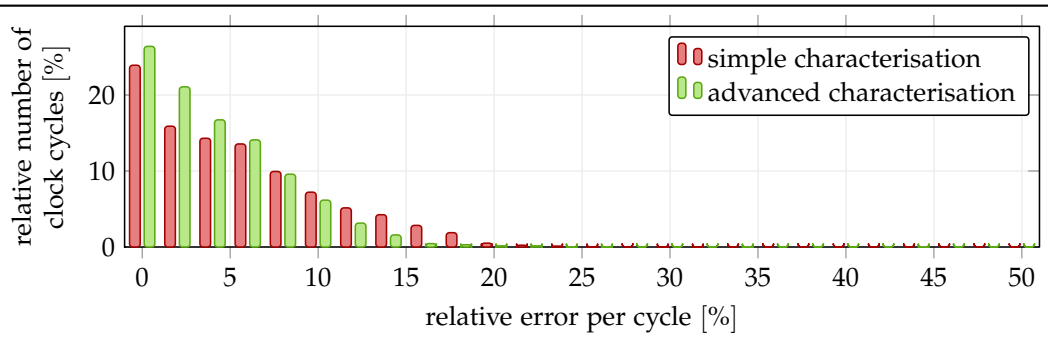


Figure D.11: Distribution of the relative error per cycle for Design IV

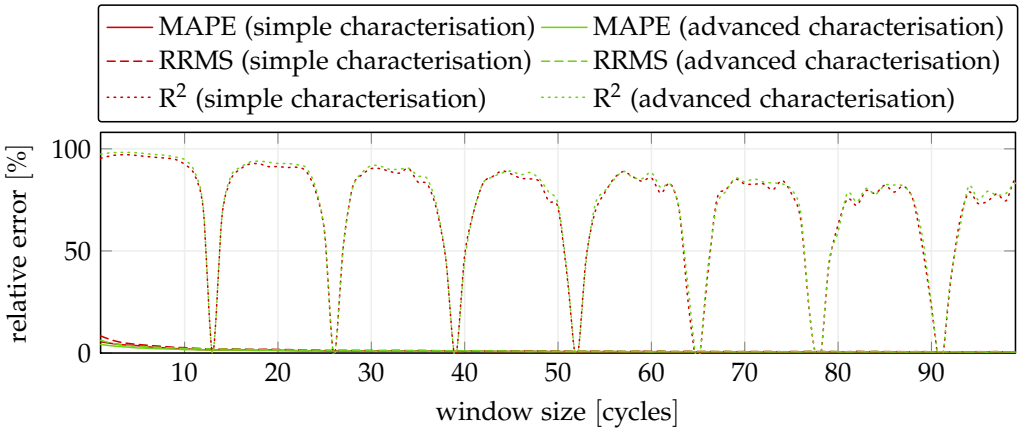


Figure D.12: Average relative error vs. sampling window size for Design IV

D.5 Detailed Evaluation of Design V

Evaluation was done by sorting 50 arrays of random size, filled with randomly generated numbers. Of course, different seeds have been used for characterisation and evaluation. Simulation covered 17 994 clock cycles, of which 200 are shown in Figure D.13. A sliding window of length four was applied to the trace for better graphical representation. On average, each generated hardware basic block estimates 28 RT components, of which ten are required for simulating the behaviour. Functional operators' and multiplexers' switched capacitance was scaled using a factor of $k_C^s \approx 1.05$. For the advanced characterisation approach a scaling factor of about $k_C^{adv} \approx 1.86$ is used.

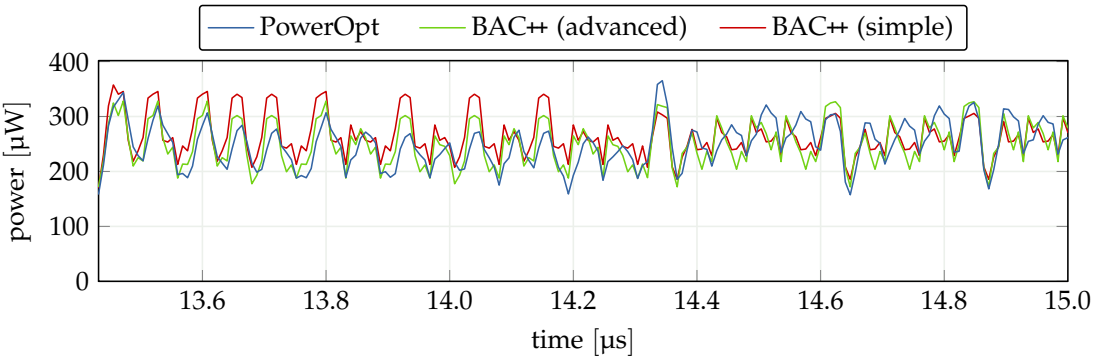


Figure D.13: Power trace for Design V

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	2.10×10^{-9}	2.11×10^{-9}	0.4740	2.11×10^{-9}	0.4740
Ctrl	1.39×10^{-9}	1.40×10^{-9}	0.4738	1.40×10^{-9}	0.4738
Op	14.30×10^{-9}	14.32×10^{-9}	0.1829	14.30×10^{-9}	0.0157
Net	7.84×10^{-9}	7.88×10^{-9}	0.5669	7.88×10^{-9}	0.5669
Reg	11.32×10^{-9}	11.36×10^{-9}	0.3141	10.39×10^{-9}	-8.2354
Leak	60.85×10^{-12}	64.25×10^{-12}	5.5910	64.25×10^{-12}	5.5910
Total	37.65×10^{-9}	37.14×10^{-9}	-1.3570	36.15×10^{-9}	-3.9911

Table D.5: Energy dissipation error per resource for Design V

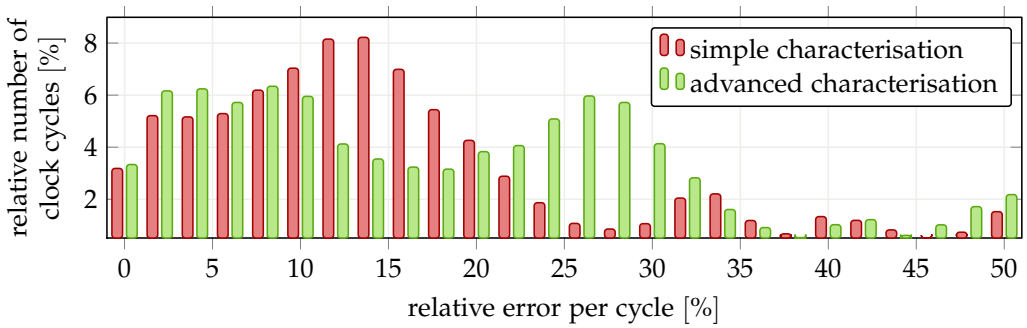


Figure D.14: Distribution of the relative error per cycle for Design V

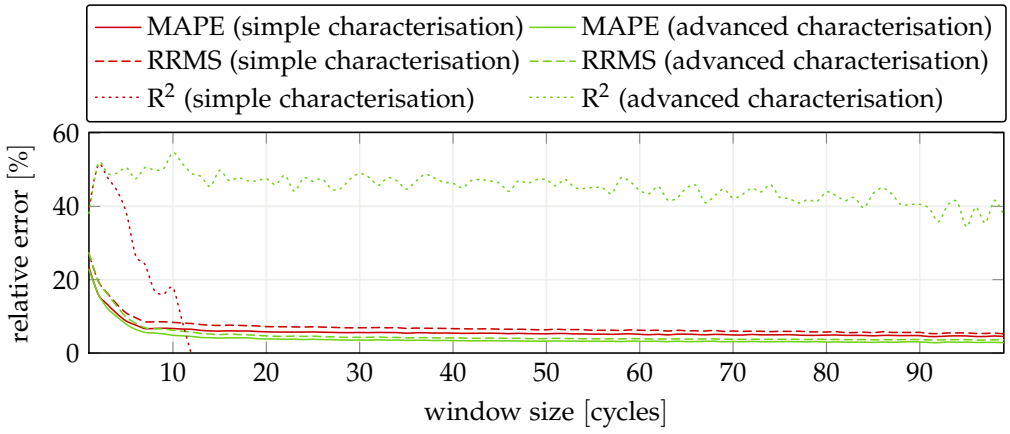


Figure D.15: Average relative error vs. sampling window size for Design V

D.6 Detailed Evaluation of Design VI

All in all, 15 runs of the encoding algorithm have been used for evaluating the design, totalling in 31 111 clock cycles to be simulated. About nine RT components are simulated per hardware basic block, while about 69 are estimated at the same time. Adjusting switched capacitance of operators and multiplexers for the simple characterisation was done using a factor of about $k_C^s \approx 0.97$. For the advanced characterisation a factor of about $k_C^{adv} \approx 1.52$ was used. A sliding averaging window of size 16 has been used for better plotting.

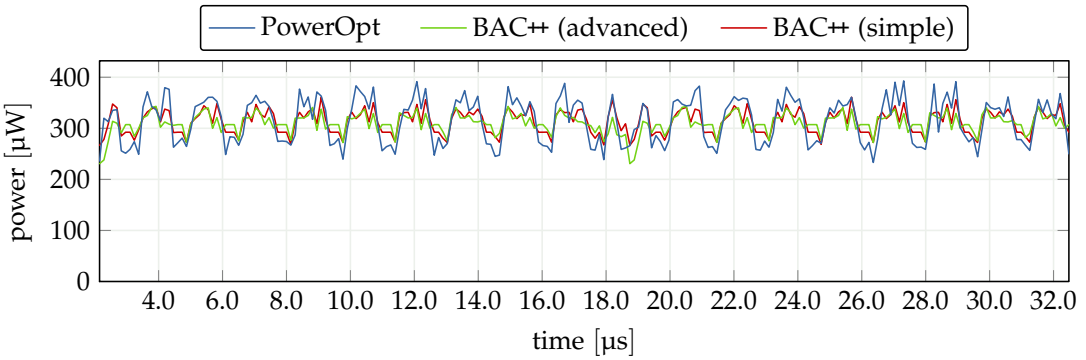


Figure D.16: Power trace for Design VI

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	4.16×10^{-9}	4.16×10^{-9}	0.0001	4.16×10^{-9}	0.0001
Ctrl	3.53×10^{-9}	3.53×10^{-9}	0.0001	3.53×10^{-9}	0.0001
Op	50.10×10^{-9}	49.84×10^{-9}	-0.5087	49.84×10^{-9}	-0.5087
Net	10.01×10^{-9}	9.84×10^{-9}	-1.7344	9.84×10^{-9}	-1.7344
Reg	10.39×10^{-9}	10.44×10^{-9}	0.4087	9.58×10^{-9}	-7.7996
Leak	91.48×10^{-12}	96.48×10^{-12}	5.4688	96.48×10^{-12}	5.4688
Total	78.43×10^{-9}	77.90×10^{-9}	-0.6759	77.05×10^{-9}	-1.7639

Table D.6: Energy dissipation error per resource for Design VI

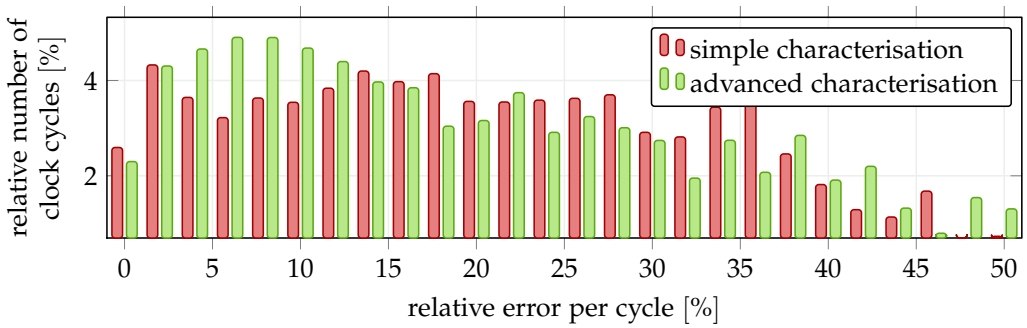


Figure D.17: Distribution of the relative error per cycle for Design VI

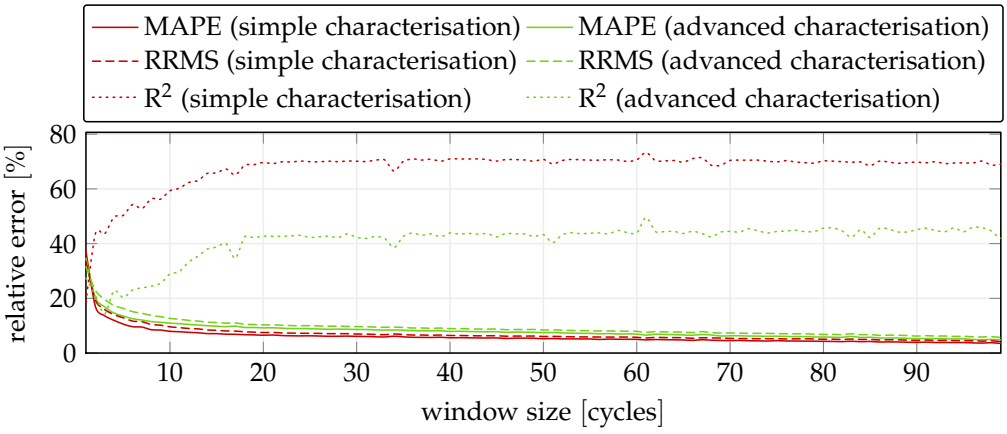


Figure D.18: Average relative error vs. sampling window size for Design VI

D.7 Detailed Evaluation of Design VII

For this design, the optimisation effort had to be set to a low level. Due to its exponential behaviour, setting the effort from `low` to `medium` will increase the time required for synthesis from 2049.32 s ≈ 34.16 minutes to 182 015 s ≈ 2.11 days. Despite the significantly higher computational effort, the effect of the high synthesis effort to the resulting RT-level implementation is negligible.

During simulation, about 41 RT components are required for simulating the behaviour in one clock cycle. At the same time about 526 components are estimated on average. The design was characterised using 100 samples. Evaluation was done by doing 1000 DCT transformations, which requires 111 000 clock cycles to be simulated.

For fine-tuning operators' and multiplexers' switched capacitance, a scaling factor of $k_C^s \approx 0.66$ was used for the simple characterisation. For the advanced characterisation a scaling factor of $k_C^{adv} \approx 1.14$ was necessary. The power trace in Figure D.19 depicts 225 cycles of the simulation, picturing two activations of the DCT.

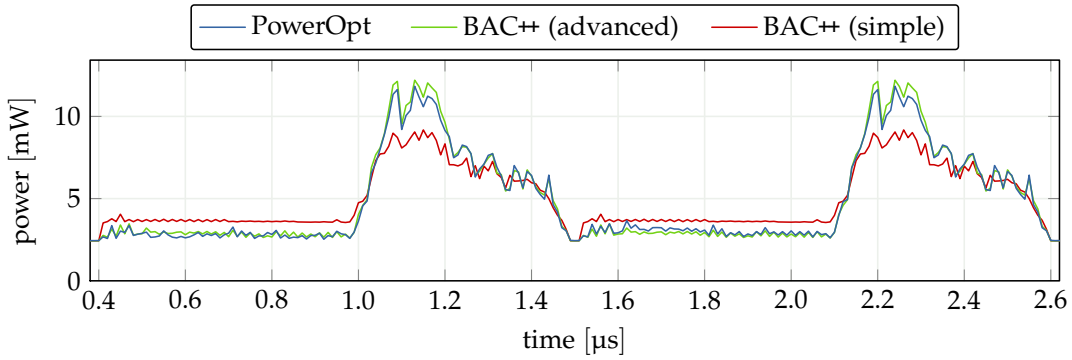


Figure D.19: Power trace for Design VII

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	96.32×10^{-9}	96.03×10^{-9}	-0.3073	96.03×10^{-9}	-0.3073
Ctrl	12.68×10^{-9}	12.64×10^{-9}	-0.3077	12.64×10^{-9}	-0.3077
Op	2.14×10^{-6}	2.14×10^{-6}	-0.0332	2.15×10^{-6}	0.2905
Net	2.59×10^{-6}	2.58×10^{-6}	-0.3314	2.58×10^{-6}	-0.3314
Reg	697.93×10^{-9}	697.41×10^{-9}	-0.0747	697.41×10^{-9}	-0.0747
Leak	12.29×10^{-9}	12.29×10^{-9}	0.0340	12.29×10^{-9}	0.0340
Total	5.56×10^{-6}	5.54×10^{-6}	-0.1995	5.55×10^{-6}	-0.0743

Table D.7: Energy dissipation error per resource for Design VII

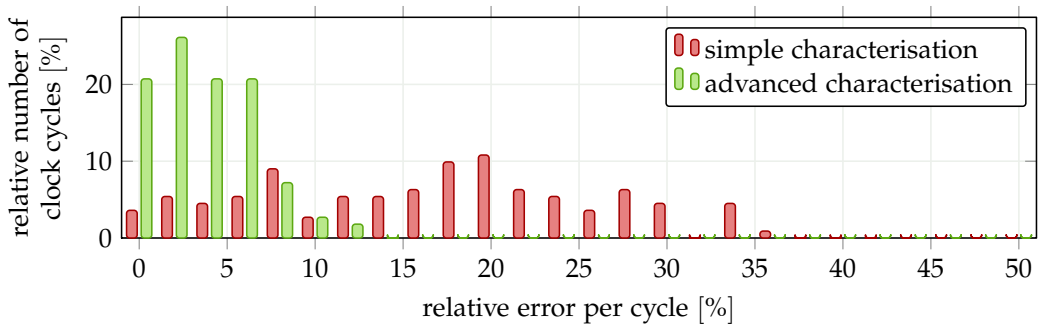


Figure D.20: Distribution of the relative error per cycle for Design VII

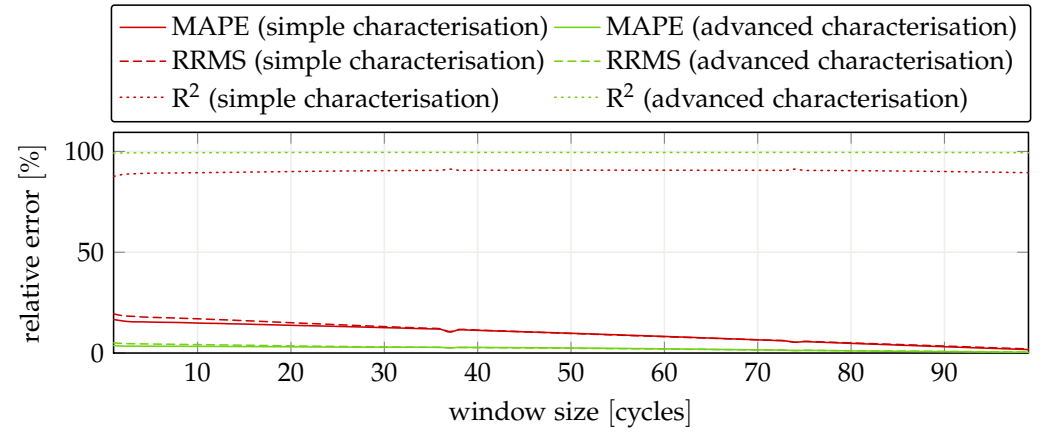


Figure D.21: Average relative error vs. sampling window size for Design VII

D.8 Detailed Evaluation of Design VIII

Characterisation was performed by simulating 25 812 clock cycles, representing three runs of the wavelet transformation. Evaluation was done by doing ten transformation with a total amount of 86 026 cycles. In each cycle, eight RT components are simulated on average, while about 78 components are estimated. Scaling factor is about $k_C^s \approx 0.64$ was used for simple characterisation. The advanced characterisation has a scaling factor of about $k_C^{adv} \approx 1.46$. In order to be able to depict one complete run of the wavelet algorithm, a sampling window of 41 clock cycles has been used.

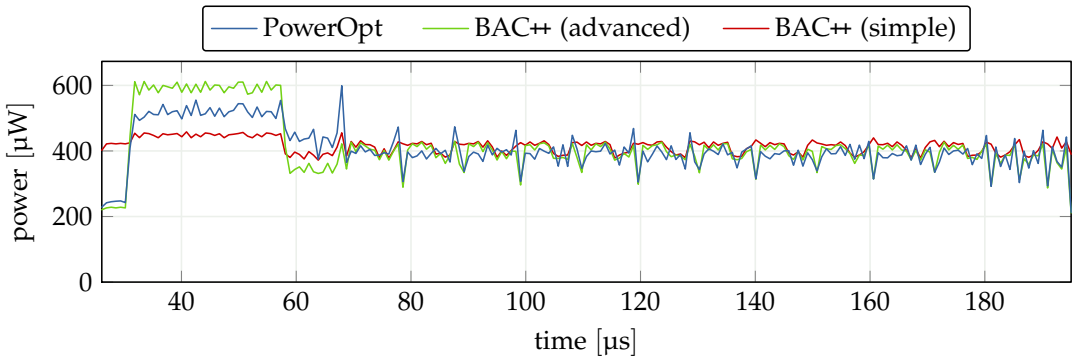


Figure D.22: Power trace for Design VIII

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	9.97×10^{-9}	10.23×10^{-9}	2.5679	10.23×10^{-9}	2.5679
Ctrl	9.54×10^{-9}	9.54×10^{-9}	-0.0162	9.54×10^{-9}	-0.0162
Op	425.25×10^{-9}	428.04×10^{-9}	0.6574	428.04×10^{-9}	0.6577
Net	184.04×10^{-9}	187.50×10^{-9}	1.8822	187.50×10^{-9}	1.8822
Reg	65.77×10^{-9}	65.74×10^{-9}	-0.0411	65.61×10^{-9}	-0.2383
Leak	3.97×10^{-9}	4.23×10^{-9}	6.4665	4.23×10^{-9}	6.4665
Total	701.69×10^{-9}	705.28×10^{-9}	0.5123	705.15×10^{-9}	0.4939

Table D.8: Energy dissipation error per resource for Design VIII

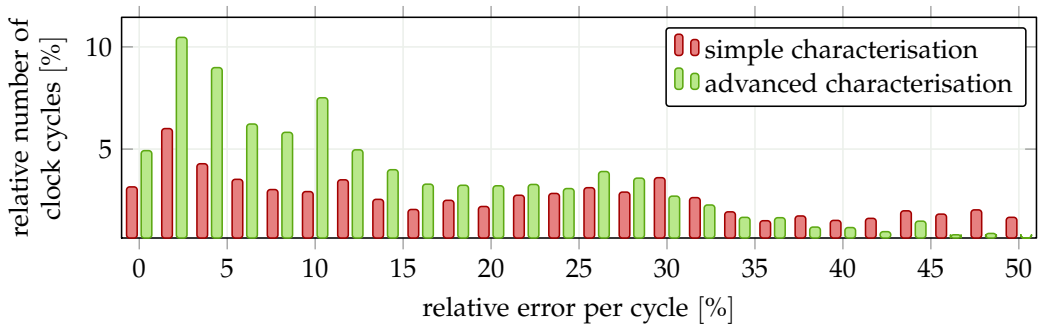


Figure D.23: Distribution of the relative error per cycle for Design VIII

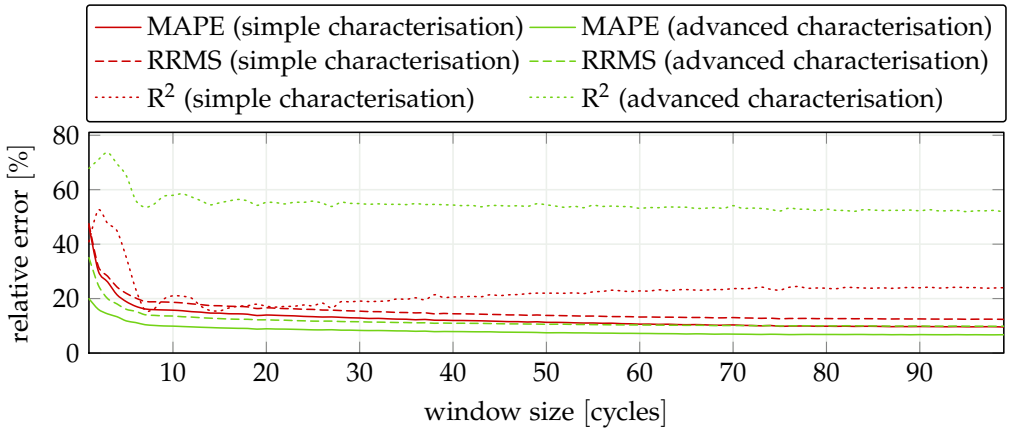


Figure D.24: Average relative error vs. sampling window size for Design VIII

D.9 Detailed Evaluation of Design IX

The designs consist of five processes. One top-level process and four behavioural processes that are executed successively. The first process performs the autocorrelation and contributes about 85 % to the total energy dissipation. As can be seen in Figure D.25, the process itself consists of different execution phases.

This design produces a large amount of hardware basic blocks, where the majority of the basic blocks are not activated during characterisation. For these blocks, the simple characterisation is used as contingency option, as described in Section 4.4.5.

Characterisation was done by performing five runs of the algorithm. Evaluation was performed by running the algorithm ten times. The generated hardware basic blocks estimate 75 RT components on average, while 21 of them are required for the behavioural simulation. For the simple characterisation approach, a scaling factor of about $k_C^s \approx 0.54$ was used, while a factor of about $k_C^{adv} \approx 1.36$ was used for the advanced characterisation process. In order to fit a complete run of the algorithm into the figure below, a sampling window with a size of 15 clock cycles was used.

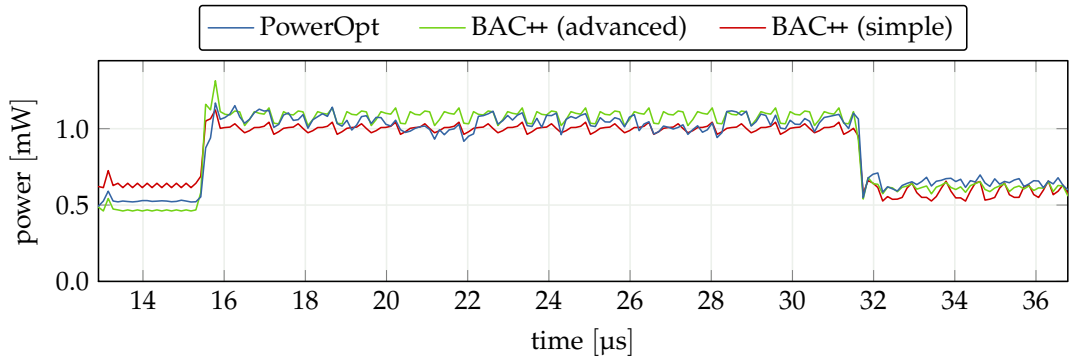


Figure D.25: Power trace for Design IX

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	29.19×10^{-9}	28.13×10^{-9}	-3.6253	28.13×10^{-9}	-3.6253
Ctrl	14.39×10^{-9}	13.88×10^{-9}	-3.5564	13.88×10^{-9}	-3.5564
Op	82.53×10^{-9}	87.97×10^{-9}	6.5966	91.69×10^{-9}	11.1076
Net	71.30×10^{-9}	73.42×10^{-9}	2.9790	73.42×10^{-9}	2.9790
Reg	37.11×10^{-9}	38.05×10^{-9}	2.5311	38.12×10^{-9}	2.7257
Leak	1.59×10^{-9}	1.62×10^{-9}	1.6572	1.62×10^{-9}	1.6572
Total	241.38×10^{-9}	243.07×10^{-9}	0.6989	246.86×10^{-9}	2.2711

Table D.9: Energy dissipation error per resource for Design IX

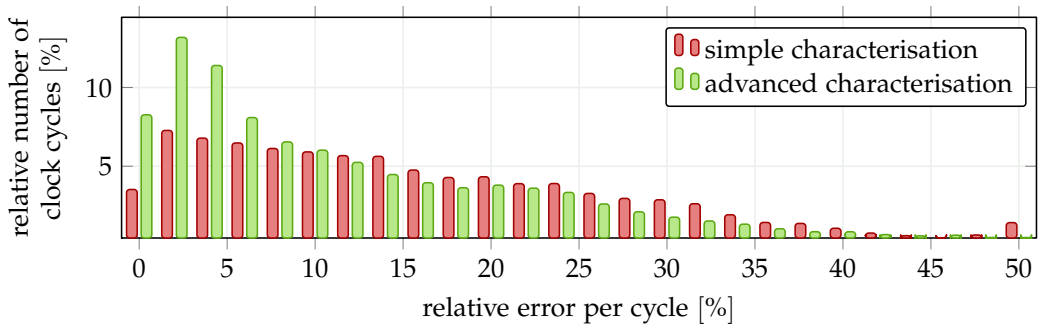


Figure D.26: Distribution of the relative error per cycle for Design IX

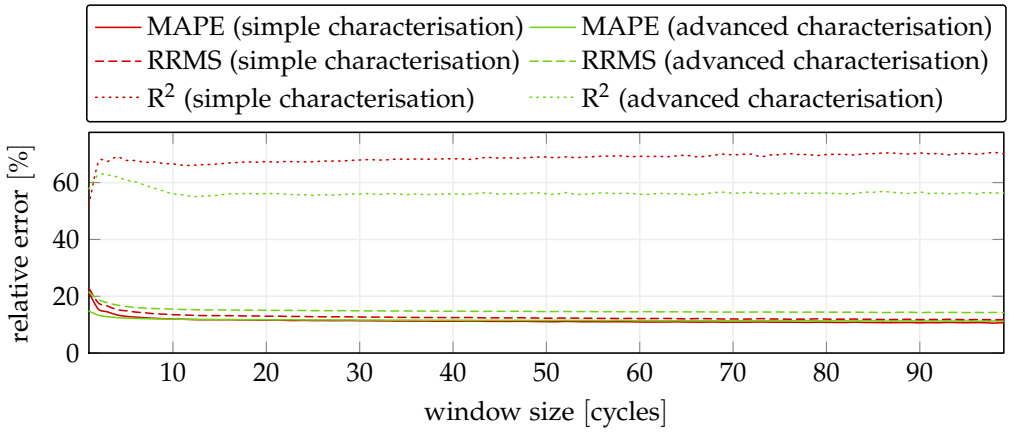


Figure D.27: Average relative error vs. sampling window size for Design IX

D.10 Detailed Evaluation of Design X

Characterisation of the design was done by processing six FFT samples, requiring a total of 63 207 clock cycles to be simulated. Evaluation was done by processing 12 samples, which required a total 125 385 cycles to be simulated. The use case has been contributed by the use case provider of the project COMPLEX and represents a car passing by. Different samples from the data set of samples have been used for characterisation and evaluation, respectively. While the generated hardware basic blocks estimate about 621 RT components on average, only 50 of them are required for the behavioural simulation.

The design requires a scaling factor of $k_C^s \approx 0.89$ for adjusting the capacitance, switched by the design's operators and multiplexers. For the advanced characterisation a scaling factor of about $k_C^{adv} \approx 1.46$ was used. Power dissipation for processing a single sample is shown in Figure D.28, using a sampling window size of 33 cycles.

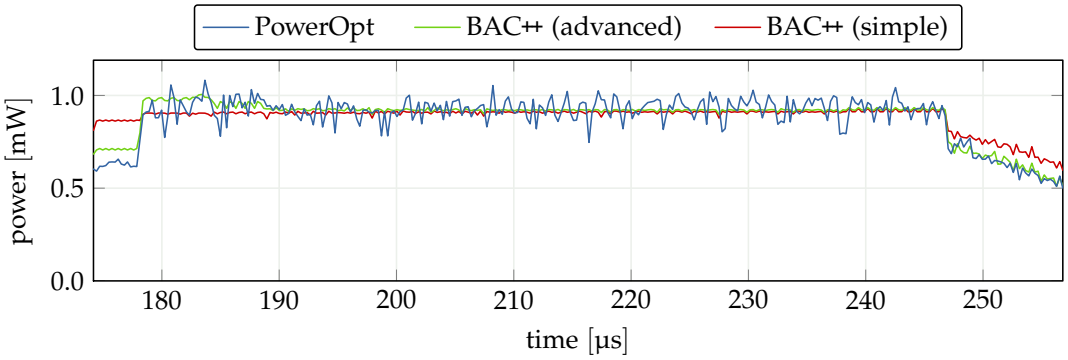


Figure D.28: Power trace for Design X

	PowerOpt	BAC++ (simple)		BAC++ (advanced)	
Resource	energy [J]	energy [J]	RE [%]	energy [J]	RE [%]
Clk	36.73×10^{-9}	36.44×10^{-9}	-0.8137	36.44×10^{-9}	-0.8137
Ctrl	14.09×10^{-9}	13.98×10^{-9}	-0.8139	13.98×10^{-9}	-0.8139
Op	408.81×10^{-9}	410.23×10^{-9}	0.3486	410.21×10^{-9}	0.3437
Net	307.60×10^{-9}	303.28×10^{-9}	-1.4047	303.21×10^{-9}	-1.4271
Reg	111.56×10^{-9}	112.91×10^{-9}	1.2109	112.25×10^{-9}	0.6166
Leak	2.07×10^{-9}	2.17×10^{-9}	4.8782	2.09×10^{-9}	0.7530
Total	883.85×10^{-9}	879.00×10^{-9}	-0.5478	878.17×10^{-9}	-0.6425

Table D.10: Energy dissipation error per resource for Design X

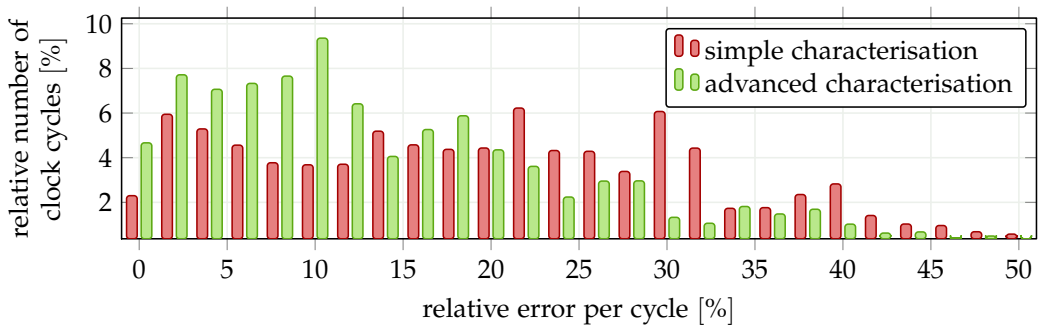


Figure D.29: Distribution of the relative error per cycle for Design X

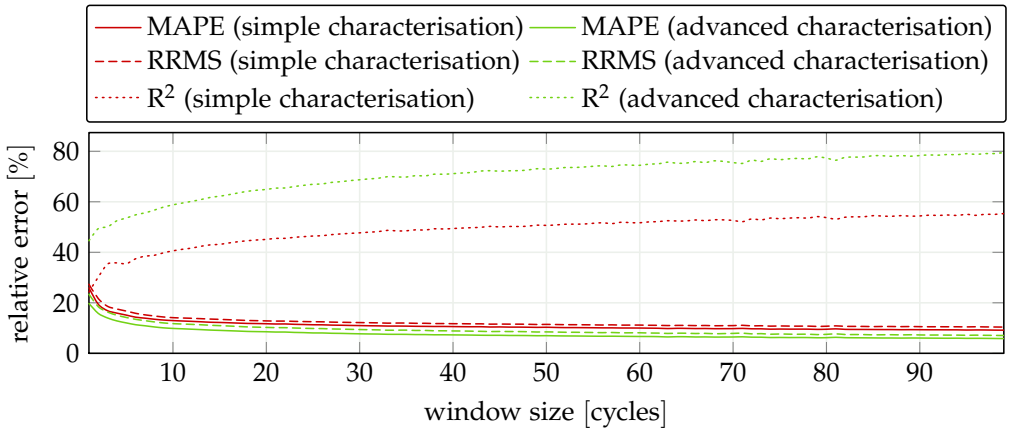


Figure D.30: Average relative error vs. sampling window size for Design X

D.11 Comparison with Logic-Level Estimation

As mentioned at the beginning of Chapter 6, comparing the results obtained from PowerOpt and BAC++ with the results obtained from a logic-level simulation is difficult. The logic-level simulation requires a significantly higher computational effort and a lot of restrictions apply. Besides the increased computational effort, several manual modifications and adoptions are necessary. Therefore, comparison with a logic-level simulation and estimation was only performed for a single example. For the comparison, Design VII was chosen, since it is a reasonable complex design and provides the most pronounced power trace. The following paragraphs describe the required modifications and discuss the achieved results.

The logic-level estimation process can be outlined as follows: It starts with a characterisation and BAC++ model generation using PowerOpt, as described in this thesis. Characterisation is done using the simple as well as the advanced characterisation approach. For both generated BAC++ models, the scaling factors are computed accordingly. Three power traces were generated, using both BAC++ models as well as PowerOpt. For generating the traces, a set of stimuli data, different from the ones used for characterisation was applied. A logic-level synthesis is performed using Synopsis Design Compiler. The generated net list is simulated using the Questa Advanced Simulator. The simulation creates a VCD file, which in turn is used by the Synopsis Power Compiler for performing the logic-level power estimation, which in turn is compared to the previously generated traces.

D.11.1 Restrictions and Required Modifications

In order to perform the desired evaluation some modifications are required. The CDB used by PowerOpt bases on a 65 nm industrial-strength technology. Due to non-disclosure restrictions, this technology is not available for a logic-level simulation and estimation. Moreover, it can be assumed that further steps have been taken to conceal the exact technology data and prevent reverse engineering of the technology. Therefore, it was necessary use a different technology during low-level estimation. Nevertheless, in order to obtain comparable values, a very similar technology was chosen. It is also a 65 nm technology, but has been characterised for a supply voltage of 1.20 V and a temperature of 25 °C. In order to achieve comparable results from PowerOpt, supply voltage and temperature have been adjusted accordingly during estimation and evaluation.

Since new settings were used during the PowerOpt-based estimation, a new BAC++ characterisation has been performed for Design VII, using the simple as well as the advanced characterisation approach. For the simple characterisation approach, a scaling factor of about $k_C^s \approx 0.66$ is required, while the advanced characterisation approach demands a scaling factor of about $k_C^{adv} \approx 1.14$. Both scaling factors are very similar to the ones identified in Section D.7. This was expected, since the generated RT-level model and thus the average switched capacitance is the same for both cases. Deviations in the switched capacitance are observed only from the fourth decimal place. These deviations are caused by second order effects.

D.11.2 Evaluation

Logic-level synthesis of the Verilog model was done in about 523.97 s using Design Compiler. Evaluation was performed with a smaller set of stimuli data as has been used in Section D.7. Only 1000 clock cycles have been simulated, which represent about eight runs of the DCT algorithm. An estimation of the complete use case, covering 111 000 clock cycles would have taken 269 013.58 s $\approx$ 3.11 days and was thus not feasible.

Not only is the required computational effort a problem. The VCD file, which is obtained from a functional simulation at logic level contains about 3.20 MB of data for each simulated clock cycle. For simulating all 111 000 clock cycles this would have led to a total file size of about 346.88 GB, which is obviously not manageable. A sample of the power trace for 225 cycles, which equals two activations of the DCT, is shown in Figure D.31.

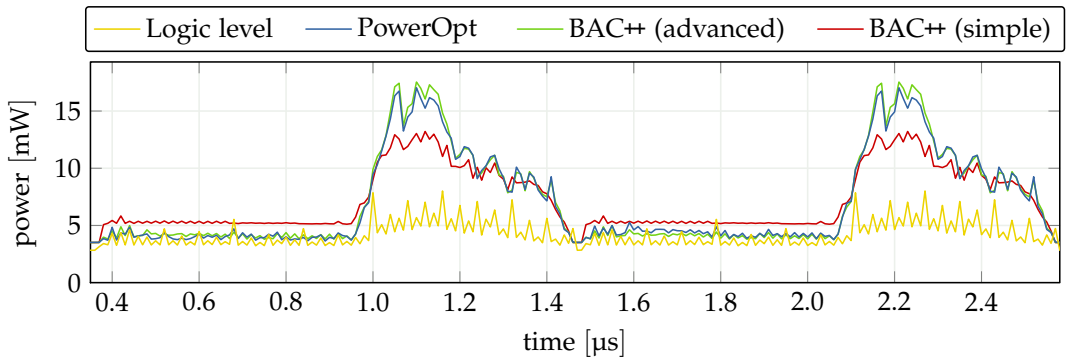


Figure D.31: Logic-level power trace — Comparison of the power traces obtained using PowerOpt and BAC++, respectively with a power trace obtained from a logic-level simulation and estimation.

Comparing the power trace obtained from PowerOpt with the trace obtained from the logic-level estimation, significant differences are apparent. The power trace obtained from the logic-level estimation shows a higher volatility regarding the cycle-by-cycle power than the one obtained from PowerOpt and both BAC++ approaches, respectively. Several factors influence the cycle-by-cycle accuracy of the estimation. PowerOpt internally uses RT-level macro models with a data abstraction that bases on the input's Hamming distance. Moreover, the logic-level power estimation uses a more accurate model for estimating the wire length, since exact wire length and thus wire load are not known until final place and route have been performed. These are only some issues that must be tackled during abstraction from logic level to RTL and making accurate estimates difficult, as shown in Section 3.1.

From Figure D.31 it is observable that the differences between the estimates highly depend on the execution phase of the design. In the first execution phase, both estimates are very close. The estimate obtained from PowerOpt is slightly higher, due to the obstacles mentioned above. This is also supported by Figure D.32 on the next page, which depicts in the blue and red bars that the PowerOpt model provides an slightly overestimation in terms of area and thus power dissipation, especially for larger RT components.

During computational phases, the power dissipation estimated at logic level is significantly lower than estimated by PowerOpt. The DCT algorithm contains a comparatively large

number of arithmetic operations with one constant input value. A deeper analysis of the Verilog model reveals that these operations also become part of the model at RTL. During logic-level synthesis, this allows an extensive optimisation of the component's internal structure, since the constant input of the corresponding RT component can be hard-wired.

The required area and thus the expected power dissipation of RT components with one fixed input can be reduced by up to 66 %, as has been shown by the FP7 ICT Project THERMINATOR [115]. During the course of the project, PowerOpt has been enhanced in order to allow for such optimisations during RT-level power estimation, as shown by the green bars in Figure D.32.

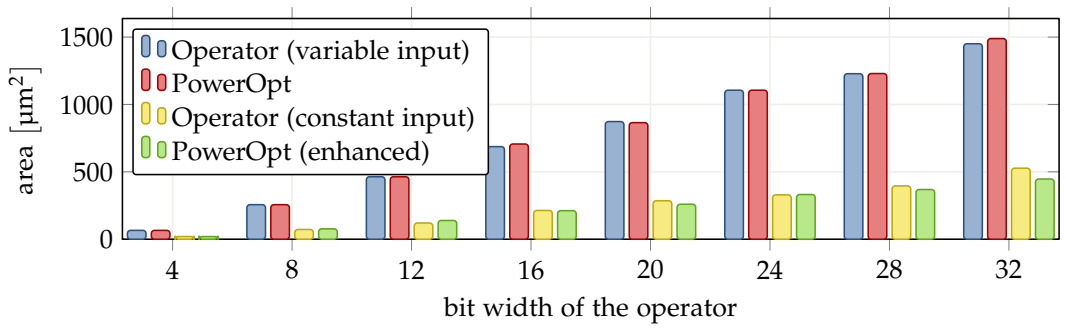


Figure D.32: Area reduction for operators with constant input — Area of a delay optimised adder for different bit widths, with and without one constant input value.

The PowerOpt version used in this thesis does not provide this feature. It thus will notably overestimate the dynamic power dissipation for arithmetic operations with one fixed input. Figure D.31 shows that the power dissipation estimated by the logic-level estimation for computational phases is about a third of the power dissipation estimated by PowerOpt and BAC++, respectively. This corresponds to the achieved optimization.

D.11.3 Summary

Summarising it can be said, that logic-level synthesis must be performed with the same technology and settings as had been used during RT-level characterisation and CDB generation. A lot of optimisation options are available to the logic-level syntheses. This makes accurate predictions of the final design and thus precise power estimates during RT-level estimation difficult. For the chosen example this is even aggravated, since features like consideration of constant inputs are not supported by the used version of PowerOpt. More meaningful results can be obtained by performing an evaluation that covers more designs and use cases and by using the latest version of PowerOpt.

The comparison of the estimates obtained from PowerOpt with the estimates obtained from BAC++ shows the same small errors as were also present for all other designs, discussed in Sections D.1 to D.10. The error that is causes by the characterisation and estimation approach, presented in this theses is negligible compared to the error due to abstraction from logic level to RTL. The accuracy of the presented approach in comparison to a logic-level estimation mainly depends on the error that is cause be the abstraction from logic level to RTL. Thus, a better low level abstraction will also significantly increase the accuracy of the approach, presented in this thesis.

Symbols

$\#_{\mathrm{H}}$	Number of hardware basic blocks.
$\#_{\mathrm{activations}}$	Number of activations.
$\#_{\mathrm{activetimes}}$	The number of times, a specific RT component or hardware basic block was active.
$\#_{\mathrm{comp}}$	The number of RT components belonging to a hardware basic block.
$\#_{\mathrm{cyc}}$	Number of clock cycles.
$\#_{\mathrm{pat}}$	Size of the pattern list of the given RT component.
$\#_{\mathrm{proc}}$	Number of processes.
a	Boolean variable assignment to the output conditions of the controller.
$\mathrm{active}^{(s,a)}$	An active path in the data path. See definition in Equation (4.11) on page 76.
activepath	An active path within a hardware basic block, which is only allowd to pass multiplexers at their selected input port. See also definition in Equation (4.11) on page 76.
α	Switching activity or probability.
C	Electrical capacitance given in farad [F]. It is the ability to store an electrical charge.
C_{l}	Switched capacitance in farad [F].
$C_{\mathrm{l}}^{\mathrm{idle}}$	Switched capacitance during idle-phases, in farad [F].
clk	A clock.
$ctrl$	A controller.
δ	State transition function of a controller's FSM.
δ	A delta function.
E	All edges of the graph representation of a data path.
E	Electrical energy given in joule [J].
E_{d}	Dynamic energy
$E^{(s,a)}$	Connections between RT components within a hardware basic block.
E_{trans}	The energy in joule [J], required for performing a power mode transition.
F	Definition of a controller's FSM.
f_{clk}	Clock frequency given in hertz [Hz] i. e., number of clock cycles per second.

f_{ref}	A reference frequency.
G	Graph representation of a RT data path.
G	Electrical conductance given in siemens [S]. It is the ease at which an electrical current can pass and is the reciprocal of the electrical resistance R.
Γ	Output alphabet of a controller.
γ	Output symbol of a controller.
H	A hardware basic block.
$H^{(s,a)}$	Graph representation of a hardware basic block, if the controller is in state s an the Boolean assignment a is applied.
I	Electrical current given in ampere [A]. It is the flow of an electrical charge.
I_{DJ}	Drain junction in ampere [A].
I_{gate}	Gate tunnelling. The combination of gate–source I_{GS}, gate–drain I_{GD}, and gate–bulk leakage I_{GB}.
I_{GB}	Gate–bulk leakage in volt [V].
I_{GD}	Gate–drain leakage in volt [V].
I_{GIDL}	Gate induced drain leakage in volt [V].
I_{GISL}	Gate induced source leakage in volt [V].
I_{GS}	Gate–source leakage in volt [V].
input	See definition in Equation (4.8) on page 76.
I_{punch}	Depletion punch-through in ampere [A].
I_{SJ}	Source junction in ampere [A].
I_{HCI}	Hot carrier injection in ampere [A].
I_{subth}	Sub-threshold current in ampere [A], if channel is closed.
k_{trans}	A scaling factor for adjusting energy dissipation during a power mode transition.
k	A generic scaling factor.
$k_{\text{C}}^{\text{adv}}$	A factor for scaling the switched capacitance of functional units and multiplexers, which has been estimated using the advanced characterisation approach.
k_{C}^{s}	A factor for scaling the switched capacitance of functional units and multiplexers, which has been estimated using the simple characterisation approach.
M	A memory.
M	A full-custom hardware module.
MAPE	Mean absolute percentage error. A definition is given in Equation (6.2) on page 130.
net	An interconnect.

O	The big O-notation $f \in \mathrm{O}(g)$ states that a given function f does not grow notably faster than function g.
ω	Output function of a controller.
P	Electrical power given in watt [W]. It is the rate at which electrical energy E is transferred by an electrical circuit.
p	A process of of a hardware module.
pat	A specific data pattern of the given RT component.
path	A path defines a set of edges directly or indirectly connecting two components of the data path.
path_H	A path inside a hardware basic block. A path is does not pass a register.
P_d	Dynamic power dissipation in watt [W]. It can be divided into power caused by charging and discharging capacities P_load, short-circuit and glitch caused power dissipation.
P_l	Static power dissipation or leakage in watt [W].
P_load	Part of dynamic power P_d that is related to charging and discharging the capacities of the design.
R	Electrical resistance given in ohm [Ω].
R^2	Coefficient of multiple determination. A definition is given in Equation (6.5) on page 131.
RE	Relative error. A definition is given in Equation (6.1) on page 130.
RMS	Root mean square error. A definition is given in Equation (6.3) on page 131.
RRMS	Relative root mean square error. A definition is given in Equation (6.4) on page 131.
R_shared	A shared register.
S	Set of states of a controller.
s	A single state of a controller.
s_0	Initial state of a controller.
(s, a)	Identification of a hardware basic block. It is a combination of the state s and variable assignment a.
$\mathrm{select}^{(s,a)}$	Determines the value of a mux-select signal, if a specific hardware basic block is active. A detailed definition is given in Equation (4.9) on page 76.
Σ	Input alphabet of a controller's FSM.
σ	Input symbol of a controller's FSM.
T	Temperature given in kelvin [K] or degree Celsius [°C].
t	A timestamp, which represents a discretized continuous time value.

t	A specific time or duration in seconds [s].
τ	Delay or timing in seconds [s] or clock cycles.
τ_{RC}	A time constant given in seconds [s], describing the time required for charging a capacitance to about 63.20 % of the difference between initial and final value.
τ_{trans}	Time in seconds [s], required for performing a power mode transition.
t_{clk}	Clock period i. e., duration of a clock cycle given in seconds [s]
V	All nodes of the graph representation of a data path.
V	The electrical voltage in volt [V] is the electrical potential difference.
v	A single RT component of the data path.
V_{C}	Constants of the data path.
V_{dd}	The supply voltage in volt [V] of a given circuit.
$\mathrm{v}(G)$	McCabe's Cyclomatic Number, measuring the complexity of the control flow within a given software.
V_{M}	Multiplexer of the data path.
V_{O}	Operations of the data path.
V_{R}	Register of the data path.
V_{ref}	A reference voltage.
$V^{(s,a)}$	Nodes of a hardware basic block.
$V_{\text{A}}^{(s,a)}$	Active nodes of a hardware basic block.
$V_{\text{E}}^{(s,a)}$	Nodes of a hardware basic block that are active due to parasitic functionality.
$V_{\text{SR}}^{(s,a)}$	A set of registers that serve as source registers for the hardware basic block i. e., provide input to the data path.
$V_{\text{TR}}^{(s,a)}$	A set of registers that serve as target registers for the hardware basic block i. e., take new values at the end of the cycle, the hardware basic block is active.
V_{ss}	Negative supply voltage or ground.
V_{th}	Threshold voltage in volt [V] of a given transistor technology.

Acronyms

AES	advanced encryption standard
ASIC	application specific integrated circuit
ASIP	application-specific instruction-set processor
BSIM	Berkeley short-channel IGFET model
CAFD	cycle-accurate functional description
CDB	RT component data base
CDFG	control and data flow graph
CES	International Consumer Electronics Show
CFG	control flow graph
CMOS	complementary metal oxide semiconductor
CPU	central processing unit
CSV	comma separated values
CTL	computation tree logic
DCT	discrete cosine transform
DMA	direct memory access
DNF	disjunctive normal form
DRAM	dynamic random access memory
DSP	digital signal processor
DVFS	dynamic voltage/frequency scaling
EDA	electronic design automation
EFSM	extended FSM
ESL	electronic system level
FDCT	fast discrete cosine transform
FFT	fast Fourier transform
FPGA	field-programmable gate array
FSM	finite-state machine
FSMD	finite state machine with data path
GPRS	general packet radio service
GPU	graphics processing unit
GSM	global system for mobile communications
HCI	hot-carrier injection
HDL	hardware description language
HTLP	hierarchical transaction-level power

IC	integrated circuit
IGFET	insulated-gate field-effect transistor
IP	intellectual property
ISS	instruction set simulator
ITRS	International Technology Roadmap for Semiconductors
LoC	Lab-on-a-Chip
LPC	linear predictive coding
LRM	local resource management
MAPE	mean absolute percentage error
MIPS	microprocessor without interlocked pipeline stages
MOSFET	metal-oxide-semiconductor field-emitting transistor
NBTI	negative-bias temperature instability
NMOS	n-channel metal-oxide semiconductor
PMOS	p-channel metal-oxide semiconductor
PMU	power management unit
PSM	power state machine
R^2	coefficient of multiple determination
RE	relative error
RMS	root mean square error
RRMS	relative root mean square error
RT	register transfer
RTL	register-transfer level
SMT	satisfiability modulo theories
SNR	signal-to-noise ratio
SoC	System-on-a-Chip
SPICE	simulation program with integrated circuit emphasis
SysML	Systems Modeling Language
TLM	transaction-level modelling
TSV	through-silicon via
UART	universal asynchronous receiver/transmitter
UML	Unified Modeling Language
VCD	value change dump
VHDL	very-high-speed integrated circuit hardware description language
WLAN	wireless local area network

Glossary

3D-stack
A three-dimensional stack of ↑*dice*.

allocation
Denotes the process of determining the amount of required resources. At RT-level it refers to determining how many ↑*functional units* of a specific type are required for implementing the behaviour.

application specific integrated circuit
An application specific integrated circuit (ASIC) is a digital IC that is designed to perform a certain and clearly distinguished task within a larger system. Typical ASICs are hardware accelerators, for example.

assignment hardware basic block
A special type of ↑*hardware basic block*. It computes the combinational input values, required by the controller.

basic block
Contains instructions that are executed atomically. Once the flow enters a basic block it leafs the block at its end without the possibility of branching or stopping. A basic block is typically bounded by conditional statements e. g., `if-then-else`-branches.

behavioural level
A description is given on page 19.

binding
Denotes the processes of assigning a given behaviour to a specific hardware resource. At RT-level that is, that a given arithmetic operation is bound to a specific ↑*functional unit* like an adder or multiplier.

clock cycle
A clock cycle denotes two transitions of the clock signal. In other words it is the time between two equal edges of the clock signal. This time is also known as *clock period*. In a typical design, a clock cycle starts with a rising edge of the clock signal.

clock domain
An area of the design running at a certain clock speed. At a border between two clock domains synchronisation must be performed. Most frequent a simple and asynchronous handshake protocol is used.

clock gating
Allows the clock signal to be gated. That is, a ↑*logic gate* is used to separate the clock signal from certain parts of the design. Typically an AND-gate is used to turn off a part of the system selectively. In the deactivated parts no ↑*parasitic functionality* can occur, since all registers are fixed to its current value.

clock-tree
A set of signals distributing the clock signal to all components of the design. A clock-tree may contain transistors allowing the turn of the clock signal for certain parts of the design. This technique is called ↑*clock gating*. A design may also contain several independent clock-trees, so-called ↑*clock domains*.

combinational macro
A macro or circuit, containing several algorithmic operations without memory (in contrast to ↑*sequential circuits*). Thus, a macro is typical enclosed by registers. The ↑*hardware basic blocks*, presented in this theses are such a macros.

computation tree logic
A temporal logic that can be used for the formal verification of computational systems. Computation tree logic (CTL) allows the specification of the temporal behaviour of a system. That is, statements about future states of the system can be made.

control and data flow graph
In a control and data flow graph (CDFG), nodes represent elementary operations such as assignments or arithmetic operation. Edges however represent control and data dependencies among nodes.

control step
A step of the controller of the design. Each control step is defined by the controllers FSM. The behaviour of the design during a certain control step is defined by the control signals enabled by the controller, activated during the particular ↑*clock cycle*.

cyber-physical system
A cyber-physical system is a composition of various distributed embedded systems, connected by a wired or wireless communication network. It often contains sensors or actors for interacting with its environment.

cycle accurate
A model is considered to be cycle accurate, if it considers each ↑*clock cycle* of the modelled design individually.

cycle-accurate functional description
A cycle-accurate functional description (CAFD) describes the behaviour of a given system with a granularity of ↑*clock cycles* i. e., it is used for performing a cycle-by-cycle simulation. All operations that occur within a ↑*clock cycle* cannot be distinguished.

data path
An RT data path is a collection of ↑*functional units* or operators, which perform the data processing in a given design. For storing information in the data path, typically registers are used. The behaviour i. e., the data flow and storage of information is controlled by a controller.

delta cycle
An intermediate step during a discrete event simulation. During a delta cycle, all intermediate values are computed. A delta cycle may cause a subsequent delta cycle e. g., if it changes a value of a signal. If no more delta cycles are required i. e., no values had been changed during the current delta cycle, a normal cycle can be performed.

delta function
This function is used for enabling or disabling terms of a mathematical equation. The delta-function typically evaluates to 1, if a specific condition is satisfied by the function's parameters. In this case, the particular term of the equation is enabled. If the given parameters do not satisfy the condition, the delta functions evaluates to 0, disabling the particular term in the equation.

design space exploration
The term design space exploration denotes the process of comparing different implementations of a given system in order to find the best solution. During design space exploration the next iteration step is computed based on the previous results.

die
A die is a small block build of semiconducting materials, implementing a functional circuit. Typically this block contains a complete processor of a SoC. A die might also contain heterogeneous modules. That is, despite computational logic it might contain analogue and mixed signal modules as well as memories. Several dice can be stacked, forming a 3-dimensional system or ↑*3D-stack*.

digital signal processor
A digital signal processor (DSP) is a processor core that is optimised for signal processing. It often contains a special architecture including circular buffers, or special instructions, which are able to operate on multiple data simultaneously.

dynamic frequency scaling
Dynamic frequency scaling (DFS) allows to change the clock frequency a logical circuit is running at.

dynamic power dissipation
Dynamic power dissipation P_d is caused by charging and discharging capacities, connected to the gate of a transistor. These capacities are charged and discharged due to the activity of the circuit build from the transistors. That is, dynamic power dissipation is directly caused by the activity and the behaviour of the particular circuit.

dynamic voltage scaling
By applying dynamic voltage scaling (DVS), supply voltage and operational frequency of a digital circuit is scaled during run-time. By adjusting the performance of the circuit to an appropriate level the ↑*power dissipation* of the circuit can be reduced.

dynamic voltage/frequency scaling
Dynamic voltage/frequency scaling (DVFS) combines ↑*dynamic frequency scaling* and ↑*dynamic voltage scaling*.

electrical level
A description is given on page 19.

electronic system level
A description is given on page 18.

entropy
A measure of the information density. The higher the entropy is, the higher is the value of the information.

extended FSM
An extended FSM (EFSM) is a conventional FSM, which has been extended with register values from the ↑*data path*. Transitions have Boolean expressions, so-called *guards*, assigned. If all guards of a transition are satisfied, the transition is fired. While bringing the machine from the current state to the next one, operations, also assigned to the transition, will be executed.

finite state machine with data path
An finite state machine with data path (FSMD) is a conventional FSM with an associated ↑*data path*. The FSM controls the data path, which performs the data processing.

functional model
The functional model contains or represents the behaviour of a module. Typically a functional model is executable like models written in an *hardware description language (HDL)*. It is also possible to provide the model in a non-executable form, like task graphs. In this thesis the term *functional model* referrers to the part of BAC++ that represents the behaviour of the characterises module.

functional unit
A component of the ↑*data path* that performs a certain functionality. Examples are adders, multipliers, etc.

glitch
A glitch is an incomplete signal transition. That is, that the output of a ↑*logic gate* starts to change its output, but then takes its original value. An example is given in Figure 2.3a on page 21.

global power manager
The global power manager implements the overall system power management policy. This is, it selects the ↑*power mode* of all modules, belonging to the system.

glue logic
A simple piece of hardware that is used for connecting two ore more complex hardware modules, whose interfaces are not directly compatible with each other.

ground voltage
The ground voltage V_{ss} or GND defined the reference point from which voltages are measured.

Hamming distance
Number of unequal bits in two consecutive bit vectors.

hardware basic block
A set of RT components that are active together. Components are considered to be active together, if they provide the input for the registers that are activated by the ↑*FSM* during the same ↑*clock cycle*. That is, the registers are enabled in the same state of the controller's FSM and if the same ↑*output condition* holds.

hazard
A complete but unnecessary transition of a ↑*logic gate*. This is typically caused by different ↑*data paths* of different length. An example of a hazard is given in Figure 2.3b on page 21.

HBB cycle
A specific execution phase in the ↑*model of computation*, presented in this thesis. During this phase the behaviour of the ↑*hardware basic block* i. e., the behaviour of the ↑*data path* is simulated.

high-level synthesis
See description of RT synthesis on page 19.

hot-spot
A small area inside the ↑*die* with a temperature that is significantly higher than the temperature of the surrounding area. Typically, hot-spots should be avoided, because they cause mechanical stress.

IR-drop
An IR-drop or voltage-drop is a drop of the ↑*supply voltage* in a certain area of the supply net.

layout synthesis
See description on page 19.

logic gate
A set of transistors implementing a Boolean logic function like AND, OR, etc. If the logic gate is available from a library, it is often called a *standard cell*.

logic level
See description on page 19.

logic synthesis
See description on page 19.

macro model
A small power model of an RT component. Typically, this model uses the actual input values of the component as input to compute the component's ↑*power dissipation*. It is also possible that some kind of abstraction, such as ↑*Hamming distance* from the input values is used.

Mealy machine
A deterministic ↑*FSM*, whose outputs depend on the actual state as well as the actual applied input symbol.

model of computation
Generally speaking, the model of computation defines how the behaviour of a system is derived from its stimuli.

Moore machine
A deterministic ↑*FSM* whose output symbol depends on the actual state, only.

output condition
A condition evaluated for obtaining the controller's output signals and the next state of the controller's FSM, respectively.

package
Contains the ↑*die/dice*.

parasitic functionality
The term parasitic functionality refers to RT components that are active, but whose results are neither needed nor used for the operation of the circuit. An example is given in Figure 2.3c on page 21.

power dissipation
Denotes the amount of electrical power that is converted into head.

power gating
A technique that allows to disconnect the gate or RT component from ↑*supply voltage* and ground, respectively.

power island
A clearly delineated area of the ↑*die* with a common supply voltage and clock frequency.

power mode model
Describes all available ↑*power modes* of a module as well as possible transitions between them.

power mode table
A table, containing all ↑*power modes* provided by a hardware module. The table also contains information about possible ↑*power mode transitions* and the associated penalty in terms of delay and power dissipation.

power mode transition
A change from one ↑*power mode* to another one. Each transition is typically connected with a penalty in terms of ↑*power dissipation* and an additional delay. Power mode transitions can only take place in certain states of the module, typically if the module is idle.

power mode
A certain configuration of a module. Typically a power mode is defined as a combination of applied ↑*supply voltage* and clock frequency. A controller is required that implements the ↑*dynamic voltage/frequency scaling* policy.

power state machine
A power state machine (PSM) is a FSM whose state denote phases in which a given system dissipates a specific power.

power & timing model
Converts information such as switched capacitance and cycle count received from the ↑*functional* and the ↑*power mode model*, respectively in order to obtain actual power in terms of watts and timing in terms of seconds, for example.

register transfer
The function of a system is described as a set of logic operations. Input values to the operation are provided by one or more register. Results are also stored in registers.

register-transfer level
Typical components are adders, multipliers, registers, etc.

RT synthesis
See description on page 19.

satisfiability modulo theories
A satisfiability modulo theories (SMT) problem is a decision problem for logical formulas with respect to combinations of background theories. Such theories can include the theory of real and integer numbers or theories of various data structures such as lists, arrays, vectors etc. In this thesis, SMT formulas are used to describe the output and next-state logic of a controller. Predicates used within the SMT formulas typically describe the state of the ↑*data path*. That is, they make statements about register values, intermediate results, or handshake signals.

scheduling
Refers to the processes of determining the order in which operations are performed. Two different types of scheduling can be identified: *static scheduling* is performed during synthesis or compile time. Once the schedule is fixed it cannot be changed. *Dynamic scheduling* however, is performed during run-time i. e., the schedule can be changed dynamically.

sequential circuit
A circuit containing operations as well as a memory (in contrast to a ↑*combinational macro*). That is, the circuit has a state and it operations depends on the actual input as well as the previous input data. The circuit can be split into ↑*data path* and *controller*.

simulation program with integrated circuit emphasis
The simulation program with integrated circuit emphasis (SPICE), is a simulation software for analogue/digital electrical circuits. Simulation is done by trying to find an approximate solution for the differential equations, describing the system. Simulations are very time-consuming but give very accurate results.

static power dissipation
Static power dissipation P_l, denotes unwanted electrical currents between the different components of a transistor, especially if the transistor should be locking.

strongly pattern dependent
A strongly pattern dependent power estimation approach requires concrete and complete signal pattern to be applied to the design.

supply voltage
The supply voltage V_{dd} is the voltage that is applied between source and drain of the gates.

system synthesis
See description on page 18.

System-on-a-Chip
A System-on-a-Chip (SoC) is a system whose components are integrated on a single IC. It may contain digital and/or analogue components.

three-address code
Three-address code is a form of representing intermediate code used to aid the implementation of code-improving transformations. Each statement typically has the general form $x := a \otimes b$, where a and b are the operands to the operator $\otimes$. The result of the operation is stored in x.

threshold voltage
The threshold voltage V_{th} is the gate voltage where an inversion layer forms at the source and drain of the transistor i. e., the depletion region starts to conduct.

use case
A use case is a defined interaction with the design under test. By using a defined input stimulus, it can be verified if the design under test shows the correct behaviour and produces the expected results.

weakly pattern dependent
A weakly pattern dependent power estimation approach requires the behaviour of the design to be specified in an probabilistic way, allowing the user to specify the typical behaviour.

zero-strength hardware basic block
A zero-strength hardware basic block does not perform any computation. It occurs, if the output symbol that is applied to the $\uparrow$*data path* does not enable any target register. In this case, no RT component is assumed to be active and thus not power is assumed to dissipate.

Bibliography

[1] COMPLEX: COdesign and power Management in PLatform-based design space EXploration. Homepage: `https://complex.offis.de/`.

[2] International technology roadmap for semiconductors (ITRS) 2008 update. Technical report, Semiconductor Industry Association, 2008.

[3] International technology roadmap for semiconductors (ITRS) 2009 edition. Technical report, Semiconductor Industry Association, 2010.

[4] International technology roadmap for semiconductors (ITRS) 2011 edition. Technical report, Semiconductor Industry Association, 2012.

[5] CHStone: A suite of benchmark programs for C-based high-level synthesis. Homepage: `http://www.ertl.jp/chstone/index.html`, January 2013. Version 1.8.

[6] Sumit Ahuja, Deepak A. Mathaikutty, and Sandeep K. Shukla. Model-based power estimation using least squares regression on FSMD models. Technical report, VirginiaTech – Virginia Polytechnic Institute and State University, 2008.

[7] Sumit Ahuja, Deepak A. Mathaikutty, Avinash Lakshminarayana, and Sandeep Shukla. Accurate power estimation of hardware co-processors using system level simulation. In Sakir Sezer, Andrew Marshall, and Thomas Buechner, editors, *Proceedings of the 21st IEEE International System-on-Chip Conference (SOCC)*, pages 399–402, September 2009. ISBN 978-1-4244-5220-0.

[8] Zaher S. Andraus and Karem A. Sakallah. Automatic abstraction and verification of verilog models. In *Proceedings of the 41st Design Automation Conference (DAC)*, pages 218–223. ACM, 2004.

[9] Wolfgang Arden, Michel Brillouët, Patrick Cogez, Mart Graef, Bert Huizing, and Reinhard Mahnkopf. More-than-moore. Technical report, Semiconductor Industry Association, 2010.

[10] Jakob Axelsson. *Analysis and Synthesis of Heterogeneous Real-Time Systems*. PhD thesis, University of Linköping, 2007.

[11] Brian Bailey, Grant Martin, and Andrew Piziali. *ESL Design and Verification: A Prescription for Electronic System-Level Methodology*. Systems in Silicon. Morgan Kaufmann, 500 Sansone Street, Suite 400, San Francisco, CA 94111, 1st edition, 2007. ISBN 978-0-12-373551-5.

[12] Nikhil Bansal, Kanishka Lahiri, Anand Raghunathan, and Srimat T. Chakradhar. Power monitors: A framework for system-level power estimation using heterogeneous power models. In *Proceedings of the 18th IEEE International Conference on VLSI Design*, pages 579–585, 2005.

[13] Karen A Bartlett, Robert K Brayton, Gary D Hachtel, Reily M Jacoby, Christopher R Morrison, Richard L Rudell, Alberto Sangiovanni-Vincentelli, and A Wang. Multi-level logic minimization using implicit don't cares. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(6):723–740, June 1988.

[14] R. Ben Atitallah, S. Niar, and J.-L. Dekeyser. MPSoC power estimation framework at transaction level modeling. In *Proceedings of the 2007 International Conference on Microelectronics (ICM)*, pages 245–248. IEEE, December 2007.

[15] Luca Benini and Giovanni De Micheli. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(2): 115–192, April 2000. ISSN 1084-4309.

[16] Luca Benini, Alessandro Bogliolo, M. Favalli, and Giovanni De Micheli. Regression models for behavioral power estimation. *Integrated Computer-Aided Engineering*, 5:95–106, April 1998. ISSN 1069-2509.

[17] Luca Benini, Robin Hodgson, and Polly Siegel. System-level power estimation and optimization. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 173–178, New York, NY, USA, August 1998. ACM Press.

[18] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):299–316, June 2000.

[19] Luca Benini, Alessandro Bogliolo, Enrico Macii, Massimo Poncino, and Mihai Surmei. Regression-based RTL power models for controllers. In *Proceedings of the 10th ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pages 147–152, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-251-4.

[20] Reinaldo A. Bergamaschi and Yunjian W. Jiang. State-based power analysis for systems-on-chip. In *Proceedings of the 40th Design Automation Conference (DAC)*, 2003.

[21] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. Regression-based RTL power modeling. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(3):337–372, July 2000. ISSN 1084-4309.

[22] Alessandro Bogliolo, Roberto Corgnati, Enrico Macii, and Massimo Poncino. Parameterized RTL power models for soft macros. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):880–887, December 2001.

[23] Nicola Bombieri, Franco Fummi, and Graziano Pravadelli. A methodology for abstracting RTL designs into TL descriptions. In *Proceedings of the 4th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 103–112, 2006.

[24] Nicola Bombieri, Franco Fummi, Graziano Pravadelli, and Joao Marques-Silva. Towards equivalence checking between TLM and RTL models. In *Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 113–122. IEEE, June 2007.

[25] Nicola Bombieri, Franco Fummi, and Graziano Pravadelli. Abstraction of RTL IPs into embedded software. In *Proceedings of the 47th Design Automation Conference (DAC)*, pages 24–29, June 2010.

[26] Nicola Bombieri, Franco Fummi, and Graziano Pravadelli. Automatic abstraction of RTL IPs into equivalent TLM descriptions. *IEEE Transactions on Computers*, 60(12):1730–1743, December 2011. ISSN 0018-9340.

[27] Nicola Bombieri, E. S. M. Ebeid, Franco Fummi, and M. Lora. On the reuse of RTL IPs for SysML model generation. In *Proceedings of the 13th International Workshop on Microprocessor Test and Verification (MTV)*, pages 54–59, 2012.

[28] Nicola Bombieri, Hung-Yi Liu, Franco Fummi, and Luca Carloni. A method to abstract RTL IP blocks into C++ code and enable high-level synthesis. In *Proceedings of the 50th Design Automation Conference (DAC)*, pages 156:1–156:9, New York, NY, USA, 2013. ACM Press. ISBN 978-1-4503-2071-9.

[29] Shekhar Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, July 1999. ISSN 0272-1732.

[30] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.

[31] Bruno Bouyssounouse and Joseph Sifakis, editors. *Embedded System Design: The ARTIST Roadmap for Research and Development*, volume 3436 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin/Heidelberg, Germany, 2005. ISBN 3-540-25107-3.

[32] Carlo Brandolese and William Fornaciari. Software energy optimization through fine-grained function-level voltage and frequency scaling. In *Proceedings of the 8thIEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 539–546, New York, NY, USA, 2012. ACM.

[33] Carlo Brandolese, Simone Corbetta, and William Fornaciari. Software energy estimation based on statistical characterization of intermediate compilation code. In *Proceedings of the 2011 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 333–338, August 2011.

[34] Maurizio Bruno, Alberto Macii, and Massimo Poncino. A statistical power model for non-synthetic RTL operators. In J.J. Chico and E. Macii, editors, *Integrated Circuit and System Design: Power and Timing Modeling, Optimization and Simulation*, volume 2799 of *Lecture Notes in Computer Science (LNCS)*, pages 208–218. Springer Verlag, September 2003. ISBN 978-3-540-20074-1.

[35] Richard Burch, Farid Najm, Ping Yang, and Timothy Tricky. A Monte Carlo approach for power estimation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 1 (1):63–71, March 1993.

[36] R.P.L. Buse and W.R. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, 2010. ISSN 0098-5589.

[37] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the USENIX ATC '10: 2010 USENIX annual technical conference*, 2010.

[38] M. A. Changizi, M. A. McDannald, and D. Widders. Scaling of differentiation in networks: Nervous systems, organisms, ant colonies, ecosystems, businesses, universities, cities, electronic circuits, and legos. *Journal of Theoretical Biology*, 218(2):215–237, 2002. ISSN 0022-5193.

[39] W. H. Cheng and B. M. Baas. Dynamic voltage and frequency scaling circuits with two supply voltages. In *Proceedings of the 2008 IEEEInternational Symposium on Circuits and Systems (ISCAS)*, pages 1236–1239, 2008.

[40] George A Constantinides, Peter Y. K. Cheung, and Wayne Luk. Synthesis of saturation arithmetic architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 8(3):334–354, 2003.

[41] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001. ISBN 9780072970548.

[42] Intel Corporation. Fun facts: Exactly how small (and cool) is 32 nanometers? Technical report, January 2010.

[43] Intel Corporation. Microprocessor quick reference guide. Homepage: `http://www.intel.com/pressroom/kits/quickreffam.htm`, February 2013.

[44] Intel Corporation. Intel ARK. Homepage: `http://ark.intel.com/`, February 2013.

[45] Philippe Coussy, Daniel D. Gajski, Michael Meredith, and Andres Takach. An introduction to high-level synthesis. *Design & Test of Computers, IEEE*, 26(4):8–17, 2009.

[46] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES – The Advanced Encryption Standard*. Springer-Verlag, Berlin/Heidelberg, Germany, 2002. ISBN 3-540-42580-2.

[47] Robertas Damaševičius and Vytautas Štuikys. Estimation of power consumption at behavioral modeling level using SystemC. *EURASIP Journal of Embedded Systems*, 2007, May 2007. Article ID 68673.

[48] Jack B. Dennis. The design and construction of software systems. In F.L. Bauer, J.B. Dennis, W.M. Waite, C.C. Gotlieb, R.M. Graham, M. Griffiths, H.J. Helms, B. Morton, P.C. Poole, and D. Tsichritzis, editors, *Software Engineering*, volume 30 of *Lecture Notes in Computer Science (LNCS)*, chapter 1.B, pages 12–28. Springer-Verlag, Berlin/Heidelberg, Germany, 1975. ISBN 978-3-540-07168-6.

[49] Nagu Dhanwada, Ing-Chao Lin, and Vijay Narayanan. A power estimation methodology for SystemC transaction level models. In *Proceedings of the 3rdIEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 142–147, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-161-9.

[50] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Power challenges may end the multicore era. *Communications of the ACM*, 56(2):93–102, February 2013. ISSN 0001-0782.

[51] Stijn Eyerman and Lieven Eeckhout. Fine-grained DVFS using on-chip regulators. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(1):1:1–1:24, April 2011. ISSN 1544-3566.

[52] William Fornaciari, Paolo Gubian, Donatella Sciuto, and Cristina Silvano. Power estimation of embedded systems: A hardware/software codesign approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(2):266–275, June 1998.

[53] Daniel D. Gajski and Robert H. Kuhn. New VLSI tools. *IEEE Computer*, 16(12):11–14, December 1983. ISSN 0018-9162.

[54] Daniel D. Gajski, Nikil D. Dutt, Allen C.-H. Wu, and Steve Y.-L. Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992. ISBN 0-7923-9194-2.

[55] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer-Verlag, 2009.

[56] Andreas Gerstlauer and Daniel D. Gajski. System-level abstraction semantics. Technical Report CECS-02-17, Center for Embedded Computer Systems, University of California, Irvine, CA 92697-3425, USA, July 2002.

[57] Andreas Gerstlauer, Suhas Chakravarty, Manan Kathuria, and Parisa Razaghi. Abstract system-level models for early performance and power exploration. In *Proceedings of the 2012 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 213–218, February 2012. invited paper.

[58] Georges Gielen, Pieter De Wit, Elie Maricau, Johan Loeckx, Javier Martin-Martinez, Ben Kaczer, Guido Groeseneken, R Rodríguez, and M. Nafría. Emerging yield and reliability challenges in nanometer CMOS technologies. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, pages 1322–1327. IEEE, 2008.

[59] Jacopo Giorgetti, Giuseppe Scotti, Andrea Simonetti, and Alessandro Trifiletti. Analysis of data dependence of leakage current in CMOS cryptographic hardware. In *Proceedings of the 17th ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pages 78–83, New York, NY, USA, 2007. ACM, ACM Press.

[60] Tony Givargis, Frank Vahid, and Jörg Henkel. A hybrid approach for core-based system-level power modeling. In *Proceedings of the 2000 Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 141–146, New York, NY, USA, 2000. ACM Press. ISBN 0-7803-5974-7.

[61] D. J. Greaves. A Verilog to C compiler. In *Proceedings of the 11th International Workshop on Rapid System Prototyping (RSP)*, pages 122–127, 2000.

[62] Dick Grune, Kees von Reeuwijk, Henri E. Bal, Ceriel J. H. Jacobs, and Koen Langendoen. *Modern compiler design*. Springer-Verlag, New York, NY, USA, 2nd edition, July 2012. ISBN 978-1461446989.

[63] Kim Grüttner, Kai Hylla, Sven Rosinger, and Wolfgang Nebel. Towards an ESL framework for timing and power aware rapid prototyping of HW/SW systems. In *Proceedings of the 2010 Forum on Specification & Design Languages (FDL)*, pages 56–61, September 2010.

[64] Kim Grüttner, Kai Hylla, Sven Rosinger, and Philipp A. Hartmann. Enabling timing and power aware virtual prototyping of HW/SW systems. In *Workshop on Micro Power Management for Macro Systems on Chip (uPM2SoC) at Design, Automation and Test in Europe (DATE)*, March 2011. Extended abstract and poster.

[65] Kim Grüttner, Philipp Hartmann, Kai Hylla, Sven Rosinger, Carlo Brandolese, William Fornaciari, Gianluca Palermo, Davide Quaglia, Wolfgang Nebel, Chantal Ykman-Couvreur, Francisco Ferrero, Raul Valencia, Fernando Herrera, and Eugenio Villar. COMPLEX: COdesign and power Management in PLatform-based design space EXploration. In *Proceedings of the 2012 Euromicro Symposium on Digital System Design (DSD)*, 2012. Invited paper.

[66] Kim Grüttner, Kai Hylla, Sven Rosinger, and Wolfgang Nebel. Rapid prototyping of complex HW/SW systems using a timing and power aware ESL framework. Number 106 in Lecture Notes in Electrical Engineering (LNEE), chapter 10, pages 157–174. Springer-Verlag, New York, NY, USA, 2012. ISBN 978-1-4614-1426-1.

[67] Kim Grüttner, Philipp A. Hartmann, Kai Hylla, Sven Rosinger, Wolfgang Nebel, Fernando Herrera, Eugenio Villar, Carlo Brandolese, William Fornaciari, Gianluca Palermo, Chantal Ykman-Couvreur, Davide Quaglia, Francisco Ferrero, and Raúl Valencia. The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, 37(8):966–980, November 2013.

[68] Subodh Gupta and Farid N. Najm. Energy-per-cycle estimation at RTL. In *Proceedings of the 1999 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 121–126, New York, NY, USA, 1999. ACM Press. ISBN 1-58113-133-X.

[69] Subodh Gupta and Farid N. Najm. Energy and peak-current per-cycle estimation at RTL. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(4):525–537, August 2003.

[70] Yuko Hara, Hiroyuki Tomiyama, Shinya Honda, and Hiroaki Takada. Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis. *Journal of Information Processing*, 17:242–254, October 2009.

[71] Juris Hartmanis and R. E. Stearns. *Algebraic structure theory of sequential machines*. Prentice-Hall international series in applied mathematics. Prentice-Hall, 1966.

[72] Ahmed E. Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering (ICSE)*, ICSE '09, pages 78–88, Washington, DC, USA, 2009. IEEE Press. ISBN 978-1-4244-3453-4.

[73] Domenik Helms. *Leakage Models for High Level Power Estimation*. PhD thesis, Carl von Ossietzky Universität Oldenburg, Uhlhornsweg 49-55, 26129 Oldenburg, Germany, October 2009.

[74] Domenik Helms, Eike Schmidt, Arne Schulz, Ansga Stammermann, and Wolfgang Nebel. An improved power macro-model for arithmetic datapath components. In Bertrand Hochet, Antonio Acosta, and Manuel Bellido, editors, *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, volume 2451 of *Lecture Notes in*

Computer Science (LNCS), pages 544–550. Springer-Verlag, Berlin/Heidelberg, Germany, 2002. ISBN 978-3-540-44143-4.

[75] Domenik Helms, Eike Schmidt, and Wolfgang Nebel. Leakage in CMOS circuits: An introduction. In *Proceedings of the 14th International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, volume 3254 of *Lecture Notes in Computer Science (LNCS)*, pages 17–35, Berlin/Heidelberg, Germany, September 2004. Springer-Verlag.

[76] Domenik Helms, Günter Ehmen, and Wolfgang Nebel. Analysis and modeling of subthreshold leakage of RT-components under PTV and state variation. In *Proceedings of the 2006 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 220–225, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-462-6.

[77] Ramin Hojati and Robert K. Brayton. Automatic datapath abstraction in hardware systems. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 98–113, London, UK, 1995. Springer-Verlag. ISBN 3-540-60045-0.

[78] Kai Hylla, Philipp A. Hartmann, Domenik Helms, and Wolfgang Nebel. Early power & timing estimation of custom hardware blocks based on automatically generated combinatorial macros. In *16. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, pages 147–158, March 2013.

[79] Ahmed Amine Jerraya, Olivier Franza, Markus Levy, Masao Nakaya, Pierre Paulin, Ulrich Ramacher, Deepu Talla, and Wayne Wolfgang. Envisioning the future for multiprocessor SoC. *IEEE Design and Test of Computers*, 24(2):174–183, March 2007.

[80] Gerd Jochens, Lars Kruse, Eike Schmidt, and Wolfgang Nebel. A new parameterizable power macro-model for datapath components. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, pages 29–36, 1999.

[81] Bernhard Katzmarski and Rainer Koschke. Program complexity metrics and programmer opinions. pages 17–26, 2012.

[82] Felipe Klein, Guido Araujo, Rodolfo Azevedo, Roberto Leao, and Luiz C. V. dos Santos. An efficient framework for high-level power exploration. In *Proceedings of the 50th Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1046–1049, August 2007.

[83] Felipe Klein, Roberto Leao, Guido Araujo, Luiz Claudio Villar dos Santos, and Rodolfo Azevedo. PowerSC: A SystemC-based framework for power estimation. Technical report, Instituto de computação, Universidade Estadual De Campinas, February 2007.

[84] Gummidipudi Krishnaiah, Preeti Ranjan Panda, Ashok Jagannathan, Sreenivas Subramoney, and Anshul Kumar. Unified modeling abstraction for fast simulation and emulation. Technical report, 2008.

[85] Marcello Lajolo, Anand Raghunathan, Sujit Dey, and Luciano Lavagno. Efficient power co-estimation techniques for system-on-chip design. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, pages 27–34, 2000.

[86] Paul Landman. High-level power estimation. In *Proceedings of the 1996 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 29–35, Piscataway, NJ, USA, 1996. IEEE Press.

[87] Paul E. Landman and Jan M. Rabaey. Architectural power analysis: The dual bit type method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(2):173–187, June 1995. ISSN 1063-8210.

[88] Hugo Lebreton and Pascal Vivet. Power modeling in SystemC at transaction level, application to a DVFS architecture. *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 0:463–466, April 2008.

[89] Henrik Lipskoch. *Optimisation of battery operating life considering software tasks and their timing behaviour*. PhD thesis, Carl von Ossietzky Universität Oldenburg, Uhlhornsweg 49-55, 26129 Oldenburg, Germany, 2010.

[90] Rafael Peset Llopis and Kees Goossens. The petrol approach to high-level power estimation. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 130–132, 1998.

[91] Daniel Lorenz, Kim Grüttner, Nicola Bombieri, Valerio Guarnieri, and Sara Bocchio. From RTL IP to functional system-level models with extra-functional properties. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. ACM, October 2012. Invited paper.

[92] Daniel Lorenz, Philipp A. Hartmann, Kim Grüttner, and Wolfgang Nebel. Non-invasive power simulation at system-level with SystemC. In José L. Ayala, Delong Shang, and Alex Yakovlev, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 7606 of *Lecture Notes in Computer Science (LNCS)*, pages 21–31. Springer-Verlag, Berlin/Heidelberg, Germany, September 2012. ISBN 978-3-642-36156-2.

[93] Enrico Macii and Massimo Poncino. *Power Macro-Models for High-Level Power Estimation*, chapter 39, pages 39–1 – 39–18. CRC Computer Engineering. CRC Press, January 2005. ISBN 978-0849319419.

[94] Enrico Macii, Massoud Pedram, and Fabio Somenzi. High-level power modeling, estimation, and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1061–1079, November 1998.

[95] Marco Maldari, Massimo Conti, M. Coppola, P. Crippa, S. Orcioni, L. Pieralisi, and C. Turchetti. System-level power analysis methodology applied to the AMBA AHB bus. In *Proceedings of the Design, Automation and Test in Europe (DATE)*, pages 32–37. IEEE, 2003.

[96] Diana Marculescu, Radu Marculescu, and Massoud Pedram. Information theoretic measures for power analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(6):599–610, June 1996. ISSN 0278-0070.

[97] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2 (4):308–320, December 1976.

[98] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[99] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8):114–117, April 1965.

[100] Farid N. Najm. Transition density: A new measure of activity in digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(2):310–323, 1993.

[101] Farid N. Najm. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):446–455, December 1994. ISSN 1063-8210.

[102] Ravi Namballa, Nagarajan Ranganathan, and Abdel Ejnioui. Control and data flow graph extraction for high-level synthesis. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 187–192, February 2004.

[103] Siva G. Narendra and Anantha P. Chandrakasan. *Leakage in Nanometer CMOS Technologies*. Series on Integrated Circuits and Systems. Springer-Verlag, New York, NY, USA, 2006. ISBN 9780387257372.

[104] Wolfgang Nebel and Jean Mermet, editors. *Low Power Design in Deep Submicron Electronics*, volume 337 of *NATO Advanced Science Institutes Series E: Applied Sciences*. Kluwer Academic Publishers, 1997. ISBN 0-7923-4569-X.

[105] Mahadevamurty Nemani and Farid N. 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, June 1999.

[106] Ralf Niemann. *Hardware/Software Co-Design for Data Flow Dominated Embedded Systems*. Kluwer Academic Publishers, 1998. ISBN 0-7923-8299-4.

[107] Gregor Nitsche, Kim Grüttner, and Wolfgang Nebel. Power contracts: A formal way towards power-closure?! In *Proceedings of the 23nd International Workshop on Power And Timing Modeling, Optimization and Simulation (PATMOS)*, 2013. Publication pending.

[108] Nachiketh R. Potlapally, Anand Raghunathan, Ganesh Lakshminarayana, Michael S. Hsiao, and Srimat T. Chakradhar. Accurate power macro-modeling techniques for complex RTL circuits. In *Proceedings of the 14thIEEE International Conference on VLSI Design*, pages 235–241, January 2001.

[109] Jan M. Rabaey, Aanantha P. Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall electronics and VLSI series. Pearson Education, Upper Saddle River, NJ, USA, 2nd edition, January 2003. ISBN 978-0130909961.

[110] Jan M. Rabaey, Daniel Burke, Ken Lutz, and John Wawrzynek. Workloads of the future. *IEEE Design and Test of Computers*, 25(4):358–365, 2008.

[111] Anand Raghunathan, Niraj K. Jha, and Sujit Dey. *High-level power analysis and optimization*. Kluwer Academic Publishers, 1998.

[112] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. Efficient RTL power estimation for large designs. In *Proceedings of the 16thIEEE International Conference on VLSI Design*, 2003.

[113] Danny Rittman. Nanometer reliability. *System Design Frontier*, 2(11):12–32, November 2005.

[114] Sven Rosinger. *RT-Level Power-Gating Models optimizing Dynamic Leakage-Management*. PhD thesis, Carl von Ossietzky Universität Oldenburg, Uhlhornsweg 49-55, 26129 Oldenburg, Germany, September 2012.

[115] Sven Rosinger. Framework overview for an all level thermal simulation of 3D SiP stacks and 2D SoCs. Technical Report THERMINATOR/D6.2.1/v1, THERMINATOR Project Office, December 2012.

[116] Sven Rosinger, Domenik Helms, and Wolfgang Nebel. RTL power modeling and estimation of sleep transistor based power gating. In Nadine Azémard and Lars Svensson, editors, *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, volume 4644 of *Lecture Notes in Computer Science (LNCS)*, pages 278–287. Springer-Verlag, 2007. ISBN 978-3-540-74441-2.

[117] Takayasu Sakurai. Alpha power-law MOS model. *IEEE Solid-State Circuits Society Quarterly Newsletter*, 9(4):4–5, October 2004.

[118] Takayasu Sakurai and A. Richard Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, 25(2):584–594, apr 1990. ISSN 0018-9200.

[119] Björn Sander, Jürgen Schnerr, and Oliver Bringmann. ESL power analysis of embedded processors for temperature and reliability estimations. In *Proceedings of the 7thIEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 239–248, New York, NY, USA, 2009. ACM Press. ISBN 978-1-60558-628-1.

[120] A. Sangiovanni-Vincentelli. Quo vadis, sld? reasoning about the trends and challenges of system level design. *Proceedings of the IEEE*, 95(3):467–506, March 2007. ISSN 0018-9219. Invited paper.

[121] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, February 2002.

[122] Alberto Sangiovanni-Vincentelli. Is a unified methodology for system-level design possible? *IEEE Design and Test of Computers*, 25(4):346–357, 2008. ISSN 0740-7475.

[123] John Sanguinetti. Abstraction and standardization in hardware design. *IEEE Design and Test of Computers*, 29(2):8–13, April 2012. ISSN 0740-7475.

[124] Eike Schmidt, Gerd von Cölln, Lars Kruse, F. Theeuwen, and Wolfgang Nebel. Memory power models for multilevel power estimation and optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(2):106–109, 2002. ISSN 1063-8210.

[125] Martin Shepperd. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, 3(2):30–36, 1988. ISSN 0268-6961.

[126] Bing J Sheu, Donald L Scharfetter, P-K Ko, and M-C Jeng. BSIM: Berkeley short-channel IGFET model for MOS transistors. *IEEE Journal of Solid-State Circuits*, 22(4):558–566, 1987. ISSN 0018-9200.

[127] Sandeep K. Shukla and Rajesh K. Gupta. A model checking approach to evaluating system level dynamic power management policies for embedded systems. In *Proceedings of the 6thIEEE International High-Level Design, Validation and Test Workshop (HLDVT)*, pages 53–57, Washington, DC, USA, November 2001. IEEE Press. ISBN 0-7695-1411-1.

[128] Ansgar Stammermann, Domenik Helms, Milan Schulte, Arne Schulz, and Wolfgang Nebel. Binding, allocation and floorplanning in low power high-level synthesis. In *Proceedings of the 2003 ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 544–550, Washington, DC, USA, 2003. IEEE Press. ISBN 1-58113-762-1.

[129] William Stoye, David Greaves, Neil Richards, and James Green. Using RTL-to-C++ translation for large SoC concurrent engineering: A case study. *Electronics Systems and Software*, 1(1):20–25, February 2003.

[130] Martin Streubühr, Rafael Rosales, Ralph Hasholzner, Christian Haubelt, and Jürgen Teich. ESL power and performance estimation for heterogeneous MPSoCs using SystemC. In *Proceedings of the Forum on Specification & Design Languages (FDL)*, pages 202–209. ECSI, September 2011.

[131] Jürgen Teich. *Digitale Hardware/Software-Systeme: Synthese und Optimierung*. 1997. ISBN 3-540-62433-3. In German.

[132] Inc. Underbit Technologies. MAD: MPEG audio decoder. Homepage: `http://www.underbit.com/products/mad/`, February 2013.

[133] Bart Vanthournout, Davide Quaglia, Philipp A. Hartmann, Carlo Brandolese, Gianluca Palermo, William Fornaciari, Héctor Posadas, Fernando Herrera, and Chantal Ykman-Couvreur. Final report on system simulation and profiling. Technical Report COMPLEX/SNPS/R/D3.1.2/1.1, COMPLEX project deliverable, February 2013.

[134] Emmanuel Vaumorin, Bart Vanthournout, Sara Bocchio, Davide Quaglia, Fernando Herrera, Pablo Peñil del Campo, Eugenio Villar, Kai Hylla, Tiemo Fandrey, Philipp A. Hartmann, and Kim Grüttner. Final report on virtual system generation. Technical Report COMPLEX/MDS/R/D2.5.3/1.0, COMPLEX project deliverable, December 2012.

[135] G. B. Vece, Massimo Conti, and Simone Orcioni. PK tool 2.0: A SystemC environment for high level power estimation. In *Proceedings of the 12thIEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, pages 1–4, December 2005.

[136] Yossi Veller, Vasile Hanga, Alexander Rozenman, and Rami Rachamim. Power modelling in circuit designs, June 2007.

[137] Qiang Wu, Yunfeng Wang, Jinian Bian, Weimin Wu, and Hongxi Xue. A hierarchical CDFG as intermediate representation for hardware/software codesign. In *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, volume 2, pages 1429–1432, June 2002.

[138] Qing Wu, Qinru Qiu, Massoud Pedram, and Chih-Shun Ding. Cycle-accurate macromodels for RT-level power analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(4):520–528, December 1998. ISSN 1063-8210.

[139] Spiros Xanthos, Alexander Chatzigeorgiou, and George Stephanides. Energy estimation with SystemC: A programmer's perspective. In *Proceedings of the 7th WSEAS International Conference on Systems, Computational Methods in Circuits and Systems Applications*, pages 7–10, July 2003.

[140] Xilinx. Pocket power estimator (PPE) app. Homepage: `http://www.xilinx.com/products/technology/power/`, February 2013.

[141] Xilinx. XPower estimator (XPE). Homepage: `http://www.xilinx.com/products/design_resources/power_central/`, February 2013.

[142] Vittorio Zaccaria, Mariagiovanna Sami, Donatelle Sciuto, and Cristina Silvano. *Power Estimation and Optimization Methodologies for VLIW-based Embedded Systems*. Kluwer Academic Publishers, 1st edition, February 2003. ISBN 978-1402073779.

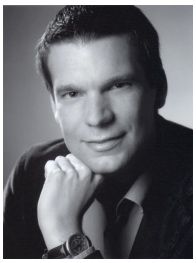
[143] Bernard P. Zeigler, Herbert Praehofer, and Tag Gon Kim. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, London, UK, 2nd edition, January 2000. ISBN 978-0-12-778455-7.

[144] Lin Zhong, Srivastava Ravi, Anand Raghunathan, and Niraj K. Jha. Power estimation for cycle-accurate functional descriptions of hardware. In *Proceedings of the 2004 ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 668–675, November 2004.

[145] Lin Zhong, Srivastava Ravi, Anand Raghunathan, and Niraj K. Jha. RTL-aware cycle-accurate functional power estimation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2103–2117, October 2006. ISSN 0278-0070.

Index

Curriculum vitae



Kai Hylla was born in 1980 in Bremen, Germany. After high school, he joint STN ATLAS Elektronik GmbH as apprentice. During his training among other things he worked on the unmanned underwater vehicle SeaFox. After finishing his apprenticeship in 2003, he started his studies at the Carl von Ossietzky University of Oldenburg with focus on embedded systems and micro robotics. Already during his studies, he dealt with modelling and simulation of microelectronic circuits at system level. Always with respect to industrial applicability. As part of his studies, together with a group of students he developed an FPGA-based autonomous underwater vehicle, including its chassis and shell, visual object recognition as well as a sonar system.

During his studies he was also with Robert Bosch GmbH, where he worked on FPGA-based prototypes of driver assistance systems, utilising computer vision. At Robert Bosch GmbH he also did his master's thesis *Erweiterung einer VHDL/SystemC-Verifikationsumgebung zur Anwendung auf MATLAB/Simulink-Modelle*, dealing with the co-simulation of multiple SystemC models inside a MATLAB/Simulink environment in the area of electronic design automation.

After receiving his diploma in 2008, he joint OFFIS – Institute for Information Technology as scientific engineer and doctoral candidate. At OFFIS he worked on several German and European research projects. These include the FP7 project *COMPLEX – COdesign and power Management in PLatform-based design space EXploration* and the BMBF project *NEEDS – Nanoelektronik-Entwurf für 3D-Systeme*, where he also was responsible for the internal project management.

Mr. Hylla is (co-)author of several scientific papers and articles. Currently he researches in the area of power estimation and its influence on the system at a high level of abstraction.

List of Own Publications

[1] Kim Grüttner, Kai Hylla, Sven Rosinger, and Wolfgang Nebel. Towards an ESL framework for timing and power aware rapid prototyping of HW/SW systems. In *Proceedings of the 2010 Forum on Specification & Design Languages (FDL)*, pages 56–61, September 2010.

[2] Kim Grüttner, Kai Hylla, Sven Rosinger, and Philipp A. Hartmann. Enabling timing and power aware virtual prototyping of HW/SW systems. In *Workshop on Micro Power Management for Macro Systems on Chip (uPM2SoC) at Design, Automation and Test in Europe (DATE)*, March 2011. Extended abstract and poster.

[3] Kim Grüttner, Philipp Hartmann, Kai Hylla, Sven Rosinger, Carlo Brandolese, William Fornaciari, Gianluca Palermo, Davide Quaglia, Wolfgang Nebel, Chantal Ykman-Couvreur, Francisco Ferrero, Raul Valencia, Fernando Herrera, and Eugenio Villar. COMPLEX: COdesign and power Management in PLatform-based design space EXploration. In *Proceedings of the 2012 Euromicro Symposium on Digital System Design (DSD)*, 2012. Invited paper.

[4] Kim Grüttner, Kai Hylla, Sven Rosinger, and Wolfgang Nebel. Rapid prototyping of complex HW/SW systems using a timing and power aware ESL framework. Number 106 in Lecture Notes in Electrical Engineering (LNEE), chapter 10, pages 157–174. Springer-Verlag, New York, NY, USA, 2012. ISBN 978-1-4614-1426-1.

[5] Kim Grüttner, Philipp A. Hartmann, Kai Hylla, Sven Rosinger, Wolfgang Nebel, Fernando Herrera, Eugenio Villar, Carlo Brandolese, William Fornaciari, Gianluca Palermo, Chantal Ykman-Couvreur, Davide Quaglia, Francisco Ferrero, and Raúl Valencia. The COMPLEX reference framework for HW/SW co-design and power management supporting platform-based design-space exploration. *Microprocessors and Microsystems: Embedded Hardware Design (MICPRO)*, 37(8):966–980, November 2013.

[6] Domenik Helms, Kim Grüttner, Reef Eilers, Malte Metzdorf, Kai Hylla, Frank Poppen, and Wolfgang Nebel. Considering variation and aging in a full chip design methodology at system level. Submitted to: Design, Automation and Test in Europe (DATE) 2014.

[7] Domenik Helms, Kai Hylla, and Wolfgang Nebel. Hybrid logical-statistical simulation with thermal and IR-drop mapping for degradation and variation prediction. In *Proceedings of the 2009 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 33–38, August 2009.

[8] Domenik Helms, Kai Hylla, and Wolfgang Nebel. Logisch-statistische simulation digitaler systeme mit temperatur- und spannungskartierung zur vorhersage von variations- und alterungseffekten. In *Zuverlässigkeit und Entwurf (ZuE)*, volume 61, pages 87–92, Berlin/Offenbach, Germany, September 2009. GMM/GI/ITG, VDE Verlag. In German.

[9] Kai Hylla, Jan-Hendrik Oetjens, and Wolfgang Nebel. Using SystemC for an extended MATLAB/Simulink verification flow. In *Proceedings of the 2008 Forum on Specification & Design Languages (FDL)*, pages 221–226, September 2008. ISBN 978-1-4244-2264-7.

[10] Kai Hylla, Jan-Hendrik Oetjens, and Wolfgang Nebel. An advanced simulink verification flow using SystemC. In Martin Radetzki, editor, *Languages for Embedded Systems and their Applications: Selected Contributions on Specification, Design, and Verification from FDL'08*, volume 36 of *Lecture Notes in Electrical Engineering (LNEE)*, chapter 5, pages 71–84. 1st edition, May 2009. ISBN 978-1-4020-9713-3.

[11] Kai Hylla, Malte Metzdorf, Armin Grünewald, Kai Hahn, Andy Heinig, Uwe Knöchel, Susann Wolf, Felix Miller Thomas Wild, Artur Quiring, Markus Olbrich, Sebastian Sattler, and Dieter Treytnar. NEEDS: Nanoelektronik-Entwurf für 3D-Systeme. In *Zuverlässigkeit und Entwurf (ZuE)*, volume 73, pages 22–29, Berlin/Offenbach, Germany, September 2012. GMM/GI/ITG, VDE Verlag. ISBN 978-3-8007-3445-0. In German.

[12] Kai Hylla, Philipp A. Hartmann, Domenik Helms, and Wolfgang Nebel. Early power & timing estimation of custom hardware blocks based on automatically generated combinatorial macros. In *16. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, pages 147–158, March 2013.

Declaration of authenticity
Ehrenwörtliche Erklärung

I herewith declare to have written this doctoral thesis only based on the sources listed and without the help of others. The work was carried out in accordance with the DFG guidelines for scientific work and without support of outside agencies or counselling services. I have not submitted or prepared the submission of this or any other doctoral thesis at the Carl von Ossietzky University of Oldenburg or any other university.

Hiermit erkläre ich, diese Doktorarbeit ohne fremde Hilfe und nur unter Verwendung der angegebenen Quellen verfasst zu haben. Die Arbeit erfolgte in Übereinstimmung mit den DFG Richtlinien für wissenschaftliches Arbeiten und ohne Inanspruchnahme externer Vermittlungs- oder Beratungsleistungen. Ich habe bis dato weder an der Carl von Ossietzky Universität Oldenburg noch an einer anderen Universität die Eröffnung eines Promotionsverfahrens beantragt oder anderweitig eine Promotion vorbereitet.

Oldenburg, den 30. September 2013

K. Hylla