NTNU
Norwegian University of
Science and Technology

Fine-Grained ASIP Power Gating

Gate-level study of the break-even point for fine-grained power gating

Anja Niedermeier

Master of Science in Electronics
Submission date: September 2009
Supervisor: Kjetil Svarstad, IET
Co-supervisor: Frank Bouwens, IMEC-NL

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

In a not too distant future, millions of tiny, invisible, self-sufficient, and communicating wireless sensors will assist our daily lives and improve the quality of life. Such wireless sensor nodes (WSN) consist out of components such as a radio for communication, sensors, and a processor for on-node processing of the data. The WSNs use scavenged energy harvested from the environment to avoid dependency of a depleting battery. The scavenged energy, however, is neither high nor continuous and requires that the aforementioned components are highly optimized for low energy consumption.

The application specific instruction set processors (ASIP) developed at Holst Centre / IMEC-NL are targeting high processing power (> 100MIPS) with a minimum amount of power (under 100uW) in the domains of biomedical and wireless communication. This requires high optimizations for both dynamic and static power. The ASIPs consist out application specific issue slots, function units, register files and/ or memories, which remain unused for other algorithms. Since leakage becomes significant if not dominant with the upcoming technologies, it is desirable to switch off unused resources at a fine-grained level.

The purpose of this thesis is to optimize a wireless communication processor for energy consumption without loss of performance by using power shut-off techniques applied to the data path of the processor. The ultra wide band processor is available as a reference generated with Target® and TSMC 90nm technology. Based on preliminary findings of simulations the processor will be optimized for energy wherever possible using fine-grained hardware techniques supported by software.

Specific tasks which are to be performed during this assignment include a detailed research of methods for power-off techniques; the results are analyzed and compared in terms of a multi-issue VLIW-processor. Based on these findings, a power-off scheme is proposed and implemented on the architectural level. Finally, the scheme is analyzed and qualified to determine the realistic power consumption benefit for the system.

Assignment given: 09. March 2009
Supervisor: Kjetil Svarstad, IET

Abstract

Power consumption in portable electronic devices is a crucial design factor. While technology at 90 nm and above is still dominated by dynamic power, it is expected that leakage power will gain importance in sub-90 nm technologies. One commonly used technique to reduce leakage power is power gating, which is still an active research topic, especially on the fine-grained level.

The purpose of this thesis was to explore the impact of fine-grained power gating on the datapath of a VLIW processor. Also, a detailed analysis of the savings versus introduced overhead was performed to derive a generic formula for a quick estimation of the energy efficiency of power gating.

During the work, a work-flow to partition the system into power domains was developed. Furthermore, a verification method was implemented that validates whether a power gated resource is scheduled by the compiler or not. A configurable HSPICE simulation flow was implemented to determine how many power switches were required for a specific power domain as well as the energy consumption to switch a power domain on.

Two processors with different usage profiles, designed with the help of different tools, were investigated in this thesis. The processors were modified to support power gating, and, furthermore, a synchronous power manager was developed. After RTL-level verification of functional correctness, the resulting systems were synthesised and placed and routed for 100 Mhz with two different 90nm TSMC libraries (low power and general purpose) to evaluate the variation between different technology flavours. The results showed, that a large contributor to the energy overhead of power gating is the dynamic power of additionally required modules, e.g., the isolation cells at the output of a power domain and the power manager. Also it was proven that the power domains need a very low duty cycle in order to apply power gating efficiently.

It has been shown that the energy overhead for fine-grained power gating is significant and it is mainly caused by additional modules that have to be added to the system. Therefore, power gating can only be beneficial on designs with sufficient large power domains with a low duty cycle. However, it must be said that power is mainly consumed by the memories. Also, for 90 nm, leakage power is a rather small fraction of the total power consumption. The possible overall savings when focussing on the datapath are therefore very limited.

Preface

Writing my Master Thesis in IMEC-NL was a great experience. I received very helpful support and guidance during my work and I also learned a lot about processor desing and low-power techniques. First of all, I would like to thank Frank Bouwens, who was my daily supervisor and a great guide through my thesis. Also he was very supportive in technical matters as well as he helped me organising the structure of my thesis. I also received great help from Jos Hulzink for the implementation of power gating in the processors itself. He also tought me a lot about ASIP design in general. A special thanks I want to give to Michael De Nil, who always helped me when I had problems with synthesis or place and route. Also, he always managed to give me new motivation when I was despairing of the tools. Also Ben Busze I would like to thank for his help with synthesis and place and route. For the circuit-level related topics I was involved in I received great help of Yu Pu, who showed me how to determine the power-on energy, and from Maryam Ashouei who helped me with question related to SPICE as well as with an estimation how many switches I needed. I would also like to thank Filipa Duarte for her help regarding academic writing. Finally, I would like to thank Jos Huisken and my professor Kjetil Svarstad for their guidance and support throughout my thesis.

After finishing my Thesis, I was encouraged to write a paper about my work. This paper was submitted to the Design, Automation, and Test in Europe (DATE) conference 2010. It can be found in the electronic appendix (filename article.pdf).

Contents

List of Figures

List of Tables

List of Abbreviations

ALU	Arithmetic Logic Unit
ASAP	As Soon As Possible
ASIP	Application Specific Instruction set Processor
CPF	Common Power Format
CPU	Central Processing Unit
DFVS	Dynamic Frequency and Voltage Scaling
DSP	Digital Signal Processor
DWT	Discrete Wavelet Transform
ECG	Electrocardiogram
EEG	Electroencephalogram
G	General purpose
GIDL	Gate Induced Drain Leakage
HDL	Hardware Description Language
HW	HardWare
I/O	Input/Output
ILP	Instruction Level Parallelism
IP	Intellectual Property
IPC	Instructions issued Per Cycle
LP	Low Power
MAC	Multiply ACcumulate
MLV	Minimum Leakage Vector
P&R	Place and Route
PD	Power Domain
RBB	Reverse Body Biasing
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SDF	Standard Delay Format
SFD	Start Frame Delimiter
SR	State Retention
SW	SoftWare
TC	Typical Case
TSMC	Taiwan Semiconductor Manufacturing Company
VCD	Value Change Dump
VHDL	Very high speed integrated circuits HDL
VLIW	Very Long Instruction Word
WC	Worst Case

Chapter 1

Introduction

Due to downscaling of technology, portable embedded systems have gained increasing popularity. Besides multimedia applications, like mobile music or gaming devices, a promising field of application is in the medical sector [1], [2]. As these systems usually are battery-powered or even use energy scavenging [3], optimisation of power consumption is crucial in order to meet the strict energy constraints.

While power consumption was historically dominated by dynamic switching power, leakage power consumption has gained more and more impact in both absolute numbers [4] as well as in power consumption per area [5] in the sub-90 nm technologies. One promising method to save leakage power is power gating [6, ch. 4], [7, ch. 10], i.e. shutting off unused blocks while they are not used.

Power gating is nowadays a commonly used technique for power management on system level [8] [9], i.e. switching off complete components of a chip like a processor or memory banks. However, it is still an active research topic on a more fine grained level like on the datapath of a processor.

1.1 Related work

In [10], an exploration of the potential of power gating applied on the level of execution units in the datapath is performed. Also, an analytical equation for the break-even point is derived. For the estimation, the authors use a state-of-the-art superscalar processor model which they calibrate against a pre-RTL processor model. In their analysis for the break-even point they assume the power consumed by the switch to be the only source of the energy overhead. They conclude that for an idle period of 10 cycles, power gating can bring benefits.

The authors of [4] also perform an analysis of the break-even point for power gating. They include, besides the power switch, also additionally required decap area in their model. They conclude that the overhead which is caused by

additional dynamic power consumed by the switch and the additional decap is too high for 130 nm technology, but they assume that for future technology it will bring benefits.

In [11], an implementation methodology for power gating and an analysis of the overhead are presented. The authors base their methodology on exploiting existing clock-gating control signals which they assume to be present in the design. Based on the clock-gating domains, they provide an algorithm to partition the system automatically into power domains. Also the control signals for power gating are derived from the clock-gating control. In their analysis of the overhead, they only consider the power switch. They applied their methodology on a 32-bit RISC embedded CPU and performed synthesis and place and route (P&R). Afterwards they performed power analysis by using Toshiba 90nm device models. They conclude that significant amounts of leakage power can be saved at a reasonable area penalty.

In [12], a more detailed analysis of power gating than in the previous papers is presented. The authors base their trade-off analysis on five factors, namely performance degradation, sleep transistor size, leakage power savings, power mode transition time and power mode transition energy. They implement a power gated design with 65 nm STMicroelectronics technology and present power numbers extracted after place and route. They conclude that they can achieve up to 75 % leakage power savings.

All of the previously presented publications have in common that they do not consider the energy consumed by the isolation cells at the boundary of a power domain. Neither, any statements are given regarding the energy consumption of additional modules, like a power manager.

1.2 Approach, contribution of this work

The hypothesis of this work is that applying power gating in the datapath of a processor can lead to energy savings. Also, it is expected that the overhead of power gating is not only determined by the energy to switch a power domain on, like concluded in previous publications on power gating, but also by additionally required modules like isolation cells.

The purpose of this thesis is therefore an exploration of the impact of power gating in the data path of a processor. Henceforth, a detailed analysis of the break-even point of power gating is derived to evaluate if power gating could bring benefits. Furthermore, a method to partition a processor into power domains, followed by the implementation and evaluation of power gating is proposed. Finally, an analysis of the main challenges of fine grained power gating in the datapath is given. The analysis was based on power figures obtained from a post-P&R netlist for 90 nm TSMC.

The procedure for implementing power gating was as follows: After a careful analysis of the power consumption of the blocks in the datapath and their duty cycle, the system is partitioned into power domains. In order to determine both the powering-on time and energy, spice simulations are performed. Further spice

simulations were performed to identify the number of required power switches. Following, power gating was integrated into the design using the Common Power Format (CPF) [13]. The resulting system was synthesised and placed and routed using the Cadence Design Tools [14]. Afterwards, the power consumption was analysed using PrimeTime from Synopsis [15]. The obtained power numbers were used for the final analysis to determine whether energy could be saved by using power gating.

1.3 Document structure

The remainder of this report is structured as follows: In chapter 2, the background for power consumption in CMOS circuits is described including a brief overview of power saving techniques. Then, in chapter 3 a detailed analysis of the benefits and costs of power gating is performed. In chapter 4, the methodology which is used in this work is explained. In chapter 5, the implementation of power gating is described. The results are presented in chapter 6, followed by a discussion in chapter 7. A short overview of possible future work is given in chapter 8. Finally, the conclusions are drawn in chapter 9.

Chapter 2

Background

In this chapter, the background theory of this work is explained. First, an introduction to the different sources of power dissipation in CMOS circuits, including a brief overview of power reduction techniques, is given in section 2.1. As power gating is primarily used in this work, the principles of power gating are explained in section 2.2. Finally in section 2.3, an overview of leakage power aware compliation techniques is given.

2.1 Power consumption in CMOS

The average power dissipation in CMOS devices (P_{avg}) can be described by the following equation [16], [17]:

$$P_{avg} = P_{short} + P_{dynamic} + P_{leak} = \frac{\beta}{12} \cdot (V_{dd} - 2V_T)^3 \cdot \frac{\tau}{T} + \alpha C_L V_{dd}^2 f + I_{leak} V_{dd} \tag{2.1}$$

P_{short} is caused by short circuit currents that occur when both the NMOS and PMOS transistor in a CMOS cell are in their conductive state for a short time during transitions. β is the gain factor of a MOS transistor, V_T is the threshold voltage, τ is the rise or fall time of a signal, T is the period time of a signal and V_{dd} is the supply voltage [18].

$P_{dynamic}$ represents the dynamic power due to switching activity in the circuit. It is determined by the node transition factor α, the load capacitance C_L, the clock frequency f and V_{dd}.

P_{leak} represents the leakage power consumption and is a product of the leakage current $I_{leakage}$ and V_{dd}.

2.1.1 Dynamic power consumption

Dynamic power is caused by charging and discharging of node capacitances in the design. To minimise dynamic power consumption, several techniques are available. In the following, an overview of the most common techniques is given.

One approach is *dynamic frequency and voltage scaling (DFVS)*. The principle is to lower the voltage and/ or frequency of a design to a level that the circuit just remains functional and reaches its timing constraints ([7], [19, ch. 4]). This is also supported by several application compilers, like presented in [20], [21] or [22].

Another technique is *clock gating*, which is applied to idle blocks of the design ([6, ch. 2], [23]. In order to reduce switching power dissipation caused by the clock, which can be up to 50 percent of the total power consumption, the clock is turned off for blocks that are not performing any useful task.

Usually, several modules share a common input source, typically a register or a memory. When the inputs of a module are changing, but the output is not needed, it is performing redundant computation during which dynamic power is consumed. In order to prevent that, the inputs of a module can be isolated using *operand isolation* [24].

Minimising the switching capacitance is another method. Hereby the switching activity or the capacitive load is reduced by several optimisation techniques, like described in for example [19, ch. 7] or [25].

Besides the mentioned methods, there are further optimisation techniques, a good summary can be found in [26] and [27]. Several methods which focus on software optimisation techniques for low power are presented in [28].

2.1.2 Leakage power consumption

According to [16], there are five major sources of leakage current, namely subthreshold current, gate leakage, pn-junction leakage, Gate Induced Drain Leakage (GIDL) and Punchthrough. Of those, leakage due to subthreshold current, which is flowing between the source and the drain when the transistor is in the subthreshold region (i.e. when the gate voltage is below the threshold voltage), has the highest impact. This current is also strongly depending on the temperature the circuit is operating at, it increases with higher temperatures [29]. A detailed analysis of leakage power can also be found in [30], [31] and [32].

While the total power consumption was historically dominated by dynamic power, leakage power gains more and more impact in sub-90nm designs [5]. Therefore, optimisation methods that focus on leakage power minimisation become of increasing interest in addition to methods for minimisation of dynamic power dissipation.

One method to reduce leakage power is *reverse body biasing (RBB)*, as explained for example in [7], [30] and [33]. The principle is to apply a deep reverse body bias during standby to increase the threshold voltage and thus force the transistor into the off region to reduce subthreshold leakage.

Leakage power consumption of a gate is depending on its input vector [34]. Another method to minimise the leakage power when a module is idle, is called *minimum leakage vector (MLV)*. The principle is to shift the input vector which causes the least leakage current into the module whenever it is idle. A method to determine the MLV for a given combinational logic, including a short review of related methods, is presented in [35].

A popular technique for leakage power reduction is *power gating*, also called *power shut-off*. In the remainder of this report, the term *power gating* will be used. The basic principle of this method is to disconnect idle blocks from the supply voltage, as described in more detailed in the next section.

2.2 Power gating

Power gating relies on switching off idle blocks in the design, thus saving leakage power. A very good overview can be found in [6] and [7].

There are two basic design approaches for power gating ([7, ch. 10.3.4.1]). One is the coarse-grained approach, in which complete blocks are switched off. An illustrating example for a system with coarse grained power gating could be a wireless sensor node ([36]) which consists of a microprocessor, a sensor and a radio. Power gating could be applied to the radio whenever no communication is needed. Or, the sensor is powered down as long as data is transmitted.

The other approach is fine-grained power gating, where individual modules within a block can be switched off while keeping other modules switched on at the same time. Fine-grained power gating could be applied to a processor with several dedicated modules, for example one multiplier and one multiply-accumulate (MAC) unit. Whenever the multiplier is used, the MAC-unit could be powered down or vice versa.

A general overview of a power gated system is illustrated in Figure 2.1. A part of the system which is in one switchable power domain is represented by *PD_switchable*. Its output signals are connected to another power domain (*PD_always_on*), which is an always-on domain in this example. It can be seen, that the outputs are isolated to prevent unknown signals propagating through the design when *PD_switchable* is switched off. Also, a power switch is inserted between the voltage supply VDD and the power domain. Furthermore, a power manager is integrated into the system to control the power gating related cells.

In the coming sections, the individual parts of the scheme and the general considerations for power gating will be described in more detail.

2.2.1 Power domain

A part of a design which is connected to the same power supply is called a *power domain*. When no power gating is used, the complete design is in the same, always-on, power domain. When power power gating is used, usually several power domain are present in the design. In Figure 2.1, all modules that are contained in the power domain *PD_always_on* are connected to the

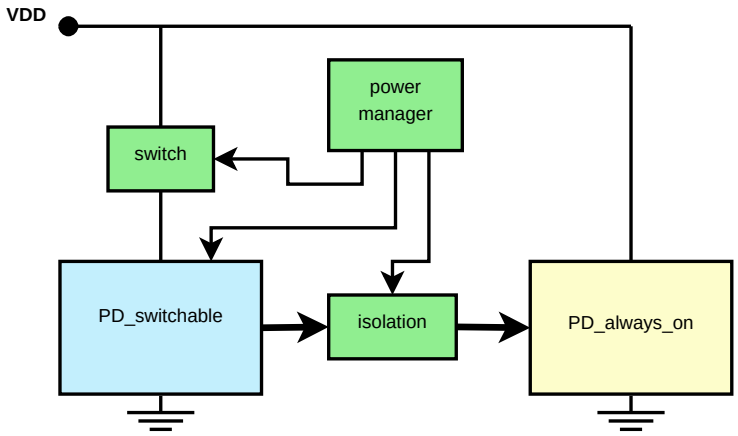


Figure 2.1: General power gating scheme

supply voltage VDD which is always on. All modules which belong to the power domain *PD_ switchable* are connected to VDD through a switch, hence they are connected to an own power supply.

2.2.2 Power switch

To cut off the power supply of a power domain, a sleep-transistor which is controlled like a switch is used. There are two possibilities to cut off the power supply of a power domain. Either, a header or a footer switch is used as depicted in Figure 2.2 and Figure 2.3, respectively. The header switch is placed between the supply voltage and the power domain, thus introducing a virtual voltage supply. The footer switch is placed between the power domain and ground, thus introducing a virtual ground. In Figure 2.1, a header switch is used. An overview of the advantages and disadvantages of both approaches is given in [6, ch. 5.1].

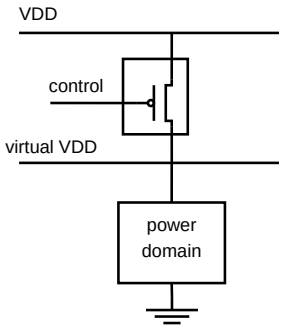


Figure 2.2: Header switch

The insertion of power switches influences the system's behaviour in multiple ways, namely rush-in current, area overhead and voltage drop.

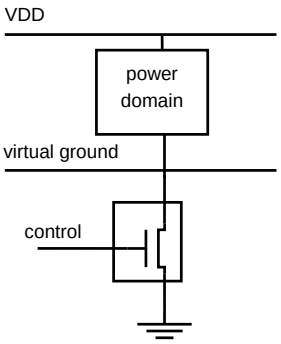


Figure 2.3: Footer switch

When a module is powered on after a sleep period, a sudden *rush-in current* is introduced. This current affects the power-on energy significantly and causes noise in the circuit. It can be controlled by connecting the power switches in a daisy-chain configuration, so that they are turned on one after another and so limit the current flow ([37]).

Power switches also introduce *area overhead* to the design which translates into a higher total power consumption. Several approaches to reduce this area overhead have been proposed, for example to connect the power switches in a cluster ([38]) or in a distributed network ([39]).

Another important factor is the *voltage drop* over the switch during active mode, if it is too high, the required frequency cannot be reached which leads to timing problems. The voltage drop is related to the number of power switches and their size [40].

2.2.3 Isolation

Interfaces between power gated domains and other domains, in Figure 2.1 between *PD_switchable* and *PD_always_on* have to be taken special care of. When a power domain is shut off, its output signals are floating, i.e. it is not known which value they have. In order to prevent unknown values propagating through the design, the output signals of the power domains have to be isolated. Therefore, the outgoing signals have to be forced to either *0* or *1* by using special isolation cells.

When output signals of switched-off power domains are forced to zero, an isolation cell like in Figure 2.4 is used. As long as the power domain is switched on, *iso* is low. As soon as it is switched off, *iso* becomes high to enforce the output of the isolation cell to be low. The truth table can be found in Table 2.1.

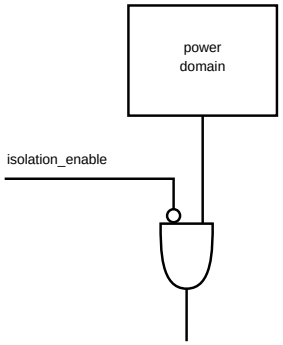


Figure 2.4: Principle of an and-isolation cell

Data	**iso**	**Out**
1	0	1
0	0	0
x	1	0

Table 2.1: Truth table of an and-isolation cell

When the output signals are forced to one, an isolation cell like in Figure 2.5 is used. The truth table is shown in Table 2.2.

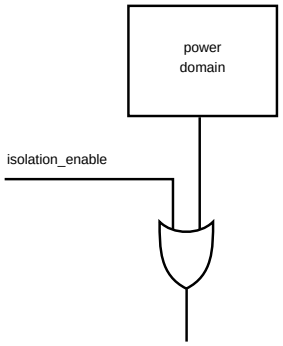


Figure 2.5: Principle of an or-isolation cell

Data	**iso**	**Out**
0	0	0
1	0	1
x	1	1

Table 2.2: Truth table of an or-isolation cell

Another consideration is whether the isolation cells should be placed at the output of the power domain, in Figure 2.1 *PD_switchable*, or at the input of the next module, which is using the output signals of the power gated module as inputs, in Figure 2.1 *PD_always_on*.

In terms of functionality, there is no difference, but as one power domain usually provides input values to multiple modules, there is more area overhead when the signals are isolated at their destination. Furthermore, analysis and verification is simplified when signals are isolated at their output. In Figure 2.1, the isolation cells are placed at the output of the power domain.

Introducing isolation cells can also affect the timing of the design and therefore the maximum achievable frequency. It can occur that due to isolation cells the timing of a path cannot be reached anymore. Hence this path becomes a critical path. This can lead to additionally required buffers or faster cells in the path, increasing power consumption. Moreover, synthesis and layout tools are more restricted in performing design optimisation.

2.2.4 State retention and restoration

When registers are shut off, they lose their internal state. When the stored value still has to be present after a shutdown period, the register's state can be retained or restored.

State retention (SR) can be implemented by three methods:

1. Software based, where the value of the register which is to be switched off is written to an external memory or another register which remains switched on.
2. Exploiting already existing scan chains by shifting the scan registers as scan-in-testing while the outputs are routed to an external memory during the power-down sequence.
3. Usage of special state retention flip flops which are capable of saving the internal state during. power-down

A detailed comparison between the methods can be found in [6, ch. 5.3]. In this work, the focus is explicitly on SR registers.

A state retention register is a special register that contains, in addition to the normal functionality, a shadow register which can retain the state during shut off. A schematic can be seen in Figure 2.6. This type of SR registers has two latches, a main latch for the active mode which is connected to the virtual voltage supply, VV_{dd}, and a SR latch which dissipates less leakage power than the main latch which is connected to an always-on voltage supply (V_{dd}). Before a SR register is shut off, the *save* signal has to be set in order to transfer the stored value from the main latch into the SR latch. At wakeup, the *restore* signal has to be set so that the data is written back.

An advantage of state retention cell compared to the other state retention methods is that the design and the design flow itself does not have to be changed, apart from assigning the *save* and *restore* signals properly. A drawback is, that the retention cells have area overhead, typically 20% or more ([6]) and require additional control signals. To avoid the use of control signals, a sense-amplifier-based state retention flip flop is proposed in [41]. The presented design has a retention time in the range of milliseconds and a small area overhead. The

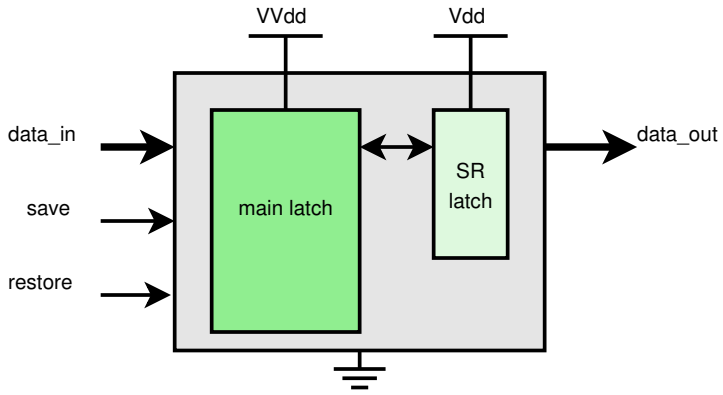


Figure 2.6: State retention register

rather long retention time, however, makes these SR registers less suitable for fast power-gating.

2.2.5 Power manager

The power manager is in charge of providing the control signals to the power switches, isolation cells, and, if present, to the SR registers. It can be implemented as a dedicated hardware module or in software. In Figure 2.1, the power manager is implemented in hardware. The control sequence for a system without SR can be seen in Figure 2.7. When registers are included in the power domain, a reset signal has to be provided before de-isolation to set the registers in the power domain in a known state. The control sequence for a system with SR can be seen in Figure 2.8.

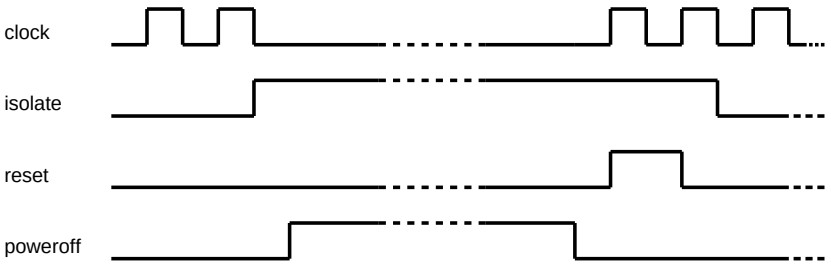


Figure 2.7: Control sequence for a design without state retention

2.3 Power aware compilation

The compiler has a large influence on the power efficiency of an application. Considering power gating, it is relevant in which cycle a resource is used and how long its idle periods are. Usually, the compiler has a lot of freedom how to

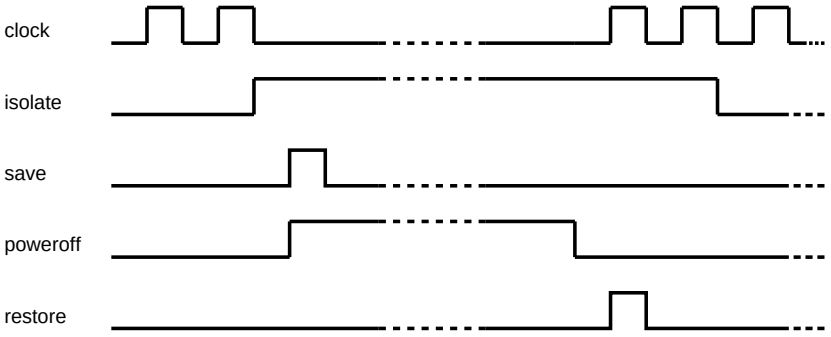


Figure 2.8: Control sequence for a design with state retention

schedule as it is only constrained by data flow and control demands. That can lead to inefficient resource usage in terms of idleness.

In the following section the different principles of leakage power aware compilation are explained. Basically, there are two principles:

1. Post-procesing: The program code is scheduled first before a post-processing algorithm is executed to analyse the generated assembly code for possibilities to shut off certain blocks. Then, power gating instructions are inserted into the assembly code.
2. Smart scheduling: Already during the initial scheduling phase, the idle periods of functional unit are increased without affecting the performance. This is achieved by smart mapping of the instructions to the hardware.

2.3.1 Method one: Post-processing

In [42], a patent is filed for automatic scheduling of power gating instructions. The principle is to translate the high-level code into assembly, followed by an analysis of the dataflow and utilisation of hardware-blocks. Based on the results, power-gating instructions are inserted into the code. In a next step, power gating instructions are merged together if possible in order to minimse instruction execution power. To determine possible savings, the leakage energy of the block is traded off against the energy required to fetch and decode a power-gating instruction. Other overhead is not considered in this approach.

In [43], several compiler techniques for power aware scheduling are described. It covers both dynamic and static power minimisation. Power gating awareness is reached by inserting power-off and power-on instructions into the assembly code.

The authors of [44] first compare hardware-based methods to analyse the resource utilisation with compiler based methods. They draw the conclusion that the compiler based methods are suited better as a compiler has an overview of the total dataflow, the hardware based methods require a certain history to base assumptions on. Then they describe their compiler approach, which starts with an initial scheduling of the code, followed by a partitioning of the

program into hot and cold blocks. Afterwards, an analysis of the utilisation of these blocks is performed. Based on a power model which considers expected idle time, power-on and power-off energy, the compiler inserts power-off and power-on instructions into the code.

In [45], a similar approach like in [44] and [42] is described. The program is scheduled first, then a data flow diagram is set up. Based on that, the compiler inserts power gating instructions into the code. Their power model takes the power-off and power-on energy into account.

All presented methods have the drawback that they do not consider the additional energy overhead which is introduced into the system due to power-gating related cells like power switches or isolation cells. Also they give the compiler during the initial scheduling all freedom. No consideration is made of possibilities to select different function units in instruction level parallelism (ILP) or delay instructions in order to leave a module switched off longer without influencing the functionality.

2.3.2 Method two: Smart scheduling

In [46], a scheduling method which increases the idle time of function units is described. The idea behind the method is that if a resource has a longer continuous idle period, it can be shut off. The approach is to schedule in a way that functional units are used in blocks. However, this method does not consider ILP.

In [47], a scheduling approach for clustered VLIW architectures is described. The principle is as follows: Whenever a functional unit is idle for one clock cycle, it is marked as sleeping. During scheduling, when there is more than one functional unit of the desired type, the one which has slept for the longest time is selected by the compiler to compensate for the wakeup-energy. The energy model that is used in this work to determine the break-even point trades off the leakage energy of the functional unit with the energy required to power it on.

Another method is developed in [48], in which a loop scheduling is described where resources are rescheduled to better locations to increase idle periods in other resources.

In [49], the authors present another leakage energy aware scheduling method, which has the focus on the utilisation of arithmetic logic units (ALUs) in loops. First they identify all the loops in the program, then they analyse the critical, most executed block in each loop. Within this block, they analyse how many ALUs are necessary, they call it instructions issued per cycle (IPC). After analysis, the program is rescheduled with the previously determined subset of ALUs. This could cause a overhead due to increased execution time when there were more ALUs necessary in the non-critical blocks of the loop then in the critical block. The scheduling algorithm therefore has to take the energy overhead caused by that into account and make sure that the savings due to shut-off ALUs are larger than the energy overhead. After rescheduling is complete, power-off and power-on instructions are inserted before and after the loops to shut off the unused ALUs. The authors base their trade-off on the leakage energy that

could be saved versus the energy overhead due to switching on a function unit and the additional energy required for extra required execution cycles. Energy overhead caused by other power gating related cells that have to be added to the design, like isolation cells, they do not consider.

Chapter 3

Analysis of the break-even point

An important task within this thesis was the evaluation whether applying power gating to a system can bring benefits. The analysis is presented in this chapter.

The chapter is organised as follows: First, the power dissipation characteristics of a system that uses power gating is described in section 3.1. In section 3.2, the energy consumption in the different states of a processor is analysed. In section 3.3, the savings are explained, followed by an analysis of the energy overhead in section 3.4. Afterwards, the break-even point for power gating based on the findings is derived in section 3.5. Finally, two example power profiles of power gated systems are illustrated in section 3.7.

3.1 Power profile of a power gated system

A digital system like a processor is usually not constantly active. It has periods where computations are performed followed by idle periods. A schematic of an example power profile of a system is illustrated in Figure 3.1. During *active*, the system is performing useful tasks. The power consumption is illustrated as an average over the complete active period. Afterwards, the system is not performing any useful tasks, i.e. it is idle. The power consumption during that period is determined by the leakage power consumption of the system, assumed that the clock is stopped, i.e. that clock gating is applied.

In Figure 3.2, the power consumption characteristics of a system that uses power gating is depicted. At t_{idle}, the system has finished the active state and is switched off. The switching off process is finished at t_{off}. Then, the system remains switched off until t_{sleep}. At that moment, it is switched on again. At t_{on}, it is again fully functional. The individual components of the power consumption will be explained in the remainder of this chapter.

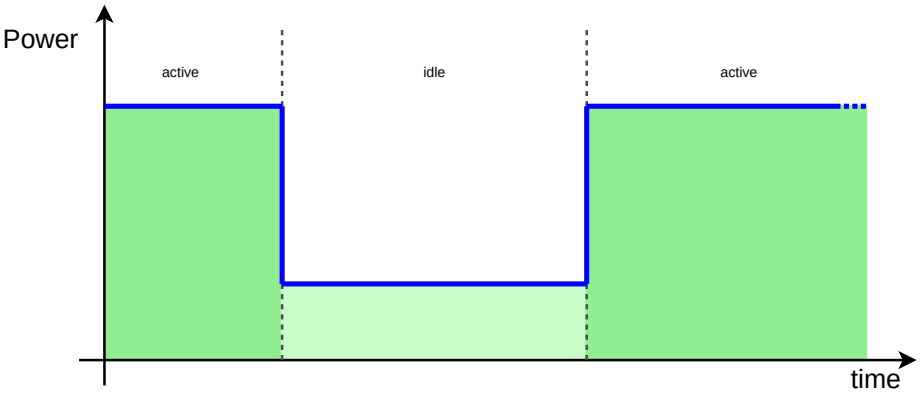


Figure 3.1: Power dissipation of a non power-gated system

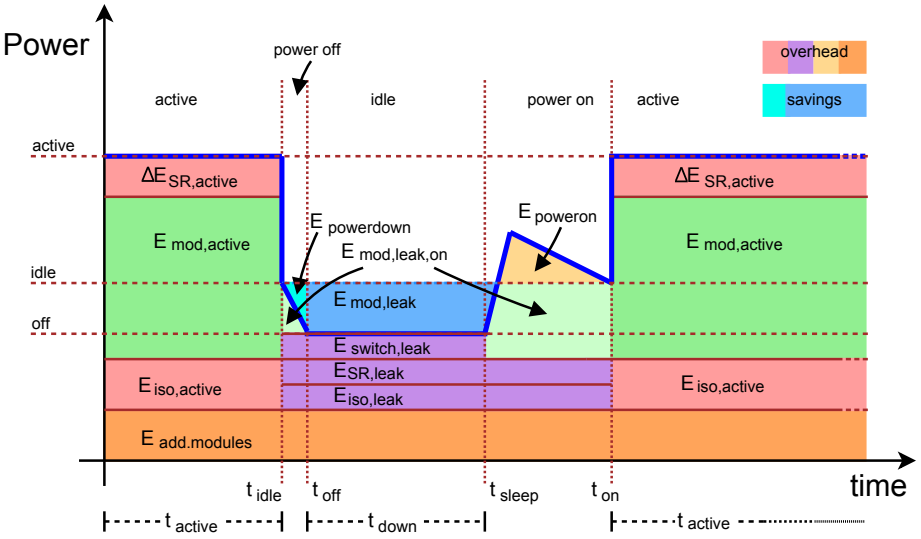


Figure 3.2: Power dissipation of a power gated system

3.2 Energy consumption in the different states

The total energy consumption of a power gated system is composed of the energy which is consumed by the power domain and the energy which is consumed by the additional power gating related modules. The modules which are constantly active, like a power manager, are consuming energy all the time ($E_{add.modules}$). The energy consumption of the remaining components depends on the state of the processors.

During t_{active}, the system is in its active state. The energy in this state is consumed by the power domain ($E_{mod,active}$), the isolation cells ($E_{iso,active}$) and the

SR registers which consume more energy than regular registers ($\Delta E_{SR,active}$).

At t_{idle}, the system has finished the active state and goes immediately to the idle state by switching off the clock. At the same time it is powered off. The powering off process takes place until t_{off}. During that time, the power domain still leaks but the leakage energy of the power domain ($E_{mod,leak,on}$) is converging to the off-level.

Then, the system remains switched off during t_{down}. The energy which is consumed depends on the leakage of the power switch(es) ($E_{switch,leak}$), the leakage of the isolation cells ($E_{iso,leak}$) and the leakage of the SR registers ($E_{SR,leak}$).

At t_{sleep}, the power domain is switched on again. The switching-on process takes until t_{on}. The energy which is consumed during that period is the leakage energy of the power domain ($E_{mod,leak,on}$) and the additional energy required to switch the power domain on ($E_{poweron}$). Afterwards, the system is fully functional.

3.3 Energy savings

The energy savings are defined as the amount of energy that is saved when power gating is applied to a system. In Figure 3.2, it is represented by the blue areas.

The energy savings of a power gated module are determined by two factors:

1. $E_{mod,leak}$, the energy which the module would consume during the idle period if it was not switched off. It is determined by the leakage power of the power domain ($P_{mod,leak}$) multiplied by time the module is switched off (t_{down}).
2. $E_{powerdown}$, which is the difference between the leakage energy the power domain would normally consume during powering off (between t_{idle} and t_{off}) and the energy which is still consumed ($E_{mod,leak,on}$). That implies that from the moment the power-down signal is given, energy is saved.

Both contributors can be combined to the total energy savings $E_{savings}$:

$$\begin{aligned} E_{savings} &= E_{mod,leak} + E_{powerdown} \\ &= P_{mod,leak} \cdot t_{down} + E_{powerdown} \end{aligned} \tag{3.1}$$

For future analysis, the equation is rewritten to

$$E_{savings} = \alpha \cdot t_{down} + \varphi \tag{3.2}$$

with $\alpha = P_{mod,leak}$ and $\varphi = E_{powerdown}$.

3.4 Energy overhead

The energy overhead is defined as the additional energy consumption of a system due to the required components for power gating. In Figure 3.2, the overhead is represented by the pink, purple, and orange areas. The energy overhead can be divided into four subgroups, namely

1. $E_{overhead,down}$, the additional energy which is consumed while the power domain is switched off. It is represented by the purple areas.
2. $E_{overhead,active}$, the additional energy which is consumed while the power domain is active. It is represented by the pink areas.
3. $E_{overhead,always}$, the additional energy which is consumed during the total run-time. It is represented by the dark orange area.
4. $E_{poweron}$, which is required to power a module back on after it has been switched off. It is represented by the light orange area.

$E_{overhead,down}$ is determined by the leakage power of the power switch(es) ($P_{switch,leak}$), the leakage power of the isolation cells ($P_{iso,leak}$), the leakage power of the SR registers ($P_{SR,leak}$), and the time during which the power domain is switched off (t_{down}). The equation for $E_{overhead,down}$ is as follows:

$$\begin{aligned} E_{overhead,down} &= E_{switch,leak} + E_{iso,leak} + E_{SR,leak} \\ &= t_{down} \cdot (P_{switch,leak} + P_{iso,leak} + P_{SR,leak}) \end{aligned} \tag{3.3}$$

$E_{overhead,active}$ depends on the energy consumed by the isolation cells during active mode ($P_{iso,active}$), the additional power consumed by the SR registers compared to what regular registers would consume during active mode ($\Delta P_{SR,active}$), and the time during which the power domain is active (t_{active}). The equation for $E_{overhead,active}$ is as follows:

$$\begin{aligned} E_{overhead,active} &= E_{iso,active} + \Delta E_{SR,active} \\ &= t_{active} \cdot (P_{iso,active} + \Delta P_{SR,active}) \end{aligned} \tag{3.4}$$

$E_{overhead,always}$ is caused by always-on components that have to be added to the design in order to enable power gating, as for example control registers or a power manager. Their power consumption ($P_{add.modules}$) and the total run time, (t_{total}) determine the total energy consumption. Summarising, $E_{overhead,always}$ is defined as follows:

$$E_{overhead,always} = t_{total} \cdot P_{add.modules} \tag{3.5}$$

$E_{poweron}$ is determined by the number and size of the power switches and the size of the power domain.

To evaluate the impact of the power gating related cells and components introduced in section 2.2, a short summary is given below which of the energy overhead components they influence.

- Power switches. They leak when the power domain is switched off, they therefore have an impact on $E_{overhead,down}$.
- Isolation cells. One factor is the energy they consume while the power domain is active. Whenever the output of the power domain switches, the isolation cells have to switch as well. This influences $E_{overhead,active}$. The other factor is the leakage energy consumed by the isolation cells while the power domain is switched off. That has an impact on $E_{overhead,down}$.
- SR registers. During active mode of the power domain, they introduce a certain energy overhead, namely the difference in energy consumption compared to regular registers. This influences $E_{overhead,active}$. The leakage energy during of the SR registers the period the power domain is switched off impacts $E_{overhead,down}$.
- Additional modules, like a power manager or function units. As they are switched during the complete run-time, they influence $E_{overhead,always}$.

Summarising, the energy overhead can be written as follows:

$$\begin{aligned} E_{overhead} &= E_{overhead,down} + E_{overhead,active} + E_{overhead,always} + E_{poweron} \\ &= t_{down} \cdot (P_{switch,leak} + P_{iso,leak} + P_{SR,leak}) \\ &+ t_{active} \cdot (P_{iso,active} + \Delta P_{SR,active}) \\ &+ t_{total} \cdot P_{add.modules} \\ &+ E_{poweron} \end{aligned} \tag{3.6}$$

For simplicity's sake for the final analysis, the individual factors are merged, which leads to the following equation:

$$E_{overhead} = t_{down} \cdot \beta + t_{active} \cdot \gamma + t_{total} \cdot \delta + \epsilon \tag{3.7}$$

with $\beta = P_{switch,leak} + P_{iso,leak} + P_{SR,leak}$, $\gamma = P_{iso,active} + \Delta P_{SR,active}$, $\delta = P_{add.modules}$ and $\epsilon = E_{poweron}$.

3.5 Derivation of the break-even point

Building on the above analysis of the savings and the overhead, an analytical equation can be derived for the minimum percentage of time that the module has to be switched off in order to gain energy savings, i.e. the break-even point. Energy is saved when the energy savings exceed the energy overhead:

$$E_{savings} > E_{overhead} \tag{3.8}$$

By using the above definitions for the savings and the overhead, and expressing t_{active} with $t_{total} - t_{down}$, a condition for the minimum down time $t_{down,min}$ can be found:

$$\begin{aligned}
\alpha \cdot t_{down,min} + \varphi &> \beta \cdot t_{down,min} + \gamma \cdot t_{active} + \delta \cdot t_{total} + \epsilon \\
t_{down,min}\,(\alpha - \beta) - \gamma\,(t_{total} - t_{down,min}) &> \delta \cdot t_{total} + \epsilon - \varphi \\
t_{down,min}\,(\alpha - \beta + \gamma) &> t_{total}\,(\gamma + \delta) + \epsilon - \varphi \\
t_{down,min} &> \frac{\epsilon - \varphi + t_{total}\,(\gamma + \delta)}{\alpha - \beta + \gamma} \qquad (3.9)
\end{aligned}$$

Some of the factors of the above analysis can be omitted because they are neglegible, namely the leakage power of the switch ($P_{switch,leak}$), the leakage power of the isolation cells ($P_{iso,leak}$), the energy to switch a power domain on after it has been switched off ($E_{poweron}$) and the energy which is saved during powering down before reaching the lowest energy state ($E_{powerdown}$). This will be shown in the results section (chapter 6). Then, the condition for $t_{down,min}$ is not longer depending on the total run time but can be expressed percentage-wise with reference to t_{total}:

$$\frac{t_{down,min}}{t_{total}} > \frac{P_{iso,active} + \Delta P_{SR,active} + P_{add.modules}}{P_{iso,active} + \Delta P_{SR,active} + P_{leak,mod} + P_{SR,leak}} \qquad (3.10)$$

Using the previously definitions for α, γ and δ and defining $\beta' = P_{SR,leak}$, the equation can be written as follows:

$$\frac{t_{down,min}}{t_{total}} > \frac{\gamma + \delta}{\gamma + \alpha + \beta'} \qquad (3.11)$$

where $\alpha = P_{leak,mod}$, $\beta' = P_{SR,leak}$, $\gamma = P_{iso,active} + \Delta P_{SR,active}$ and $\delta = P_{add.modules}$

3.6 Absolute energy savings

For the calculation of the absolute energy savings, the break-even point is a first indication. After the break-even point for a given system has been calculated, it can be determined whether the system can be switched off for a sufficient long time in order to gain energy savings. When this is the case, it also has to be calculated how big the total energy savings are. This is done the following way:

The absolute savings are the difference between the possible energy savings and the introduced overhead:

$$E_{savings,tot} = E_{savings} - E_{overhead} \qquad (3.12)$$

Using the definitions for α, β, γ and δ, the equation can be rewritten:

$$
\begin{aligned}
E_{savings,tot} &= E_{savings} - E_{overhead} \\
&= \alpha \cdot t_{down} + \varphi - (t_{down} \cdot \beta + t_{active} \cdot \gamma + t_{total} \cdot \delta + \epsilon) \\
&= t_{down} \cdot (\alpha - \beta + \gamma) - t_{total} \cdot (\gamma + \delta) + \varphi - \epsilon
\end{aligned} \tag{3.13}
$$

An illustration can be seen in Figure 3.3. The slope of the line is determined by the term $\alpha - \beta + \gamma$, the offset is determined by $t_{total} \cdot (\gamma + \delta) + \varphi - \epsilon$. The bigger α, i.e. the leakage power of the power domain, the bigger is the steepness of the line, i.e. the energy savings.

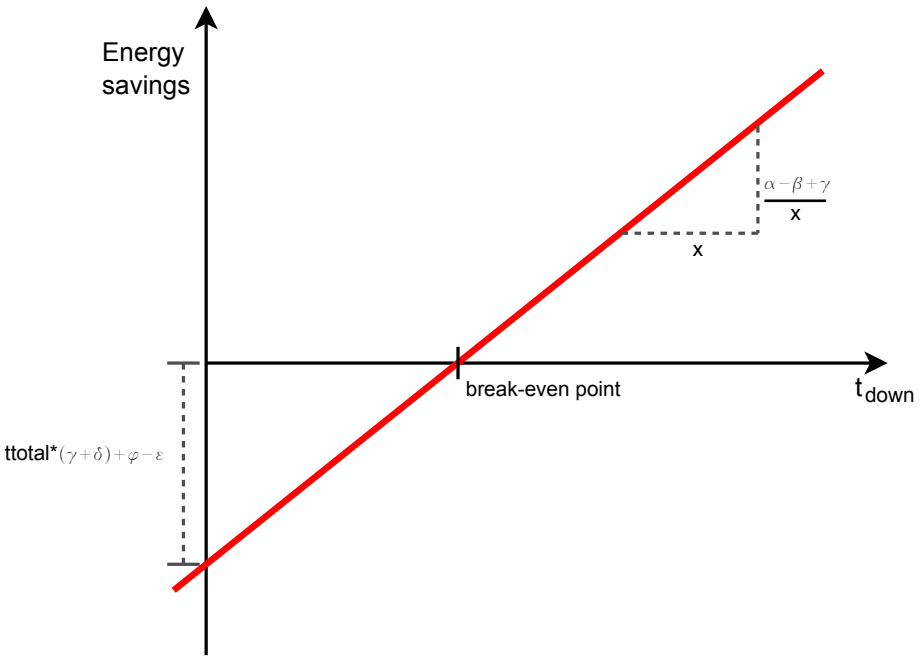


Figure 3.3: Illustration of the absolute energy savings

3.7 Example for a power gating scheme

To illustrate the power characteristics of power domains, two examples are illustrated. The first example depicts the power consumption scheme of a power domain where power gating leads to energy savings, the second example shows a power domain where power gating leads to extra energy consumption. Both examples are for illustration purposes and not based on numbers obtained from experiments. The color scheme corresponds to the color scheme used in Figure 3.2.

In Figure 3.4 the power characteristics of a power domain which can be switched off for a significant amount of time is displayed. It can be seen that the leakage

energy (blue) of the power domain is relatively high and the introduced overhead (orange, pink, purple) rather low. Therefore, power gating is justified in this system.

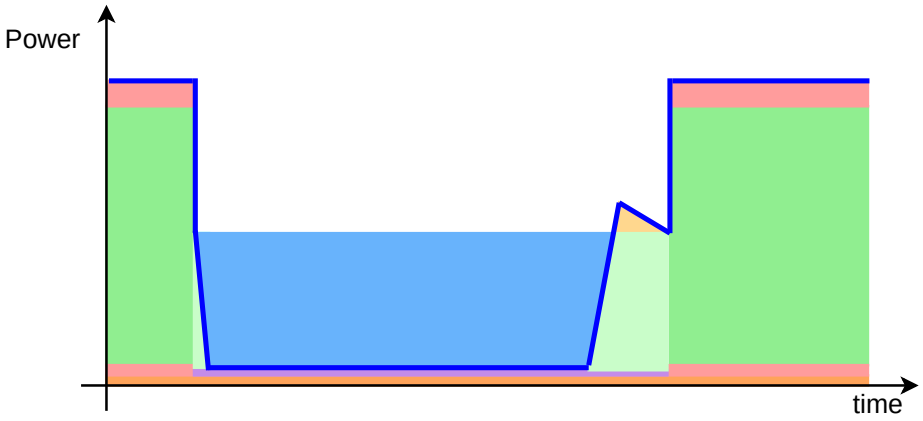


Figure 3.4: Example of a power gated module where power gating makes sense

As a counterexample, in Figure 3.5 the power power distribution of another power domain is shown. There it can be seen, that the introduced energy overhead, especially caused by additional modules and the isolation cells, is larger than the savings, which implies that power gating is not efficient in this case.

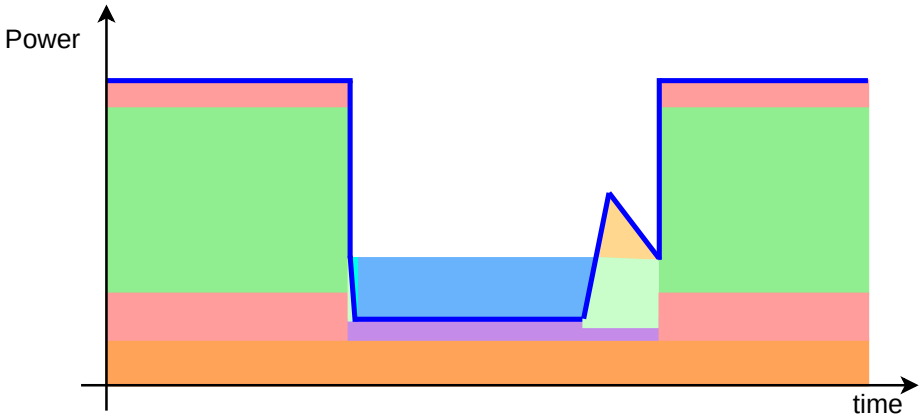


Figure 3.5: Example of a power gated module where power gating does not makes sense

Chapter 4

Methodology

In this chapter, the methodology that has been developed and used during this work is explained. It is a design flow that can be applied to a system, in this work a processor, to implement fine-grained power gating. In Figure 4.1, an overview is depicted. In the flow, the system is partitioned into power domains, then power gating is implemented into the design. Finally, a verification is performed to prove that the system is still functionally correct, furthermore, the energy savings are evaluated.

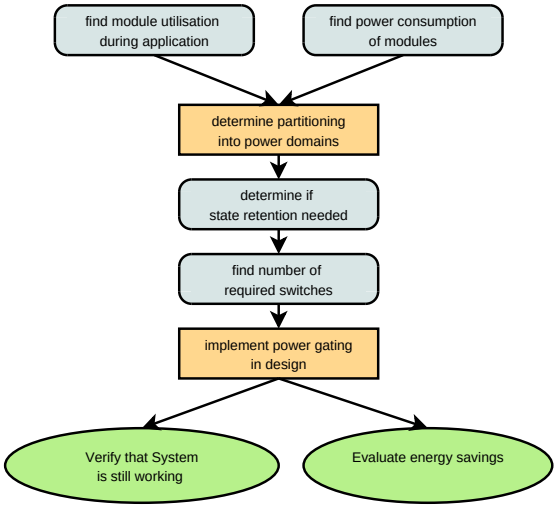


Figure 4.1: General power gating design flow

The chapter is organised as follows: In section 4.1, the tools that were used during this work are introduced. In section 4.2, the methodology to obtain power consumption values is described. In section 4.3, it is explained how the module utilisation is determined. Based on those section, the partitioning into power domains is described in section 4.4. Afterwards, the method to determine if state retention (SR) is required in section 4.5 is presented. A description how the number of required power switches can be identified is presented in

section 4.6. In section 4.7, the workflow for implementing power gating into a design on RTL level is described. In section 4.8 it is explained how it can be verified that the system still is functionally correct. Finally, the evaluation of energy savings is explained in section 4.9.

4.1 Tools used in this work

In this section, a brief introduction of the tools that were used in this work is given.

- Target IP Designer [50], a tool for the design of application specific instruction set processors (ASIP). It uses a high-level description language (nML, [51, ch. 4]) to specify the functionality and architecture of the desired processor. From this description, synthesisable HDL code and a compiler are generated. Also, a cycle-accurate and instruction-accurate simulator are provided that can be used for verification and debug purposes. A typical design flow for the design of a processor with Target IP Designer is presented in [52].
- Silicon Hive's Hive Logic [53], also a tool for the design of ASIPs. Like in Target, the architecture of the desired processor is described in a Silicon Hive proprietary description language called TIM (The Incredible Machine). From the description, the HDL code, compiler and simulator are generated. A description of the design flow of Hive Logic can be found in [54].
- The Cadence [14] design tools, including NCSim (for rtl- and netlist simulations), RTL Compiler (for synthesis) and Encounter (for place and route).
- PrimeTime [55] from Synopsis [15], used for the extraction of power numbers after layout.
- HSPICE [56] from Synopsis, used for circuit-level simulations required to obtain accurate numbers for the analysis.
- Matlab from MathWorks [57], used for plots.
- tcl, a scripting language used for automation of design processes or simulations.

4.2 Determination of power consumption values

For extraction of power numbers, the flow which is depicted in Figure 4.2 was used. Initially, a netlist after place and route and a SDF (standard delay format) [58] file which defines the delays of each gate in the design is required, i.e. the complete design flow from RTL to layout has to be executed (refer section 4.7). A simulation has to be executed with a realistic application to obtain the switching activity. During simulation, a VCD (value change dump) file is generated containing the switching activity of all signals and nodes. This file is

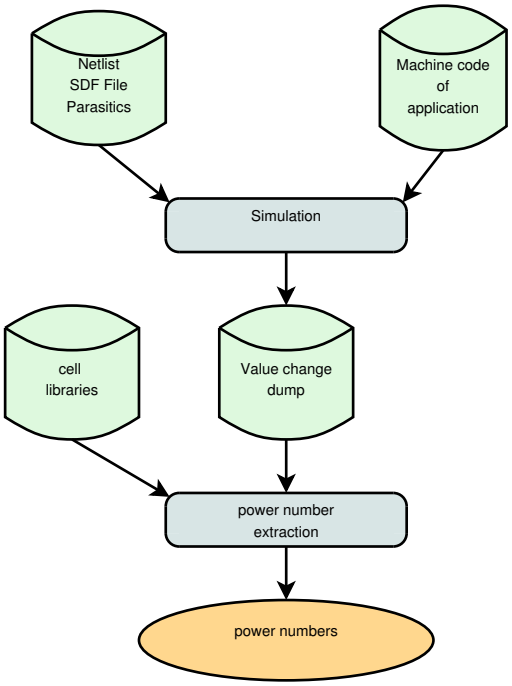


Figure 4.2: Flow for the extraction of the power numbers

then used to extract power numbers with primetime. The power consumption is broken down into *static power* which consists of mainly subthreshold leakage, *internal power* which is consumed within the boundaries of a cell including power due to charging and discharging of internal capacitances and short circuit currents, and *switching power* which is caused by charging and discharging of load capacitances at the output of a cell.

When the power consumption of all modules is known, the main consumers of leakage power have to be identified, as the focus is on leakage power consumption is this work. For this purpose, a tcl-script was written which produced a report with the required information at a glance. In the script, the modules of interest are defined and then the power concumption report generated by PrimeTime is scanned for these modules. Then, the power numbers are written out and the leakage power consumption related to the total leakage power consumption is calculated. The script can be found in the electronic appendix (filename get_power_numbers.tcl), a code snippet for the calculation of the relative leakage power consumption of a module can be found in Listing 4.6.

```
# get total leakage power as reference
if {[string match ${top_module_name} [lindex $words 0]] == 1} {
  set total_leakage [lindex $words 3]
}

if {[string match ${module_name} [lindex $words 0]] == 1} {
  set rel_leakage [expr [lindex $words 4]/$total_leakage*100]
  puts $rel_leakage
}
```

Code Snippet 4.1: Extract relative leakage power consumption of report file

4.3 Resource utilisation over time

The complete flow to determine the resource utilisation over the complete run-time of the application is illustrated in Figure 4.3.

When the power consumption of the resources is known, it has to be determined if they have sufficient idle periods where they can eventually be switched off. For this purpose, a script was written during this thesis that could provide this information. For this script, the following information is required: First it has to be determined which resources are used for which instruction (where one instruction represents all operations used during one cycle). The second information is the sequence in which the instructions are executed during the complete run-time of the application.

For this purpose, Target IP Designer provides useful features. One is a mechanism that gives information about resource usage for each instruction. The desired resources that are to be monitored have to be specified and the result will be information about which instruction requires the desired resource. The result will be saved in a *resource usage file*. Another useful feature is that a log file of the register content is generated for the complete run-time with the corresponding program counter (PC) value. The result is stored in a *register log file*.

These two files can be exploited to find the utilisation of specific resources over time. The register log file is parsed for the value of the program counter which is also listed in the resource usage file. If a specific resource is used for the specific program counter, a '1' is saved in a *result file*, otherwise a '0'. A code snippet for the explained procedure is shown in Code Snippet 4.4. The complete scripts can be found in the electronic appendix (folder resource_usage).

```
1 set curr_PC [lindex [split [lindex $words 1] ()] 1]
2 puts $results_file "PC: ${curr_PC}, cycle: [lindex $words 0] => [lindex
      $units ${curr_PC}]"
3 if {[string length [lindex $units ${curr_PC}]] > 0} {
4  puts -nonewline $units_results_file "1 "
5 } else {
6  puts -nonewline $units_results_file "0 "
7 }
```

Code Snippet 4.2: Link resource usage file with register log file

This result file can be used in Matlab to plot the usage over time for an easy overview, the matlab script can be found in the electronic appendix (filename plot_usage.m).

4.4 Partitioning into Power Domains

In this section, the steps that are required to divide a system into different power domains are explained. The modules with a significant leakage power consumption should be considered as possible power domains. Also, it is important how the modules are utilised during the application. For an initial partitioning of the system, modules with similar utilisation profiles and a relatively high leakage power consumption should be grouped into power domains. However,

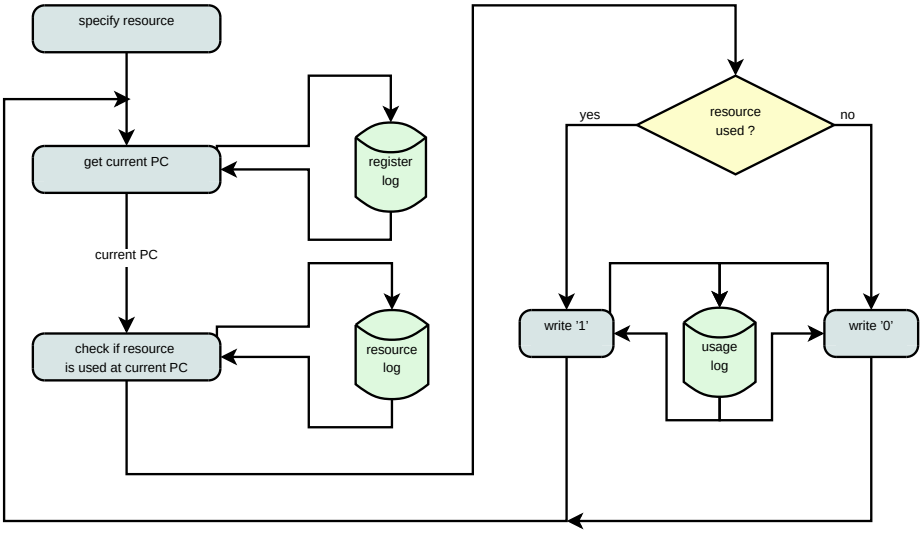


Figure 4.3: Flow for the determination of the resource usage over time

the final partitioning requires good knowledge of the system and the executed application.

4.5 Determination if state retention is required

When the power domains are determined, it has to be decided whether state retention (SR) is required. When a power domain contains registers, it has to be analysed if their internal state has to be retained during shutdown. For example, a power domain can contain registers that are used to store temporary variables of the application. When this power domain is switched off during the the execution of the application, it is most likely that the state has to be retained. However, when a power domain contains registers that are of a special type, for example vector registers, it can be different. A possible use-case is that an application consists of different function calls. Only one of the functions requires the presence of vector registers, otherwise they are not needed. Then, the state does not have to be retained.

In the end, extensive knowledge of the system and the application is required to determine if state retention is required or not. As it was described in chapter 3, SR registers also come with a certain overhead. For the final decision, several possibilities should be evaluated.

4.6 Number of required switches

An important issue for power gating is the number of required power switches for a power domain. It has to be ensured that the voltage drop over the switches

is low enough that the timing constraints can still be reached. For that purpose, the circuit is modelled as shown in Figure 4.4. R_{on} is the equivalent resistance of the power domain in active mode.

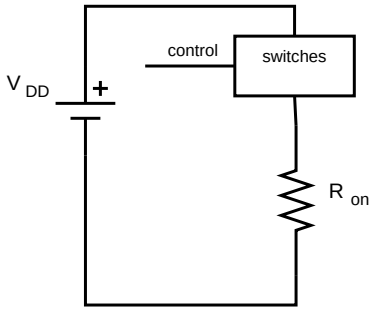


Figure 4.4: Equivalent circuit for determination of required number of switches

To determine R_{on}, the design is simulated with a testbench with high load for the power domain. Afterwards, power numbers are extracted. Then the equivalent resistance can be calculated using the basic formula

$$R_{on} = \frac{{V_{DD}}^2}{P_{total}} \tag{4.1}$$

A disadvantage of this method is, that it is application dependent. Therefore, the testbench has to be carefully chosen to make sure that a sufficient high R_{on} is found, otherwise the switches are eventually dimensioned too small. An advantage is, that it is a very fast method and it can easily be integrated into the existing design flow.

When the value for R_{on} is known, the circuit from Figure 4.4 is simulated in HSPICE using the spice-model of the power switch provided by the vendor. The simulation is performed using a configurable flow, where the desired number of power switches can be specified as well as R_{on}. During the simulation, it is checked if the voltage drop over the switches is low enough in order to reach the timing constraints. The scripts can be found in the electronix appendix (folder number_of_switches).

The relevant parts of the script to generate the circuit including R_{on} and power switches is shown below:

```
# generate model of power domain
puts $outfile "r1 vvdd_0 vss ${REQ_CUR}"

# generate switch net
for { set loopcount 0 } { $loopcount < ${row} } { incr loopcount } {
 puts $outfile "xswitch${loopcount}_1 nsleepin0 ${loopcount}_01 vss vdd
      vvdd ${SWITCH}"
 for { set i 1 } { $i < ${column} } { incr i } {
 puts $outfile "xswitch${loopcount}_[expr ${i}+1] ${loopcount}_[expr ${i
      }-1]${i} ${loopcount}_${i}[expr ${i}+1] vss vdd vvdd ${SWITCH}"
 }
}
```

Code Snippet 4.3: Generation of the spice circuit

The spice netlist for the simulation is as follows:

```
*** supply voltage and VSS ***

vdd vdd 0 1.2
vss vss 0 0

********** circuit **********

.include switch_chain.sp
.include circuit.sp

vin_nsleepin0 nsleepin0 0 dc 1.2

.END
```

Code Snippet 4.4: Spice netlist for the simulation for the determination of the required number of switches

4.7 Implementation of power gating

When the preparation steps which were presented in the previous sections are finished, the implementation itself has to be performed. For this purpose, the processor has to be modified with the required changes to support power gating. The resulting design has to be verified to make sure it is still functionally correct, for example by RTL-level simulations or by simulation tools provided by the processor-design-tool itself. In a next step, the additional modules, if required, have to be designed and also verified. Afterwards, they have to be integrated into the system followed by another verification of the resulting design.

The final step in the implementation is to integrate power gating into the actual design flow. For that, the scripting language CPF, which stands for *Common Power Format*, is used. It was initially designed by Cadence Design Systems, later it was transferred to the Silicon Integration Initiative (Si2) [59] which is now in charge of controlling the ongoing development of the CPF standard. CPF is used during all steps of the design flow, beginning with simulation RTL level where a power domain which is switched off is represented by all signals going to X. During synthesis, the definitions in the CPF file are used to insert isolation cells and to replace normal flip flops with state retention flip flops where required. In the place and route step, the actual power switches are inserted and all control signals are routed.

The complete design flow is is depicted in Figure 4.5. First, the new processor architecture has to be described using the processor modelling language (i.e. nML for the use of Target IP Designer and TIM for Hive Logic). Then, this information is used by the processor design tool to generate the HDL code (in this work, VHDL was used) of the RTL model of the processor. This RTL model, together with the CPF definitions, the design constraints and the technology libraries is used to performe the synthesis of the processor. Synthesis was performed with RTL Compiler by Cadence. The outcome is a synthesis netlist. Afterwards, the synthesis netlist, together with the CPF definitions, the design constraints and the technology libraries is used to perform place and route (P&R). Place and route was performed using Encounter by Cadence. The

outcome of this step is a place and route - netlist, information about the parasitics in the design and a SDF (standard delay format) file in which the delays of each cell in the design are stored.

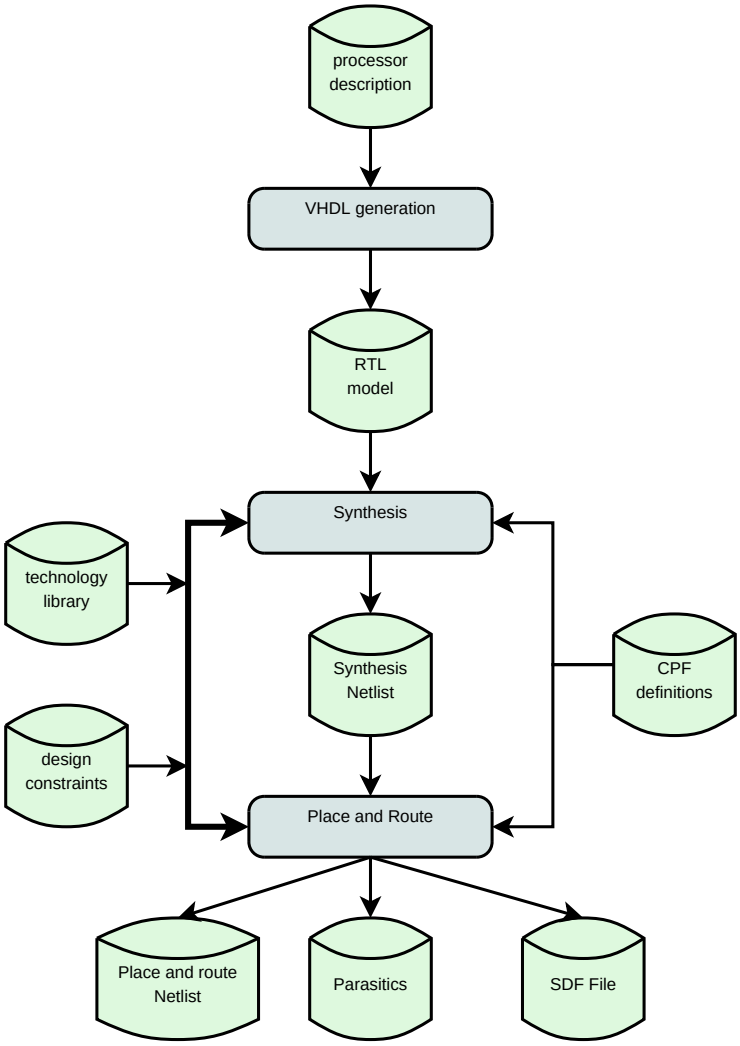


Figure 4.5: Design implementation flow

4.8 Verification that the system still works functionally correct

After the implementation of power gating it has to be verified that the system still works functionally correct. This is achieved by simulating the application on the post-P&R netlist. The application should therefore be self-testing or it should compute a specific result at the end which can be verified.

While a resource is switched off, it cannot be used. However, the compiler is not necessarily aware of that. Therefore, it has to be made sure that whenever a switch off signal for a specific power domain is given, it is not scheduled before it is switched on again. In Target IP Designer, there are several methods available. For example, a specific class of instructions, which have to be defined, can be deactivated for a function. This will be explained more detailed in section 5.4.

Sometimes, however, it is not possible to deactivate a specific power domain for scheduling, for example when it is switched off within a function. But when the application designer knows for sure, that the resource is not used it can still be switched off. But still a verification method is needed to verify the resource is really not used. For this purpose, a script was written in this Thesis. It takes the resource usage over time file, which is generated with the script explained in section 4.3. For each cycle, it is checked if a power domain is switched off. If yes, a flag is set. In the following cycles, it is checked for each instruction whether the specific resources are scheduled. If yes, than the power-down scheme is incorrect and a violation is reported. If it is not used, the resource can savely be switched off. The software flow can be seen in Figure 4.6. The scripts can be found in the electronic appendix (folder resource_not_used).

4.9 Evaluate energy savings

In this section, the methodology for the evaluation of possible energy savings is described. In order to determine the impact of power gating, the values for the factors which dictate the break-even point, as described in chapter 3, have to be known.

As a reminder, the factors for the determination of the break-even point according to Equation 3.9 are summarised shortly:

- $P_{mod,leak}$: the leakage power consumption of the power domain
- $P_{switch,leak}$, $P_{iso,leak}$, $P_{SR,leak}$: the leakage power of the power switch(es), isolation cells, and SR registers, respectively
- $P_{iso,active}$, $\Delta P_{SR,active}$: the power consumption of the isolation cells during active mode and the power consumed by the SR registers compared to what regular registers would consume during active mode
- $P_{add.modules}$: the power consumption of additional modules
- $E_{powerdown}$: the energy which is still consumed during powering down
- $E_{poweron}$: the energy required to switch on a power domain
- t_{total}: the total runtime

$P_{mod,leak}$, $P_{iso,active}$, and $P_{add.modules}$ can be determined by power number extraction as described in section 4.2. $P_{switch,leak}$, $P_{iso,leak}$, and $P_{SR,leak}$ can either be determined by power number extraction or the information can be extracted from the corresponding datasheets. $\Delta P_{SR,active}$ has to be determined by comparing the power consumption values of SR registers with power consumption values of the corresponding, non-SR registers provided in the datasheet.

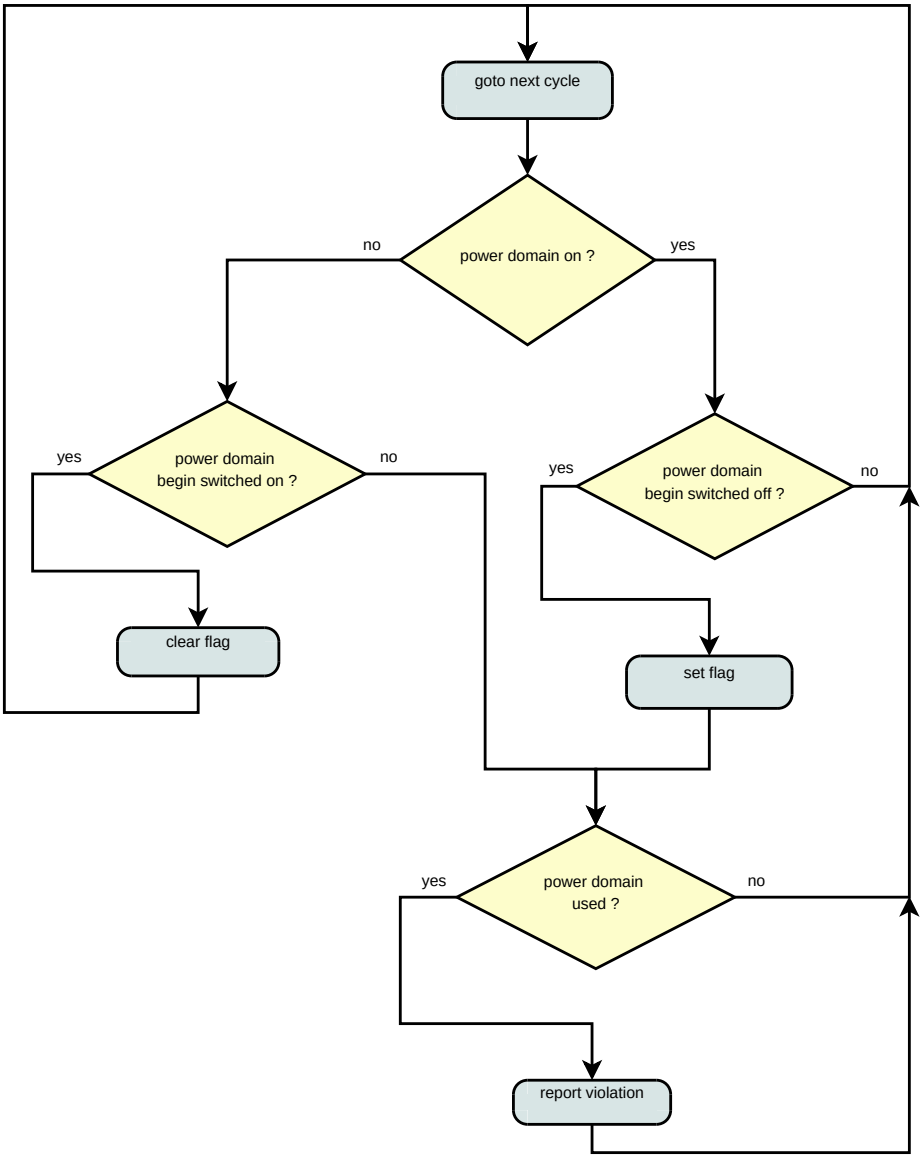


Figure 4.6: Flow for the verification that a resource is not used while it is switched off.

$E_{powerdown}$ and $E_{poweron}$ have to be determined using HSPICE simulations. This will be described in the following section. t_{total} is the total runtime of the application which can be determined by the required number of cycles.

4.9.1 Power-on /-off energy

In this section the determination of $E_{powerdown}$, $E_{poweron}$ and the time required to switch on a power domain is described. A configurable HSPICE simulation flow for accurate analysis was developed during this work.

Furthermore, the time required to switch on a power domain is identified during this flow. When fine-grained power gating is applied to the datapath in a processor, the power-on and power-off times are important. Depending on the time it takes to power down a power domain, it has to be determined how often and for how long it can be powered down. When it takes only a few clock cycles, it can be powered down more often than if it takes a long time. Power-off time affects the power-on time; when the power domain is not fully discharged, it takes shorter to switch it on.

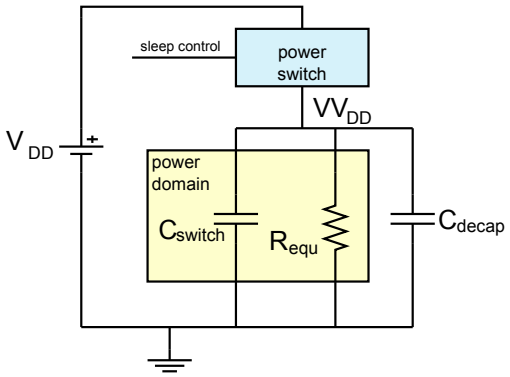


Figure 4.7: Spice model of the circuit

The power domain is modelled as shown in Figure 4.7. The power gated module is represented by its switching capacitance C_{switch} and equivalent resistance R_{equ}. In addition, the decoupling capacitor C_{decap} is present in the model. V_{DD} and VV_{DD} represent supply voltage and virtual supply voltage, respectively.

The switching capacitance can be determined with help of the following expression for the switching power:

$$P_{switching} = C_{switching} \cdot V_{DD}^2 \cdot f \tag{4.2}$$

where $P_{switching}$ stands for the total switching power of a module, $C_{switching}$ for the total switching capacitance, V_{DD} for the system's supply voltage and f for the clock frequency. This equation can be converted in order to calculate $C_{switching}$:

$$C_{switching} = \frac{P_{switching}}{V_{DD}^2 \cdot f} \tag{4.3}$$

The equivalent resistance R_{equ} can be found as follows:

$$R_{equ} = \frac{V_{DD}^2}{P_{leakage}} \tag{4.4}$$

where $P_{leakage}$ represent the leakage power consumed by the module.

The values for $P_{switching}$ and $P_{leakage}$ have to be determined using the method described in section 4.2. The supply voltage V_{DD} is technology dependend and the clock frequency f is a design constraint that has to be specified.

With the known switching capacitance, C_{decap} can be estimated with the following rule of thumb according to [60]:

$$C_{decap} = 9 \cdot C_{switching} \tag{4.5}$$

Furthermore, the way the power switches are connected and how many switches are used has an influence on the power-on energy. A schematic of the switches used in this work is depicted in Figure 4.8. The pin *Vdd* is connected to the always-on voltage supply, *VVdd* is the pin for the virtual voltage, *Vss* is the connection to ground, *sleep in* is the input for the control signal, and *sleep out* is connected to *sleep in* and can be used as an control input for the next switch.

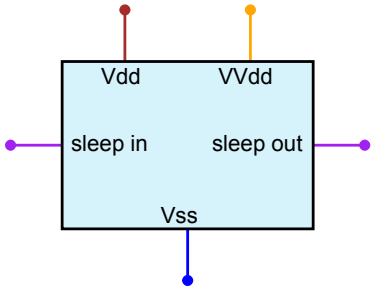


Figure 4.8: Schmeatic of the power switch

The switches are usually connected in a daisy chain with one switch controlling the next switch in the chain like it can be seen in Figure 4.9. The switches are modelled using the accurate spice model provided by the vendor. The daisy chain ensures that the power domain is not charged at once, but slowly to confine the power-on peak current. The *sleep in* signal of the first switch in the chain is connected to the sleep control signal of the power gating control unit. The *sleep out* signal is then connected to the next switch in the chain, thus introducing a delay in the switch-on process. It is crucial that the delay of the switch is larger than the time required for charging the module, otherwise daisy chaining does not have any effects on the peak current.

From the technology point of view, the size of the sleep transistors inside the power switch has a large impact on factors like peak current and power-on time. However, in this work 90 nm LP off-the-shelf power switches are used and modifying its transistor dimensions is out of scope for this thesis.

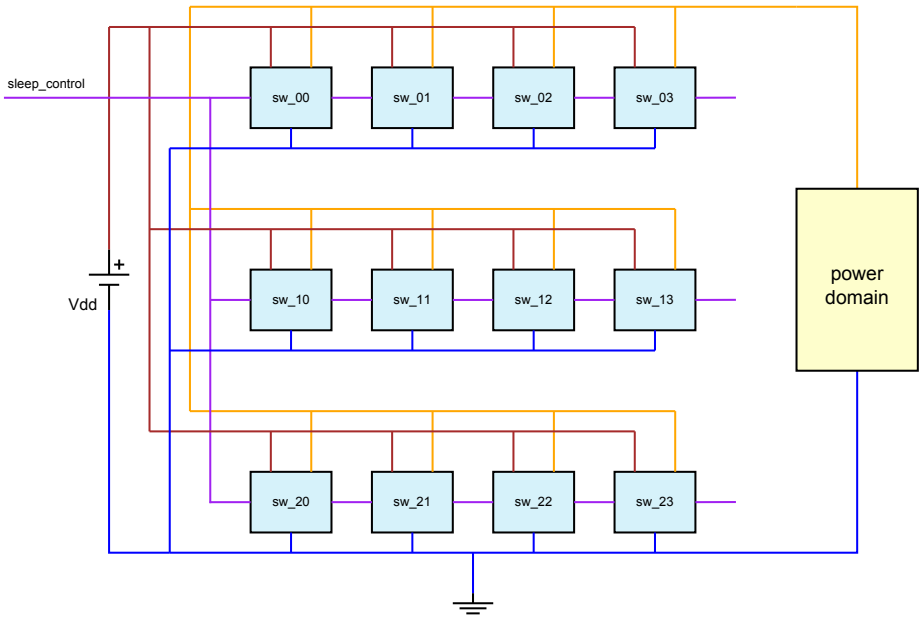


Figure 4.9: Spice model of the power gated block with a net of switches

The simulation flow is sketched in Figure 4.10. All required scripts were written in this thesis. They can be found in the electronic appendix (folder energy_and_time).

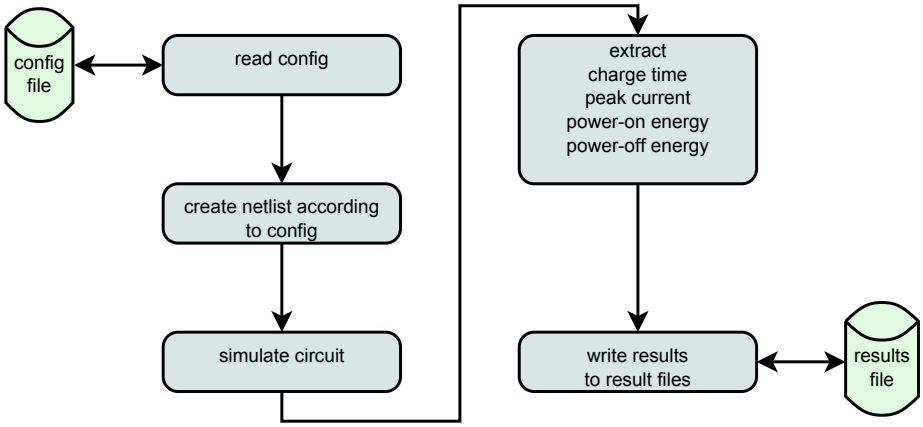


Figure 4.10: Spice simulation flow

In the config file, the values for C_{decap}, $C_{switching}$ and R_{equ} are specified. Also the configuration of the switch-net is specified in rows and columns. Based on that information, the circuit is generated. The relevant code snippets are shown below.

```
# generate model of power domain
puts $outfile "cswitch vvdd_0 vss ${CSW_CUR}"
puts $outfile "cdecap vvdd_0 vss ${CDEC_CUR}"
puts $outfile "r1 vvdd_0 vss ${REQ_CUR}"

# generate switch net
for { set loopcount 0 } { $loopcount < ${row} } { incr loopcount } {
 puts $outfile "xswitch${loopcount}_1 nsleepin0 ${loopcount}_01 vss vdd
     vvdd ${SWITCH}"
 for { set i 1 } { $i < ${column} } { incr i } {
 puts $outfile "xswitch${loopcount}_[expr ${i}+1] ${loopcount}_[expr ${i
     }-1]${i} ${loopcount}_${i}[expr ${i}+1] vss vdd vvdd ${SWITCH}"
 }
}
```

Code Snippet 4.5: Generation of the spice circuit

The relevant excerpts from the spice netlist for the simulation are presented below:

```
******** measurements ********

.tran 1n 302u

* charge time => time required for powering on a power domain
.MEAS tcharge WHEN V(vvdd)="vdd*0.95" RISE=2

* peak current during powering on
.MEAS TRAN PeakI MAX I(vsense) FROM=300us TO=302us

* energy for powering on
.MEAS TRAN ton WHEN I(vdd)="1.05*ILeak" RISE=1 TD=300us
.MEAS TRAN Ipenalty INTEG I(vdd) FROM=300us TO=ton
.MEAS E_on_penalty PARAM='-Ipenalty*1.2'

* energy during powering-off
.MEAS toff WHEN I(vdd)="ILeak*0.05" RISE=1 TD=2us
.MEAS TRAN Ioff INTEG I(vdd) FROM=2us TO=toff
.MEAS E_off PARAM='-Ioff*1.2'

*** supply voltage and VSS ***

vdd vdd 0 1.2
vss vss 0 0

********** circuit **********

.include switch_chain.sp
.include circuit.sp

* sleep control
vin_nsleepin0 nsleepin0 0 pulse 0 1.0 0 250p 250p 2u 300u
```

Code Snippet 4.6: Spice netlist for the simulation

4.9.2 Evaluate total energy savings

When the indidual components for the possible savings and the introduced overhead are determined, the total energy savings can be calculated. For that purpose, a tcl script was written during this thesis. It requires the values for the components for the savings and overhead, introduced as α, β', γ, and δ in chapter 3, those values are stored in a config file which is sourced by the script. In the script, Equation 3.13 is applied.

```
foreach domain $POWER_DOMAINS {
 puts [lindex $MODE $count]
 set SAVINGS [expr [lindex $RUN_TIME $count] * [lindex $OFF_TIME_PERCENT
      $count] /100 * [lindex $LEAKAGE $count]]
 puts "savings: $SAVINGS J"
 set OVERHEAD [expr [lindex $RUN_TIME $count] * [lindex $ADD_MODULES
      $count] + [lindex $RUN_TIME $count] *(100-[lindex
      $OFF_TIME_PERCENT $count])/100* [lindex $ISO_DYNAMIC $count] + [
      lindex $OFF_TIME_PERCENT $count]/100 * [lindex $RUN_TIME $count] *
       [lindex $ISO_LEAKAGE $count] ]
 puts "overhead: $OVERHEAD J"
 set BENEFIT [expr $SAVINGS - $OVERHEAD]
 puts "BENEFIT: [expr ${BENEFIT}] J\n"
 incr count
}
```

Code Snippet 4.7: TCL script for the evaluation of total energy savings

To visualise the possible energy savings depending on the off-time, a simple matlab script was written to plot the characteristics. The script is shown below. In the script, the values of α, β, γ, δ, ϵ and φ, which are named a, b, c, d, e, and f, respectively, in the script, have to be defined.

```
% define required variables here

x = tdown .* (a - b + c) - ttotal * (c+d) + f - e;
plot(x)
```

Code Snippet 4.8: TCL script for the evaluation of total energy savings

The scripts can be found in the electronix appendix (folder energy_savings).

Chapter 5

Implementation of Power Gating

In this chapter, the actual implementation of power gating is described. First, the different processors which were used during this thesis are explained in section 5.1. Then, the partitioning into power domains is described in section 5.2. In section 5.3 and section 5.4, two different implementation approaches for power gating are presented. Finally, the actual implementation of the power domains into the processors is described in section 5.5.

5.1 Architecture description

In this work, two different processors were used to explore the possibilities of power gating. The processors were designed for different purposes and with the help of different tools. In the following section, they will be explained in more detail.

5.1.1 Ultra wide band processor

The processor which is used in this work is an improved version of the Ultra Wide Band (UWB) processor designed at IMEC-NL as presented in [61]. The development of the processor was performed using the Target IP Designer (see section 4.1). Originally, the processor contained four issue slots, two for scalar operations and two for vector operations. The improved version has merged one of the scalar slots and one of the vector slots, leading to a three issue slot processor with one issue slot for scalar operations only, one combined issue slot for both scalar and vector operations, and one for vector operations exclusively. A schematic can be seen in Figure 5.1, the dotted rectangles represent the issue slots. The scalar issue slots contain an address generator for load/ store operations and two ALUs which can work in parallel. The vector issue slots contain a vector adder, an address generator for vector load/store operations,

and several function units which were implemented to execute specific operations during different stages in the UWB application of which the biggest one is a correlator. Furthermore, three vector registers are present. For this work, also a multiplier was added to the processor in order to make the processor suitable for a wider range of applications.

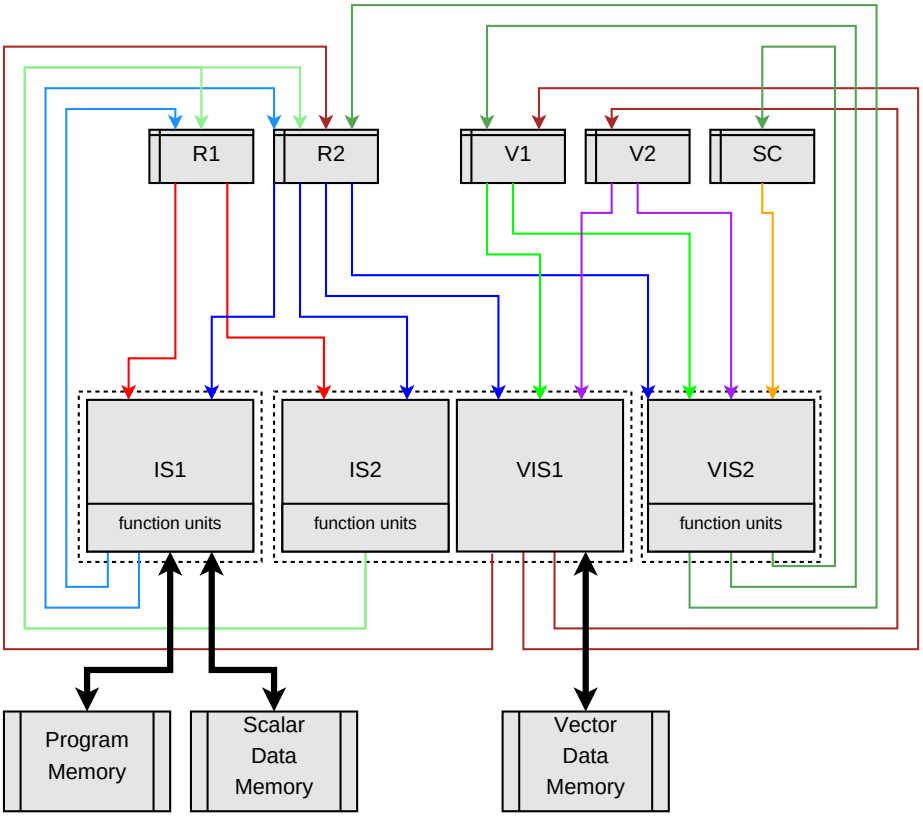


Figure 5.1: Overview of the UWB processor

UWB Application The processor was originally designed for an UWB application. The application iterates through several states, the corresponding state machine can be seen in Figure 5.2. First, an *initialisation phase* is executed during which the system is set up. Afterwards, a *noise estimation* is performed to determine the threshold values for *signal detection* which is executed afterwards. The number of iterations of the signal detection state depends on when a signal is detected. As soon as the signal detection was successful, the system is performing a *fine delay search* to perform accurate synchronisation. The mentioned states can be summarised as *Synchronisation / Timing Acquisition Phase*. Following, a *start frame delimiter (SFD) search* is performed to detect the end of the header. Finally, the data is decoded in the *payload demodulation* phase. The number of iterations depends on the length of the received message.

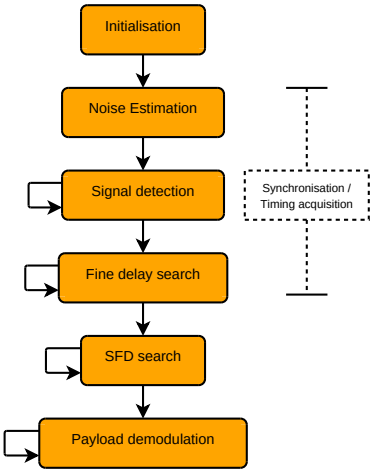


Figure 5.2: States of the UWB application

DWT Application The other application which is run on the processor is a discrete wavelet transform (DWT) like it is also used for JPEG2000. In the application, first a forward transform is executed and afterwards an inverse transform to check the results.

The DWT application was mapped on the processor to have a use-case of an algorithm which does not use the vector issue slots and registers, as they are a significant contributor to the leakage power consumption. The algorithm only relies on the normal scalar part of the processor, hence the vector part could be switched off. Therefore, it can be evaluated if it could make sense to have processors for several applications during which only their dedicated hardware is active.

For this work, the following use-case is assumed: First, the UWB receiver application is executed, afterwards, the DWT algorithm is performed.

5.1.2 Biomedical DSP

The second processor which was explored during this work was a biomedical digital signal processor (DSP), also designed at IMEC-NL. It was developed using the Silicon Hive's Hive Logic (refer section 4.1). The processor is part of a bigger design as depicted in Figure 5.3. Besides the processor (red block), it contains also device handlers (gray blocks) to be able to receive data from attached sensors and control and management blocks (green blocks).

The processor itself contains four issue slots, one RISC-like (*RISC*, one for load/ store operations and address calculations (*LD / ST*), one for EEG applications (*EEG*) and one for communication with devices (*I/O*).

The processor is designed for two different applications, one is an EEG (for detecting brain waves) algorithm based on the work presented in [62], the other one an ECG (for detecting heart beats) algorithm based on the algorithm devel-

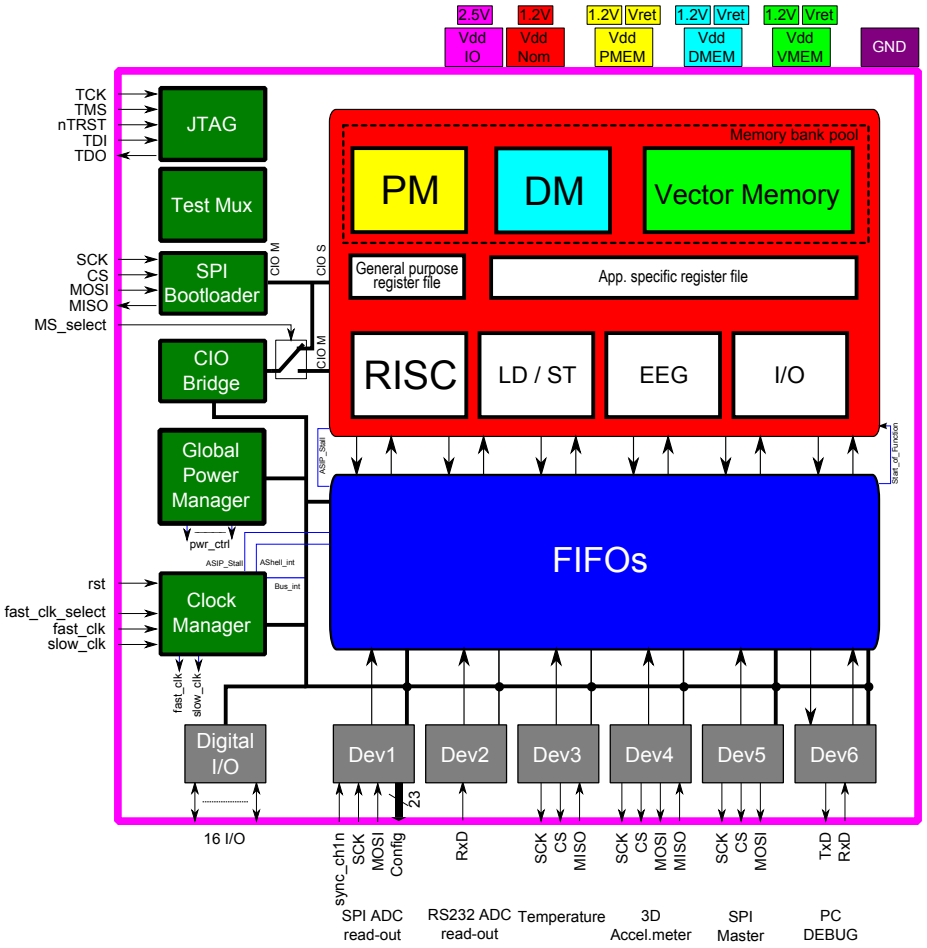


Figure 5.3: Biomedical DSP overview

oped in [63]. The processor can only be used for either of them simultaneously, i.e. it executes either the EEG application or the ECG application. For both applications, the system works as follows: The external sensors measure data and send it via the device handlers into the FIFO-buffers. When enough data is collected, the data is processed.

5.2 Partitioning into power domains

In this section the actual partitioning of the design into power domains is described. It is based on the methodology presented in chapter 4.

5.2.1 UWB processor

First, the power consumption of the modules in the datapath of the UWB processor is analysed. Then, the utilisation of the most significant modules is examined.

The analysis of the power consumption of the processor showed that the vector issue slots and the vector registers are a significant contributor to leakage power consumption. Within the vector issue slots, the main consumer of power is the vector adder and the correlator. In the scalar issue slots, the multiplier is significant.

The next important step is the determination of the utilisation of the modules. The two applications (UWB and DWT) are analysed in terms of the modules which were presented in the previous section.

The correlator is used during the complete runtime of the UWB application, so it is not a candidate for power gating, at least not during the UWB application. The vector adder is only used during two states, namely *signal detection* and *Fine delay search*. The vector issue slots including vector registers are utilised heavily during the UWB application, during the DWT application they are, as intended idle. The multiplier is used only during the DWT application but not in the UWB application. For the three relevant blocks, i.e. the vector adder *vec*, the multiplier *mul* and the vector issue slots including the vector registers *VIS*, the utilisation during selected phases within the UWB application and the DWT application is analysed further.

The resource utilisation of the blocks during the complete algorithm can be seen in Figure 5.4.

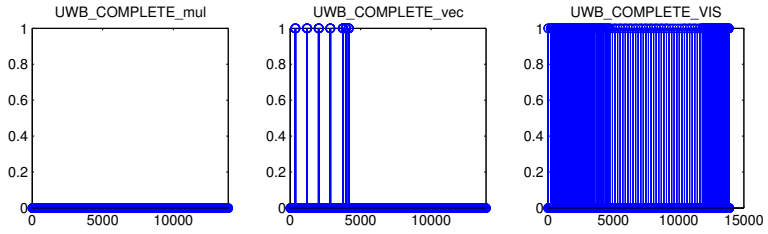


Figure 5.4: Resource utilisation of the power domains during complete UWB algorithm

The detailed resource usage durin the *signal detection* phase can be found in Figure 5.5, the utilisation during the *fine delay search* phase is shown in Figure 5.6.

The utilisation of the complete DWT algorithm can be seen in Figure 5.7.

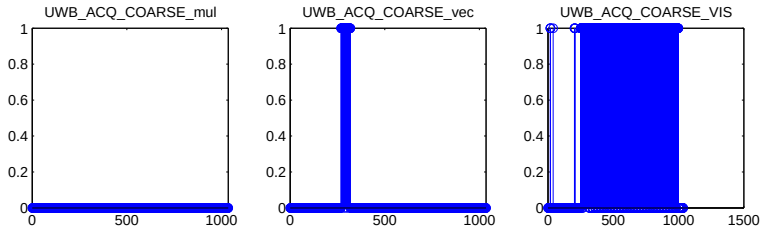


Figure 5.5: Resource utilisation of the power domains during signal detection in UWB application

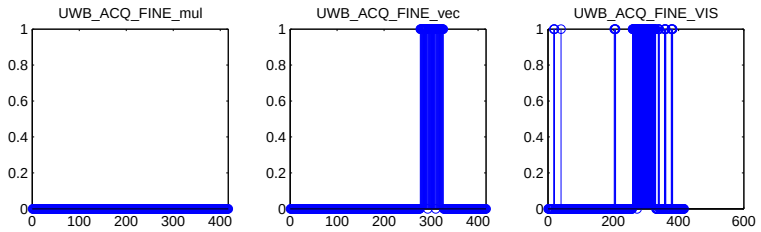


Figure 5.6: Resource utilisation of the power domains during fine delay compensation in UWB application

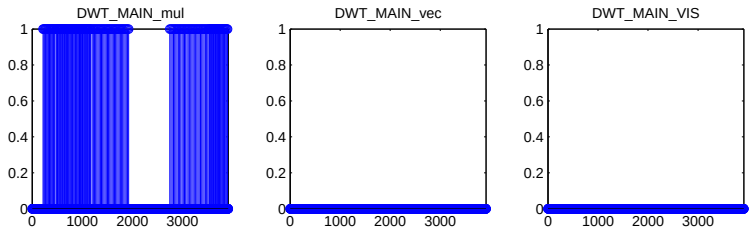


Figure 5.7: Resource utilisation of the power domains during complete DWT algorithm

Grouping

Based on the results from the power numbers and the utilisations, a partitioning into power domains can be performed. The vector adder is put into one power domain (*PD_vec*), the multiplier is also taken as one power domain (*PD_mul*). The vector issue slots are grouped to a big power domain, *PD_VIS*. Here, also the registers *V1*, *V2* and *SC* are included, hereby it is assumed that no state retention is required as this power domain can only be switched off during the DWT application. Also, whenever *PD_VIS* is switched off, *PD_vec* can be switched off too, as it requires the vector registers and the control logic within *PD_VIS*. The remaining modules are in the always-on power domain *PD_top*. A complete schematic about the power domains can be seen in Figure 5.8. An

overview of the power domains is also presented in Table 5.1.

Table 5.1: Power domains in the UWB processor

Power Domain	Modules	Gates	Outputs
PD_vec	vector adder	898	96
PD_mul	multiplier	1202	32
PD_VIS	vector issue slots, vector registers	13718	144

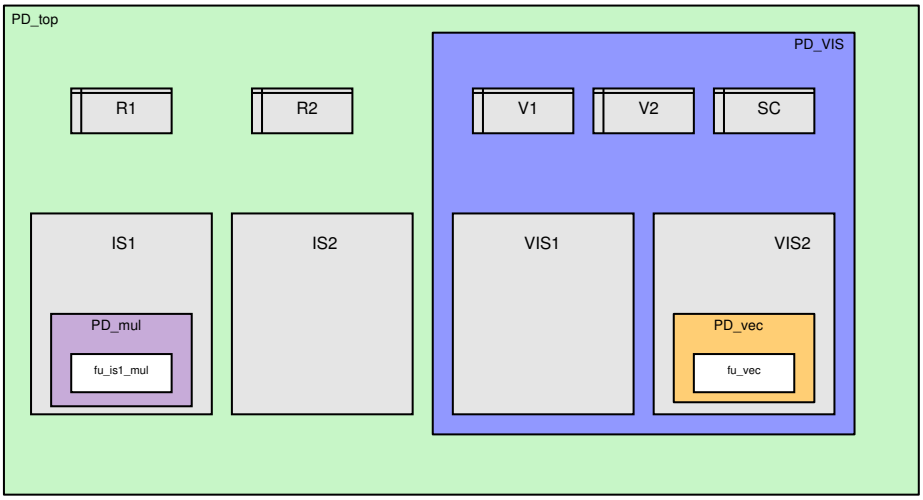


Figure 5.8: Power Domains in the UWB processor

Isolation

As explained in subsection 2.2.3, it is necessary to isolate the output of a power domain if the output signals go into a power domain which can be switched on. However, it has to be taken care that no isolation cells are put into the design that are not required as they contribute to the overhead (refer chapter 3).

The power domains *PD_VIS* and *PD_mul* have to be isolated against the always-on power domain *PD_top*. *PD_vec* has to be isolated against *PD_VIS*. *PD_VIS*, however, does not have to be isolated against *PD_vec* although it has output signals going to it. The reason is that whenever *PD_VIS* is switched off, *PD_vec* is switched off as well. The resulting isolation scheme can be seen in Figure 5.9.

Based on these findings, a power-off scheme for the UWB processor can be proposed. *mul* can be switched off during the complete UWB application. *vec* only needs to be switched on in *signal detection* and *Fine delay search* within the UWB application, otherwise it can also be switched off. *VIS* is used during the complete UWB application, but not in the DWT application, so it can be

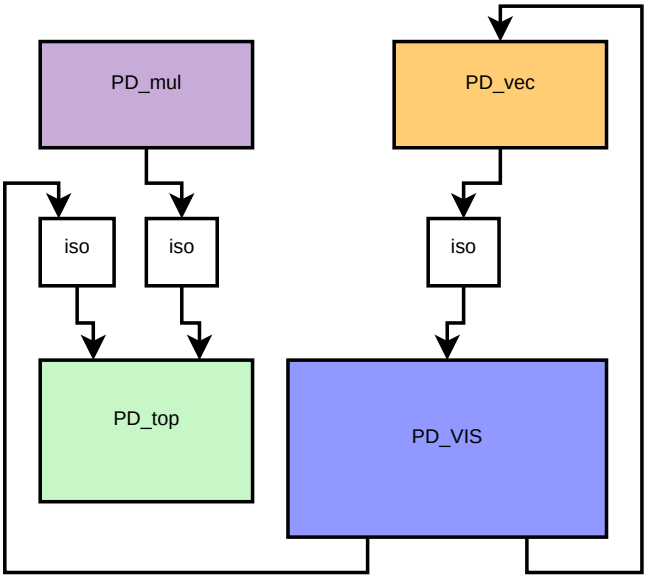


Figure 5.9: Isolation scheme for the UWB processor

switched off during the DWT application. An overview of the proposed power-off scheme can be found in Figure 5.10.

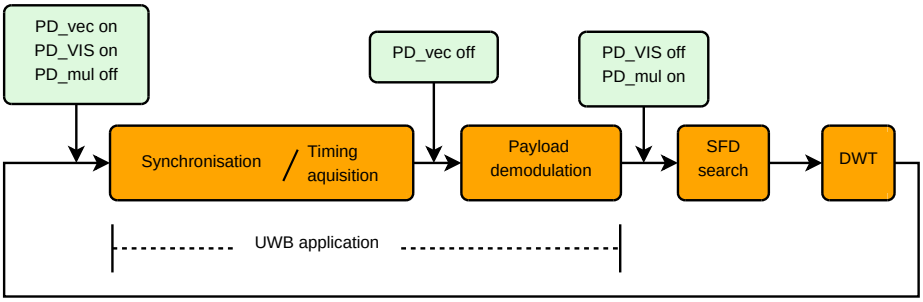


Figure 5.10: Possible power off scheme for the UWB processor

Number of switches

5.2.2 Biomedical DSP

As explained before, does the biomedical DSP contain four issue slots. Three of them, namely the *RISC* issue slot, the *LD / ST* issue slot and the *I/O* issue slots are used constantly during the applications. The remaining issue slot, *EEG*, is only used depending on the application. It was mentioned before, that the processor can either execute an EEG algorithm or an ECG algorithm. Therefore, the *EEG* issue slot was taken as the only power domain (*PD_EEG*) within this processor.

When the ECG algorithm is executed, the utilisation of the *EEG* issue slot is zero. During the EEG algorithm, it is used blockwise but only 2.6 % of the time. A schematic of the utilisation during the EEG algorithm is depicted in Figure 5.11. During the *idle* period, the processor is waiting for new input data. In that time, the issue slot *EEG* is not needed, only the remainding parts of the processor, represented by the green part in the graph. During the *R-peak search* period, the actual brain wave detection computation is performed and *EEG* is needed, represented by the blue part in the graph.

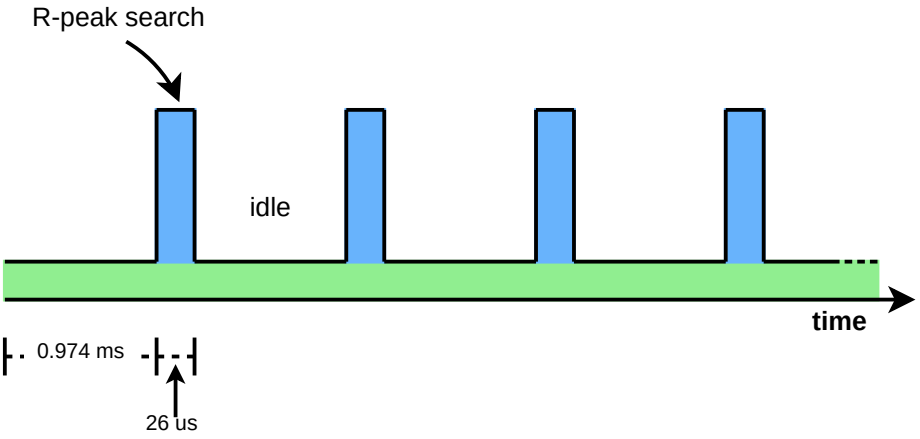


Figure 5.11: EEG algorithm

The proposed power-off scheme is therefore quite simple: When the ECG algorithm is executed, *PD_EEG* can be switched off completely. The power-off scheme for the EEG algorithm is depicted in Figure 5.12.

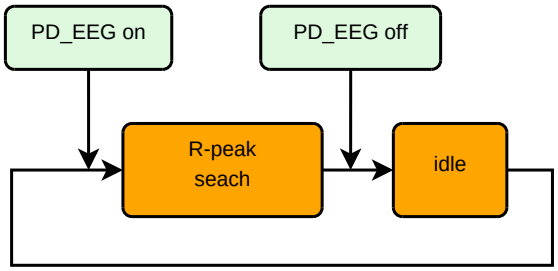


Figure 5.12: EEG power off scheme

Also isolation is simpler than for the UWB processor, as there is only one power domain, all its output have to be isolated.

5.3 Proposed power gating topology 1: HW based power manager

In this section, the first implementation is described. The proposed architecture relies on a dedicated control module which manages the power shutdown and power-up sequence as described in Figure 2.7. An overview can be seen in Figure 5.13. In the scheme, two switchable power domains (*PD_1* and *PD_2*) and one always-on power domain *PD_always_on*) are illustrated. During this work, the hardware based topology was only implemented for the UWB processor and not for the biomedical processor.

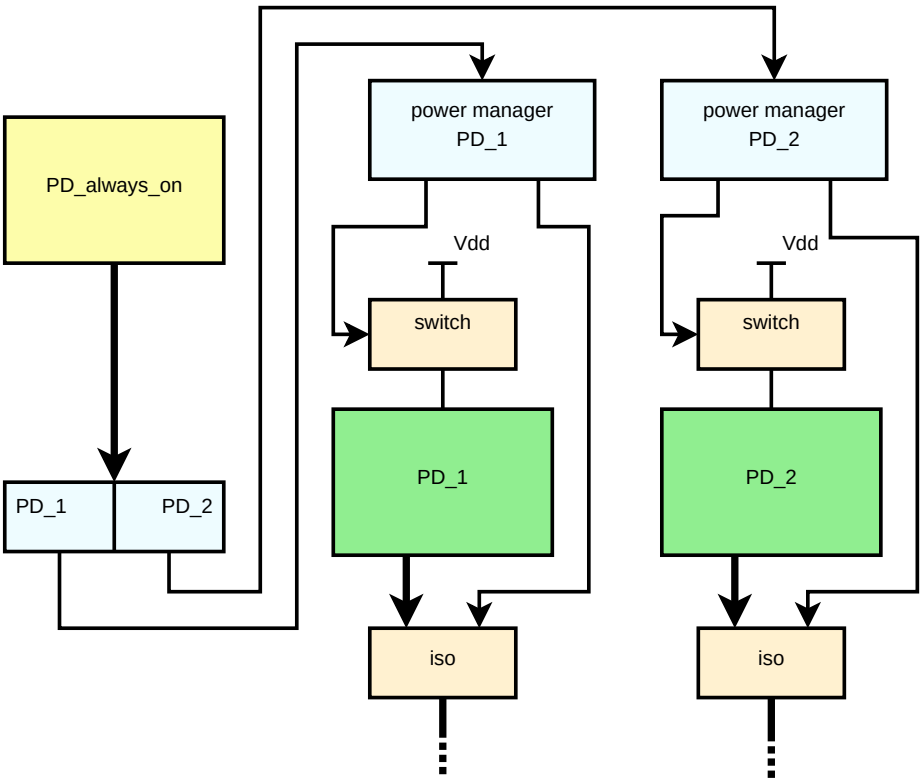


Figure 5.13: HW based power manager

5.3.1 HW implementation

In this section, the implementation of the additional modules which are added to the UWB processor is described. As mentioned before, has this processor been designed with the Target IP Designer which relies on the processor modelling language nML. In this language, all changes to the processor have been modelled. In the following section, they will be presented.

Power gating control register In order to control the on/off status of a power domain, i.e. if it is currently switched on or off, the current desired status of the power domain has to be stored. In this implementation, that was ensured with a control register. As the datapath of the processor is 16-bit wide, an additional 16 bits control register was added to the processor as shown in Figure 5.14. Its purpose is the control of the current status of the power domains. It contains one bit per power domain, *bit set* indicates that the power domain is shut off, otherwise it is on.

In this design, only three power domains were present, leading to a certain overhead in bits in the control register. It would also have been possible to implement a smaller register, but then an extra datapath for the desired number of bits and type-conversions would have to be implemented, too. This would presumably lead to additional power consumption and also a higher design-effort. However, in this implementation, the bits of the power domains were also made individually accessible, for which a dedicated function unit was implemented. The design choices were motivated by the possibility to explore the different impacts on the final power consumption of the control register.

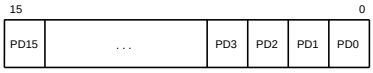


Figure 5.14: Power Gating Control Register

As explained in section 5.2.1, the UWB processor contains three switchable power domains. The bits of the register that are used to control the current status of the power domains, are each defined as single bit registers such that they can be accessed independently in the application. The single bits are concatenated to form a 16-bit register in order to enable single-cycle access to all bits simultaneously. The unused bits in the register are filled with dummy bits. Those can be used in the future when further power domains are implemented. The implementation can be seen below.

```
1  reg PG_VIS<uint1>;
2  reg PG_vec<uint1>;
3  reg PG_muk<uint1>;
4  reg PG_dummy<uint13>;
5
6  reg PG<width>
7  {
8    PG_VIS; PG_vec; PG_mul;
9    PG_dummy;
10 }
11 read(pg_r pg_c) write(pg_w);
12 outport pg_c_out<word>;
```

Code Snippet 5.1: Declaration of the control register

In order to be able to access the single bits registers from the application, each register has to be uniquely encoded in the instruction word. The implementation is shown in the code snippet below. In the lines 1 to 3, it is defined which register is accessed (*value*) and the corresponding name used in the assembly code (*syntax*). Following, it is defined which bit-pattern is assigned to which register. For example, the register *PG_VIS* is encoded with the bit pattern "00".

```
mode pg_VIS_reg() { value : PG_VIS; syntax : "PG_VIS"; }
mode pg_vec_reg() { value : PG_vec; syntax : "PG_vec"; }
mode pg_mul_reg() { value : PG_mul; syntax : "PG_mul"; }

mode pg_reg(pg_VIS_reg | pg_vec_reg | pg_mul_reg)
{
  image :
        "00" :: pg_VIS_reg
    |   "01" :: pg_vec_reg
    |   "10" :: pg_mul_reg
  ;
}
```

Code Snippet 5.2: Merge individual bits to one register

Additional instructions In order to be able to give specific power down instructions in the program code, four instruction were declared. Two instructions operate with a 16-bit value, one read and one write instruction, the other two are able to set and clear a specific bit. The 16-bit instructions rely on the existing register move operations, the single-bit instructions require a dedicated function unit, as described above.

The instruction to read a 16-bit value from the register is defined as follows, the syntax of the assembly instruction is '`mv t, PG`', where `t` defines the destination register.

```
opn pg_inst_r(t: opn_alut_is1)
{
  action {
    stage E1:
      alur_is1 = pg_r = PG;
      alut_is1 = alur_is1;
      t;
  }
  syntax : "mv "t", PG";
  image  : t class(pg);
}
```

Code Snippet 5.3: Read a value from the control register

The instruction to write a 16-bit value to the register is defined as follows, the syntax of the assembly instruction is '`mv PG, r`', where `r` defines the source register.

```
opn pg_inst_wr(r: opn_alur_is1)
{
  action {
    stage E1:
      r;
      alut_is1 = alur_is1;
      PG = pg_w = alut_is1;
  }
  syntax : "mv PG, "r;
  image  : r class(pg);
}
```

Code Snippet 5.4: Instruction to write a value to the control register

To be able to access the single bit registers, a dedicated functional unit (*fu_clearset*) was implemented.

The instruction to set a single bit is shown below. The syntax of the instruction is defined as 'sb_PG, pgreg' where pgreg follows the definition of the mode, as described in section 5.3.1.

In line 1, the function unit is defined. In the lines 3 to 5, the behaviour of the primitive operation is declared. In the remainder, the data flow (*action*), assembly syntax (*syntax*), and instruction word encoding (*image*) are defined.

```
1  fu fu_clearset;
2
3  uint1 setbit() {
4   return 1;
5  }
6
7  opn pg_inst_sb(pgreg: pg_reg)
8  {
9    action{
10     stage E1:
11       pgreg = pg_trn = setbit() @ fu_clearset;
12   }
13   syntax : "sb_PG, "pgreg;
14   image  : "000"::pgreg;
15 }
```

Code Snippet 5.5: Instruction to set a bit in the control register

Finally, the instruction to clear a single bit is defined as shown below. The syntax of the instruction is defined as 'cb_PG, pgreg' where pgreg follows the definition of the mode, as described in section 5.3.1.

```
1  uint1 clearbit()
2  {
3   return 0;
4  }
5
6  opn pg_inst_cb(pgreg: pg_reg)
7  {
8   action{
9    stage E1:
10    pgreg = pg_trn = clearbit() @ fu_clearset;
11  }
12  syntax : "cb_PG, "pgreg;
13  image  : pgreg class(pg_up);
14 }
```

Code Snippet 5.6: Instruction to clear a bit in the control register

In order to force the compiler to use those instructions, a variable has to be assigned to the corresponding storage element, i.e. the control register. The code for that looks like this:

```
INT16 chess_storage(PG) pg_reg;
```

In this case, the variable pg_reg is assigned to the power gating control register **PG**. In order to write a specific value to the register, the line

```
pg_reg = 0x01;
```

can be inserted into the code. The effect would be, that the value 0x01 is written into the register, using the previously defined instruction.

In order to set one specific bit, the compiler has to be forced to use the previously defined instruction. Therefore, a one-bit variable is declared and assigned to

the corresponding storage and the primitive `setbit()` is called. Therefore, a variable assigned to the specific bit in the register would have to be defined:

```
uint1 chess_storage(PG_VIS) pg_VIS_reg;
```

Then, the previously declared instruction can be used to set the bit:

```
pg_VIS_reg = setbit();
```

During the hardware generation of the function units in Target IP Designer, a framework for the described function unit is generated. The entity and architecture including the input signals and output signals and required control signals are implemented automatically. The relevant part of the generated framework is shown in Code Snippet 5.7. The functionality itself, however, has to be implemented manually. For the function unit *fu_clearset*, it is quite trivial as it only sets a '1' or a '0' to its output. The relevant part of the implementation is shown in Code Snippet 5.8. The only line that had to be changed was line four, in the generated default implementation, all output signals are set to '0'.

```
1     case bin_selector_E1 is
2       when "01" =>
3         clearbit(pg_trn_out);
4       when "10" =>
5         setbit(pg_trn_out);
6       when others =>
7         null;
8     end case;
```

Code Snippet 5.7: Framework of the function unit fu_clearset

```
1   procedure setbit(
2     signal pg_trn_out : out t_uint1) is
3   begin
4     pg_trn_out <= '1';
5   end setbit;
6
7   procedure clearbit(
8     signal pg_trn_out : out t_uint1) is
9   begin
10    pg_trn_out <= '0';
11  end clearbit;
```

Code Snippet 5.8: Functionality of the function unit fu_clearset

The dedicated power gating module

To implement the control sequence that was explained in subsection 2.2.5, a dedicated power gating control module, a power manager, was designed. It uses the current value of the corresponding bit for the power domain in the power gating register as input. For each power domain one power gating module is instanciated.

A block diagram can be seen in Figure 5.15. The inputs of the power gating module are the system clock (*clk*), a reset input (*rst*) to set the module in an initial state and the power gating control signal (*pg_ctrl*), which is connected to the corresponding bit for the power domain in the control register, as depicted in Figure 5.13. The outputs are the control signal for the isolation cells (*iso*), the control signal for the power switch (*ps*) and the reset signal (*pd_reset*) for

the power domain that is required when the power domain contains registers that have to be set to a known state after the power domain has been switched on after it was switched off.

The state machine for the power gating module can be seen in Figure 5.16. After reset, the module is in the state *UP*, i.e. the corresponding power domain is switched on. While the control signal *pg_ctrl* signal remains '0', the power gating module remains in this state, its output signals are all zero. As soon as the control signal switches to '1', the modules goes into the *ISOLATE* state. In this stage, the isolation control signal *iso* is set to '1' to enable isolation of outputs of the power domain. When the control signals stays '0', the module switches to the next stage, *DOWN*, otherwise it goes back to the state *UP*, meaning that the switch off-procedure was cancelled.

In the *DOWN* state, *iso* remains '1' and also the control signal of the power switch, *ps*, is set to '1' to disconnect the power domain from the power supply, i.e. switch it off. The modules stays in this state as long as the control signal remains '1'. As soon as it is set to '0', the switch-on procedure has to be executed. For that purpose, the module switches to the *POWER_UP* state. In this state, *iso* is still '1', but *ps* is set to '0' in order to switch the power domain on. Also, the reset signal *pd_reset* is set to '1' to reset the registers in the power domain. When the control signal remains '0', i.e. the switchin on procedure is not cancelled, the system proceeds to the next state (*RESET_STATE*), otherwise it goes back to the *DOWN* state.

In *RESET_STATE*, the *pd_reset* is set to '0'. By doing that, the reset is finished. Next, the system goes to the *UP* in which also *iso* is set back to '0', i.e. the outputs of the power domain are not isolated anymore.

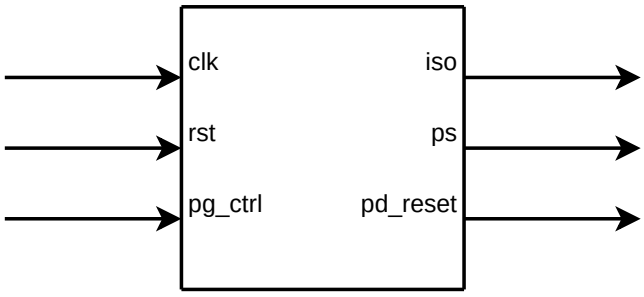


Figure 5.15: Block diagram of power gating control unit

5.3.2 SW implementation

The final step of the implementation of power gating into the processor was the actual implementation of the power-shutdown signal in the application. For that, the previously defined instructions to write to the power gating control register were used as explained. One main problem is that it has to be ensured that the instruction is scheduled at the correct position. The instruction to switch off a power domain has to be scheduled at a specific place within the application, as well as the switch-on instruction. For the switch-on instruction

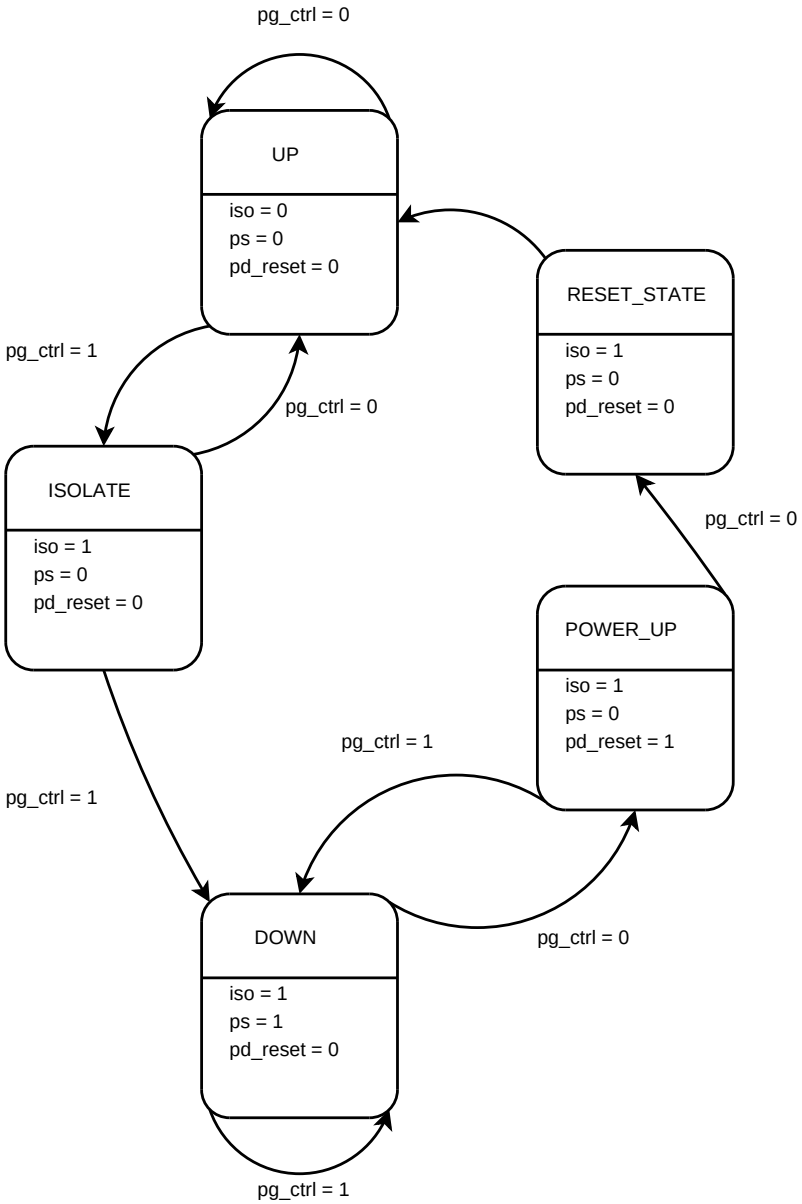


Figure 5.16: State machine of power gating control unit

it is of great importance that it is executed sufficient long before the modules within the power domain, that was switched off, are scheduled again by the compiler. For that, Target provides some useful features. It can be defined that instructions from a certain kind have to be scheduled ASAP, which can be used for the power-off instructions. But this still cannot ensure a scheduling at a specific position in the program flow. So, in this thesis another option was used, which is a seperator-instruction that is inserted in the program code. Everything before this instruction has to be scheduled before the separator. This can be

used to force the compiler to schedule the power-down and -up instruction at specific places. A disadvantage of this method is that the compiler loses freedom to generate a dense schedule when shutting off and powering on power domains. An illustration for power gating power domain *PD_VIS* is given below:

```
<code which is using VIS>

// switch off VIS
chess_separator_scheduler();
pg_VIS_reg = setbit();

chess_separator_scheduler();

<code which is not using VIS>

// switch on VIS
chess_separator_scheduler();
pg_VIS_reg = clearbit();
chess_separator_scheduler();

<code which is using VIS>
```

Code Snippet 5.9: Force compiler to schedule power-gating instructions at a specifig position

The example above shows the case that a power domain is switched off within a function. However, it could also be the case that a specific power domain has to be switched off for a specific function. For example, an algorithm consists of different function calls which are executed sequentially. In one of the functions, one power domain can be switched off because it is not needed. For this case, Target provides a useful feature that enables the programmer to disable a certain function-class for a specific function. Therefore, the *class* attribute has to be used during definition of the instructions. In the illustration for the definition of the read-instruction for the control register, shown in Code Snippet 5.3, the instruction was added to the class *pg*. In the same way, all instructions that are executed on modules belonging to a specific power domain, can be grouped in one class. As an example, let us assume, all instructions of *PD_vec* are assigned to the class *vec*. In the code below, they are disabled for the function *fct_1()*. In the main function, this function is executed after *PD_vec* has been switched off. After execution of *fct_1()*, *PD_vec* is switched on again.

```
void fct_1(void) property(disable_instruction_class_vec)
{
 <code>
}

int main()
{

 <code>

 switchoff_vec();

 fct_1();

 switchon_vec();

 <code>

 ...

}
```

Code Snippet 5.10: Disable instruction class for a specific function

5.4 Proposed power gating topology 2: SW based power manager

In this section, an alternative control topology is presented. The method presented in the previous section relies on a dedicated power gating control unit. However, it is also possible to implement the power shutdown and power-up sequence completely in software, hence the extra hardware for the control unit becomes obsolete. A schematic can be found in Figure 5.17. This topology was implemented for both the UWB processor and the biomedical processor.

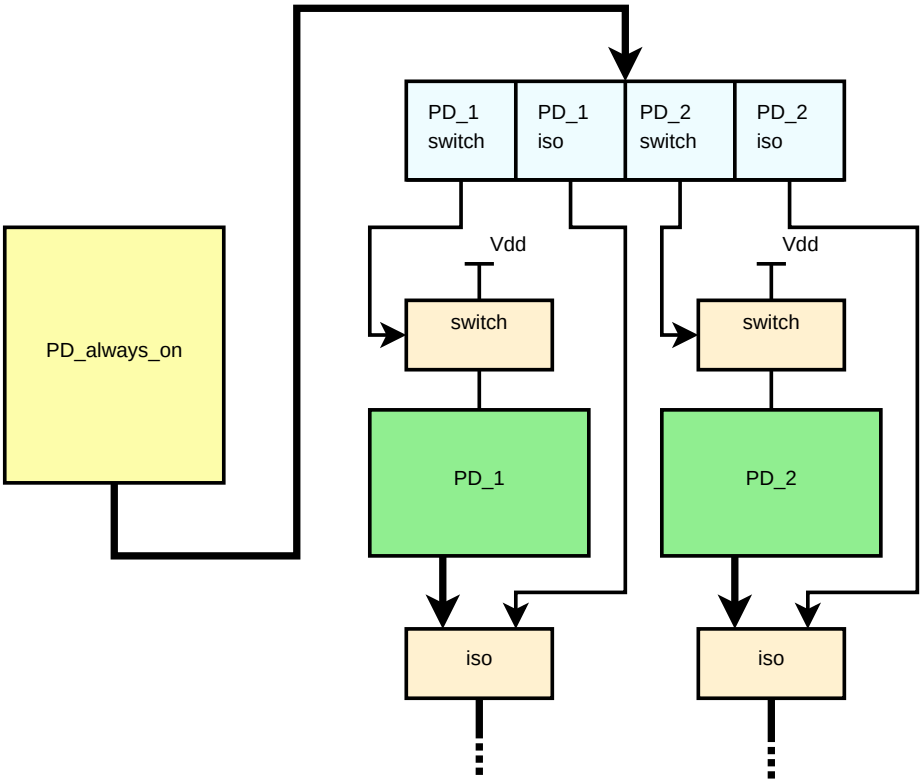


Figure 5.17: SW based power manager

5.4.1 Implementation on the UWB processor

In this section, the implementation of the SW based power manager on the UWB processor is explained. First, the additionally required modules, i.e. the necessary changes in the hardware of the processor, are explained. Besides, a comparison to the implementation of the power gating topology which relied on the HW based power manager is drawn. Afterwars, the implementation in the software itself it described.

Additional modules For this approach, no power gating control unit is needed. Furthermore, the power gating control register is implemented in a different way. In the first implementation, the register was split into single bit to access each bit separately. This, however, had the disadvantage that clock gating cannot be applied reasonably, leading to a large overhead due to the continously switching clock input. This also removes the need for a separate function unit to set or clear a bit (*fu_ clearset*). Clock gating is usually implemented using a special clock-gating latch. This latch also consumes power when the clock is deactivated. When the clock-gated part of the design is sufficient large, the additional power consumed by the clock gating latch is clearly justified by the power which is saved due to clock gating. However, when only one single flip-flop is clock gated with a latch, the overhead power consumption of the clock gating latch does not exceed the savings that can be obtained.

In this implementation, more bits are needed for each power domain. One for isolation, one for power-shutoff and, if there are registers in the power domain, a bit that controls the reset. The principle is illustrated in Figure 5.17. The bit-configuration is depicted in Table 5.2, *ps* represents the switch-off control signal that is connected to the power switch, *iso* is the isolation control signals which is connected to the isolation block and *rst* is the reset signal which is connect to the reset signal of the power domain (if available).

Bit	**Function**
0	ps *PD_VIS*
1	iso *PD_VIS*
2	rst *PD_VIS*
3	ps *PD_vec*
4	iso *PD_vec*
5	ps *PD_mul*
6	iso *PD_mul*
7-15	unused

Table 5.2: Bit configuration for the control register

Additional instructions The instructions to read and write a 16-bit value from and to the register remain the same, the instructions to clear and set one bit are dismissed.

SW implementation The implementation in the application code becomes a bit more complicated in this method. Whenever a value is written to the control register it has to be ensured, that no bits, which are responsible for the control of other power domains, are changed accidently. That can either be done by carefully writing the correct value to the register while remembering which bits are already set. This, however, is very error-prone. The other possibility is to read the value first and then perform a locical *or*-operation (to set specific bits) or an *and*-operation (to clear specific bits). This method is safer than the first one but also requires more cycles. Furthermore, the correct power gating sequence, which was presented in Figure 2.7 has to be implemented in

the application, i.e. the bits of the control register have to be set/cleared in the correct order following Figure 2.7.

Below, the implementation for the power domain *PD_VIS* is shown. The required instructions to switch the power domain off and on have been combined to a function each to simplify the implementation in the application code. The funtion to switch *PD_VIS* off is shown below:

```
1  inline void shutdown_VIS()
2  {
3    // shut off PD_VIS and PD_vec
4    chess_separator_scheduler();
5
6    // isolate PD_VIS, PD_vec
7    pg_reg = pg_reg | 0x0012;
8    chess_separator_scheduler();
9
10   // switch off PD_VIS, PD_vec
11   pg_reg = pg_reg | 0x0009;
12   chess_separator_scheduler();
13 }
```

Code Snippet 5.11: Function to switch off VIS

The function to switch *PD_VIS* on is implemented as follows:

```
1  inline void switchon_VIS()
2  {
3   // switch on
4   pg_reg = pg_reg & ~0x0009;
5   chess_separator_scheduler();
6
7   // reset
8   pg_reg = pg_reg | 0x0004;
9   chess_separator_scheduler();
10  pg_reg = pg_reg & ~0x0004;
11  chess_separator_scheduler();
12
13  // de-isolate
14  pg_reg = pg_reg & ~0x0012;
15  chess_separator_scheduler();
16 }
```

Code Snippet 5.12: Function to shut down VIS

The actual implementation in the application code is shown below.

```
1 <code which uses VIS>
2
3 shutdown_VIS()
4
5 <code which does not use VIS>
6
7 switchon_VIS()
8
9 <code which uses VIS>
```

Code Snippet 5.13: Function to shut down VIS

5.4.2 Implementation on the Biomedical DSP

The software based power manager approach depicted in Figure 5.17 was also implemented in the biomedical DSP. This processor was designed using Hive Logic. In the following section, the modifications which were applied to the processor are presented.

HW implementation

As explained in subsection 5.2.2, the processor only has only one power domain. Furthermore, this power domain does not require state retention. To control the power switches and the isolation cells, a 2-bit control register was added to the datapath. This was performed by the following declaration in the description file of the processor. In line 1, the number of bits in each register is defined. In line 2, the number of registers in the register file is declared. Afterwards, a register file with one read port and one write port is instantiated.

```
1 signed rf_pg_ctrl_size := 2;
2 signed rf_pg_ctrl_cap := 1;
3
4 RFwc1x1 rf_pg_ctrl <rf_pg_ctrl_size,rf_pg_ctrl_cap> (is_scalar.op_pg);
```

Code Snippet 5.14: Declaration of the control register

In order to write a value to the register, a special *pass-unit* was implemented. The declaration can be seen below. In the lines 1 to 3, the semantics for the pass-operation are defined. In line 5, the function unit is defined with two arguments (the width of an integer and the width of the control register). In line 7, the in and output ports are defined. In line 10, the timing information for the operation is declared. In line 11, the actual operation is instantiated and renamed to clarify the purpose of the operation (*std_pass_pg*). Line 12 tells the tools that the VHDL for this operation is user-defined.

```
1 OP std_pass_u  (Unsigned A) -> (Unsigned R) {
2    SEM R(A)= {R = (unsigned<Width:=(Width(R))>)A; };
3 };
4
5 FU HLUD_cfu_pass_pg <signed intWidth, signed pgWidth>
6
7 ( portW<intWidth> ip0 ) -> ( portW<pgWidth>op0 )
8
9 {
10        Cycle[0] := {ip0, op0};
11        std_pass_pg :    op0 = std_pass_u (ip0);
12        Implementation := external;
13 };
```

Code Snippet 5.15: Declaration of the pass unit

The pass unit is then instantiated in the datapath of the processor. First, the parameters for the instantiation of the function unit are defined in lines 1 and 2. Then, an instantiation of the function unit with the name *pgu* is created.

```
1 signed intWidth := 32;
2 signed pgWidth := 2;
3
4 HLUD_cfu_pass_pg pgu <intWidth, pgWidth> (ip0);
```

Code Snippet 5.16: Instantiation of the pass unit

SW implementation

The previously defined instruction *std_pass_pg* can be used to write a value to the control register. As illustration, it is shown how the power domain in

the biomedical DSP can be switched off. Bit 0 controls the isolation cells, bit 1 controls the power switches.

```
1   // pg write
2   int a;
3
4   // isolate power domain, set bit 0
5   a = 0x1;
6  L01: pg=OP_std_pass_pg(a) VOLATILE;
7
8   // switch power domain off, set bit 1
9   a = 0x3;
10 L02: pg=OP_std_pass_pg(a) VOLATILE AFTER (L01,2);
11
12  init_system() AFTER (L02,1);
```

Code Snippet 5.17: Instanciation of the pass unit

The variable *a* is used to specify the value which is written to the control register *pg*. By using the operation `pg=OP_std_pass_pg(a) VOLATILE;` the compiler is forced to use the previously defined dedicated pass-unit which is connected to the control register. By doing that, the value which is stored in *a* is transferred to the control register *pg*. The `VOLATILE` attribute is used to prevent that the instruction is optimised away by the compiler.

L01 and *L02* are labels used to control the sequence of the instructions. In the code above, line 8 is scheduled at least two clock cycles after L01 (by using the attribute `AFTER`). By doing so, it is ensured that the power domain is first isolated before being switched off. The function to initialise the system (*init_system()*) is scheduled at least one clock cycle after *L02*.

In Hive Logic, a specific function unit can be deactivated for an application. This is performed by passing `'-fdisable-fu <function_unit_name>'` as argument to the compiler, where `<function_unit_name>` has to be replaced with the name of the function unit that has to be deactivated.

5.5 CPF Flow

When the modifications presented in the previous section are performed and the system is simulated to verify correct functionality, the required steps to synthesise and place and route the design have to be performed. The general flow has been presented in Figure 4.5.

To implement the power domains into the design, CPF is used. The basic principle was explained in section 4.7. The required declarations are described in the following.

First, the power gating related cells have to be declared. During this work, isolation cells and power switch cells had to be defined:

```
1 define_isolation_cell -cells <cell_name> -power <power_supply_pin>
      -ground <ground_pin> -enable <iso_pin>
2 define_power_switch_cell -cells <cell_name> -power_switchable <
      switchable_power_supply_pin> -power <power_supply_pin>
      -stage_1_enable <ps_pin> -stage_1_output <out_pin> -type header
```

Code Snippet 5.18: Declaration of the power gating cells

Then, the power and ground nets have to be defined for place and route. In the example below, an always-on power supply, a ground connection and the switchable power supply for the power domain *PD_VIS* are defined. The power supply voltages are 1.2 V.

```
create_power_nets -nets VDD -voltage 1.20
create_power_nets -nets VDD_VIS -voltage 1.20 -internal
create_ground_nets -nets VSS
```

Code Snippet 5.19: Declaration of the power and ground lines

Following, the power domains have to be defined. As an example, *PD_VIS* is illustrated. The power domain contains the instances *inst_VIS inst_reg_V1 inst_reg_V2* and *inst_reg_SC*. The control signal for the power switch is *pg_vis/ps*. The virtual supply voltage is connected to the pin *VDD_VIS*.

```
create_power_domain -name PD_VIS -instances {inst_VIS inst_reg_V1
    inst_reg_V2 inst_reg_SC} -shutoff_condition pg_vis/ps
create_global_connection -domain PD_VIS -net VDD -pins <power_supply_pin
    >
create_global_connection -domain PD_VIS -net VDD_VIS -pins <
    switchable_power_supply_pin>
create_global_connection -domain PD_VIS -net VSS -pins <ground_pin>
update_power_domain -name PD_VIS -internal_power_net VDD_VIS
```

Code Snippet 5.20: Declaration of the power domains

Afterwards, the isolation rules have to be defined, i.e. which signal is used to control the isolation and to which value the signals have to be forced. Below, the definition for *PD_VIS* is shown. The isolation cells are put at the output of the power domain, specified by the condition *-from PD_VIS -to PDtop*. The control signal for the isolation cells is *pg_vis/iso*.

```
create_isolation_rule -name ISO_VIS_low -from PD_VIS -to PDtop
    -isolation_condition pg_vis/iso -isolation_output low
update_isolation_rules -names ISO_VIS_low -location to -prefix iso_VIS
```

Code Snippet 5.21: Declaration of the isolation

Finally, the power switches have to defined. The implementation is shown for *PD_VIS*. They are connected to the supply voltage *VDD*.

```
create_power_switch_rule -name PS_VIS -domain PD_VIS -external_power_net
    VDD
update_power_switch_rule -name PS_VIS -cells  <cell_name>
```

Code Snippet 5.22: Declaration of the power switches

The script with the above defintions is used during different steps in the design flow, as explained in section 4.7.

Chapter 6

Results

In this chapter, the results for the power consumption breakdown of the overhead ($P_{switch,leak}$, $P_{iso,leak}$, $P_{iso,active}$ and $P_{add.modules}$) and the savings ($P_{mod,leak}$) of power gating, as introduced in chapter 3 are presented. $P_{add.modules}$ is composed of the power consumption of the control register ($P_{ctrl.reg}$) and, if available, the power manager ($P_{pwr.man}$). For the power consumption, both the relative distribution as well as the total numbers are shown. Furthermore, the results of the HSPICE simulations regarding the power-on and power-off energy ($E_{poweron}$ and $E_{powerdown}$, respectively) and the required time to switch on a power domain are shown. Also, the required number of switches for each power domain is presented.

For both processors, the power numbers for the analysis of the break-even point were determined as follows: The power consumption of the isolation cells during active mode was extracted during an active period of the application. Therefore, a state of the application was taken that was representative for the time the power domain is switched on. The power numbers for the additional numbers were extracted during the complete run-time of the application as they are switched on all the time. The leakage power numbers were extracted during the time of the application when the power domain was idle and could be switched off as the leakage power is also depending on the input values.

In this chapter, the results are only presented, a detailed discussion is performed in the next chapter. However, a short summary is given for the results already in this chapter in order to point the reader to the most relevant aspects of the presented graphs and numbers.

6.1 UWB processor

In this chapter, the results for the UWB processor are presented. It was synthesised with 90nm TSMC libraries in two different flavours, namely LP (low power) and G (general purpose) [64]. For the G library, the power numbers were extracted for the worst case (WC), i.e. $1.08V, 125°C$. For the LP library, both the WC and the typical case (TC), i.e. $1.2V, 25°C$ were extracted. The reason

to extract power numbers for both the typical and worst case is that leakage power has an exponential relation to temperature. Therefore, the temperature is expected to have great influence on the break-even point.

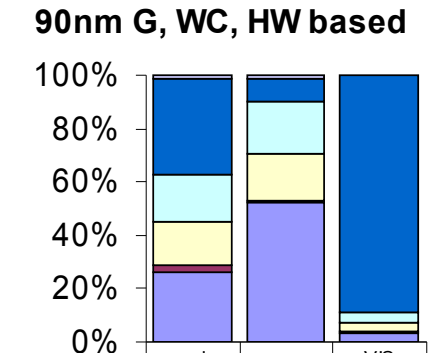
The results are presented for the three cases for both implementations, the hardware based power manager as explained in section 5.3 and the software based power manager, as illustrated in section 5.4.

6.1.1 HW based power manager

In the following, the results for the hardware based power manager implementation are presented.

G library, WC measurements

The breakdown of the savings and the overhead for the UWB processor, implemented with the G library, for the WC is presented in Figure 6.1.



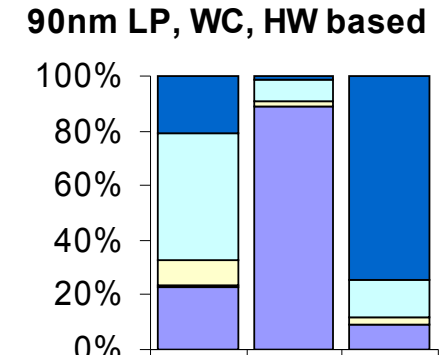
	mul	vec	VIS
P switch,leak	4.68E-07	4.68E-07	4.68E-07
P mod,leak	1.64E-05	3.65E-06	1.85E-04
P pwr.man	7.93E-06	7.93E-06	7.93E-06
P ctrl.reg	7.34E-06	7.34E-06	7.34E-06
P iso,leak	9.69E-07	2.12E-07	8.15E-08
P iso,active	1.19E-05	2.16E-05	7.45E-06

Figure 6.1: Results for G, WC, HW in [J]

It can be seen that for the power domains *PD_mul* and *PD_vec*, the possible power savings, i.e. the leakage power ($P_{mod,leak}$), which is the dark blue section, is smaller than the power consumption overhead, represented by the remaining sections. For *PD_VIS*, the power consumption overhead is small compared to the possible power savings.

LP library, WC measurements

The WC results for the implementation with the LP libraries can be seen in Figure 6.2.



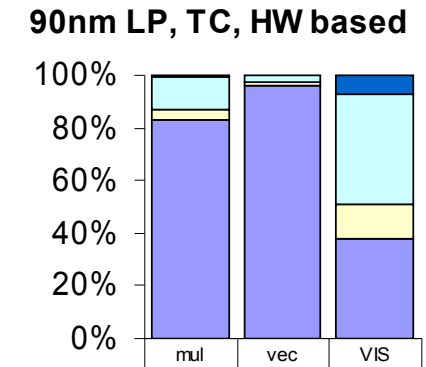
	mul	vec	VIS
P switch,leak	2.86E-08	2.86E-08	8.58E-08
P mod,leak	1.97E-06	6.36E-07	2.41E-05
P pwr.man	4.35E-06	4.35E-06	4.39E-06
P ctrl.reg	8.81E-07	8.71E-07	8.80E-07
P iso,leak	1.29E-08	1.98E-07	6.04E-08
P iso,active	2.18E-06	4.72E-05	2.95E-06

Figure 6.2: Results for LP, WC, HW in [J]

Compared to the implementation with the G libraries, it can be seen that the absolute values for the power consumption are smaller. The distribution of the components of the overhead is similar with slight variations.

LP library, TC measurements

The results for the typical case and the LP library are depicted in Figure 6.3.



	mul	vec	VIS
P switch,leak	1.20E-09	1.20E-09	1.20E-09
P mod,leak	2.40E-07	7.88E-08	9.37E-07
P pwr.man	5.43E-06	5.43E-06	5.43E-06
P ctrl.reg	1.61E-06	1.61E-06	1.61E-06
P iso,leak	1.19E-08	6.97E-08	4.63E-09
P iso,active	3.52E-05	1.84E-04	4.93E-06

Figure 6.3: Results for LP, TC, HW in [J]

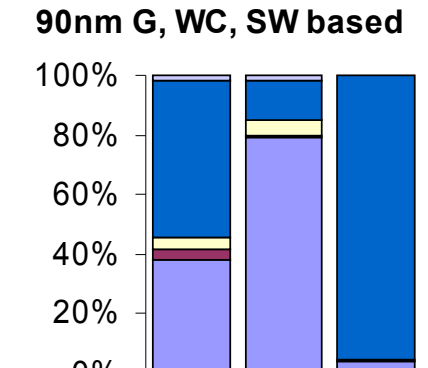
For the measurements for the typical case it can be seen that the leakage power consumption of the power domain ($P_{mod,leak}$) is much smaller than for worst case measurements, in this case on average a factor of 8. Also, the power consumption overhead is much bigger than the possible power savings for all power domains.

6.1.2 SW based power manager

In this section, the results for the software based power manager implementation are shown.

G library, WC measurements

The results for the WC measurements and the implementation with the G library can be found in Figure 6.4.



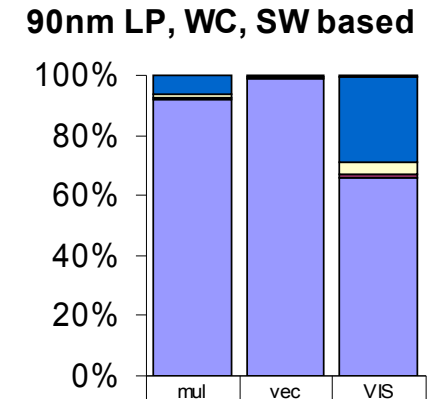
	mul	vec	VIS
Pswitch,leak	4.68E-07	4.68E-07	4.68E-07
Pmod,leak	1.64E-05	3.65E-06	1.85E-04
Pctrl.reg	1.35E-06	1.35E-06	1.35E-06
Piso,leak	9.69E-07	2.12E-07	8.15E-08
Piso,active	1.19E-05	2.16E-05	7.45E-06

Figure 6.4: Results for G, WC, SW in [J]

Compared to the first implementation presented in the previous section, it can be seen that the overhead has been reduced significantly. However, the power consumption overhead still exceeds the possible power savings for the power domains *PD_mul* and *PD_vec*.

LP library, WC measurements

The WC results for the implementation in the LP library can be found in Figure 6.5.



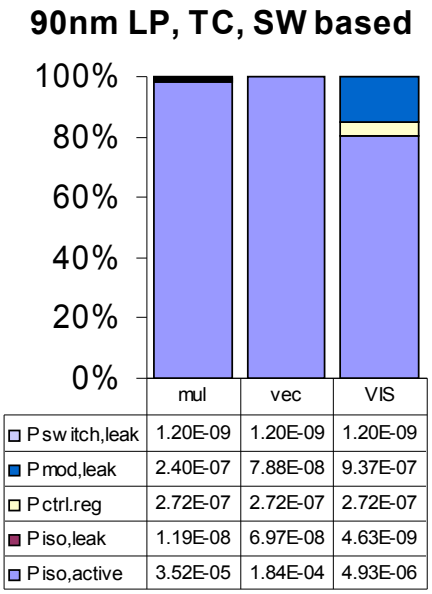
	mul	vec	VIS
P switch,leak	2.86E-08	2.86E-08	2.86E-08
P mod,leak	1.98E-06	7.04E-07	1.73E-06
P ctrl.reg	2.73E-07	2.73E-07	2.73E-07
P iso,leak	2.88E-07	6.53E-07	5.48E-08
P iso,active	2.87E-05	1.14E-04	4.04E-06

Figure 6.5: Results for LP, WC, SW in [J]

Compared to the implementation with the G libraries, the leakage power, i.e. the possible savings, is smaller for the LP libraries in both absolute numbers and also in relation to the overhead. For all power domains, the power consumption overhead exceeds the possible power savings.

LP library, TC measurements

The results for the TC and for the LP library are presented in Figure 6.6.



	mul	vec	VIS
Pswitch,leak	1.20E-09	1.20E-09	1.20E-09
Pmod,leak	2.40E-07	7.88E-08	9.37E-07
Pctrl.reg	2.72E-07	2.72E-07	2.72E-07
Piso,leak	1.19E-08	6.97E-08	4.63E-09
Piso,active	3.52E-05	1.84E-04	4.93E-06

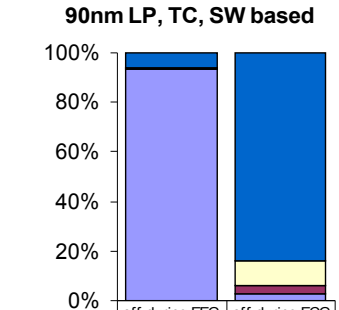
Figure 6.6: Results for LP, TC, SW in [J]

For the TC measurements, the leakage power consumption of the power domain is much lower than for WC measurements, like it has already be seen in the first implementation approach. The overhead is almost exclusively determined by the power consumed by the isolation cells during active mode.

6.2 Biomedical DSP

In this section, the results for the biomedical DSP are presented. This system was only synthesised for the LP library, also the power numbers were only extracted for the TC. This was done because there was already enough data present from the UWB to analyse the impact of different library flavours (G vs. LP) and different temperature conditions (WC vs TC). The purpose of the implementation of power gating on the biomedical DSP was to explore how easy the implementation could be ported to another processor-design software (in this case, from Target IP Designed to Hive Logic) and how different processor design tools influence the break-even point itself. Also the duty cycle of the power domain is much lower than for the UWB processor which enables an analysis for that case (i.e. a power domain with a low duty-cycle).

In subsection 5.2.2, the two use-cases for the power-off scheme were presented. The results for the first case, i.e. the EEG algorithm is executed and the power domain is shut off whenever possible during the application, are presented in the left bar of Figure 6.7. The results for the second use-case, i.e. the ECG algorithm is executed and the power domain can be shut off completely, are depicted in the right bar of Figure 6.7.



	off during EEG	off during ECG
Pswitch,leak	1.20E-09	1.20E-09
Pmod,leak	3.85E-07	3.85E-07
Pctrl.reg	2.72E-08	4.80E-08
Piso,leak	1.61E-08	1.36E-08
Piso,active	5.88E-06	1.36E-08

Figure 6.7: Results for LP, TC, SW in [J]

It can be seen that for the *off during EEG* case, the overhead is dominated by the power consumed by the isolation cells ($P_{iso,active}$). Also, the possible savings, i.e. $P_{mod,leak}$, are rather small in relation. However, for the *off during ECG* case, the possible savings are dominant in the power breakdown.

6.3 Results from the spice simulations

The results presented above were extracted using PrimeTime using netlist-simulations and the library-information of the standard cells. For the simulation, the LP libraries were used. However, in order to find out the power-on and power-off energy ($E_{poweron}$ and $E_{powerdown}$, respectively), the required time to switch on a power domain ($t_{switchon}$ are shown and the required number of switches $N_{switches}$, spice simulation were performed. The used methodology was described in subsection 4.9.1 and section 4.6, respectively.

The results for both processors are presented in Table 6.1.

	PD_mul	**PD_vec**	**PD_VIS**	**PD_EEG**
$t_{switchon}$	1.1 ns	2.5 ns	1.9 ns	1.7 ns
$E_{poweron}$	6.4 pJ	1.5 pJ	11.8 pJ	14 pJ
$E_{powerdown}$	0.4 pJ	0.04 pJ	0.4 pJ	0.4 pJ
$N_{switches}$	1	1	1	1

Table 6.1: Results of the spice simulations

6.4 Results for the break-even point

Having obtained all the necessary results, the determination of the break-even point, i.e. the minimum down times, can be calculated. In Table 6.2, both the extensive equation (Equation 3.9) as well as the simplified equation (Equation 3.11) are used. The results are labeled $t_{down,min,ext}$ and $t_{down,min,simple}$, respectively.

The results for the UWB processor are shown in Table 6.2.

	PD_mul	**PD_vec**	**PD_VIS**
90G, WC, HW			
$t_{down,min,ext}$ [%]	99	146	12
$t_{down,min,simple}$ [%]	96	146	12
90LP, WC, HW			
$t_{down,min,ext}$ [%]	179	110	31
$t_{down,min,simple}$ [%]	179	110	30
90LP, TC, HW			
$t_{down,min,ext}$ [%]	119	104	204
$t_{down,min,simple}$ [%]	119	104	204
90G, WC, SW			
$t_{down,min,ext}$ [%]	49	92	5
$t_{down,min,simple}$ [%]	47	91	5
90LP, WC, SW			
$t_{down,min,ext}$ [%]	95	100	76
$t_{down,min,simple}$ [%]	94	100	75
90LP, TC, SW			
$t_{down,min,ext}$ [%]	100	100	89
$t_{down,min,simple}$ [%]	100	100	89

Table 6.2: Results of the break-even point of the UWB processor

The results for the biomedical DSP are presented in Table 6.3.

	off during EEG	**off during ECG**
$t_{down,min,ext}$ [%]	95	16
$t_{down,min,simpl}$ [%]	94	15

Table 6.3: Results of the break-even point of the biomedical DSP

It is noticeable that for some cases the required minimum down time is above 100 %. This indicates, that the overhead will always exceed the possible savings, i.e. the power domain can never be switched off sufficiently long to gain energy savings. Furthermore, most of the remainding required down times (i.e. those which are not above 100 %) are rather high. This gives an indication that the overhead of power gating is significant.

6.5 Detailed analysis of the break-even point for selected cases

In order to analyse the different components of the overhead and savings better, an detailed energy breakdown was performed. Therefore, it was analysed how the different factors of energy savings and of the energy overhead were composed when the respective power domain was switched off for exactly the minimum down time, as presented in the results. For the UWB processor, the power domain *PD_VIS* was analysed for the cases G, WC, HW (G libraries, WC power numbers, HW based power manager), to have a scenario for the G library, LP, WC, HW (LP libraries WC power numbers, HW based power manager), to have a worst case scenario for the LP library and LP, TC, SW, (LP libraries, TC power numbers, SW based power manager) to have a typical case scenario for the LP library. The biomedical DSP was analysed for both existing use-cases.

In the presented graphs, the savings are shown in the left side of the graph. They are composed of the leakage energy of the power domain which could be saved ($E_{mod,leak}$) and the energy which is saved during switching off ($E_{powerdown}$). The overhead is represent by the right side of the graph. It is composed of the leakage energy of the switch and the isolation cells ($E_{switch,leak}$ and $E_{iso,leak}$, respectively), the energy composed by the additional modules ($E_{ctrl.reg}$ and $E_{pwr.man}$ if available), the energy consumed by the isolation cells during active mode ($E_{iso,active}$) and the energy required to switch a power domain on ($E_{poweron}$).

The results for the UWB processor, power domain *PD_VIS*, G library, WC, HW based power manager can be found in Figure 6.8. For the G library, the energy to switch on a power domain could not be measured, as the spice models for the power switches were unfortunately not available.

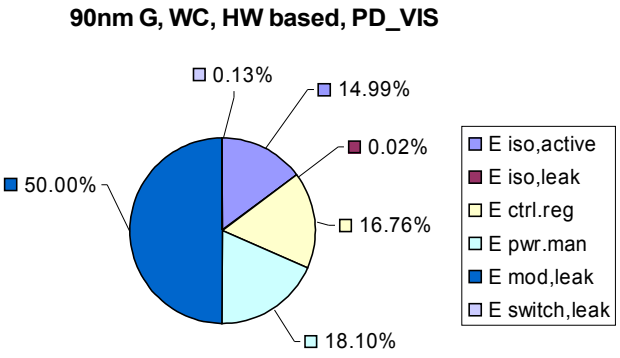


Figure 6.8: Detailed analysis for G, WC, HW in [J]

The results for the UWB processor, power domain *PD_VIS*, LP library, WC power numbers and HW based power manager implementation are depicted in Figure 6.9.

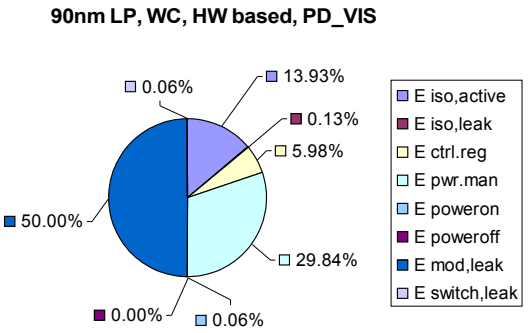


Figure 6.9: Detailed analysis for LP, WC, HW

The results for the UWB processor, power domain *PD_ VIS*, LP library, TC power numbers and SW based power manager are illustrated in Figure 6.10.

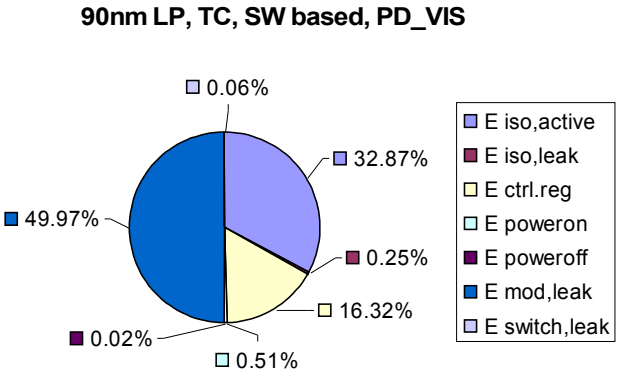


Figure 6.10: Detailed analysis for LP, TC, SW

The results for the biomedical DSP for the second use-case, i.e. the power domain is switched off during the EEG algorithm, are presented in Figure 6.11.

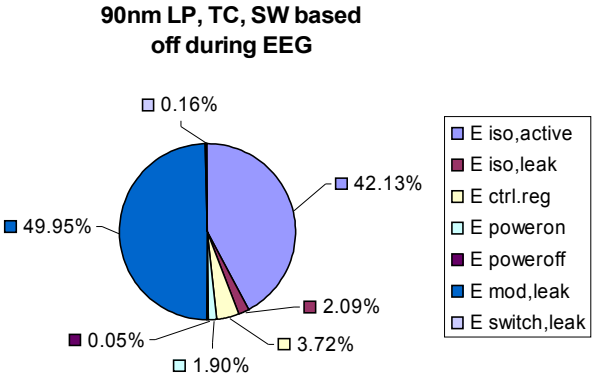


Figure 6.11: Detailed analysis, off during EEG

Finally, the results for the biomedical DSP, use-case one, i.e. the power domain is switched off during the ECG application, are shown in Figure 6.12.

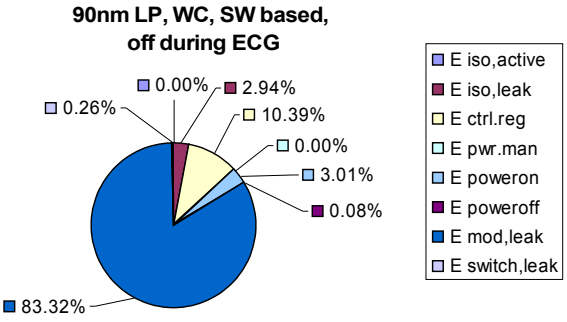


Figure 6.12: Detailed analysis, off during ECG

6.6 Absolute energy savings

In this section, the possible absolute energy savings are presented for the most relaistic cases, i.e. the implementation with the LP libraries and the SW based power manager and power number extraction for the typical case.

For the UWB processor, no energy savings can be achieved for that case. The required minimum down times (presented in Equation 3.9, in the block labelled **90LP, TC, SW**) for *PD_mul* and *PD_vec* are 100 %, for *PD_VIS* 89 %. However, *PD_VIS* cannot be switched off for the required time, as it has a high duty cycle (refer subsection 5.2.1).

For the biomedical DSP, the analysis of the total energy savings can be performed, as the duty cycles of the power domain are low enough to be switched off for the required minimum down time like presented in Table 6.3. For the first use-case, i.e. the power domain was switched off during execution of the EEG algorithm, the energy savings are evaluated per brain wave detection - iteration, i.e. per 1 ms (refer subsection 5.2.2). For the second use-case, i.e. the power domain is switched off complete during execution of the ECG algorithm, the energy savings were evaluated per iteration of the heart-beat detection algorithm. The principle of the ECG application is, like for the EEG application, that the system waits for input data, then it performes the required computations, then it waits again for input data. The waiting period for the ECG algorithm is 4.9694 ms, the computation period is 39.6 us. The results for the possible energy savings compared to the total energy consumption in the datapath, are shown in Table 6.4.

	off during EEG	off during ECG
possible off-time	97.4 %	100 %
total savings	1.79E-10 J	8.55E-10J
total energy consumption	7.53E-7 J	3.06E-6 J
percentage	0.024 %	0.028 %

Table 6.4: Results for the possible absolute energy savings in the biomedical DSP

Chapter 7

Discussion

In this chapter, the results which were presented in the previous chapter, are analysed and discussed.

For this chapter, the naming conventions which were already used in the previous chapters for the individual components of the overhead are used. As a short reminder, they are listed below:

- $P_{mod,leak}$: leakage power of the power domain
- $P_{switch,leak}$: leakage power of the power switch(es)
- $P_{iso,leak}$: leakage power of the isolation cells
- $P_{iso,active}$: power consumption by the isolation cells during active mode
- $P_{add.modules}$: power consumption of additional modules
- $E_{powerdown}$: energy saved during powering down
- $E_{poweron}$: energy required to switch a power domain on

In literature about power gating, it is stated that the overhead is dominated by the energy to switch a power domain on (refer section 1.1). The energy which is consumed by modules that have to be added to the system in order to enable power gating, like isolation cells, are not considered. In this work, however, they are also taken into account for the derivation for the break-even point.

7.1 UWB processor

First, the results obtained from the implementation on the UWB processor are analysed.

7.1.1 HW based power manager

In this section, the results for the hardware based power manager are discussed.

G libraries, WC

The first implementation of the HW based power manager on the UWB processor was performed using general purpose (G) libraries. These libraries are primarly designed for speed and not for a low power consumption. The results of the power consumption break down of the components to the savings and overhead, were shown in Figure 6.1. The power numbers were extracted for the worst case (WC).

For the power domain *PD_mul*, the overhead is mainly caused by the power consumption of the additional modules, in this implementation the power manager ($P_{pwr.man}$) and the control register ($P_{ctrl.reg}$). Also, the power consumption of the isolation cells during active mode and the additional modules affects the overhead. The power consumption of the additional modules, however, has a greater influence on the overhead as the power is consumed constantly during the complete run-time of the application (i.e. they introduce a constant overhead) while the power consumption of the isolation cells is only relevant during active mode. When the power domain is switched off, the isolation cells only leak, which is, as it can be seen in the graph, not a significant factor. Furthermore, it is noticeable that the leakage power of the power domain is smaller than the introduced overhead. The minimum down time, presented in in the first column, first two rows in Table 6.2, also shows that the power domain basically would have to be switched off during almost the complete run time (99 %) in order to save energy.

The power distribution for *PD_vec* is similar to *PD_mul* with the main difference that the minimum down time, shown in the second column, first first block (labelled **90G, WC, HW**), Table 6.2 is above 100 %. That means, that an energy benefit can never be reached as the overhead will always be bigger than the savings.

The power domain *PD_VIS* has a relatively high leakage power consumption, which also leads to a relatively small minimum down time of 12 %, presented in the third column, first block, Table 6.2.

For *PD_VIS*, also a more detailed analysis was performed to further evaluate the break-even point. For that purpose, the minimum down time, i.e. the break-even point, was used to calculate the distribution of the components of the energy savings and the energy overhead according to Equation 3.1 and Equation 3.6, respectively. As mentioned before, the spice simulations to determine $E_{poweron}$ and $E_{powerdown}$ could not be performed as the spice models of the power switches were not available. Therefore, those components are missing in the analysis. The result was presented in Figure 6.8. It is shown that one third of the overhead is caused by the active power of the isolation cells and two third by the additional modules.

LP libraries, WC

As the G libraries are, as mentioned, not optimised for power consumption, another synthesis was performed, this time using low-power (LP) libraries. The power numbers were extracted for the worst case (WC). The power consumption

values are shown in Figure 6.9. It can be noted that the absolute values are lower, as expected. Moreover, the leakage power of the power domain is, in relation to the overhead power, smaller than for the G libraries. This also reflects on the minimum down times, shown in the second block (labelled **90LP, WC, HW**). Only for the power domain *PD_ VIS* could an energy saving be achieved (if it can be switched off for more than 30 % of the time), for the other power domains the overhead always exceeds the savings.

For *PD_ VIS*, also a detailed analysis of the break-even point was performed. The values for the energy consumption were calculated for the case that savings and overhead are equal. Additionally, the spice simulations could be performed as the spice models of the power switches were available for the LP libraries. Therefore, the number of switches, the power-on time and $E_{poweron}$ and $E_{powerdown}$ could be determined. The results for the energy breakdown are shown in Figure 6.10. The overhead is dominated by the power consumed by the additional modules, the major part is caused by the power manager, the control register is a less significant contributor. Also the power consumed by the isolation cells during active mode is a noticeable contributor to the overhead. The energy required to switch on the power domain ($E_{poweron}$) was only responsible for 0.06 % of the overhead, i.e. it is neglegible. This is an interesting observation, $E_{poweron}$ has been stated as the main overhead in previous works on power gating (refer section 1.1).

LP libraries, TC

Leakage power is strongly depending on the temperature, refer subsection 2.1.2. Therefore, the chosen temperature during extraction of power numbers has a strong influence on the break-even point. The temperature for the WC is $125°C$ where it is $25°C$ for TC. According to [29], the percentage of the leakage power consumption compared to the total power consumption increases from 6 % for $30°C$ to 56 % for $110°C$ for 100nm technology. So it is expected that also for the UWB processor the break-even point will change significantly when the analysis is performed for the typical case. Furthermore, a temperature of $25°C$ is a more realistic case for the UWB processor as it is designed to be used within a Body Area Network (refer [61]) which implies an operating temperature which is in the range of the human body temperature.

The results for the power breakdown was presented in Figure 6.3. It is noticeable that the leakage power consumption is much lower than for the WC measurements. Like for the previous cases, the power consumed by the isolation cells has strong influence on the overhead. The minimum down times, presented in the block labelled **90LP, TC, HW** in Table 6.2 are all above 100 %. That means, for none of the power domains is it possible to save energy by applying power gating which is not very surprising as $P_{mod,leak}$ is almost not visible in the graph.

7.1.2 SW based power manager

In this section, the results for the implementation with the software based power manager on the UWB processor are discussed. Besides the different power manager, also the control register was implemented in a different way to make clock gating applicable. Due to those changes in the implementation, the constant part of the energy overhead is expected to be reduced significantly.

G libraries, WC

First, the results for the implemention with the G libraries is discussed. The power breakdown can be found in Figure 6.4. It can be seen that, compared to the HW based implementation, the overhead of the additional modules was reduced significantly leaving the isolation cells as main contributor to the overhead. The minimum down times, shown in the block labelled **90G, WC, SW** in demonstrate that it is possible to save energy for all power domains, *PD_mul* requires a minimum down time of 49 %, *PD_vec* 92 %, which is still quite high and *PD_VIS*, only 5 %. The results for the power consumption are also a clear indicator that the power consumption of the isolation cells during active mode are an important contributor to the overhead.

LP libraries, WC

The results for the implementation with the LP libraries and WC power number extraction are shown in Figure 6.6. It can be seen that the overhead is almost exclusively caused by the isolation cells, only for *PD_VIS* also the power consumption of the control register is visible in the graph. The minimum down times for the power domains, shown in the block labelled **90LP, WC, SW** show that for *PD_mul* a minimum down time of 95 % is required, which means that it has to have a very low duty cycle and that the overhead will always be quite significant. *PD_vec* even requires a down time of 100 %, which means that no energy savings are possible. This also corresponds to the observation that $P_{mod,leak}$ is not visible in the graph. For *PD_VIS*, a minimum downtime of 76 % is required.

LP libraries, TC

The last case for the UWB processor is the most interesting one. As mentioned before, are power numbers which were extracted at TC much more realistic than WC power numbers. The results for the power consumption breakdown are shown in Figure 6.6. It can be seen that the isolation cells are the almost exclusive contributor to the power overhead of all power domains.

The leakage power consumption of the power domain, which could be saved (i.e. $P_{mod,leak}$) is significantly smaller than the power consumption of the isolation cells during active mode ($P_{iso,active}$). For *PD_VIS*, $P_{mod,leak}$ is a factor of 5 smaller than $P_{iso,active}$, for *PD_mul* it is a factor of 150 smaller, and for *PD_vec* it is even a factor of 2300 smaller. Also for this case, $P_{mod,leak}$ is not visible in

the graph. The minimum down times, presented in the block labelled **90LP, TC, SW** demonstrate that only for *PD_VIS* energy savings are possible if *PD_VIS* can be switched off for more than 89 % of the time. For the other power domains, a minimum down time of 100 % is required which means that power gating will never lead to energy savings.

For *PD_VIS*, also a detailed analysis of the break-even point was performed. Like for the previous detailed analysis, the energy consumption values were calculated for the break-even point case, i.e. the power domain was switched off for 89 % of the time. The result is depicted in Figure 6.10. The overhead is dominated by the isolation cells but also the control register has a relevant effect. This, however, shows that the isolation cells play an important role as they have so much influence on the energy overhead even if the power domain is only switched on for 11 % of the time. Furthermore, the energy to switch on a module ($E_{poweron}$) is neglegible compared to the other contributors to the overhead.

Energy overhead caused by isolation cells

A surprising observation for both implementations (HW based and SW based power manager) is the large difference in power consumption of the isolation cells between the power domains. For *PD_vec* it is a factor of in average almost 40 compared to *PD_VIS*, which has 1.5 times so many output signals. This is caused the fact that the isolation block of *PD_vec* are in a critical path in the design, therefore extra buffers were required to meet timing constraints. Therefore, it is not only the isolation cells that consume additional power but also buffers

7.2 Biomedical DSP

In this section, the results for the biomedical DSP are discussed. In this processor, only one power domain is used. This power domain contains the EEG issue slot. As explained before, two different use-cases were analysed in this work. The first scenario was that the power domain is switched off during the EEG algorithm which uses the power domain during a short period. The other scenario that it was switched off completely while an ECG algorithm was executed that did not require the power domain. The processor was implemented using the LP libraries, all results were obtained for TC power number extraction. The results for the power breakdown were shown in Figure 6.7.

The left bar displays the results for the first scenario, i.e. the power domain was switched off during execution of the EEG application. The overhead is clearly dominated by the power consumption of the isolation cells during active mode, the control register does not have an important effect in this design. The minimum down time for that case, presented in the column labelled **off during EEG**, shows that the power domain requires a minimum down time of 95 %. This is not surprising, considering the strong dominance of the power overhead compared to the savings. The break-even point is analysed further, using the

same methodology as for the detailed break-even point analysis for the UWB processor. The results are presented in Figure 6.12. The energy overhead is dominated by the energy which is consumed by the isolation cells during active mode, the control register only has a small influence. Noticeable in this implementation, compared to previous analysis, is that the leakage of the isolation cells has a noticeable influence (2 %) and also the energy to switch the power domain on has a more significant influence (2 %) than for the previous implementations. Still, the factors are marginal compared to the energy consumed by the isolation cells during active mode.

The results for the second scenario, i.e. the power domain is powered down complete, are presented in the right bar. This is an interesting case as it is different from the previously discussed scenarios. As the power domain is not used at all during the application, the isolation cells do not switch, hence they do not consume active power, only leakage power. The minimum down time, shown in the column labelled **off during ECG** is still 16 %.

The detailed analysis was performed for the case that the power domain is switched off constantly. The results were presented in Figure 6.12. The results reveal that the energy overhead is dominated by the energy consumed by the control register, but also $E_{poweron}$ has a noticeable influence of 3 %. As the power domain is switched off completely, no energy overhead is caused by the isolation cells during active mode as they never have to switch. Their leakage energy ($E_{iso,leak}$), however, contributes to the overhead with 3 %.

The results for the achievable total energy savings, presented in Table 6.4, demonstrate that the possible energy savings are only marginal. Even when the power domain can be switched off complete, only savings of 0.0028 % can be reached. For this analysis, the energy savings were only evaluated on the energy consumption of the datapath. However, the datapath itself is only a small consumer of the total energy, the majority of the energy is consumed in the memories. This implies that the energy savings, that could be reached by applying fine-grained power gating on the datapath, are even smaller than the presented numbers. In any case, it is obvious that a energy saving of 0.0028 % is not a significant saving.

7.3 Results from the spice simulations

The results for the spice simulations which determined the number of switches, required switch on time, $E_{poweron}$ and $E_{powerdown}$ were shown in Table 6.1. $E_{poweron}$ and $E_{powerdown}$ were already discussed in the previous sections. The switch-on times demonstrate that each power domain can easily be switched on during one clock period. Both processors were synthesised for 100 MHz (which corresponds to a clock period of 10 ns) and the power-on times are never above 2 ns.

Furthermore, only one power switch is required per power domain. This is surprising on the one hand as in literature it is usually claimed that multiple switches are needed, even hundreds to thousands of switches are mentioned ([65]). On the other hand, the switches that are used are relatively big and are

designed to have a very low voltage drop. Also, the power domains are quite small compared to a power domain which includes a memory, for example.

7.4 Comparison of the detailed versus the simplified formula for the break-even point

In section 3.5, the equation derived for the break-even point (Equation 3.9) was simplified to Equation 3.11 by neglecting the leakage power of the switch ($P_{switch,leak}$), the leakage power of the isolation cells ($P_{iso,leak}$), the energy to switch a power domain on after it has been switched off ($E_{poweron}$) and the energy which is saved during powering down before reaching the lowest energy state ($E_{powerdown}$). In Table 6.2 and Table 6.3, where the break-even points for the UWB processor and the biomedical DSP, respectively, are presented using both the simplified and the detailed formula, it can be seen that the results do not differ significantly. Also, in the graphs where the power breakdown and in the detailed analysis of the energy overhead it became clear that the assumptions that were made during the simplification of the equations were valid.

7.5 General analysis of the overhead

In previous publications on the break-even point of power gating it was concluded, that the overhead is mainly caused by the required energy to switch a power domain on ($E_{poweron}$). Factors like isolation cells or additional modules were not considered. During this work, it became clear that the main part of the overhead was missing in the previous publications as $E_{poweron}$ is marginal compared to the energy consumed by the isolation cells during active mode and the additionally required modules.

The power which is consumed by additional modules is very implementation dependent. The HW based power manager, which was used in the first implementation approach, only contained a simple state machine but still its power consumption was a significant contributor to the overhead. This problem can be solved by using a software based power manager, but that also comes with certain drawbacks. The implementation of the power-gating sequence into the algorithm has to be perfomed with care, furthermore, additional cycles are introduced into the application. Also, this method is error-prone. Therefore, the hardware based power manager could be improved further for lower power consumption. One simple method would be to clock it with a lower frequency. This, however, would lead to a longer power-on time. For example, a processor operating at 100 MHz uses a power manager which is clocked at 1 MHz. When the processor sends a power-on signal for a certain power domain, it takes until the next clock edge of the slower power manager before the power on sequence starts. Depending on the sequence itself (i.e. is state retention required, are only isolation cells and power switches to be controlled?), it takes further slow clock cycles. When power gating is applied on the datapath, these delays are usually not acceptable. Another possibility would be to implement the power manager

asynchronously. This leads to further implementation problems, for example, a certain delay is required for the control signals of the isolation cells and the power-switches. But the implementation of a fixed delay in an asynchronous circuit is not trivial.

The number of isolation cells depend on the number of outputs of a power domain. The more outputs, the more isolation signals. However, this is not the only impact they have on the introduced overhead. It could happen that their delay increases the timing of a path in such a way that the timing constraints cannot be met anymore. Then, the design compiler inserts additional buffers to improve timing. Those additional buffers, however, also consume power. This was the case in the power domain *PD_vec*.

For all analysed cases, except for *PD_VIS* for the implementation with the G libraries for the WC, the required minimum down times were relatively high (between 30 % up to 99 % if the down times above 100 % are ignored as they cannot be reached). That implies that the power domains need a low duty cycle for power gating to become beneficial. In the UWB processor, *PD_mul* and *PD_vec* had a low duty cycle. However, for both power domains the overhead was too high compared to the possible savings in the majority of the analysed cases. Also for the two most realistic cases, i.e. the analysis for the TC, the minimum down time was 100 % or above. *PD_VIS* on the other hand has a very high duty cycle during the application, hence it cannot be switched off for the required time was shown in subsection 5.2.1.

Another important result of this work is that the leakage power for 90nm is still quite low compared to the active power. In the break-even point analysis for power gating, the leakage power of the power domain (which could be saved) is traded off agains active power (which is additionaly consumed). As the active power is much higher than the leakage power, fine-grained power gating can hardly become beneficial. For future technologies, this might look different. It is expected that the leakage power will gain much more influence on the total power consumption than for 90 nm (refer chapter 1).

Chapter 8

Future work

During this work it was demonstrated that power gating in the datapath of a processor does not lead to significant energy savings in the used designs. One reason is that leakage power consumption for the used technologies (90 nm TSMC) is still quite low compared to the total power consumption. Furthermore, the introduced overhead is relatively high compared to the possible savings. Moreover, the main source of power consumption is usually not the datapath but the memories, I/O pads and the clock-tree.

Therefore, the presented break-even point analysis could be used to evaluate the energy savings when power gating is applied on other parts of the processor as it is expected that greater energy savings can be obtained.

A significant overhead was caused by the power manager. Therefore, the power manager could be improved. For example, an asynchronous implementation could be designed including an voltage-sensing circuit which detects when a power domain is fully powered on.

The isolation cells were a significant contributor to the overhead due to their dynamic power consumption during the active mode of the power domain. Therefore, an interesting approach on circuit level would be to design more power-efficient isolation cells.

The developed scripts require manual control. Automating the implemetation and the estimation of power savings could minimise the burden for the designer.

Although beyond the work of this thesis, a power aware compiler could increase the energy efficiency significantly. The discussed previous work on compiler techniques could be combined to design a compiler which takes the possibilities of power gating into account already during the scheduling phase and inserts automatically power gating instruction into the assembly code.

Chapter 9

Conclusion

In this work, it was studied whether applying fine-grained power gating on the datapath of a processor can lead to energy savings. A detailed analysis of the possible energy savings as well as the introduced energy overhead was performed. Based on the findings, an analytical equation of the break-even point of power gating was derived. A novel work-flow to implement and evaluate power gating was developed and applied to two different processors.

Analysis of the introduced overhead of power gating revealed that the main contributor is not the energy to switch a power domain on, as it was concluded in previous publications. Instead, the overhead is dominated by power consumed by additional modules, like the isolation cells or a power manager. The obtained results also demonstrated that the power domains require a low duty cycle. The determined minimum switch-off time for the different power domains for the most representative cases was in the range of 90 % (for the cases where the minimum down time did not exceed 100 %).

An important factor that made power gating at best marginally beneficial was the fact that leakage power is still very low when compared to the total power consumption for 90 nm technologies. In the evaluation of the break-even point, the possible savings, i.e. the leakage power of the power domain, were compared with the overhead, i.e. the dynamic power of isolation cells and additional modules. This combination makes power gating just marginally beneficial.

It was shown that the break-even point is influenced by many different factors. The temperature has a strong influence on the leakage power, hence on the possible savings. The implementation of the additional modules also has a significant impact on the overhead. Also the number of necessary isolation cells and whether they are placed in a critical path in the design, thus possibly leading to the need of additional buffers, has a large impact on the overhead.

For future research, it would be interesting to use sub-90 nm libraries, as it is expected that leakage power will gain a much greater influence on the total power consumption due to downscaling of technology. Also, the design of isolation cells could be improved for active power consumption. The power manager could be implemented in a more power efficient way.

Bibliography

[1] R. Schmidt, T. Norgall, J. Moersdorf, J. Bernhard, and T. von der Gruen, "Body Area Network BAN–a key infrastructure element for patient-centered medical applications." *Biomedizinische Technik. Biomedical engineering*, vol. 47, p. 365, 2002.

[2] B. Gyselinckx, C. Van Hoof, J. Ryckaert, R. Yazicioglu, P. Fiorini, and V. Leonov, "Human++: autonomous wireless sensors for body area networks," in *Custom Integrated Circuits Conference, 2005. Proceedings of the IEEE 2005*, 2005, pp. 13–19.

[3] J. Paradiso and T. Starner, "Energy scavenging for mobile and wireless electronics," *IEEE Pervasive Computing*, pp. 18–27, 2005.

[4] H. Jiang, M. Marek-Sadowska, and S. Nassif, "Benefits and costs of power-gating technique," in *2005 IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings*, 2005, pp. 559–566.

[5] G. Panic, Z. Stamenkovic, and R. Kraemer, "Power gating in wireless sensor networks," in *Wireless Pervasive Computing, 2008. ISWPC 2008. 3rd International Symposium on*, 2008, pp. 499–503.

[6] M. Keating, *Low Power Methodology Manual for System-on-chip Design*. Springer, 2007.

[7] J. Frenkil and S. Venkatraman, "Power Gating Design Automation," *chapter in Closing the Power Gap between ASIC and Custom, Springer*, 2007.

[8] N. Pantazis and D. Vergados, "A survey on power control issues in wireless sensor networks," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 4, pp. 86–107, 2007.

[9] M. Seok, S. Hanson, Y. Lin, Z. Foo, D. Kim, Y. Lee, N. Liu, D. Sylvester, and D. Blaauw, "The Phoenix Processor: A 30pW platform for sensor applications," in *2008 IEEE Symposium on VLSI Circuits*, 2008, pp. 188–189.

[10] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose, "Microarchitectural techniques for power gating of execution units," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2004, pp. 32–37.

[11] K. Usami and N. Ohkubo, "A design approach for fine-grained run-time power gating using locally extracted sleep signals," in *Computer Design, 2006. ICCD 2006. International Conference on*, 2007, pp. 155–161.

[12] A. Sathanur, A. Calimera, A. Pullini, L. Benini, A. Macii, E. Macii, and M. Poncino, "On quantifying the figures of merit of power-gating for leakage power minimization in nanometer CMOS circuits," in *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008*, 2008, pp. 2761–2764.

[13] http://www.si2.org/?page=811.

[14] http://www.cadence.com/.

[15] http://www.synopsys.com/.

[16] D. Helms, E. Schmidt, and W. Nebel, "Leakage in CMOS circuits-An introduction," *Lecture notes in computer science*, pp. 17–35, 2004.

[17] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, 1992.

[18] H. Veendrick, "Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits," *IEEE Journal of Solid-State Circuits*, vol. 19, no. 4, pp. 468–473, 1984.

[19] A. Chandrakasan and R. Brodersen, *Low power digital CMOS design*. Kluwer Academic Pub, 1995.

[20] D. Zhurikhin, A. Belevantsev, A. Avetisyan, K. Batuzov, and S. Lee, "Evaluating power-aware optimizations within GCC compiler."

[21] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proceedings of the 1998 international symposium on Low power electronics and design*. ACM New York, NY, USA, 1998, pp. 197–202.

[22] D. Mosse, H. Aydin, B. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," in *Workshop on Compilers and Operating Systems for Low-Power (COLP'01900)*, 2000.

[23] F. Fallah and M. Pedram, "Circuit and system level power management," *Kluwer Academic Pub*, pp. 373–412, 2002.

[24] M. Muench, B. Wurth, R. Mehra, J. Sproch, and N. Wehn, "Automating RT-level operand isolation to minimize power consumption in datapaths," in *Proceedings of the conference on Design, automation and test in Europe*. ACM New York, NY, USA, 2000, pp. 624–633.

[25] J. Oh and M. Pedram, "Gated clock routing minimizing the switched capacitance," in *Design, Automation and Test in Europe, 1998., Proceedings*, 1998, pp. 692–697.

[26] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proceedings of the IEEE*, vol. 83, no. 4, pp. 498–523, 1995.

[27] S. Devadas and S. Malik, "A survey of optimization techniques targeting low power VLSI circuits," in *Proceedings of the 32nd ACM/IEEE conference on Design automation*. ACM New York, NY, USA, 1995, pp. 242–247.

[28] R. Mehta, R. Owens, M. Irwin, R. Chen, D. Ghosh, E. Technol, and C. Campbell, "Techniques for low energy software," in *Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on*, 1997, pp. 72–75.

[29] F. Fallah and M. Pedram, "Standby and active leakage current control and minimization in CMOS VLSI circuits," *IEICE Transactions on Electronics*, vol. 88, pp. 509–519, 2005.

[30] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," *Proceedings of the IEEE*, vol. 91, no. 2, pp. 305–327, 2003.

[31] W. Elgharbawy and M. Bayoumi, "Leakage sources and possible solutions in nanometer CMOS technologies," *IEEE Circuits and Systems Magazine*, vol. 5, no. 4, pp. 6–17, 2005.

[32] A. Keshavarzi, K. Roy, and C. Hawkins, "Intrinsic leakage in low-power deep submicron CMOS ICs," in *International Test Conference*, 1997, pp. 146–155.

[33] T. Kuroda, T. Fujita, S. Mita, T. Nagamatsu, S. Yoshioka, K. Suzuki, F. Sano, M. Norishima, M. Murota, M. Kako *et al.*, "A 0.9-V, 150-MHz, 10-mW, 4 mm 2, 2-D discrete cosinetransform core processor with variable threshold-voltage (VT) scheme," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp. 1770–1779, 1996.

[34] M. Johnson, D. Somasekhar, and K. Roy, "Models and algorithms for bounds on leakage in CMOS circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 6, pp. 714–725, 1999.

[35] A. Abdollahi, F. Fallah, and M. Pedram, "Runtime mechanisms for leakage current reduction in CMOS VLSI circuits," *Low Power Electronics and Design*, pp. 213–218, 2002.

[36] B. Krishnamachari, *Networking Wireless Sensors*. Cambridge University Press New York, NY, USA, 2005.

[37] A. Calimera, L. Benini, and E. Macii, "Optimal MTCMOS reactivation under power supply noise and performance constraints," in *Proceedings of the conference on Design, automation and test in Europe*. ACM New York, NY, USA, 2008, pp. 973–978.

[38] R. Vilangudipitchai and P. Balsara, "Power switch network design for MTCMOS," in *VLSI Design, 2005. 18th International Conference on*, 2005, pp. 836–839.

[39] C. Long and L. He, "Distributed sleep transistors network for power reduction," in *Design Automation Conference, 2003. Proceedings*, 2003, pp. 181–186.

[40] J. Kao, S. Narendra, and A. Chandrakasan, "MTCMOS hierarchical sizing based on mutual exclusive discharge patterns," in *Proceedings of the 35th annual conference on Design automation*. ACM New York, NY, USA, 1998, pp. 495–500.

[41] S. Henzler, G. Georgakos, M. Eireiner, T. Nirschl, C. Pacha, J. Berthold, D. Schmitt-Landsiedel, I. AG, and G. Munich-Neubiberg, "Dynamic state-retention flip-flop for fine-grained power gating with small design and power overhead," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 7, pp. 1654–1661, 2006.

[42] Y. You, C. Huang, J. Lee, C. Wang, and K. Chuang, "Power-gating instruction scheduling for power leakage reduction," Jul. 27 2006, uS Patent App. 11/493,765.

[43] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. Irwin, D. Duarte, and Y. Tsai, "Exploiting VLIW schedule slacks for dynamic and leakage energy reduction," in *Microarchitecture, 2001. MICRO-34. Proceedings. 34th ACM/IEEE International Symposium on*, 2001, pp. 102–113.

[44] S. Rele, S. Pande, S. Onder, and R. Gupta, "Optimizing static power dissipation by functional units in superscalar processors," *Lecture Notes in Computer Science*, pp. 261–275, 2002.

[45] Y. You, C. Lee, and J. Lee, "Compilers for leakage power reduction," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, no. 1, pp. 147–164, 2006.

[46] K. Vardhan and Y. Srikant, "Transition aware scheduling: increasing continuous idle-periods in resource units," in *Proceedings of the 2nd conference on Computing frontiers*. ACM New York, NY, USA, 2005, pp. 189–198.

[47] R. Nagpal and Y. Srikant, "Compiler-assisted leakage energy optimization for clustered VLIW architectures," in *Proceedings of the 6th ACM & IEEE International conference on Embedded software*. ACM New York, NY, USA, 2006, pp. 233–241.

[48] M. Wang, Z. Shao, C. Xue, and E. Sha, "Real-Time Loop Scheduling with Leakage Energy Minimization for Embedded VLIW DSP Processors," in *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007*, 2007, pp. 12–19.

[49] H. Kim, N. Vijaykrishnan, M. Kandemir, and M. Irwin, "Adapting instruction level parallelism for optimizing leakage in VLIW architectures," in *Proceedings of the 2003 ACM SIGPLAN conference on Language, compiler, and tool for embedded systems*. ACM New York, NY, USA, 2003, pp. 275–283.

[50] http://www.retarget.com/.

[51] P. Mishra and N. Dutt, *Processor Description Languages*. Morgan Kaufmann, 2008.

[52] H. Maréchal, "ASIP design methodology with Target2019s Chess/Checkers retargetable tools," in *Proc. Intl. Signal Processing Conf., Santa Clara*, 2006.

[53] http://www.siliconhive.com/.

[54] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal, "An fpga design flow for reconfigurable network-based multi-processor systems on chip," in *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, 2007, pp. 1–6.

[55] http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx.

[56] http://www.hspice.com.

[57] http://www.mathworks.com/.

[58] http://www.eda.org/sdf/.

[59] http://www.si2.org/.

[60] Xiongfei Meng, "Power Distribution and Decap Design," Dept. of ECE. University of British Columbia, Lecture notes.

[61] C. Bachmann, A. Genser, J. Hulzink, M. Berekovic, and C. Steger, "A Low-Power ASIP for IEEE 802.15.4a Ultra-Wideband," in *Design, Automation and Test in Europe (DATE) 2009 Proceedings*, 2009.

[62] S. Cirit, J. Penders, M. Ashouei, J. Hulzink, M. de Nil, J. Huisken, and U. Hofmann, "Development of Real-Time EEG Application on an Ultra-Low- Power DSP," in *Proceedings of 7th ODES Workshop on Optimizations for DSP and Embedded Systems*, 2009.

[63] J. Martinez, R. Almeida, S. Olmos, A. Rocha, and P. Laguna, "A wavelet-based ECG delineator: evaluation on standard databases," *IEEE Transactions on Biomedical Engineering*, vol. 51, no. 4, pp. 570–581, 2004.

[64] http://www.tsmc.com/download/english/a05_literature/90nm_Brochure.pdf.

[65] K. Shi and J. Li, "A wakeup rush current and charge-up time analysis method for programmable power-gating designs," in *2007 IEEE International SOC Conference*, 2007, pp. 163–165.