TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JOONAS MULTANEN

HARDWARE OPTIMIZATIONS FOR LOW-POWER PROCESSORS

Master of Science thesis

Examiner: D.Sc. Pekka Jääskeläinen
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 14th May 2015

ABSTRACT

JOONAS MULTANEN: Hardware Optimizations for
Low-Power Processors
Tampere University of Technology
Master of Science thesis, 51 pages
May 2015
Master's Degree Programme in Electrical Engineering
Major: Programmable devices and platforms
Examiner: D.Sc. Pekka Jääskeläinen
Keywords: power optimizations, transport triggered architecture

Power consumption is a key aspect in contemporary processor design. Small processor systems, such as mobile devices, benefit from power optimization and smart design in the form of increased battery life. Due to their small size, they often have tight thermal constraints, which optimizing for power helps to meet. Power optimization can be done on all design abstraction levels. On the architecture level, choosing an optimal architecture for low power may not be straightforward. *Transport Triggered Architecture (TTA)* processors utilize *Instruction Level Parallelism (ILP)* efficiently and are by nature a good choice for low power designs. They enable the designer to implement various power and performance optimizations, some of them unique to TTAs. In this thesis, a literary review of the most commonly used power and energy optimization methods was carried out. Next, four power optimizations on the *Register Transfer Level (RTL)* were implemented to *TTA-based Co-design Environment (TCE)*, a toolset for designing and programming TTA processors developed at *Tampere University of Technology (TUT)*. Synthesis was performed with Synopsys Design Compiler to analyze the results. The effect of the synthesis tool's automatic optimizations was also analyzed. The optimizations were synthesized on three cores designed for different use cases. All of the cores benefited from the optimizations, yielding up to 26% decrease in power consumption with 3% area overhead.

TIIVISTELMÄ

JOONAS MULTANEN: Laitteisto-optimoinnit matalan tehonkulutuksen prosessoreille
Tampereen teknillinen yliopisto
Diplomityö, 51 sivua
Toukokuu 2015
Sähkötekniikan koulutusohjelma
Pääaine: Ohjelmoitavat alustat ja laitteet
Tarkastajat: TkT Pekka Jääskeläinen
Avainsanat: tehonkulutusoptimoinnit, siirtoliipaisuarkkitehtuuri

Modernien prosessorien suunnittelussa tehonkulutuksen huomioonottaminen on tärkeää. Pienet prosessorijärjestelmät, kuten mobiililaitteet, hyötyvät tehonkulutusoptimoinnista pidemmän akunkeston muodossa. Optimointi auttaa myös vastaamaan mobiililaitteiden usein tiukkoihin lämpösuunnittelurajoituksiin. Tehonkulutusoptimointeja voidaan tehdä kaikilla suunnittelun abstraktiotasoilla. Arkkitehtuuritasolla, optimaalisen arkkitehtuurin valitseminen ei usein ole suoraviivaista. *Siirtoliipaisuarkkitehtuuria* käyttävät prosessorit hyödyntävät *käskytason rinnakkaisuutta* tehokkaasti ja ovat hyvä valinta matalan tehonkulutuksen sovelluksiin. Niitä käyttäen prosessorisuunnittelijan on mahdollista toteuttaa erilaisia tehonkulutus- ja suorituskykyoptimointeja, joista osa on siirtoliipaisuarkkitehtuurille yksilöllisiä. Tässä diplomityössä tehtiin ensin kirjallisuuskatsaus yleisimmin käytetyistä tehonkulutus- ja suorituskykyoptimoinneista. Näistä neljä toteutettiin Tampereen Teknillisessä Yliopistossa kehitettyyn *TTA-based Co-design Environment (TCE)* -kehitysympäristöön, joka mahdollistaa TTA-prosessorien suunnittelun ja ohjelmoinnin. Vaikutusten analysoimiseksi optimoinnit toteutetiin kolmeen eri tarkoitusta varten suunniteltuun prosessoriytimeen, jotka syntesoitiin Synopsys Design Compilerilla. Kaikki ytimet hyötyivät optimoinneista, saavuttaen parhaassa tapauksessa 26% tehonkulutuspienennyksen, pinta-alan kasvaessa 3%.

PREFACE

This MSc thesis was completed in the Department of Pervasive Computing at Tampere University of Technology in 2014-2015.

I would like to thank Prof. Jarmo Takala for the chance to work on the project for this thesis. I would also like to thank my advisor Pekka Jääskeläinen, D.Sc., for his guidance, advice and insightful ideas on the thesis.

I would also like to thank my coworkers in the Customized Parallel Computing group for the motivated atmosphere and all the help and advice I have received while working there. Especially Timo Viitanen, Henry Linjamäki and Heikki Kultala for their advice and expertise on software tools and hardware.

Finally, I want to thank my family and friends for their support and always helping me remember that there is also life outside of studies.

Tampere, July 13, 2015

Joonas Multanen

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CMOS	Complementary MOS
CU	Control Unit
FDSOI	Fully Depleted Silicon On Insulator
FPU	Floating Point Unit
FU	Function Unit
IC	Interconnection
ICG	Integrated Clock Gating
IU	Immediate Unit
LSU	Load-Store Unit
MCU	Micro Controller Unit
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MSB	Most Significant Bit
NMOS	N-channel MOSFET
NOP	No Operation
PC	Program Counter
PMOS	P-channel MOSFET
RF	Register File
RTL	Register Transfer Level
SAIF	Switching Activity Interchange Format
SDR	Software Defined Radio
SIMD	Single Instruction Multiple Data
SRAM	Static Random Access Memory
SVTL	Semi Virtual-Time Latching
TCE	TTA-based Co-design Environment
TTA	Transport Triggered Architecture
TUT	Tampere University of Technology
VCD	Value Change Dump
VHDL	Very high speed integrated circuit Hardware Description Language
VLIW	Very Long Instruction Word

C	capacitance
f	frequency
P	power
V	voltage
α	activity

1. INTRODUCTION

Power consumption in today's integrated circuits and especially processors is a key aspect when considering design constraints and parameters. The power density of modern processor cores is often compared to be similar to the one of a rocket combustion engine's. In mobile devices, space for cooling and battery capacity are limited and power consumption directly affects the thermal constraints and battery lifetime. In large scale data facilities such as server centers, cooling costs directly affect the energy bill. Larger and larger amounts of data are required to be processed, while the time-to-market of processor devices needs to be kept short. This creates pressure for logic reusability and scalability for different applications.

Transport Triggered Architecture (TTA) [5] processors are a simple and power-efficient type of statically scheduled processors, utilizing a high level of *Instruction Level Parallelism (ILP)*. Their aim is to minimize the amount of control hardware, leading to low power consumption. Their modular structure and scalability make them a good choice for low power applications. They also offer room for various power optimizations and even enable the use of some TTA-specific optimizations.

Synthesis tools automatically perform power and area optimizations on the logic and transistor levels. However, for minimal power consumption, the choices of the processor designer have a major impact. It is up to the designer, for example, what type and number of logic blocks or buses are used in the design, or what kind of instruction and data encoding is used. The effect of combining these choices on the power consumption is not always clear and may not always be easy to estimate beforehand. In this thesis, power consumption optimizations were reviewed from literature and implemented on TTA cores. The optimizations were integrated to the *TTA-based Codesign Environment (TCE)*, a software toolset developed at *Tampere University of Technology (TUT)* to design and program TTA processors, and are now automatically included in the generated processors.

The structure of this thesis is as follows. Chapter 2 first describes the components and fundamentals of power consumption in processors and in integrated circuits in general. Next, different processor types are explained. TTAs are presented as a subtype of statically scheduled processors. Chapter 3 provides a review of common techniques and methods on different design abstraction levels for reducing design power consumption. Chapter 4 introduces the benchmark designs, programs and synthesis tools used in the thesis. Initial power and area measurements are presented, followed by measurements for automatic synthesis tool options. Chapter 5 describes the power optimizations implemented in this thesis and presents the results in terms of area usage, power consumption and program cycle count. Chapter 6 concludes the thesis.

2. POWER CONSUMPTION IN PROCESSORS

In order to understand how the power consumption of an integrated ircuit can be lowered, it is important to know how the power is dissipated. Modern integrated circuits consists of *Complementary Metal Oxide Semiconductor (CMOS)* technology, which consists of *N-channel MOS (NMOS)* and *P-channel MOS (PMOS)* transistors. The power consumed on an integrated circuit depends on the electrical characteristics of these transistors.

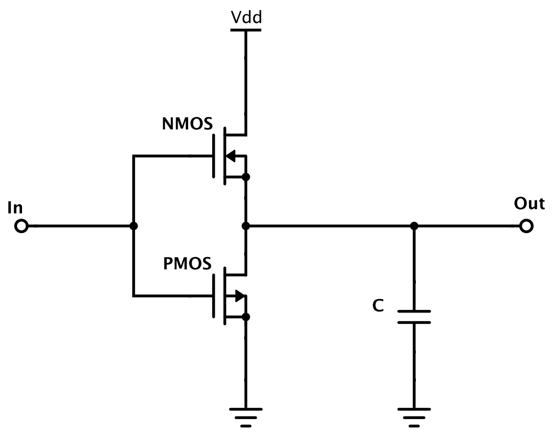


Figure 2.1 *CMOS inverter. The voltage level at port 'In' determines the voltage seen at load capacitance C (port 'Out').*

To explain CMOS operation by means of a simple example, a CMOS *inverter* gate is presented in Figure 2.1. An inverter is a simple gate, which implements the logic *not* operation. When a voltage representing logic '1' is driven to port 'In', logic '0' is seen at the 'Out' port. The inverter consists of an NMOS and a PMOS transistor. In this inverter, like in all CMOS gates, when the NMOS is conducting, the PMOS is not and vice versa. Hence the name *complementary.*

This chapter first describes different categories of power consumption in integrated circuits and processors implemented with CMOS technology. Next, different processor types and are presented, along with an explanation of TTAs. Then, energy consumption and parasitic effects in power consumption are described.

2.1 Components of Power Consumption

Power dissipation in integrated circuits is usually divided into two [16, p. 30] categories: static and dynamic power consumption. Dynamic power is further divided into two categories: short circuit power and switching power. Static power can be divided into two categories based on where in a CMOS the power dissipation occurs: subthreshold leakage and gate leakage.

2.1.1 Dynamic Power

A CMOS gate changes its state by charging its load to the level of supply voltage, or to ground. The amount of power consumed is proportional to the supply voltage, frequency and the load capacitance.

Switching Power

Transistors can be modeled as switches having two states: open when not conducting and closed when not conducting.

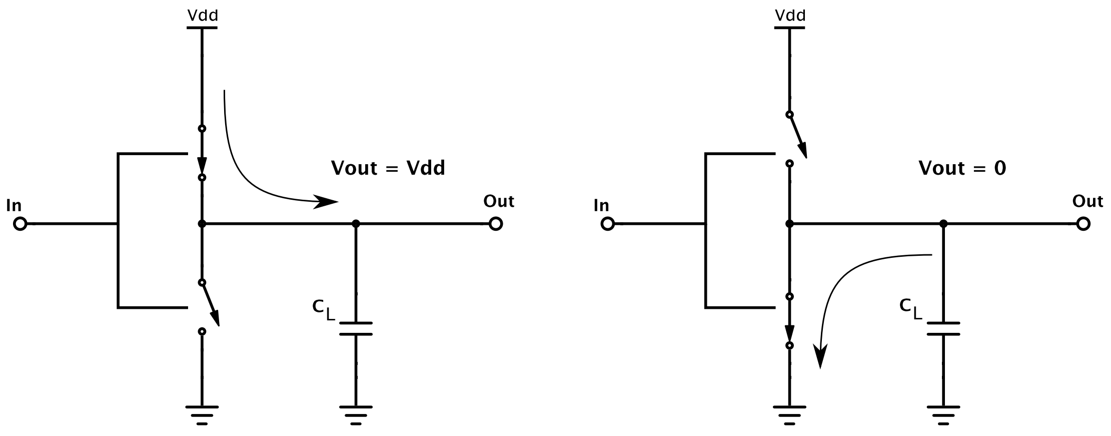


Figure 2.2 *CMOS inverter switching states.*

In Figure 2.2 on the left side the NMOS is closed and PMOS is open, allowing electrical current (direction shown by arrow) to charge the load capacitance C. This means a logic '1' at the output. When the CMOS changes its output from logic '1' to '0', the NMOS is opened and PMOS closed, allowing the discharging of the load capacitance to ground level.

Power consumed in an integrated circuit due to *switching activity* can be written as [16, p. 215]:

$$P_{dyn} = C_L V_{dd}^2 f_{0\to1} \tag{2.1}$$

where transitions from logic '0' to logic '1' are considered, since electrical energy is drawn from the power source at this time. Equation 2.1 only gives power consumption for a CMOS that changes its output state every clock cycle. Usually this is true for only the clock signal, since other signals usually have conditions for changing. If the rate of state changes of the CMOS is considered, we get

$$P_{dyn} = \alpha C_L V_{dd}^2 f_{0\to1}, \tag{2.2}$$

where α is the *activity* of the CMOS. Activity can have a value between 0 and 1, where 0 means the state is never changed and 1 means that the state changes on every clock cycle.

Short Circuit Power

When the state of a CMOS gate changes, both the NMOS and PMOS are briefly in a conductive state at the same time, because real life transistors have finite rise and fall times. This leads to a direct path for electrical current to flow from the supply voltage to ground level. This is presented in Figure 2.3, where the curved arrow shows the direction of the short circuit current.

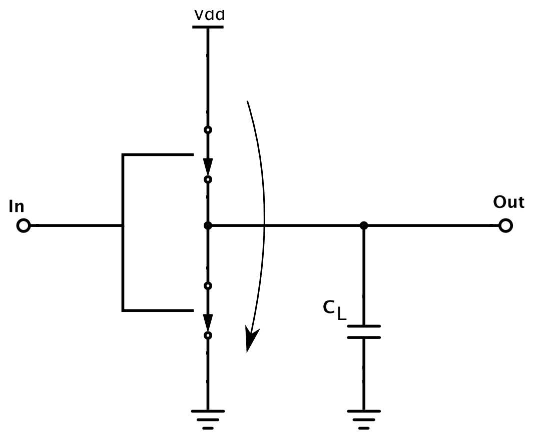


Figure 2.3 *A CMOS inverter in the middle of transition between states.*

The current consumed by the CMOS during this time is called short circuit power, or direct path power and can be written as [16, p. 220]:

$$P_{dp} = t_{sc}V_{dd}I_{peak}f = C_{sc}V_{dd}^2 f \tag{2.3}$$

According to equation 2.3, power consumed during the short circuit period is directly proportional to the time of the short circuit, supply voltage, and the peak current, which is determined by the saturation current of the transistor [16, p. 221].

2.1.2 Static Power

Transistors can be thought of as switches, but being semiconductors, they are not ideal. Due to their electrical qualities, static power is still consumed when they are not switching states. This is due to *leakage currents.* Leakage power consumption can be divided into *subthreshold leakage* and *gate leakage*, and the total power consumption is [16, p. 223]:

$$P_{stat} = I_{stat}V_{dd} \tag{2.4}$$

where I_{stat} is the total current between supply voltage rail and ground voltage rail when the transistor is in idle state, and V_{dd} is the supply voltage.

Subthreshold Leakage

As transistor technology size scales down, channel lengths are reduced, forcing their threshold voltages to be lowered. This increases the subthreshold leakage, since the transistor channel is always somewhat conducting.

Subthreshold leakage occurs below the transistor threshold voltage, that is, when the transistor is in idle state. Electrons drift between the drain to source terminals, forming a leakage current.

Gate Leakage

The gate is separated from the source and drain terminals by gate oxide. As transistor sizes get smaller, the gate oxide thickness must be reduced to maintain performance. This increases gate leakage due to electron tunneling effect.

Gate leakage was significantly lowered starting from 45 nanometer technology [13].

This is due to development of insulators with a higher dielectric constant, known as *high-k dielectrics.* Materials with a high dielectric constant reduce the effect of tunneling and therefore leakage. Currently, the next transistor technologies for low-power applications seem to be *FDSOI* [24] and *FinFET* [15].

2.1.3 Capacitive Effects

In digital logic, *fan-out* is the number of gate inputs that a logic gate output is connected to. Each connection adds to the total capacitance seen by the logic gate and, therefore, increases the power consumption and delay of the gate. *Fan-in* is the number of inputs going to a logic gate, and its effect on the *intrinsic capacitance* of the logic is quadratic, due to the additional transistors needed to implement the gate. Gates with a fan-in greater or equal to four have an excessive delay [16, p. 249] and should be avoided.

In addition to load capacitance, *coupling capacitance* (*crosstalk*) is present in integrated circuits [16, p. 447]. If two wires are placed in parallel close to each other, and only one of them changes its value, the other wire will see the state change as noise. The power consumed in the two wires is more than it would take to charge a single wire because of the capacitive coupling.

2.1.4 Energy Consumption

Energy consumed for a given period of time in a system is the time multiplied by the average power consumption. Momentary power consumption is important when considering safe operating temperatures for an integrated circuit. The energy consumption becomes important when the chip is a part of a battery-powered system. The less energy the chip consumes, the longer the battery will last, or the smaller the energy bill will be.

Optimizing for low power consumption only, disregarding the impact on performance, can lead to increased total energy consumption. For example, an architecture with a small number of registers can consume less power than an architecture with a larger number of registers. However, if the number of registers is too small for typical programs run on the processor, the execution time of the program in clock cycles can increase. This is due to the increased amount of memory writes (*spilling*) and reads that are necessary to store the data that would have fitted to the processor with more registers. The total energy used by a program may increase if the increase in the program cycle count is relatively larger than the decrease in average power.

2.2 Processor Types and Power Consumption

Modern *multiple-issue processors* exploit *Instruction Level Parallelism (ILP).* Multiple-issue processors can be divided into two [9, p. 182] categories: *superscalar* and *Very Long Instruction Word (VLIW)* processors. Depending on how the instructions are scheduled, they can be divided into *statically scheduled* and *dynamically scheduled* processors. This section first provides an overview of the building blocks found in these two processor types. Next, dynamic and static scheduling are presented. Then, *Transport Triggered Architecture (TTA)* processors are presented.

2.2.1 Common Components

At least to some extent, processors usually utilize similar building blocks. Here, the basic components used in processors are presented.

Register Files

In order to avoid often expensive memory stores and loads, *Register Files (RFs)* can be used to temporarily store data. They are a collection of registers inside a processor core, that are fast to access. Register files can have multiple input and output ports that allow simultaneous access to the RF. At hardware level, register files consist mostly of flip-flops storing the data. For low-power applications, register file size should be carefully considered, as mentioned in Subsection 2.1.4

Function Units

The actual operations on data in processors are performed by *Function Units (FUs)*, that can often perform multiple operations. Typical FUs in processors are *Arithmetic Logic Units (ALUs)*, *Load-Store Units (LSUs)* and *Floating Point Units (FPUs).*

For low power consumption, the algorithms for FU operations should be efficient. Often, there is a trade-off between power consumption and silicon area for different algorithms.

Control Unit

The *control unit (CU)* is responsible for fetching instructions to the processor core and decoding them into signals to control other parts of the core. The CU keeps track of the *Program Counter (PC)* and handles program control operations such as *branching*. Control unit design can have a large effect on core power consumption, since it is continuously active when running a program.

Buses

Data and instructions in processors are transported on *buses*. Bus design has an impact on both the performance and the power consumption in a processor. For example, data can be transported with a *serial* or a *parallel* protocol. For a given frequency, a parallel bus allows more data to be transported per time unit. However, a parallel bus requires more wires and can consume unnecessary power if a serial or a smaller bus would satisfy the transport need.

2.2.2 Dynamically Scheduled Processors

Scheduling a program means mapping the program operations onto the hardware resources of the target machine and deciding in which order they are executed. Dependencies between instructions affect the freedom of scheduling, when an instruction uses the result of another instruction. A program can be scheduled differently to different processors, since the scheduling relies on the available hardware resources on each processor. It is not uncommon for an instruction to take multiple cycles to finish, leaving clock cycles for instructions not dependent on it to be executed meanwhile. Scheduling a program efficiently usually leads to reduced runtime, since the number of idle cycles is reduced. This may increase power consumption momentarily, but is likely to lead to a reduced energy cost.

In *dynamic scheduling* [9, p. 181] instructions are scheduled runtime, in the hardware. If an instruction stalls the pipeline in a processor, all instructions after it are stalled. For example, reading data from data memory can stall the pipeline. Processors often implement more than one level of memory hierarchy and a cache miss can stall the processor for a longer period of time compared to a cache hit. However, if the next instructions do not depend on the stalling instruction, they could be executed. Dynamically scheduled processors address this with *out-of-order* execution. where the instructions are executed in a different order then they are fetched in. This way, some dependencies unknown at the compile time can can be handled efficiently. Dynamic scheduling allows good portability of code, since a program compiled for one pipeline can efficiently run on another pipeline.

Dynamic scheduling requires control logic, such as an instruction buffer after the fetch/decode units which, consisting mostly of registers to hold instruction values, consumes a large amount of power. If a stall is present in the program code, the control logic tries to avoid it by reorganizing the instructions, possibly leading to a decrease in overall cycle count.

A superscalar processor can be dynamically or statically scheduled. Statical scheduling and its differences to dynamic scheduling are presented next.

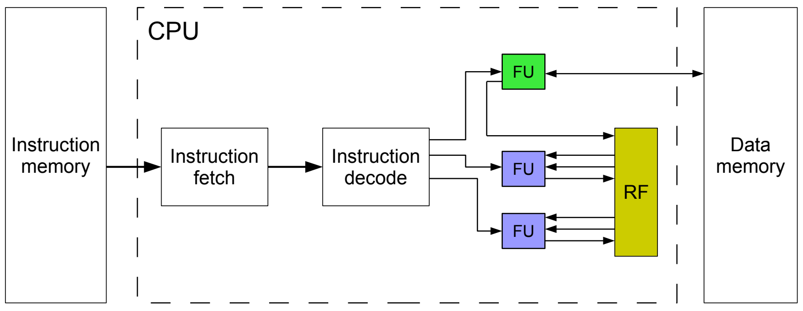
2.2.3 Statically Scheduled Processors

Static scheduling [9, p. 182] is done during compile-time of the program. Here the responsibility of scheduling is given to the compiler, removing control logic from the hardware and adding complexity to the compiler. However, all dependencies cannot be determined during compile-time, for example if they involve a complex dynamic memory reference. In terms of performance, dynamically scheduled processors outperform statically scheduled processors when running general-purpose code. In terms of power consumption, however, statically scheduled processors are better due to the lack of expensive branch control logic and hardware needed for out-of-order execution.

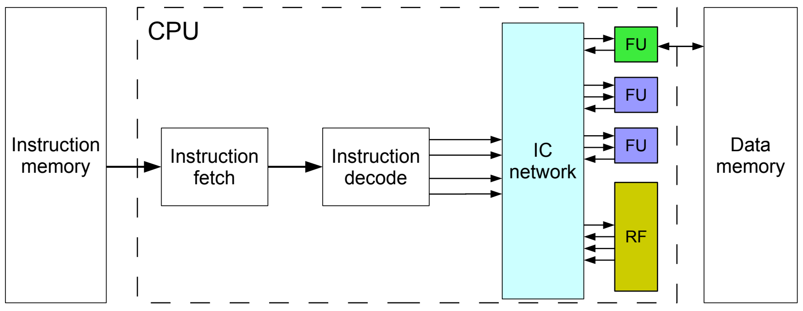
A traditional example of a statically scheduled processor is VLIW. In VLIW processors, it is up to the compiler to arrange the instructions so that there are no hazards between them. The structure of a VLIW is presented in Figure 2.4(a). A VLIW issues a fixed (usually large) number of instructions every clock cycle, even if some of the instructions are *No Operations (NOPs)*.

VLIWs usually implement a *bypassing network* [5, p. 86] to *forward* values between FUs, when the result of an FU is needed by another FU. This reduces the program cycle count because otherwise the intermediate result would need to be stored to the RF and read from there to the next consumer. A challenge in scalability of VLIWs is, that the bypass network grows quadratically [5, p. 92], when adding FUs to the processors. This affects especially the area and power of large processors.

VLIWs present a good choice for implementing low-power applications. However, they leave room for improvement. TTAs try to address some problems in VLIWs and they are presented next.



(a) VLIW



(b) TTA

Figure 2.4 *VLIW and TTA comparison.*

Transport Triggered Architecture

In 1976, a control processor architecture using a "MOVE architecture" was presented by Lipovski [11]. Here operations were triggered by moving operands to inputs of FUs, making the operations *transport triggered* in contrast to the traditional design paradigm, *operation triggered* execution. Corporaal [6] later studied TTAs extensively and proposed them as an improvement to VLIWs. A software environment for automatic generation of TTAs, *TTA-Based Co-design Environment (TCE)* [21], is originally based on the Delft University of Technology MOVE project [20] and can be used to design and program TTA-based processors. TCE development was started in 2002 at *Tampere University of Technology (TUT).*

TTAs, like VLIWs, are a subclass of statically scheduled processors. However, TTAs have an *exposed datapath*, where the *Interconnection (IC)* network can be controlled by the programmer. Comparison to VLIW is presented in Figure 2.4. TTAs attempt to address some issues in VLIWs, such as the scaling bottleneck. This means, that when a VLIW is scaled up in size, RF complexity increases by requiring additional RF ports when adding FUs. In TTAs the number of RF ports required to keep the FUs busy does not depend directly on the number of FUs. TTAs also utilize *software bypassing*, where bypassing is controlled by the compiler, in contrast to the bypass network in VLIWs. Software bypassing can allow reduced IC connectivity, since the explicit programming model allows more precise control of data transports compared to VLIWs.

In TTAs, the only machine instructions used are *move* and NOP. Moves happen between component ports, that can be either inputs or outputs. At least one of the input ports in a function unit must be *triggering*, which means that when data is moved to that port, the operation defined will execute. This allows TTA scheduling to have more freedom compared to a VLIW, since operands can be moved to and from FU ports at different times. A small example TTA is presented in Figure 2.5. This particular TTA implements two FUs: an ALU and an LSU.

The instruction word size in a TTA depends on the size of *move slots*. Move slots are defined for each bus separately. Their sizes are determined by the number of FU input and output ports, RF sizes, number of connections to the data buses, size of short immediates and the number of opcodes an the FUs implement. The size of the slot must allow describing of all possible moves from *source* to *destination* ports, that are connected through that bus.

In TTAs, an *Interconnection (IC)* network connects the CU, FUs and RFs in a processor. The interconnection consist of *data buses* transporting data, which are connected to other components by *sockets*. Interconnection design has an effect on both the performance and the power consumption in a processor. For example, an interconnection with an excessive amount of data buses consumes unnecessary power. If the amount of data buses is too small, the scheduling freedom decreases and program cycle count can increase, possibly leading to increased energy consumption.

For optimal power consumption, the IC *connectivity* must also be considered. In TTAs, buses are connected to FUs and RFs by *sockets*. Too much of connectivity is undesirable, since it increases the IC area and fan-in and fan-out of the logic gates, and therefore power consumption. TCE implements the IC as a combinatorial network and if it is very large, it can become the critical path in the design and therefore lower the maximum clock frequency. A very small amount of connections can also be harmful to the performance of the processor, because the compiler has less opportunities to parallelize the source code and therefore execution time is increased.

The TCE TTA template supports two ways to transport constant values in instructions [22, p. 12]. *Short immediates* are encoded into the move slot's source field directly. Larger constant values are transported with *long immediates*. They are written directly to the *Immediate Unit* (IU) by the CU, when it detects an instruc-

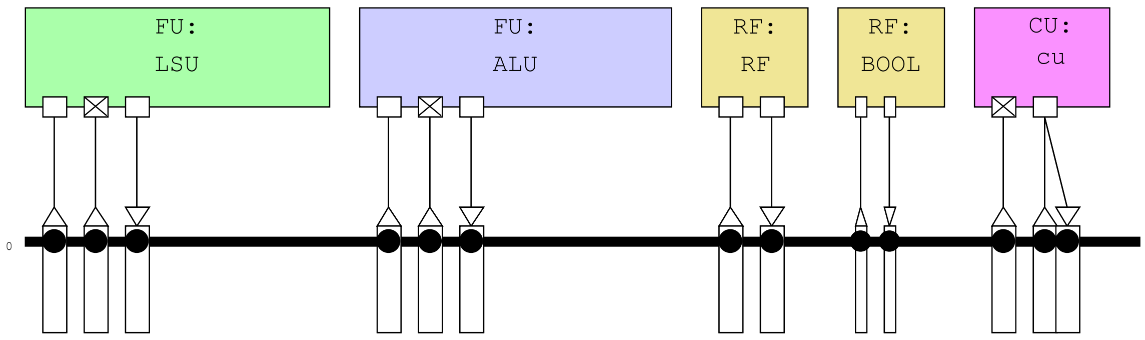


Figure 2.5 *An example TTA in TCE Processor Designer view.*

tion containing a long immediate. The IU only has read ports, since writing to them is done by the CU only.

Due to their low-power characteristics, modularity and simple structure, TTAs are an excellent choice for mobile devices and data processing oriented applications, where low power and high performance are crucial, such as accelerating operations of *Digital Signal Processing* (DSP) and *Software Defined Radio* (SDR).

3. TECHNIQUES AND METHODS FOR POWER-EFFICIENT PROCESSOR DESIGN

This chapter is a review of previously developed techniques and studies on power optimizations in integrated circuits. Optimizations in an integrated circuit design can be done on different levels of abstraction. This chapter describes those levels and gives examples of the most commonly used techniques on each one. The levels are listed as separate sections starting from the highest abstractional level, the system level, and ending on the lowest, transistor level.

Practical implementations of power-saving methods are often a trade-off between area, power consumption and speed. For example, extra control logic may need to be synthesized, which consumes static and dynamic power, so care must be taken not to increase the overall power consumption. Power optimizations can also have a performance cost. A sequential system's maximum clock frequency depends on the single longest path between two registers. If the extra logic increases the length of this *critical path*, the highest possible clock frequency must be lowered.

However, power optimizations don't always come with a trade-off. For example, an inefficient algorithm may waste power, speed and area and provides an opportunity for optimization. Or at the logic level, wires may simply be routed in an efficient way minimizing parasitic components in power consumption.

3.1 System Level

System level is the highest abstractional level where power optimizations can be done. Here the computational, communication and storage components of the system are taken into account, in addition to the physical power and clock distribution networks.

Bus Segmentation

The idea of *bus segmentation* is to divide a large bus into several tree-structured bus segments, taking advantage of local communication of function units [3]. This reduces the bus power consumption, because now only a segment of the bus changes its state at a time. The larger the bus is, the more load capacitance it will have, and segmenting it will avoid charging unnecessary parts where data doesn't need to be transmitted to at certain times.

Clock Frequency and Voltage Scaling

According to 2.1, The dynamic power of a CMOS circuit depends directly on clock frequency and is proportional to the square of operating voltage. This means that power and energy can be saved by reducing them, during both system design and run-time. In this section, different methods of voltage and frequency scaling are explained.

In ***Static Voltage Scaling*** *(SVS)*, separate supply voltages are fixed to each block according to performance needs, forming different voltage domains. The lower the needed operating frequency, the lower the supply voltage can be set. For example, a special function unit implementing a bottleneck function for a system might need to operate at a high frequency and therefore need a high supply voltage.

Static voltage scaling brings design challenges, such as requiring level shifters when signals go from one voltage domain to another, or the need for multiple voltage regulators.

When high performance isn't needed, both the clock frequency and operating voltage of a design can be lowered to save energy. This is called ***Dynamic Voltage and Frequency Scaling*** *(DVFS)*. If the run-time clock frequency is lowered, the throughput and power consumption of a block can be lowered. However, for a fixed size task, energy used by the system will stay the same, or increase because leakage current is still present. If we reduce both the operating voltage and clock frequency at the same time, we can achieve both power and energy savings.

One approach to perform simultaneous voltage and frequency scaling in a system is to define a fixed amount of modes the system can operate in. For example, these modes could be defined for low, moderate and high workloads. Each mode would have a voltage-frequency pair to be used, where the voltage is selected based the on frequency required. Because the actual clock frequency of the system depends on process variation, operating temperature and operating voltage, safe operation requires a margin for the operating voltage. Because of this, voltages corresponding to each frequency are larger than they could be.

Adaptive Voltage Scaling *(AVS)* extends DVFS by adding a control loop to the system. This addresses the problem with large safety margins in supply voltage. In AVS, the system's actual clock frequency and operating voltage are constantly monitored and fed back to a power management unit, which in turn adjusts the voltage to optimal for a given frequency. Because of the need to monitor the actual clock frequency, AVS is more complex to implement than DVS and therefore used less.

Power Gating

In order to eliminate both dynamic and leakage power consumption, the supply voltage to an unused block in a system can be disconnected. *Power gating* uses transistors implemented into the power distribution network to accomplish this. The designer needs to evaluate an enable condition for the logic blocks to be turned off.

Power gating causes registers in the powered down region to lose their states, which may need to be stored somehow when power is connected again. This can be done with a *retention memory*, where all the states of registers are stored just before power-down. Another method to store the values is by *retention registers.* They are special registers close to the actual register that store their value, when logic is powered down. Both of these methods cause area overhead and therefore add to power consumption.

3.2 Algorithm Level

A way to reduce switching activity and thus power consumption in a processor is to optimize different algorithms. At the *Algorithm level* for example instruction decoding, encoding and fetching can be implemented in various manners, that may be optimized for speed, area or power consumption. An algorithm can also be inefficient in a way that an inefficient implementation could occupy more area and take more clock cycles than an efficient implementation.

Instruction Encoding

Various methods have been used for power-efficient *instruction encoding*. Instruction encoding can be aimed to reduce instruction word length and therefore instruction memory size, to minimize instruction bus switching activity and to minimize crosstalk between parallel bus lines. Efficiency of instruction encoding depends on the program being executed, so it may not be always beneficial to implement, because both encoding and decoding have to be implemented and the extra logic required by this consumes both static and dynamic power.

Reducing the instruction word size (*instruction compression*) can affect power consumption indirectly, since it can lead to reduced instruction memory size and smaller instruction bus width. Smaller memory size means less leakage power when idle and less dynamic power when fetching instructions. This applies for both the instruction memory and instruction bus. Various approaches to instruction compression have been applied, for example *dictionary compression* [8], [10] or *Huffman* [1] encoding.

Reducing the amount of switching activity on the instruction bus can lead to significant power savings. The *hamming distance* of two consecutive instructions is the amount of bits that change between the instructions. Each bit change corresponds to a wire charged from ground level to supply voltage, or the opposite. If the hamming distance of consecutive instructions can be minimized, power and energy consumption of the instruction bus can be reduced. Various encoding algorithms to minimize these bus signal changes exist and their effectiveness usually depends on the program being executed. These include *bus invert* encoding [17], *Gray* encoding [18] *and Beach* encoding [2].

Reducing the parasitic coupling capacitance is one target of optimization in an IC. As mentioned in 2.1.3, At the algorithm level, some approaches [12] to reducing the coupling capacitance have been developed.

3.3 Architecture Level

At the *architecture level*, the choice of function units and buses, memories and caches has an impact on power consumption. Optimal performance and power consumption on this level requires experience from the designer, since the combined effect can be difficult to predict.

Pipelining

In a pipelined processor, different stages of an operation are executed in parallel. For example, in a processor with a pipeline depth of five, up to five instructions can be executed simultaneously. This increases the latency (amount of clock cycles) of a single operation, but also increases throughput (amount of data processed per clock cycle). *Throughput* is the amount of output produced by a component in a given time. Pipelined processors have better throughput than single-cycle processors, since less time is needed to execute a single operation. However, the time from triggering an operation to receiving the result increases with pipelining, if we assume the same clock frequency in both cases.

However, pipelining adds complexity to the control logic, because of *hazards* which lead to *stalling* of the pipeline. *Data hazards* can occur when an instruction depends on the result of a previous instruction. A *Structural hazard* means that the architecture of the processor can't support some combination of instructions. *Control hazards* occur when a branch is taken. If a branching condition is met, the program counter is increased or decreased by some value. To resolve a hazard, a stall stops the pipeline.

The increased complexity of control logic and pipeline registers leads to area overhead and an increase in power consumption. Pipelining can allow higher clock frequencies for increased performance, since the critical path of a single operation can be lowered. This way, the throughput can be increased with the increased maximum clock frequency. However, pipelining combined with increasing parallelism, can also be used to reduced overall power consumption, if the supply voltage can be lowered along with lowering the clock frequency [4, p. 55]. It may be applicable, if there is only a requirement for the throughput, or *Instructions Per Clock Cycle* (IPC), and not for the time of a single operation. For example, if an operation forming the critical path can be divided into pipeline stages, the stages will have timing slack, possibly allowing the supply voltage to be lowered. Or, if we can add another function unit implementing the same operation in parallel, the clock frequency can be lowered.

In TTAs, pipelining is done in function units. Function units can be single-cycle, multi-cycle or pipelined. Single-cycle FUs output the result after one clock cycle after an operation is triggered, whereas multi-cycle FUs take more clock cycles. Multi-cycle FU may not accept new input operands if an operation is being executed. A pipelined FU can start an operation every clock cycle and in a function unit with n stages, n operations can be executed simultaneously, in different stages of the operation. An FU can also be partially pipelined.

Loop Caches

The purpose of a *cache* is to provide a small, fast-to-access memory located close to the function unit fetching data from a memory or a cache. An instruction cache stores instructions that are about to be executed and is useful, when a program contains loops or instructions that are repeated often.

A *loop cache*, or a *loop buffer* stores a loop, which is executed repeatedly in a program. This means that the instructions are fetched from the main instruction memory or a cache only once and decoded, after which they will be held in the loop cache until the next loop needs to be fetched. Decoding can also be done after the loop cache. A loop cache can reduce the power consumption, since there will be less memory accesses and less instruction decoding. Choosing the optimal size for the loop cache requires consideration of typical programs executed and the amount and length of loops they have, since a loop cache of excessive size can waste power. A loop cache of insufficient size may not work optimally with large loops. Combined with power gating of the instruction fetch and decode units, loop caches can provide an efficient way to save power. However, implementing a loop cache in a design has some area overhead, which leads to an increase in the power consumption.

3.4 Register Transfer Level

At the *Register Transfer Level (RTL)*, the designer's decisions determine, what kind of logic will be synthesized. This allows various power-saving opportunities.

Clock Gating

In sequential logic, the value of register outputs can change only on the clock edge. When the output of a register changes, energy is dissipated because of load capacitance seen by the register. When the clock is gated off, the registers timed with that clock will keep their state and not dissipate any dynamic power. However, unlike in power gating, leakage power is still consumed because the logic is still connected to supply voltage. *Clock gating* also reduces the clock tree power consumption, because it stops the clock signal switching inside gated logic blocks.

During synthesis, gating components are placed into the clock tree. The number of gating components depends on the size of the logic block each component controls. Extreme case of clock gating would be to add a gating component to each register in the design. Each of these components needs an enable condition for stopping the clock signal. Inserting gating components to the clock tree affects the timing of the design, because they have a finite propagation delay and therefore increase the skew of the clock signal. Therefore the clock tree must be balanced in the place and route phase of the design flow.

Clock gating can be implemented as AND gates, NOR gates, latch based AND gates, latch based NOR gates, and multiplexer based clock gating. Synthesis software tools use *Integrated Clock Gating (ICG)* cells found in technology libraries to automatically implement gating. For this, the software needs to determine a condition when a clock signal can be stopped. An RTL designer can instantiate the ICG cells manually for single registers or logic blocks, if enable conditions for gating can be determined. This can lead to increased power saving, because the number of gating components can be reduced.

Logic Selection

Synthesis tool implementation of logic in a design depends on the RTL code. In a *Very high speed integrated circuit Hardware Description Language (VHDL)* process, for example, an *if-else* statement will be inferred as a priority encoder, whereas a *case* statement will be inferred as a multiplexer. The RTL code written by the designer affects the inferred logic and therefore the area, performance and power consumption of the design.

Operand Isolation

In a digital design, unused parts of the data path consume unwanted power if its signals change their state. For example, a logic block in a design may not be used during some time period, but its inputs may still be toggled, which can propagate unwanted switching activity into that block. To stop this switching activity, *operand isolation* can be implemented with AND gates, latches or multiplexers. To stop toggling of inputs, an enable condition has to be determined.

Operand isolation adds extra logic to the design. This leads to an increase in the total area of the design. Extra logic in turn consumes both dynamic and static power, so care must be taken not accidentally increase the total power consumption when implementing operand isolation.

Pulse Latching

Generally clock triggered flip-flops are used as registers to store values in digital designs. This is because flip-flop timing can be analyzed with *Static Timing Analysis (STA)*. An alternative to flip-flops are latches. Unlike flip-flops, latches are not synchronous. *Pulse-latches* can be used similarly to flip-flops to achieve synchronous operation while reducing the dynamic power consumption. They require a pulse generator to generate pulses from a clock signal. The inputs of the latch are updated only when the pulse signal is active.

3.5 Logic Level

At the logic level, power optimizations are done for logic gates formed by CMOS transistors. Optimizations can be replacing logic gates with more efficient ones, optimizing placement of gates, or changing the hierarchy of logic blocks. In this section, the most commonly used techniques are reviewed.

Hierarchy Ungrouping

Large digital designs usually have different levels of hierarchy, meaning that they have a top level block, which has the inputs and outputs for the design, and consists of smaller logic blocks implementing some functionality. These can consist of more logic blocks, forming the hierarchy of the design. *Ungrouping* removes the hierarchy in a design, allowing software tools to optimize designs across the logic block boundaries and improving resource sharing between blocks.

Wire Length

The longer a wire is in an integrated circuit, the more load capacitance it has. This offers an opportunity for power optimization, because high activity wires can be shortened, reducing their load capacitance and therefore power consumption according to 2.1. Optimizing wire length also means optimizing logic placement, as the logic gates connected by a high activity wire need to be close to each other in order to shorten the wire.

Software tools can automatically perform logic placement optimizations during the place & route phase. For this they need information about the switching activity in a design, which can be produced from RTL or gate level simulation. Different simulations may result in different switching activity depending on the simulated software and its input data.

Wire Spacing

As mentioned in Section 3.2, the distance between wires running in parallel affects the power consumption because of coupling capacitance. Place & route tools can perform wire placement optimizations on the logic level by avoiding long wires side by side.

3.6 Transistor Level

This section focuses on techniques used in transistor fabrication. From a system designer's point of view, transistor level optimizations are limited to selecting appropriate technology libraries. Silicon manufacturers offer various *processes* and libraries, which define how the design will be implemented physically.

Choice of Process

In integrated circuit fabrication, process is the physical method used to implement a design onto silicon. Silicon manufacturers offer various processes for different use cases, usually optimized for low power, high performance, small area or a combination of these. It is up to the ASIC system designer to choose the appropriate process for a given application. Development of process technology scales transistor *gate lengths* downwards, with smallest gate length in commercial applications being 14nm [14].

Multi Channel Length libraries

Optimization with multiple libraries can be done for transistor channel length. Transistors with a short channel can change their state at higher frequencies than a transistor with a long channel, but the trade-off for performance is high leakage current. Synthesis software can be used to map the implemented logic to transistors with short channel, when high performance is needed. When timing is not critical, longer channel transistors can be used to reduce power consumption.

Multi Voltage Threshold libraries

The threshold voltage, V_t is the level of voltage needed for a transistor to become conductive. In low-V_t transistors, the leakage power consumption is large, but performance is high. That is, the transistor can be run at higher frequencies and still meet timing requirements. In high-V_t transistors, the leakage power consumption is small but performance is low.

Multi-V_t optimization can be done with synthesis software tools. Standard cell libraries with different threshold voltages are given to the tool. The software calculates critical paths in the design and, for logic that requires high performance, maps that logic to low-V_t standard cells. By default, High-V_t cells are used where the performance requirements aren't high, because they offer smaller leakage power consumption.

Number of Tracks

Standard cell libraries can include implementations for different number of tracks for a cell, for example 8-track or 12-track cells. Number of tracks is the amount of wires that can pass through it. For example, supply voltage and ground would need a track in a cell. A higher number of tracks corresponds to higher driving strength, that is how much logic is the cell able to feed power to from it's output.

Cells with a high number of tracks can also run at higher clock frequencies, but have a higher leakage power consumption than ones with a lower track count. Cells can have varying widths, but have a fixed height according to the number of tracks, so that they can be organized into even rows during the place & route phase.

4. BASELINE MEASUREMENTS AND AUTOMATIC SYNTHESIS TOOL OPTIMIZATIONS

This chapter presents the initial synthesis results for three benchmark architectures. First, the baseline measurements with no options implemented are examined. After this, the effect of the synthesis tool, Synopsys Design Compiler, are observed.

4.1 Benchmark Architectures, Programs and Initial Measurements

In order to measure the effects of power optimizations, three different TTA processors designed using TCE were used. Different use cases were targeted when choosing the architectures. The first processor, is meant for *Micro Controller Unit (MCU)* usage and is targeted to have the lowest power consumption and area. The second processor's use case is in *Digital Signal Processing (DSP)* and is larger in both power consumption and area. The last and largest processor is the *Software Defined Radio (SDR)*, *Single Instruction Multiple Data (SIMD)* processor with support for hardware floating point operations. These processors will be referred to here as *MCU*, *DSP* and *SDR*.

Table 4.1 *Summary of the benchmark architectures.*

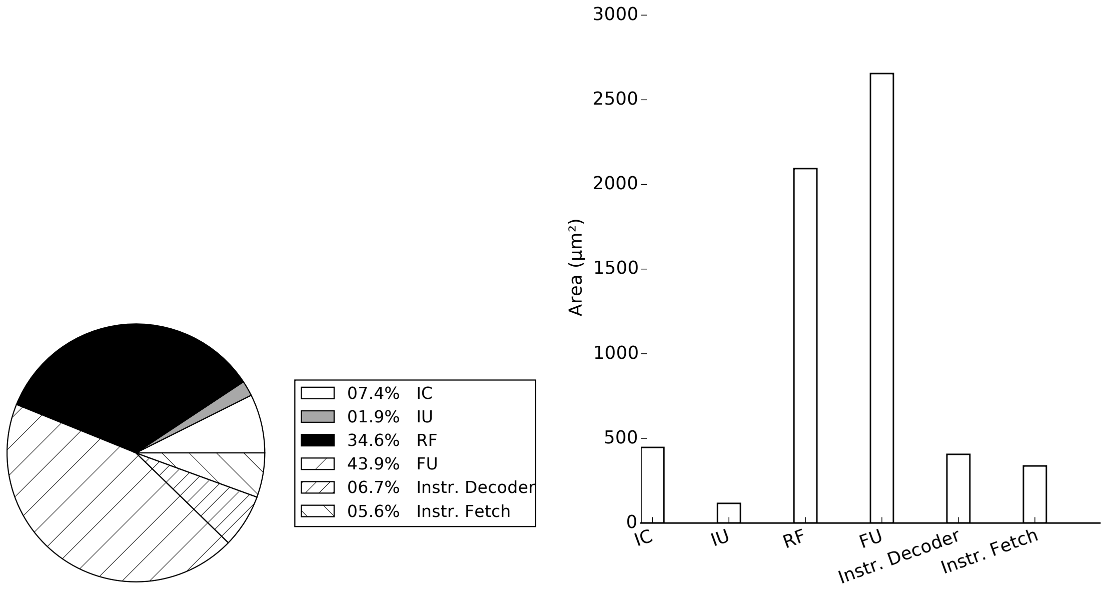
processor	MCU	DSP	SDR
target clock frequency	50 MHz	500 MHz	1GHz
instruction width	40b	335b	128b
transport buses	3x32b	18x32b	3x32b, 4x512b
registers	2x1b, 16x32b	6x1b, 3x14x32b	2x1b, 32x32b, 32x512b

The measurements for power consumption were performed for synthesized designs using a 28 nm *Fully Depleted Silicon On Insulator (FDSOI)* standard cell *Application Specific Integrated Circuit (ASIC)* technology. Details for the used architectures are listed in Table 4.1.

For power estimations, Power Compiler [19], used internally by Design Compiler, separates power usage to tree categories: Static power, switching power and internal power.

4.1.1 Initial Results

Initial area distributions after synthesis for the benchmark architectures can be found in Figures 4.1 (MCU), 4.2 (DSP), 4.3 (SDR).



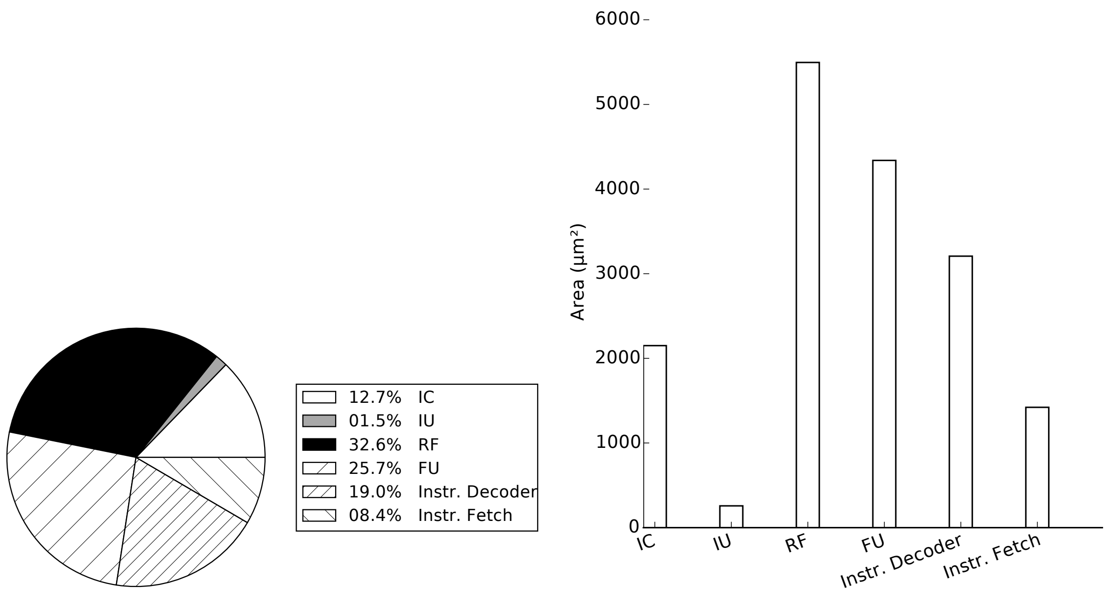
(a) (b)

Figure 4.1 *Baseline area distribution for MCU*

From these Figures can be seen that FU and RF components occupy most of the area, in our benchmark architectures FUs occupy between 25.7% to 52.3% and RFs between 32.6% to 39.3%. The proportion of control logic, instruction fetch and instruction decoder (and decompressor), seems to depend greatly on the size of the instruction word.

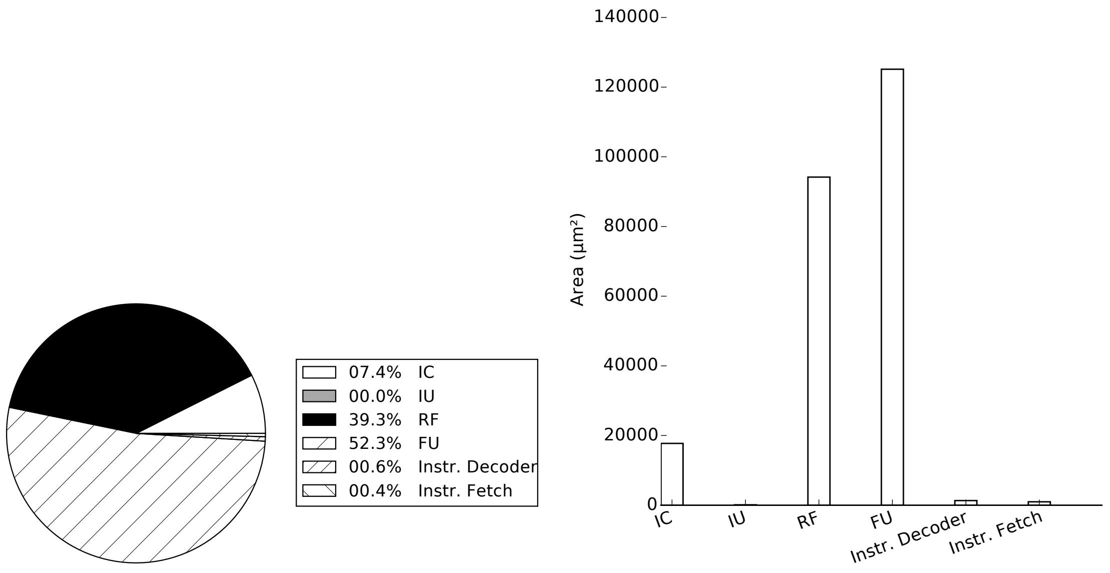
The measurements here for power consumption should be observed with caution, since in real world designs, automated synthesis tool optimizations have a great effect on the power consumption and area occupation. Results here are for later comparison to the effect of the automatic optimizations.

The baseline power distributions for each benchmark architecture and program are presented in Figures 4.4 for *MCU* core, 4.5 for *DSP* core and 4.6 for *SDR* core. From these can be seen, that the power consumption consist mostly of *internal*



(a) (b)

Figure 4.2 *Baseline area distribution for DSP*



(a) (b)

Figure 4.3 *Baseline area distribution for SDR.*

power consumption. This is due to the cells not being gated and burning up power unnecessarily when not needed.

Table 4.2 lists the total baseline area and power for each architecture. the *DSP* core is 2.5 times larger than the *MCU* core and the *SDR* core 39.5 times larger than the *MCU* core.

Table 4.2 *Baseline area and power for the benchmark architectures.*

processor		MCU	DSP	SDR
area (μm^2)		6073	17011	239991
power (mW)	Coremark	0.42		
	DSPStone matrix	0.43		
	AES		10.7	
	JPEG		10.5	
	FFT		10.4	
	SDRkernel1			257
	SDRkernel2			252

4.2 Power Optimizations in Design Compiler

This section describes different power optimizations in Synopsys Design Compiler. Underneath Design Compiler's user interface, Synopsys Power Compiler does optimizations regarding power consumption.

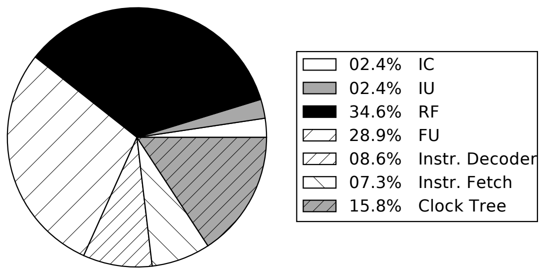
Certain optimizations require information about the switching activity to generate accurate power estimations. *Switching Activity Interchange Format (SAIF)* files [19, p. 63] can be used for this. For user defined signals in a design, the SAIF file provides a summary of time spent at logical 0, time spent at logical 1, time spent at unknown state, total toggle count and toggle count to/from unknown state.

SAIF files can be produced from *Value Change Dump (VCD)* files, or from simulation directly. For user defined signals, a VCD file contains the status of the signal at every clock cycle. The size of a VCD file depends on the size of the design and duration of the simulation, so it can become very large compared to a SAIF file of the same simulation.

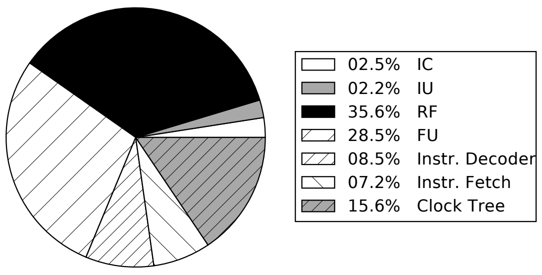
The effects of optimizations are not always visible in the logic block they are targeted to. For example, the reduction in power when gating an FU for switching activity may be seen only in the FU, or both the IC and the FU. Therefore, it is often best to observe the total effect on the core.

4.2.1 Design Compiler Options

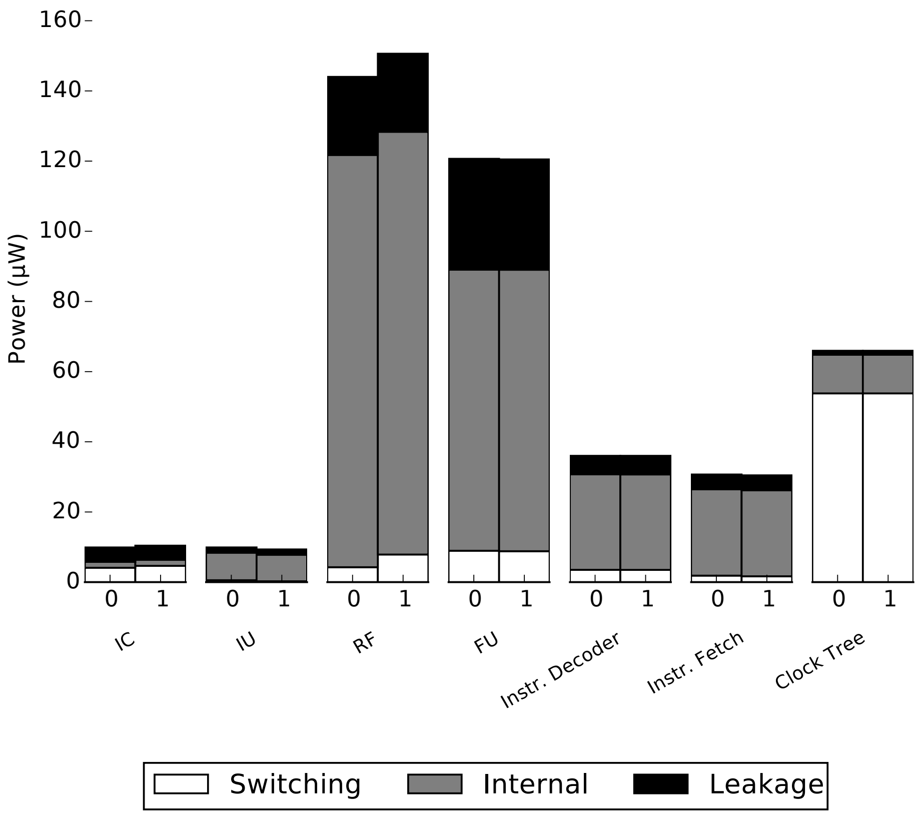
The different optimizations for power and area found in Design Compiler are introduced here one by one, after which power and area measurements are presented. The commands and switches to enable the optimizations are finally summarized in Table 4.3.



(a) Coremark



(b) DSPStone matrix

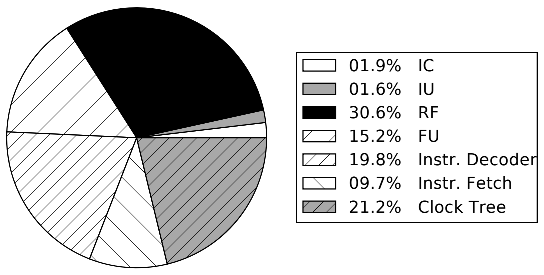


(c) Power components for Coremark (0) and DSPStone matrix (1)

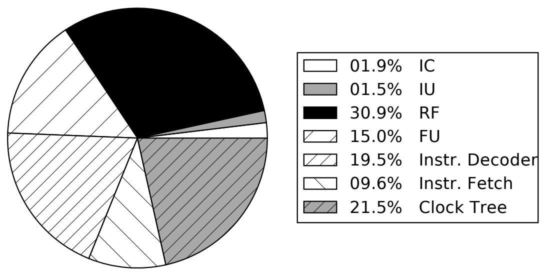
Figure 4.4 *Baseline power distribution for MCU core with Coremark 4.4(a) and DSPStone matrix 4.4(b)*

Leakage Power Optimization

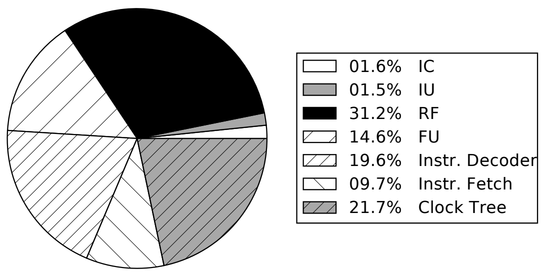
For non-critical paths in a design, Power Compiler can perform leakage optimization by replacing low-Vth, high leakage cells with high-Vth, low leakage cells. Leakage power optimization is automatically enabled for Design Compiler tools, except for DC Expert. This optimization requires multi-Vth libraries to be defined.



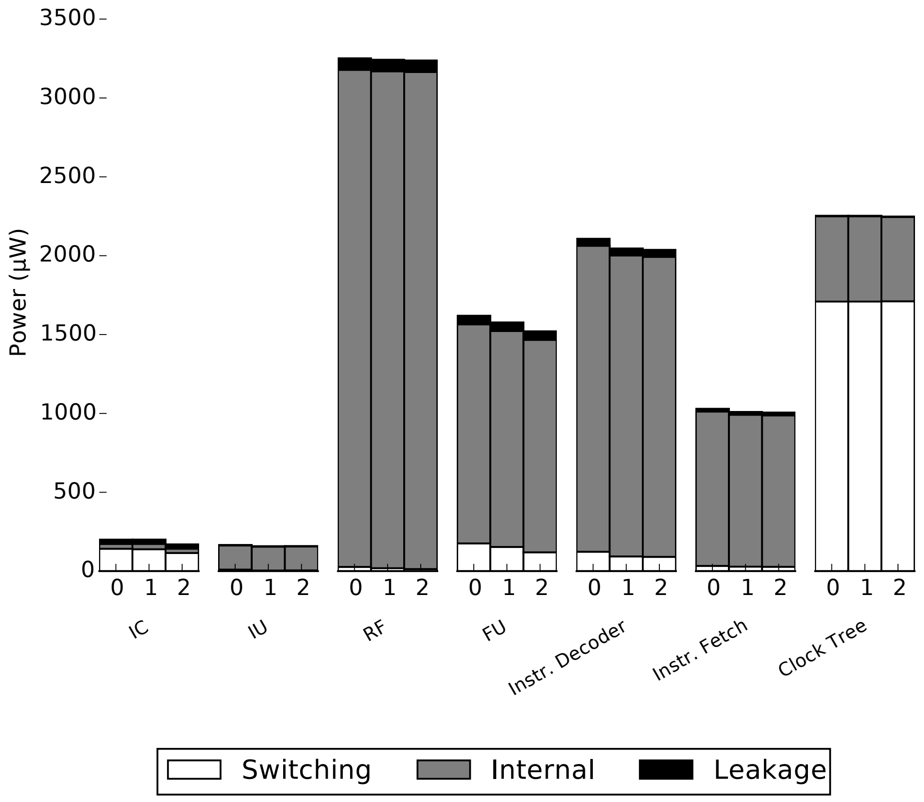
(a) AES



(b) JPEG

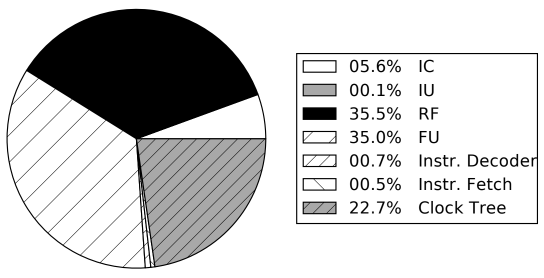


(c) FFT

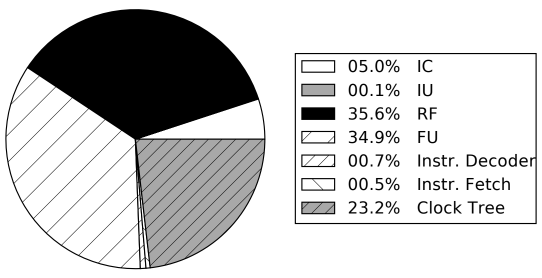


(d) Power components for AES (0), JPEG (1) and FFT (2)

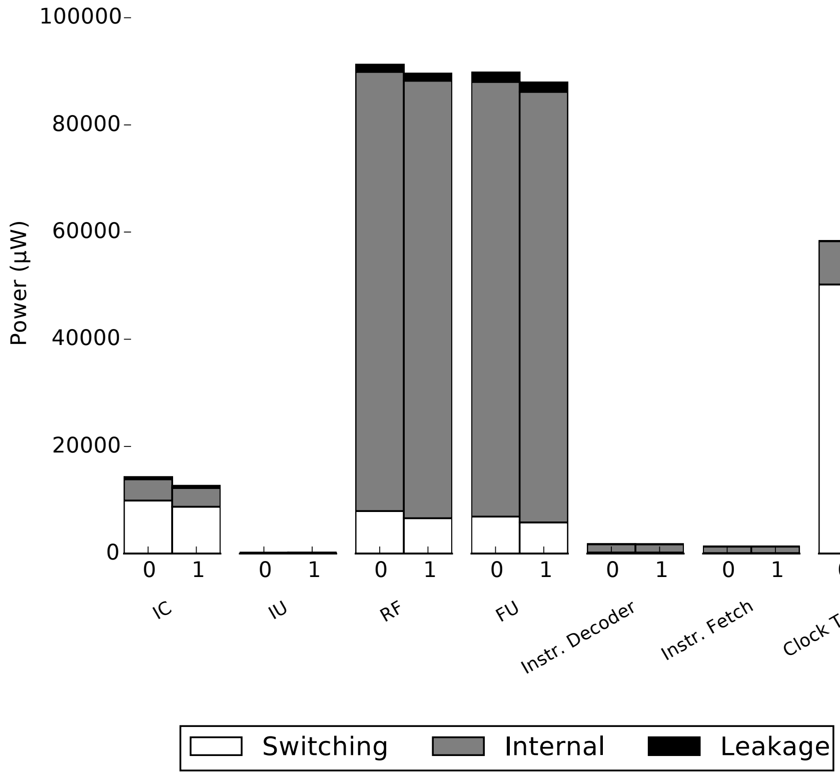
Figure 4.5 *Baseline power distribution for DSP core with AES 4.5(a), JPEG 4.5(b) and FFT 4.5(c)*



(a) SDRkernel1



(b) SDRkernel2



(c) Power components for SDRkernel1 (0) and SDRkernel2 (1)

Figure 4.6 *Baseline power distribution for SDR core with SDRkernel1 4.6(a) and SDRkernel2 4.6(b)*

Dynamic Power Optimization

If not annotated by the designer, Power Compiler will use default values for switching activity information. For better dynamic optimization, SAIF files should be used, otherwise the tool will assign a default toggle rate value for all input ports. Requires switching activity information. This optimization requires multi-Vth libraries to be defined.

***Table 4.3** Options and switches to enable power optimizations in Design Compiler.*
†: requires Multi-Vth libraries

optimization	command	switch	default
leakage †	set_leakage_optimization true		true
dynamic †	set_dynamic_optimization true		true
clock gating	set compile_clock_gating _through_hierarchy true		false
	compile_ultra	-gate_clock	disabled
hierarchy ungroup	-no_autoungroup		enabled
	ungroup	-start_level n	

Clock Gating

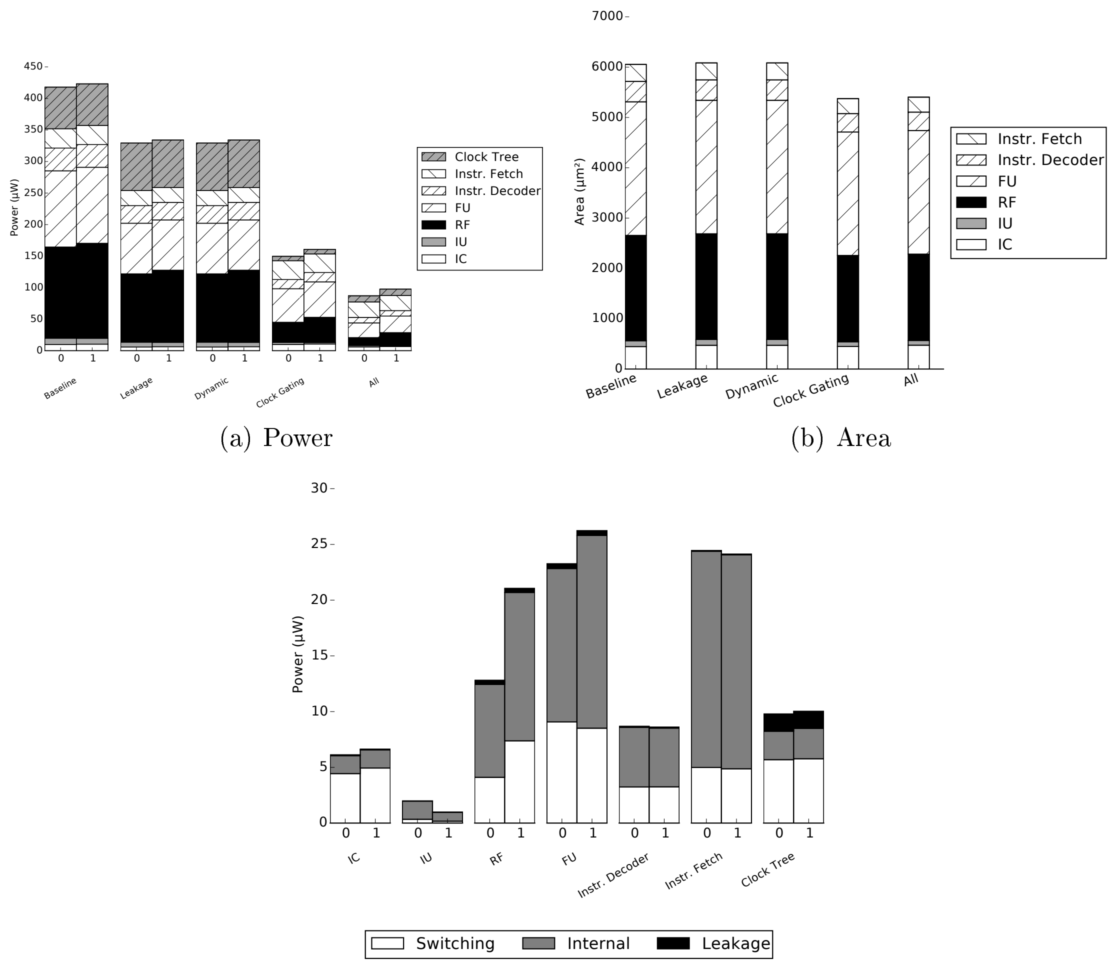
Power Compiler can automatically insert clock gating to a design. This will infer clock gating elements specific to the standard cell library used. Power Compiler performs clock gating only to registers, whose enable signal is synchronous with its clock.

Hierarchy Ungrouping

Design Compiler Ultra removes design hierarchy by default. This can make the design schematic hard to read, since boundaries of logic blocks, such as register files or function units in a TTA processor, will be removed and the logic inside them placed onto the same hierarchy level. However, this can allow logic to better use shared resources. This option is enabled by default, but can be disabled, or enabled only for design hierarchy levels below a level given as an argument to the switch. This may be useful, when for example effect of an optimization needs to be analyzed for a single module. For this thesis, this option was disabled to observe optimization effects for individual components.

Operand Isolation

Design Compiler supports automatic insertion of operand isolation logic. However, this feature was not used in this thesis due to it requiring a separate license. Instead, this optimization was implemented manually and will be presented later in the thesis.



(a) Power

(b) Area

(c) Power components with all Design Compiler optimizations applied.

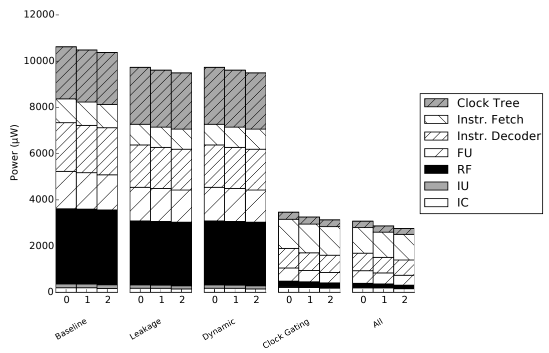
Figure 4.7 *Effect of Design Compiler optimizations on power and area for MCU core with Coremark (0) and DSPStone matrix (1)*

4.2.2 Effect of Design Compiler Optimizations

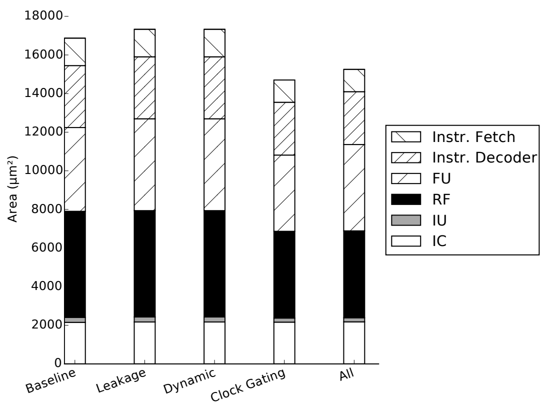
Results for power consumption and area occupation after synthesis for *MCU*, *DSP* and *SDR* core are presented in Figures 4.7, 4.8 and 4.9 respectively. Observing the power consumption with all Design Compiler optimizations applied serves as a good starting point, since these optimizations can be assumed to be used in all reasonable designs.

In general, leakage and dynamic power optimizations seem to have a small effect compared to clock gating in a design. Alone, the automatic clock gating more than halves the power consumption in all the test cases for all designs measured here.

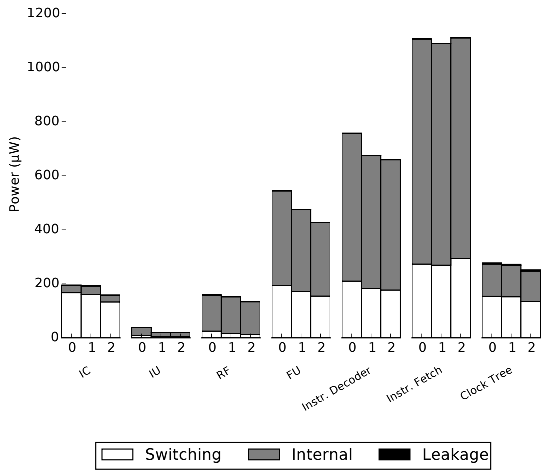
For the *MCU* core, the components consuming most of the power are RFs, FUs



(a) Power



(b) Area

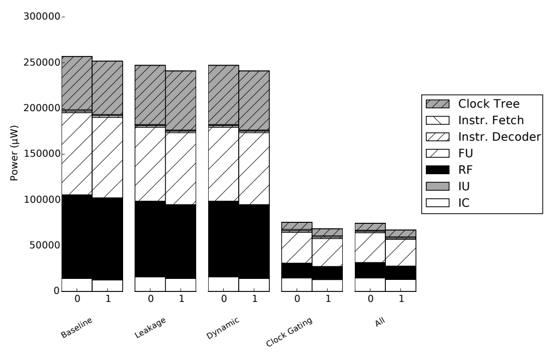


(c) Power components with all Design Compiler optimizations applied.

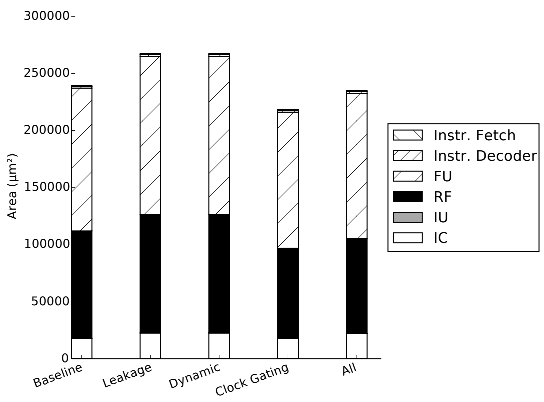
Figure 4.8 *Effect of Design Compiler optimizations on power and area for DSP core with AES (0), JPEG (1) and FFT (2)*

Table 4.4 *Area and power with Design Compiler optimizations applied.*

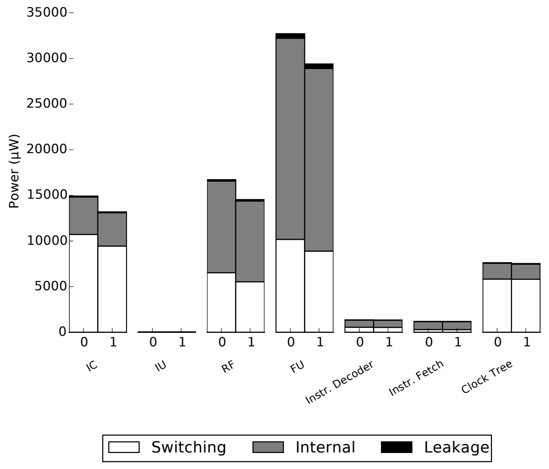
processor		MCU	DSP	SDR
area (μm^2)		5422	15390	234846
power (mW)	Coremark	0.0889		
	DSPStone matrix	0.0993		
	AES		3.11	
	JPEG		2.90	
	FFT		2.79	
	SDRkernel1			74.5
	SDRkernel2			67.0



(a) Power



(b) Area



(c) Power components with all Design Compiler optimizations applied.

Figure 4.9 *Effect of Design Compiler optimizations on power and area for SDR core with SDRkernel1 (0) and SDRkernel2 (1)*

and instruction fetch unit (Fig. 4.7(c)). This is good, since RF and FU utilization should be high for efficient core usage. This way, power is consumed in the actual computation instead of control logic. The instruction fetch unit is active on every clock cycle for the *MCU* core and therefore consumes a relatively large portion of the total core power.

For the *DSP* core, instruction fetch, instruction decoder and FUs take up most of the power consumption. This is due to the core having 18 data buses, therefore, having a 335-bit instruction word. The unit stores this word into a register and because the consecutive instructions can be very different, the register's value changes frequently, burning up power.

For the *SDR* core, IC, FU and RF components dominate the power consumption. Here the instruction fetch and decode's portion of the power consumption is relatively small, although having an 128-bit instruction word. This can be explained due to the core having a 32x512b register and four 512-bit data buses, as listed in Table 4.1. In addition, the core has a 32-way, 16-bit floating point FU. Compared to these, the instruction fetch power consumption is small.

From Figures 4.7(b), 4.8(b) and 4.9(b) can be seen that the area occupation increases when using leakage and dynamic power optimization. This is due to the synthesis tool replacing high-leakage standard cells with larger but slower low-leakage cells. This is only possible, if there is enough slack in the timing of the logic to be replaced.

High FU power consumption in all three designs is a good indicator of efficient FU utilization in TTAs. Avoiding large control logic by static scheduling helps in accomplishing this. After the Design Compiler optimizations, it seems that the power optimizations should be targeted to the instruction fetch, RF and FU components. Moreover, optimizing the instruction fetch power consumption often has an impact on the instruction memory consumption, which can be more than the total core power consumption.

5. IMPLEMENTED POWER OPTIMIZATIONS

This chapter describes the power optimizations that were implemented to TCE, or individual HDL modules. First, the individual optimizations are presented along with instructions on how to enable them. Results for individual optimizations are presented. The chapter is concluded with a summary of all optimizations combined and their results.

5.1 Function Unit Operand Isolation

Function units in TTAs often implement more than one possible operation. For example, a standard ALU shipped with TCE can contain around 20 different operations. Depending on the implementation, changes in input operands to an FU may be propagated to all the operations in it, although only one operation's result is used. This means unwanted switching activity that can be eliminated. Depending on the opcode input, unwanted operations can be determined. In TTAs produced with TCE, the opcode signal can be used to determine if the input should be propagated to an operation or not.

Operations can have varying levels of complexity, using different amounts of logic. Depending on the power consumption of the logic implementing the operation, it may not be useful to gate the operation. For example, simple logic operations such as OR, NOT or AND of the input operands are simple in terms of logic and seemed unwise to be gated.

For the thesis, this optimization was implemented for the ALUs of *MCU* and *DSP* cores. *SDR* core was left out because its ALU's portion of the power consumption was very small, around 1.7% for the vector ALU and only around 0.3% for the scalar ALU with both test programs. Thus, the effect would have been quite insignificant for the total power consumption. In addition to the ALUs, the *SDR* core had a vector floating point unit, but this optimization was not applicable to it, since many of its operations shared computation logic already. The best results for operand isolation were obtained by leaving the simple operations, such as AND, OR, XOR, as they were and by grouping similar operations together. When an operation was not used

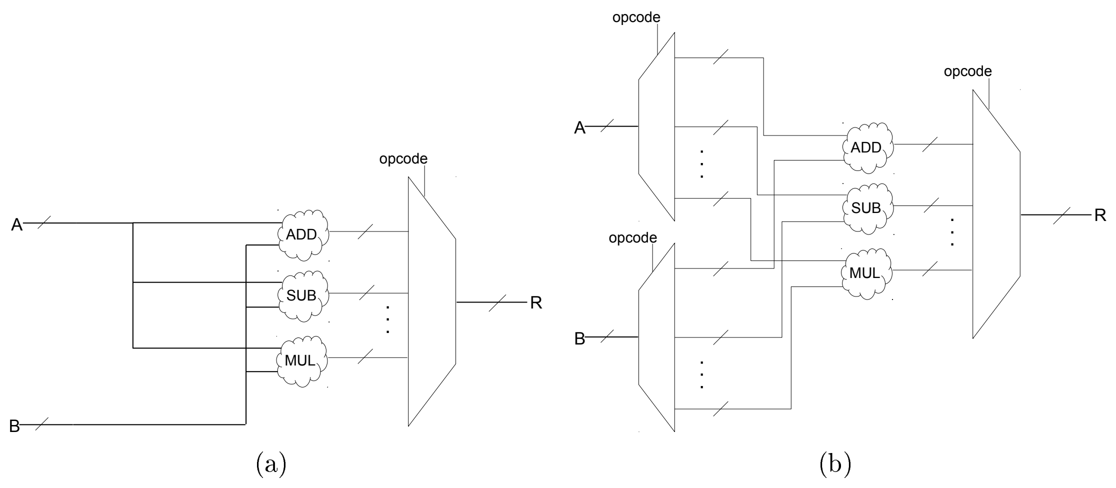


Figure 5.1 *Isolating the input operands in TTA function units. No operand isolation 5.1(a) and operand isolation applied 5.1(b)*

Table 5.1 *Effect of operand isolation on core total power and area.*

processor		MCU		DSP	
			Δ(%)		Δ(%)
area (μm^2)		5453	+0.6	15618	+1.48
power (mW)	Coremark	0.0857	-3.60		
	DSPStone matrix	0.0970	-2.31		
	AES			3.02	-3.05
	JPEG			2.82	-3.00
	FFT			2.74	-1.58

(according to the opcode), logic '0' was fed to it, minimizing the bit switches. An example of grouping operations in the *DSP* core is joining *ADD* and *SHIFT-LEFT-ADD* operations. Both of these utilize an adder, with the latter shifting its input data logically left before the addition. Thus, it makes sense to isolate the inputs to the adder as a group.

Table 5.1 summarizes the results for this optimization. For the *MCU* core, achieved savings in total core power consumption were 3.60% with Coremark and 2.31% with DSPStone matrix, with a total area increase of 0.6%. Although the total core savings aren't large in proportion, the ALU power consumption decreased 13.4% and 10.9% for the test programs. With possible later optimizations and power savings, these reductions will become more significant.

For the *DSP* core, power savings of 3.05%, 3.00% and 1.58% for AES, JPEG and FFT programs, respectively. This brought an area increase of 1.48%. ALU power decreases for the test programs were 14.4%, 14.3% and 13.4%.

5.2 Register File Input Port Data Gating

Register file and function unit inputs in TTAs can be connected to the same data buses, generating spurious input switching when it is not wanted. New values to register files and function units are loaded when their *load* is set active. In [7], RF input data was gated with the load signal using AND gates to reduce the switching capacitance of the IC network. RFs are usually utilized quite heavily. The more the RF operations there are in a program, the less effective RF data gating becomes, since the gating logic becomes less effective and is just consuming power itself. However, for all of the benchmark programs used the data gating resulted effective in reducing the dynamic power consumption.

Table 5.2 *Effect of register file datapath gating on core total power and area.*

processor		MCU		DSP		SDR	
			Δ(%)		Δ(%)		Δ(%)
area (μm^2)		5436	+0.26	15579	+1.22	240276	+2.31
power (mW)	Coremark	0.0845	-4.95				
	DSPStone matrix	0.0933	-6.04				
	AES			3.09	-0.77		
	JPEG			2.87	-1.00		
	FFT			2.76	-0.80		
	SDRkernel1					68.8	-7.73
	SDRkernel2					62.4	-6.87

Datapath gating was implemented to the TCE's processor generator and is now included in the generated processors automatically. The gating block is inserted between the IC sockets and RFs during processor generation. This type of datapath gating was also tried on function units, but it did not always decrease power consumption. Therefore, the datapath gating was implemented only for RFs. The implementation idea is presented in Fig. 5.2, where each bus wire is fed through an AND port to reduce switching activity. Using multiplexers was also tried in the RTL code, but the synthesis result was still an AND gate, leading to the same results.

The RF power consumption for the *MCU* core decreased 44.1% when running Coremark and 46.4% with DSPStone Matrix. Decrease for the *DSP* core was 38.6% with AES, 41.0% with JPEG and 42.3% with FFT. Decrease for SDR core was 21.8% with SDRkernel1 and 18.2% with SDRkernel2.

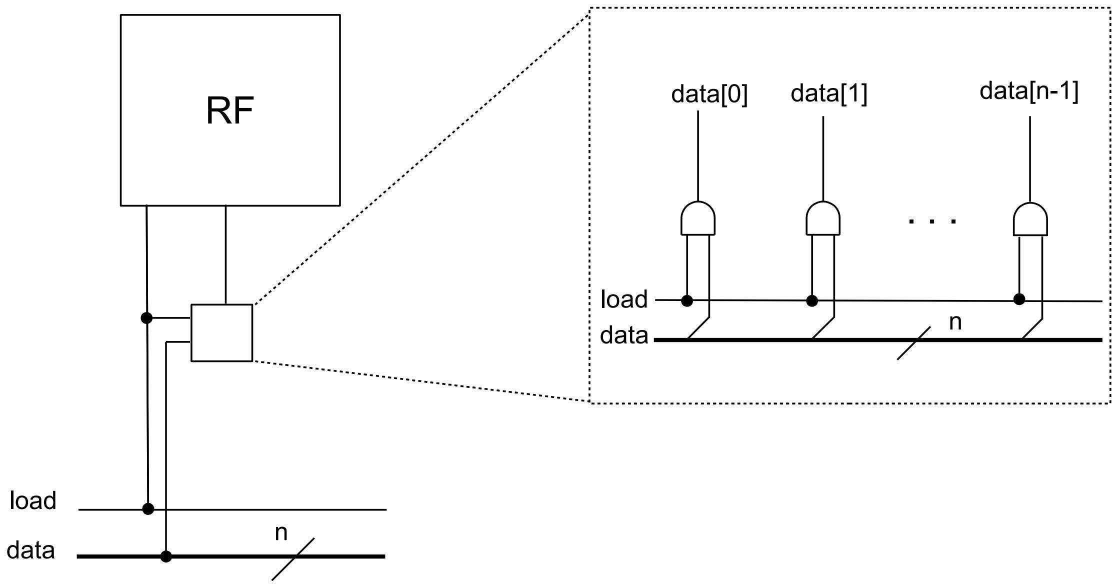


Figure 5.2 *Register file datapath gating using AND gates.*

5.3 Pipelined Function Unit Clock Gating Enablement

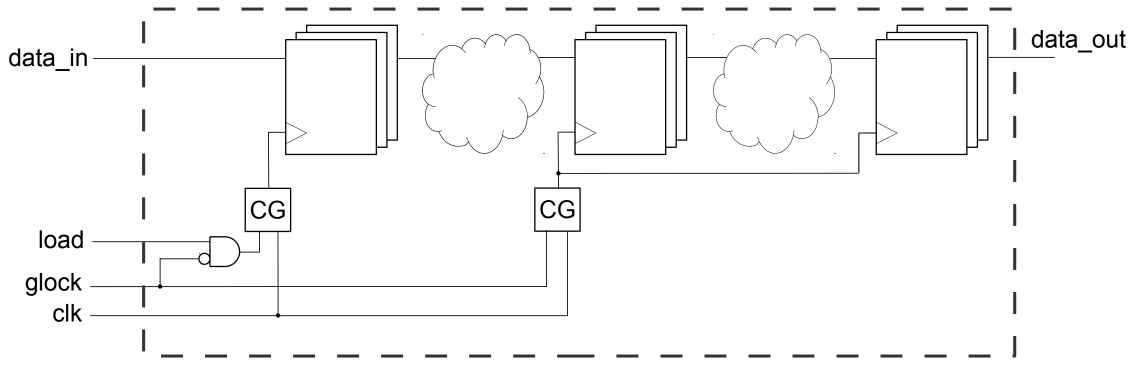
This optimization was only implemented for the *SDR* core, since it featured a three-stage pipelined floating point function unit. Before the enhancement, this FU alone consumed around 29% of the total TTA core power with the SDRkernel1 and SDRkernel2 test programs.

Table 5.3 *Effect of enhancing the clock gating enablement on core total power and area.*

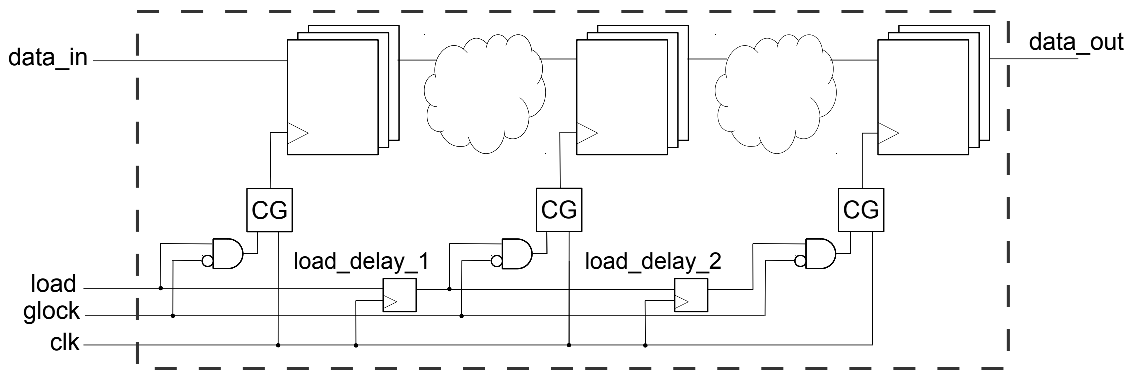
processor		SDR	
			Δ(%)
area (μm^2)		233880	+0.73
power (mW)	SDRkernel1	67.2	-9.85
	SDRkernel2	59.9	-10.6

```
...
load : IN std_logic;
...
signal load_delay1_r, load_delay2_r : std_logic;
...

stage1: PROCESS (clk, rstx)
    BEGIN  -- PROCESS stage1
      IF (rstx = '0') THEN
         load_delay1_r <= '0';
       ELSIF (clk'event AND clk = '1') THEN
         IF (glock = '0') THEN
           load_delay1_r <= load;
           IF load = '1' THEN
             ...
           END IF;
         END IF;
      END IF;
    END PROCESS stage1;

stage2: PROCESS (clk, rstx)
    BEGIN  -- PROCESS stage2
      IF (rstx = '0') THEN
        load_delay2_r <= '0';
      ELSIF (clk'event AND clk = '1') THEN
        IF (glock = '0') THEN
          load_delay2_r <= load_delay1_r;
          IF load_delay1_r = '1' THEN
            ...
          END IF;
        END IF;
      END IF;
    END PROCESS stage2;

stage3: PROCESS (clk, rstx)
    BEGIN  -- PROCESS stage2
      IF (rstx = '0') THEN
        ...
      ELSIF (clk'event AND clk = '1') THEN
        IF (glock = '0') then
          IF load_delay2_r = '1' THEN
            ...
          END IF;
        END IF;
      END IF;
    END PROCESS stage2;
```

Program 5.1 *Propagating the load signal to three individual pipeline stages for efficient clock gating.*



(a) Clock gating after synthesis tool automatic optimization.



(b) Enhanced clock gating.

Figure 5.3 *Clock gating enhancement of a pipelined FU. In (a), the load signal is used to clock gate the input registers. Later stages are only gated by global lock. In (b), the load signal is propagated to the pipeline stages for more accurate clock gating.*

In [5], several pipeline latching disciplines were proposed for TTAs, out of which *Semi Virtual-Time Latching (SVTL)* was proposed optimal for pipelined function units. In SVTL, only moves to a trigger input port start operations as opposed to *True Virtual-Time Latching (TVTL)*, where moves to all input ports start operations. Initially in our design, only the first pipeline stage was clock gated using the *load* signal in addition to the global lock signal. This is presented in Fig. 5.3(a). The latter stages were gated using only the global lock signal, resulting in inefficient clock gating. This could be fixed by propagating the load signal to the latter pipeline stages. This requires a one bit wide register for each pipeline stage and an AND port to determine the clock gating enable condition. This is presented in Fig. 5.3(b).

A VHDL code example of enabling clock gating individually for pipeline stages is presented in Program 5.1. Similar to Fig. 5.3, the pipeline has three stages, which are here separated into their own VHDL processes for the sake of clarity. the *load* signal is an input port to the FU, indicating loading of input data and the start of an operation. This signal is then propagated to the latter stages with *load_delay1_r* and *load_delay2_r* signals. During reset (*rstx*, active low), they are assigned to '0'. The synthesis tool automatically deduces the clock gating condition from *IF-THEN* clauses in each sequential process, for example for process *stage1*, these are on lines 12 and 14. Registers, that are written into inside these to clauses are gated.

5.4 Register File Banking

In contemporary ASICs, using custom RF cells instead of standard cells is usually advisable due to their optimized structure for area, delay and power consumption. These custom cells, when ordered from a semiconductor manufacturer, might already be banked for optimal performance. However, in some cases it might not be possible to achieve the optimal RF size with the custom cells, since in TTAs the RF size can have a great impact on the program cycle count. Too small size may cause extra write operations to memory and too large can cause extra power consumption.

TTAs produced with TCE take the register write address from the opcode port. The *Most Significant Bits (MSB)* were used to determine the bank number to write into. A VHDL generic *bank_size* was introduced to define the size of a single bank in the RF. Since RFs in TCE can have arbitrary depths, the banking algorithm divides the first banks into equal sizes and leaves the last one smaller, if the RF size is not divisible by the defined bank size.

Table 5.4 *Effect of register file banking on power and area.*

processor		MCU		DSP		SDR	
			Δ(%)		Δ(%)		Δ(%)
area (μm^2)						239881	+2.10
power (mW)	Coremark	0.0827	-6.97				
	DSPStone matrix	0.0898	-9.57				
	AES			3.03	-2.63		
	JPEG			2.81	-3.07		
	FFT			2.71	-2.84		
	SDRkernel1					66.3	-11.0
	SDRkernel2					59.4	-11.3

Results for the total core power consumption for the test designs with RF banking are presented in Table 5.4.

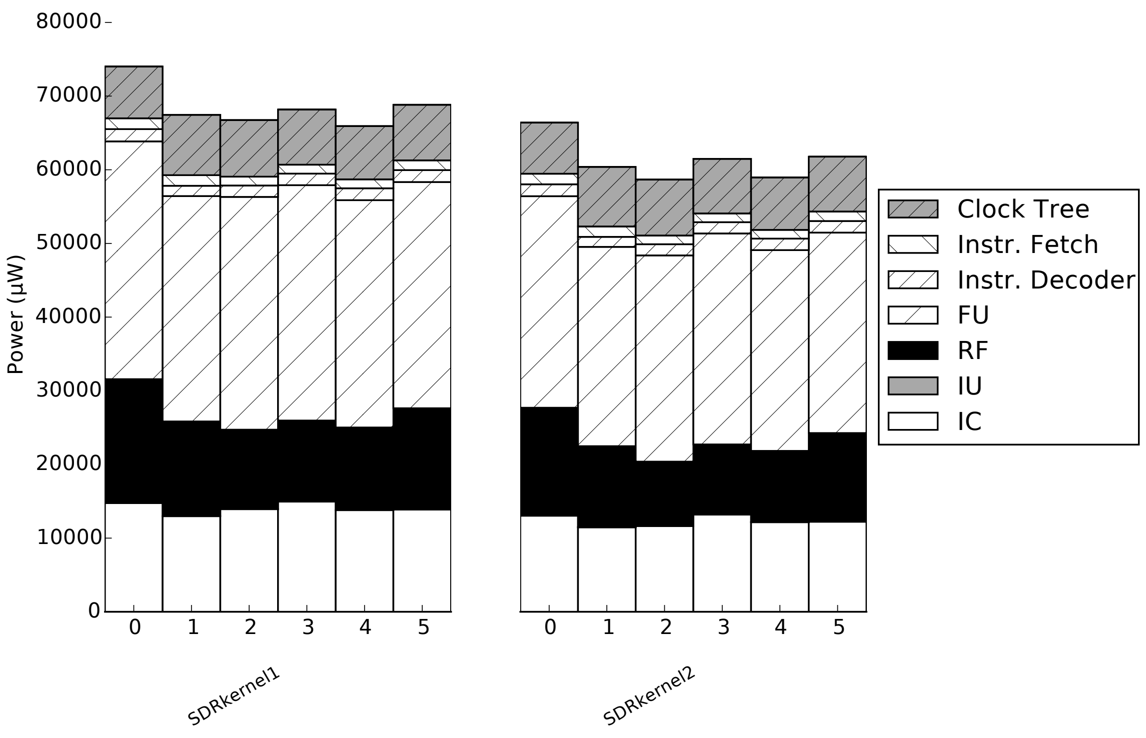


Figure 5.4 *Effect of banking the 32x512b register file on the SDR core power consumption. Number of banks on the x-axis.*

The lowest total core power consumption for the *MCU* core's 16x32b RF were achieved with bank size of four and, thus, number of banks being eight. With this configuration, the RF power consumption decreased by 46.1% with Coremark and 44.0% with DSPStone matrix.

The optimal bank size for the *SDR* core's 32x512b register file was found to be two, thus, the number of banks being 16. The total power consumption decreased by 11.0% for SDRkernel1 and 11.3% for SDRkernel2, while total area increased by 2.10%. The RF power consumption decreased 34.2% and 35.6% for the two benchmark programs. The *SDR* core benefited from RF banking the most. In Fig. 5.4, power consumption core with different RF bank sizes is compared. It can be seen, that not only the RF power decreases, but the IC power as well, due to the smaller load capacitance seen by the IC.

For the *DSP* core's 14x32b RF, optimal bank size was two, having seven banks. The RF power consumption for the benchmark programs decreased as follows: AES 35.3%, JPEG 33.6 and FFT 36.9%

5.5 Loop Buffer

The total energy saved by using a loop buffer in this particular case can be estimated if we assume that for a design without a loop buffer, an instruction is fetched from the instruction memory (SRAM) on every clock cycle. Estimating the instruction memory read energy with Cacti [23] for a 1024-byte, 1-bank, 32nm 64-bit *Static Random Access Memory (SRAM)*, we got 1.5pJ per read. For the loop buffer read energy, for a buffer depth of 32 and instruction width of 40, from synthesis results we obtained 0.28pJ. This was reached by synthesizing the *MCU* core and providing switching activity data from the loop buffer active time. The energy for a given number of instructions read from the loop buffer was calculated and divided by the amount of instructions read. The cost of energy per read for the instruction memory can be estimated to be more than five times larger compared to the loop buffer in this case.

Modeling an L1 cache with Cacti, using a 1-bank, 1024-byte, 40-bit, 32nm cache with direct associativity and 1 read/write port, we get a read energy consumption of 12.0nJ. This is 42.9 times higher than the loop buffer read energy.

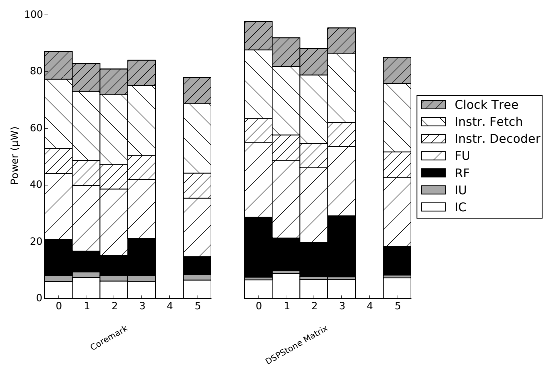
At the time of writing this thesis, programs for architectures with a loop buffer could only be scheduled using the TCE bubblefish2 scheduler. Therefore, the design without loop buffer was also scheduled with it. With the DSPStone matrix benchmark, using the loopbuffer, the cycle count decreased from 466627 to 369011, over 20%. However, using the default TCE scheduler, a cycle count of 293794 can be reached. The decrease in cycle count for the bubblefish2 scheduled programs can be explained due to missing jump condition examination at the end of the code loops. Also, the jump operation by default has a latency of four. If the schedule is not optimal, these delay slots can add to the program cycle count when comparing to one with a loop buffer.

Now, taking into account the reduced cycle count with a loop buffer and that the loop buffer is read approximately 87% of the time in this benchmark and the rest is reads from the instruction memory, we can estimate the total energy consumption to be 4.3 times larger when not using a loop buffer. The loop buffer control logic of course adds some power consumption and should be taken into consideration, but at the same time instruction fetch power consumption decreases.

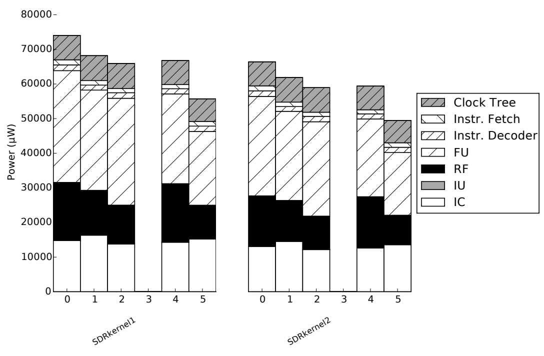
As a conclusion, using a loop buffer seems to reduce total energy consumption when comparing to using an SRAM or a cache to fetch instructions from. However, care should be taken when choosing the loop buffer depth, since larger depth infers more registers to it.

5.6 Summary of Results

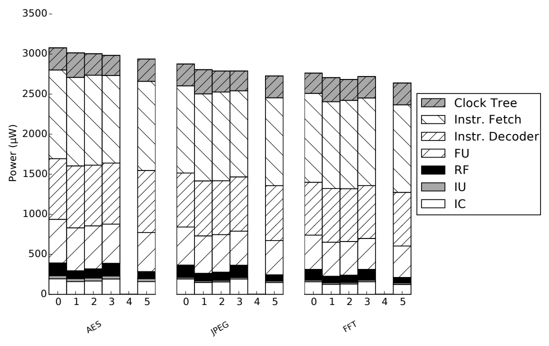
This section lists the results for all of the previous optimizations implemented together. The effect of using a loop buffer was not included to get comparable results, since the TCE default scheduler is not compatible with it at the moment.



(a) MCU



(b) SDR



(c) DSP

Figure 5.5 *Summary of implemented power optimizations for the three cores. 0 - Baseline, 1 - RF input port datapath gating, 2 - RF banking, 3 - FU operand isolation, 4 - VFPU clock gating enhancement, 5 - All optimizations combined.*

The effect of power optimizations was the smallest for the *DSP* core. In the best case with JPEG benchmark, its power consumption decreased by 4.31% with an area overhead of 4.18%. The relatively low decrease is due to to the large instruction word size and, thus, large instruction fetch and decode power consumption. The *MCU* and *SDR* core had initially larger RF and FU power consumption and since the optimizations were targeted to these components, benefited proportionally more. The *SDR* benefited most of the optimizations. This is due to it having a rather large RF word size. It was also the only core to which the pipelined FU clock gating enhancement was implemented. The *SDR* core's power consumption decreased by 26.1% in the best case with SDRkernel2, with an area overhead of 3.09%. Synthesizing all optimizations together lowered the power consumption more than any of

Table 5.5 Results for all power optimizations combined. 1 - RF input port data gating, 2 - RF banking, 3 - FU operand isolation, 4 - VFPU clock gating enhancement

processor		MCU		DSP		SDR	
optimization	1	✓		✓		✓	
	2	✓		✓		✓	
	3	✓		✓			
	4					✓	
			Δ(%)		Δ(%)		Δ(%)
area (μm^2)		5558	+2.51	16061	+4.18	242339	+3.09
power (mW)	Coremark	0.0796	-10.45				
	DSPStone matrix	0.0867	-12.7				
	AES			3.00	-3.76		
	JPEG			2.78	-4.31		
	FFT			2.68	-3.73		
	SDRkernel1					56.4	-24.8
	SDRkernel2					50.1	-26.1

the optimizations individually. This is good, since synthesis can sometimes have effects that are difficult to predict beforehand. Results for the cores are summarized in Table 5.5.

Future Work

Research regarding automatic power optimizations for TCE is ongoing. After the power optimizations implemented in this thesis, all the cores still have quite high instruction fetch and decode power consumption, which is typical for programmable designs. Next, reducing the instruction fetch power consumption will be investigated by implementing some of the instruction encoding algorithms mentioned in 3.2. The less complex encoding algorithms will be tried out first, possibly followed by the more complex but possibly more efficient algorithms. At the compiler level, loop buffer utilization is being optimized.

6. CONCLUSIONS

In this thesis, a literature review of the most commonly used techniques and methods for power optimizations on design abstraction hierarchy levels was carried out first. Next, four optimizations on the register transfer level were implemented to three Transport Triggered Architecture cores, which were chosen to represent use cases in microcontrollers, digital signal processing and software defined radio. The effect of optimizations was analyzed by synthesizing the cores with Synopsys Design Compiler.

Function unit operand isolation prevents input data from being propagated to unused operations in function units. Operations similar to each other were grouped together for best results. Example for implementation was presented in this thesis. Register file input port data gating decreases the interconnection network load capacitance by preventing unnecessary switching activity in register files and was implemented to the TTA-based Co-Design Environment processor generator. Clock gating for individual pipeline stages was enabled for the software defined radio core and an example of the implementation was presented in this thesis. Register file banking was implemented and best power consumption was achieved with bank sizes two and four, depending on the core. For best results, cores should be synthesized with various bank sizes to compare results. However, the differences in power consumption between bank sizes are small and banking the register file at least once helps.

Power consumption for all the cores decreased with the implemented optimizations. The best case reduction was 26%, with an area overhead of 3%. The worst case reduction was 3.7% with an area overhead of 4.2%. After the implemented optimizations, instruction fetching and decoding consume majority of the power in the microcontroller and digital signal processing cases. For the software defined radio case, the function units consume majority of the power, indicating good hardware utilization. Next, to address the instruction fetching and decoding power consumption, the focus of work will be on instruction (and data) encoding.

The actual work and writing the thesis took eight months. A large portion of the time was spent on automating the synthesis of the cores. This required a lot of trial and error, since the author was not familiar with Design Compiler and there did not seem to be much hands-on information on using it available publicly. The work could have been accelerated by more carefully specifying what was needed from the automatic synthesis and plot generation in the beginning of the project.

BIBLIOGRAPHY

[1] M. Bene, S. M. Nowick, and A. Wolfe, "A Fast Asynchronous Huffman Decoder for Compressed-Code Embedded Processors," in *in proceedings of the 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Washington, DC, Mar. 1998, pp. 43–56.

[2] L. Benini, G. De Mecheli, E. Macii, M. Poncino, and S. Quer, "Power optimization of core-based systems by address bus encoding," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 6, no. 4, pp. 554–562, Dec. 1998.

[3] J.-Y. Chen, W. ben Jone, J. shyan Wang, H. i Lu, and T. F. Chen, "Segmented bus design for low-power systems," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 7, no. 1, pp. 25–29, Mar. 1999.

[4] D. G. Chinnery and K. Keutzer, *Closing the Power Gap between ASIC and Custom - Tools and Techniques for Low Power Design.* Springer, 2007.

[5] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA.* Chichester, England: John Wiley & Sons, Ltd., 1998.

[6] H. Corporaal and H. J. Mulder, "MOVE: A Framework for High-performance Processor Design," in *proceedings of ACM/IEEE Conference on Supercomputing*, New York, NY, Nov. 1991, pp. 692–701.

[7] Y. He, D. She, B. Mesman, and H. Corporaal, "MOVE-Pro: A low power and high code density TTA architecture," in *proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, Samos, Greece, July 2011, pp. 294–301.

[8] J. Heikkinen, A. Cilio, J. Takala, and H. Corporaal, "Dictionary-based program compression on transport triggered architectures," in *International Symposium on Circuits and Systems*, vol. 2, May 2005, pp. 1122–1125.

[9] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*, 3rd ed. Morgan Kaufmann Publishers, 2003.

[10] H. Lekatsas, J. Henkel, and W. Wolf, "Code compression for low power embedded system design," in *proceedings of the 37th Annual Design Automation Conference*, New York, NY, June 2000, pp. 294–299.

[11] G. J. Lipovski, "Architecture of a simple, effective control processor," in *Second Symposium on Micro Architecture*, 1976, pp. 187–194.

[12] L. Macchiarulo, E. Macii, and M. Poncino, “Low-energy encoding for deep-submicron address buses,” in *proceedings of International Symposium on Low Power Electronics and Design*, Huntington Beach, CA, Aug. 2001, pp. 176–181.

[13] K. Mistry, C. Allen, C. Auth, and B. e. a. Beattie, “A 45nm logic technology with high-k+ metal gate transistors, strained silicon, 9 cu interconnect layers, 193nm dry patterning, and 100% pb-free packaging,” in *proceedings of International Electron Devices Meeting*, Washington, DC, Dec. 2007, pp. 247–250.

[14] S. Natarajan, M. Agostinelli, S. Akbar, M. Bost, A. Bowonder, V. Chikarmane, S. Chouksey, A. Dasgupta, K. Fischer, Q. Fu, T. Ghani, M. Giles, S. Govindaraju, R. Grover, W. Han, D. Hanken, E. Haralson, M. Haran, M. Heckscher, R. Heussner, P. Jain, R. James, R. Jhaveri, I. Jin, H. Kam, E. Karl, C. Kenyon, M. Liu, Y. Luo, R. Mehandru, S. Morarka, L. Neiberg, P. Packan, A. Paliwal, C. Parker, P. Patel, R. Patel, C. Pelto, L. Pipes, P. Plekhanov, M. Prince, S. Rajamani, J. Sandford, B. Sell, S. Sivakumar, P. Smith, B. Song, K. Tone, T. Troeger, J. Wiedemer, M. Yang, and K. Zhang, “A 14nm logic technology featuring 2nd-generation FinFET, air-gapped interconnects, self-aligned double patterning and a 0.0588μm^2 SRAM cell size,” in *IEEE International Electron Devices Meeting*, San Francisco, CA, Dec. 2014, pp. 3.7.1–3.7.3.

[15] E. Nowak, I. Aller, T. Ludwig, K. Kim, R. Joshi, C.-T. Chuang, K. Bernstein, and R. Puri, “Turning silicon on its edge [double gate cmos/finfet technology],” *IEEE Circuits and Devices Magazine*, vol. 20, no. 1, pp. 20–31, Jan. 2004.

[16] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital integrated circuits- A design perspective*, 2nd ed. Prentice Hall, 2004.

[17] M. R. Stan and W. P. Burleson, “Bus-invert coding for low-power i/o,” *IEEE Transactions on Very Large Scale Integrated Systems*, vol. 3, no. 1, pp. 49–58, Mar. 1995.

[18] C.-L. Su, C.-Y. Tsui, and A. M. Despain, “Saving power in the control path of embedded processors,” *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 24–30, Winter 1994.

[19] *Power Compiler User Guide*, Synopsys, Inc., Sep 2014, version J-2014.09, 454 p.

[20] *Move Project Home at TUT*, [Online], Tampere University of Technology, Available: https://www.cs.tut.fi/~move/, referenced 6/25/2015.

[21] *TTA-Based Co-design Environment Home Page*, [Online], Tampere University of Technology, Available: http://tce.cs.tut.fi/index.html, referenced 6/25/2015.

[22] *TTA-based Co-design Environment v1.9 User Manual*, [Online], Tampere University of Technology, 2014, version 29, 141 p. Available: http://tce.cs.tut.fi/user_manual/TCE.pdf, referenced 6/25/2015.

[23] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. Jouppi, *CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model*, HP Labs, Available: http://www.hpl.hp.com/research/cacti/, referenced: 6/26/2015.

[24] O. Weber, E. Josse, F. Andrieu, A. Cros, E. Richard, P. Perreau, E. Baylac, N. Degors, C. Gallon, E. Perrin, S. Chhun, E. Petitprez, S. Delmedico, J. Simon, G. Druais, S. Lasserre, J. Mazurier, N. Guillot, E. Bernard, R. Bianchini, L. Parmigiani, X. Gerard, C. Pribat, O. Gourhant, F. Abbate, C. Gaumer, V. Beugin, P. Gouraud, P. Maury, S. Lagrasta, D. Barge, N. Loubet, R. Beneyton, D. Benoit, S. Zoll, J.-D. Chapon, L. Babaud, M. Bidaud, M. Gregoire, C. Monget, B. Le-Gratiet, P. Brun, M. Mellier, A. Pofelski, L. Clement, R. Bingert, S. Puget, J.-F. Kruck, D. Hoguet, P. Scheer, T. Poiroux, J.-P. Manceau, M. Rafik, D. Rideau, M.-A. Jaud, J. Lacord, F. Monsieur, L. Grenouillet, M. Vinet, Q. Liu, B. Doris, M. Celik, S. Fetterolf, O. Faynot, and M. Haond, "14nm FDSOI technology for high speed and energy efficient applications," in *in proceedings of Symposium on VLSI Technology*, June 2014, pp. 1–2.