UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

On the design of power- and energy-efficient functional units for vector processors

Ivan Ratković

On the Design of Power- and Energy-Efficient Functional Units for Vector Processors

Ivan Ratković

Department of Computer Architecture

Universitat Politècnica de Catalunya - BarcelonaTech

A thesis submitted for the degree of

Doctor of Philosophy in Computer Architecture

October, 2016

Advisor: Dr. Adrián Cristal
Co-Advisor: Dr. Oscar Palomar
Co-Advisor: Dr. Osman S. Unsal
Co-Advisor: Prof. Mateo Valero

Abstract

Vector processors are a very promising solution for mobile devices and servers due to their inherently energy-efficient way of exploiting data-level parallelism. While vector processors succeeded in the high performance market in the past, they need a re-tailoring for the mobile market that they are entering now. Functional units are a key components of computation intensive designs like vector architectures, and have significant impact on overall performance and power. Therefore, there is a need for novel, vector-specific, design space exploration and low power techniques for vector functional units.

We present a design space exploration of vector adder (VA) and multiplier unit (VMU). We examine advantages and side effects of using multiple vector lanes and show how it performs across a broad frequency spectrum to achieve an energy-efficient speed-up. As the final results of our exploration, we derive Pareto optimal design points and present guidelines on the selection of the most appropriate VMU and VA for different types of vector processors according to different sets of metrics of interest.

To reduce the power of vector floating-point fused multiply-add units (VFU), we comprehensively identify, propose, and evaluate the most suitable clock-gating techniques for it. These techniques ensure power savings without jeopardizing the performance. We focus on unexplored opportunities for clock-gating application to vector processors, especially in active operating mode. Using vector masking and vector multi-lane-aware clock-gating, we report power reductions of up to 52%, assuming active VFU operating at the peak performance. Among other

findings, we observe that vector instruction-based clock-gating techniques achieve power savings for all vector floating-point instructions. Finally, when evaluating all techniques together, the power reductions are up to 80%.

We propose a methodology that enables performing this research in a fully parameterizable and automated fashion using two kinds of benchmarks, synthetic and "real world" application based. For this interrelated circuit-architecture research, we present novel frameworks with both architectural- and circuit-level tools, simulators and generators (including ones that we developed). Our frameworks include both design- (e.g. adder's family type) and vector architecture-related parameters (e.g. vector length).

Additionally, to find the optimal estimation flow, we perform a comparative analysis, using a design space exploration as a case study, of the currently most used estimation flows: Physical layout Aware Synthesis (*PAS*) and Place and Route (*PnR*). We study and compare post-PAS and post-PnR estimations of the metrics of interest and the impact of various design parameters and input switching activity factor (α_I).

Acknowledgements

Pursuing a Ph.D. is a multi-year endeavour that can turn into a tedious and never ending journey. That was not my case, and I am thankful to many people without whom I would not have been able to complete my Ph.D. studies. While it is not possible to make an exhaustive list of names, I would like to mention a few. Apologies if I forget to mention any name below.

Firstly, I would like to express my sincere gratitude to my advisors Oscar Palomar, Osman Unsal, Adrian Cristal, and Mateo Valero for all the help and guidance they provided during my Ph.D. studies. Their support, confidence, and sound technical advice have played a major role shaping my research ideas into the contributions expressed in this thesis.

I am thankful to Professor Borivoje Nikolić, who accepted me kindly and provided me an opportunity to join Berkeley Wireless Research Center, UC Berkeley as a visiting scholar. I had a great and productive time in Berkeley thanks to Borivoje's positive attitude and enthusiasm.

My sincere thanks go to my former supervisor during my studies at the University of Belgrade, Veljko Milutinović, for his guidance and support.

I would also like to thank Professor Carlos Alvarez his useful advice and fruitful discussions.

I would also like to acknowledge all my colleagues from the office in Barcelona Supercomputing Center that helped me throughout my Ph.D. studies; for their insights and expertise in technical matters, and for their unconditional support that has been crucial to keep me sane. My special thanks go to my colleagues from the "Vector Group", Milan Stanić, Milovan Đurić, Timothy Hayes, Nikola Bežanić, and Tassadaq Hussain for the stimulating discussions, for the sleepless nights we were working together before deadlines, and for all the fun we have had in the last six years. I acknowledge to my colleagues Srđan Stipić and Javier Arias for great technical help and tips. Besides, I am thankful to Brian Richards, Vladimir Milovanović, Yunsup Lee and all my other colleagues from Berkeley Wireless Research Center.

Finally, I would like to thank to my friends and family for supporting me during this endeavour. I especially thank my parents Lidija and Živorad and my sister Svetlana for their unconditional love and care.

This thesis has been supported by the cooperation agreement between the Barcelona Supercomputing Center and Microsoft Research, by the Ministry of Science and Technology of Spain under the FPU National Plan (FPU12/06157), by the European Union (FEDER funds) under contracts TIN2007-60625, TIN2008-02055-E, TIN2012-34557, and TIN2015-65316-P, and by the European Union's Seventh Framework Programme (FP7/2007- 2013) under the ParaDIME project (GA no. 318693).

Contents

1 Introduction

This thesis concerns the following two problems: selecting optimal vector functional unit (VFU) structure according to a variety of metrics and lowering VFU power; it also contributes methodologies that allow designers to explore two problems in a fully parameterizable, automated and comprehensive way. In this introductory chapter first we present the motivation of the thesis (Section 1.1) followed by an overview on vector processors (Section 1.2), clock-gating background (Section 1.3), and finally the thesis overview and contributions (Section 1.4).

1.1 Motivation

After the technology switch from bipolar to CMOS, in the 1980s and early 1990s, that enabled low power processing, processor designers had high performance as the primary design goal, leaving power and area as secondary goals. However, when it became apparent that further technology feature size scaling according to Moore's law (Figure 1.1) will lead to extreme power density (which could become extremely difficult or too costly to cool), power- and energy-efficiency turned into the primary design constraint for almost all computer systems.

Nevertheless, different computer systems have different demands. Let us consider two fast-growing markets such as mobile devices and data centers. The main design goal of mobile, battery constrained, devices is to achieve energy efficiency. However, computer systems like data centers suffer from both power density and

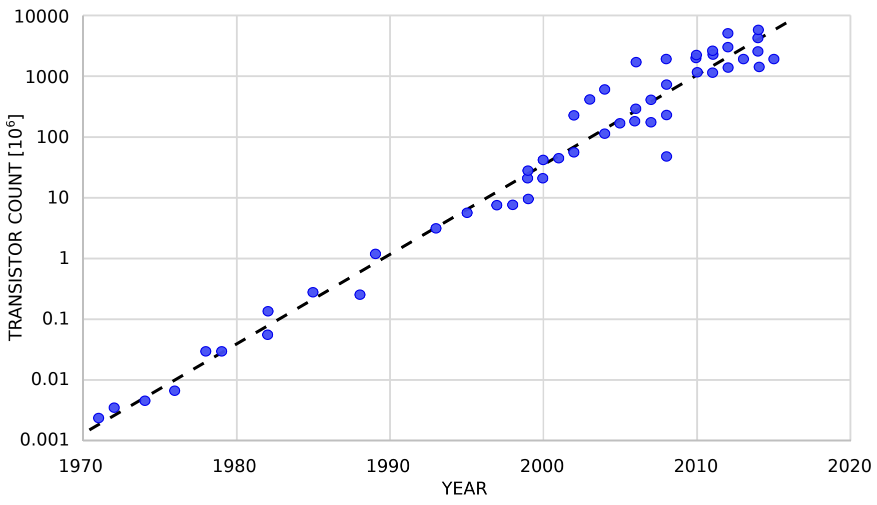


Figure 1.1: Moore's law [76, 77] illustrated on Intel's microprocessors transistor count. The number of transistor doubles every two years. Adapted from [78].

energy consumption issues. The power density issue is related to cooling problems and it has recently become even more important with the appearance of the concept of "dark silicon"[1] [35].

Nonetheless, it is still expected that each new generation of microprocessors has higher performance than the previous one. Since frequency does not scale with technology in an energy-efficient way anymore, we need changes at the architectural level that allow faster execution without a power increase [16]. In other words, we want higher performance without an excessive increase of power

Vector processors are an inherently energy-efficient architecture [7, 67, 66, 65, 118] for applications that exhibit data-level parallelism (DLP), i.e. that operate on vectors of independent elements (described in Section 1.2). It has been observed that the best vector-based machines are generally faster and/or more energy-efficient than scalar multicore processors [66]. In some cases, scalar multicores perform better though at a greater energy cost. There are server and mobile workloads with a lot of DLP. For example, Facebook's face recognition feature, running on data centers; or Google's offline voice typing system introduced in Android 4.1 (Jelly Bean). Taking all this into account, we consider that low power vector proces-

[1]Dark silicon (the term was coined by ARM [6]) is defined as the fraction of die area that goes unused due to power, parallelism, or other constraints.

sors are quite interesting designs for mobile devices and servers. These computer systems have fairly different design goals than early vector processors that were used almost exclusively for supercomputing. Although considered as a power and energy-efficient solution for workloads that exhibit data-level parallelism, vector processors were not explored sufficiently from a low power perspective in the past. Their designers prioritized performance without caring much about power, and this is not desirable in today's energy-conscious climate. Therefore, modern vector processors require a rethink and new design-space explorations of their functional units in an energy- and power-efficient manner.

There are architecture-related characteristics, specific only to vector processors, that affect functional unit (FU) usage. For example, the activity factor (described in Section 3.2.1) of a VFU is affected by architectural parameters, such as the maximum vector length or number of vector lanes. We have observed that the data of a vector are correlated, so the activity factor of a VFU is less than that of a functional unit operating with scalar data. An additional characteristic of VFUs is that they are typically fully pipelined. The reason for that is that in vector processing the critical metric is the throughput, so we typically we operate on long vectors (a long sequence of the same consecutive operations). For that reason, non-pipelined arithmetic units are not efficient in vector processing. Here we have the opposite situation than in scalar processing where we most care about the latency. Obviously, vector and scalar arithmetic units have different design goals. Therefore, an arithmetic unit's structure that is optimal for scalar architecture may not be optimal for the vector one.

FUs take a significant part in modern processor budget (Figure 1.2), especially in the case of vectors (Figure 1.3). Moreover, it has been observed that FUs are processor's thermal hotspots [99]. VFUs are likely to have even higher temperature since, due to its faster execution, a vector core dissipates more power than the equivalent scalar one [67]. This makes the concern about the hotspots more significant.

To narrow down the discussion, we focus on the most common arithmetic operations in vector processing, present in practically all vector workloads: integer and floating point addition and multiplication.

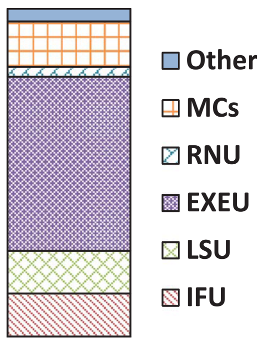


Figure 1.2: Power breakdown of an Alpha-based FinFET core. Core components: integrated memory controller (MC), renaming unit (RNU), execution unit (EXEU), load-store unit (LSU), and instruction fetch unit (IFU). Adapted from [107].

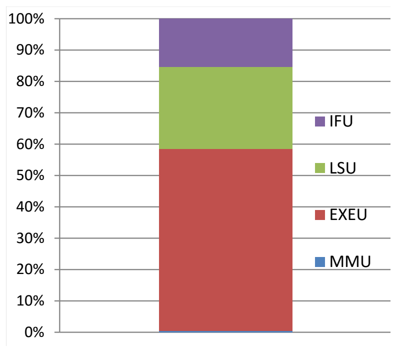


Figure 1.3: Power breakdown of ARM-based vector core [103] averaged for a number of benchmarks. Core components: IFU, LSU, EXEU, and memory management unit (MMU).

Addition is one of the most used operations in both general-purpose and application-specific processors and this also applies to vector processors. This operation is included in practically all arithmetic algorithms. Moreover, it is often in the processor's critical path [117]. Therefore, selecting an appropriate adder unit is of crucial importance for an energy-efficient vector processor design.

Multiplication is a fundamental operation in many algorithms that exhibits a lot of DLP, such as image processing. Its importance has increased in the last decade due to ubiquitous media processing. Additionally, multiplication is a common operation in most signal processing algorithms. Multipliers have large area, long latency and consume considerable power. They are a major source of power dissipation in processors specialized for digital signal processing workloads. High power dissipation in these structures is mainly due to the switching of a large number of gates during multiplication. In addition, much power is also dissipated due to a large number of spurious transitions on internal nodes. Consequently, low power multiplier design has been an important part of low power very-large-scale integration (VLSI) system design.

Floating point fused multiply-add, being a power consuming FU, deserves special attention. Abundant floating-point fused multiply-add (FMA) is typically found in vector workloads such as multimedia, computer graphics or deep learning workloads [102]. Although in the past FMAs have been used for high-performance, they recently have been included in mobile processors as well [91, 65, 118]. Moreover, energy- and power-efficient mobile processors are entering the server market as well [108]. In contrast to high-performance vector processors (e.g. NEC SX-series [74] and Tarantula [36]) that have separated units for each floating-point operation, (e.g. separate floating-point (FP) adder and multiplier) mobile vector processors' resources are limited. Thus, we typically have a single unit per vector lane capable of performing multiple FP operations rather than separate FP units [65, 118]. Additionally, having the FP computations localized inside the same unit reduces the number of interconnections, which is both power- and performance-efficient. Apart from that, FMA units offer better accuracy (single, instead of two round/normalize steps) and improved performance (shorter latency) compared to a multiplication followed by an independent addition.

A well-known method to reduce switching power in FUs (and other synchronous designs) is clock-gating (described in Section 1.3). In addition to its straightforward application to idle units, there are unexplored opportunities for this method, especially when considering active operating mode. Furthermore, there are characteristics of vector processors that provide additional clock-gating opportunities. Moreover, in addition to reducing dynamic power, clock-gating can also reduce static (leakage) power since leakage associated with CMOS devices is exponentially dependent on temperature [51, 61][1].

[1]Although leakage power has increased its importance (or weight) in recent years, when we assume low leakage libraries, it is still an order of magnitude less than dynamic power. The introduction of FinFET, high k devices, and FD-SOI has played an important role in maintaining the leakage low.

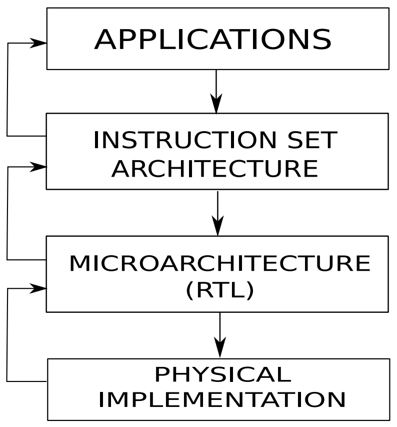


Figure 1.4: A holistic, multi-layered approach in processor design optimizations.

From the aforementioned facts we can conclude that improving energy- and power-efficiency by exploring and selecting optimal VFUs structure depending on the architectural parameters and workloads, and by applying low power techniques such as advanced clock-gating is of crucial importance. To achieve optimal results, a holistic optimization approach should be employed, i.e. bridging the gap between circuits and system should be accomplished (Figure 1.4).

We need a novel methodology that enables a joint circuit circuit-system design space explorations and optimizations using low power techniques such as clock-gating. For such a methodology we need a novel integrated framework with both architectural- and circuit-level tools. The framework should include both design- (e.g. arithmetic family type) and vector architecture-related parameters (e.g. vector length). This is necessary for aforementioned exploration as we need to observe how architectural-level parameters (e.g. vector length) affect the circuit-level metrics (e.g. adder's power dissipation) and how circuit-level parameters (e.g. adder's clock cycle) impact the execution time of a microbenchmark.

The final goal of a design-space exploration is to obtain metrics of interest (power, timing, area). Although it is not possible to measure precisely the metrics of interest of a design until it is fabricated, there are various estimation methods aimed for different design phases, which differ in levels of accuracy and estimation speed. The most accurate way to simulate a design is to use transistor-level post-place and route (PnR) data and SPICE. However, such detailed simulation is not possible in early phases of the design process. Moreover, it is impractical for a large number of test vectors and numerous design points as its computational complexity leads to an unaffordable long time frame to get the results. On the other hand, estimation processes done in early phases (e.g. post-synthesis results) are less detailed which makes them faster. Yet, decreased accuracy of these processes may lead a designer to wrong conclusions.

The most widespread estimation flows work at the gate-level. Assuming this modeling granularity, PnR models are more accurate than synthesized ones as they are closer to fabricated chips. Conversely, synthesis estimation flow deals with simpler models and requires fewer steps, which makes it faster but less accurate than PnR flow. Traditional synthesis tools use wire-load models based on fanouts which do not provide accurate wire delay information. Wiring delay cannot be ignored, and it even increases its importance with further technology scaling [57]. Physical layout Aware Synthesis (**PAS**) tends to overcome the drawbacks of traditional synthesis tools and to provide post-PAS results which are closer to post-PnR ones. The main advantage of PAS tools over the traditional synthesis flows is their floorplan awareness that provides more realistic interconnect modeling.

Selecting an appropriate estimation method for a given technology and design is of crucial interest as the estimations guide future project and design decisions. The accuracy of the estimations of area, timing, and power (metrics of interest) depends on the phase of the design flow and the fidelity of the models.

Considering design methodologies to tackle the aforementioned problems, we consider reusability and parameterization not just as an on-going trend but as a requirement. The design process has to evolve to provide a way to achieve power- and performance-efficient designs that are cost-effective. The traditional solution to face this problem, building application specific hardware instances, becomes exceedingly expensive. Therefore, there is a need for digital design rethink. An emerging solution is to design fully parameterizable design flows with hardware generators included, instead of the specific hardware instances [95, 78].

1.2 Vector Processors Background

Vector processors operate on vectors of data within the same instruction[1]. The key idea behind vector processors is to collect a set of data elements in memory, place them into a large register, operate on them, and then store them back. Vector instructions are a compact way of capturing DLP, and are a simple, scalable way to organize large amounts of computation. Vector processing receives its name because it is designed to work on long one-dimensional arrays, or vectors, of data.

Vector instruction set architecture (ISA)s provide an efficient organization for controlling a large amount of computation resources. Vector instructions offer a good aggregation of control by localizing the expression of parallelism. Furthermore, vector ISAs emphasize local communication and provide excellent computation/area ratios. Vector instructions express DLP in a very compact form, thus removing much redundant work (e.g. instruction fetch, decode, and issue). For example, a vector floating-point FMA instruction (`FPFMAV`) indicates the operation (FMA), three source vector registers and one destination vector register. Thus, tuples of three elements, one from each source register, are the inputs for the vector fused multiply-add unit (VFMA), and the result is written to the destination. All tuples can be processed independently, and multiple elements could be accommodated in a vector register.

The register file is designed so that a single named register holds a number of elements. The entire architecture is designed to take advantage of the vector style in organizing data. Additionally, the memory system of vector processors allows efficient strided and indexed memory access. The number of elements of a vector register is denoted by the maximum vector length (MV_L), and in this thesis it ranges from 16 to 128, thus including in the study both long vector and relatively short vector designs. Occasionally fewer elements than the MV_L are used, which reduces the effective vector length (EV_L). The vector register file typically consists of eight or more vector registers [49]. The number and the length of vector registers to provide is a key decision in the design of a vector unit. The configuration of a

[1] On the contrary, a "traditional", non-vector, processor can be defined as a processor that operates on scalar values, hence known as scalar processors.

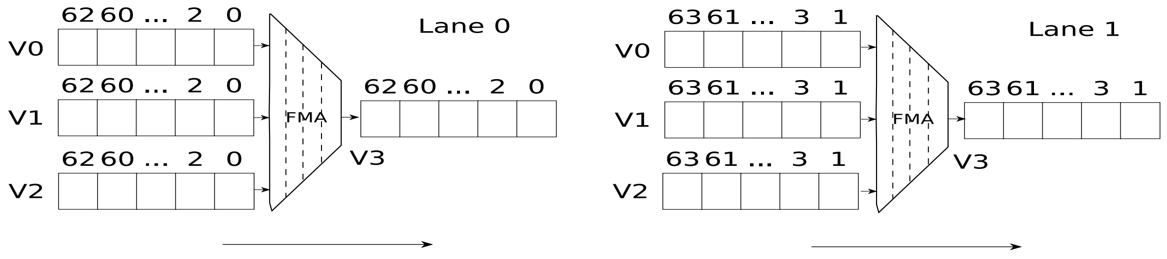


Figure 1.5: A 2-lane, 4-stage VFMA (MV_L=EV_L=64) executing `FPFMAV V3<-V0,V1,V2`.

vector register file is the programmer-visible partitioning of the vector element storage into vector registers of a given MV_L.

The vector execution model streamlines one vector register element per cycle to a fully pipelined VFU. As a result, the execution time of a vector instruction is the start-up latency (number of stages) of the VFU plus the EV_L. A common technique to reduce this time is to implement multiple vector lanes through replicated lock-stepped VFUs. Each lane accesses its own "slice" of the vector register file, which reduces the need for increasing the number of ports, typically associated with a larger number of VFUs. Lock-stepping the lanes simplifies control logic and is power-efficient. These concepts are illustrated in Figure 1.5. Using multiple lanes can increase the energy-efficiency of a vector architecture (sections 4.3.2 and 5.4).

Two techniques that significantly increase the performance are vector chaining and dead-time elimination. Chaining is vector equivalent of data forwarding; it allows a vector operation to start as soon as the individual elements of its vector source operand become available. Therefore, the results from the first functional unit in the chain are forwarded to the second functional unit. Dead-time elimination allows the processor to use ALU immediately after the current instruction.

Additionally, an interesting feature that vector processors typically offer is a vector mask control. It uses an MV_L-bit vector mask register (VMR) for indicating which operations of the vector instruction are actually performed. In other words, vector instructions operate only on the vector elements whose corresponding entries in the VMR are '1'.

Conventional vector processors should not be confused with single instruction multiple data (SIMD) multimedia extensions such as AVX-512 [9] that are an alternative way to exploit DLP and indicate operations to perform on multiple elements[1]. The main difference of these extensions with a conventional vector processor is that they exploit subword-SIMD parallelism and are typically implemented with multiple VFUs that operate on all independent elements in parallel. Having a VFU per element to operate on all them in parallel would be inefficient for vector processors because they operate on much longer vectors. Instead, VFU is fully pipelined, and the elements of the vector register are streamlined to the unit, one per cycle, possibly using a small number of vector lanes.

The reference vector architecture that we use in this thesis is described in Chapter 2.

1.2.1 Power- and Energy-Efficiency

Vector processors provide power- and energy-efficiency in several ways [7]:

- **Instruction fetch.** The most obvious reduction is in instruction fetch, decode, and dispatch, as mentioned above. For vectorizable code, a vector unit significantly reduces the number of instruction fetches. Vector instructions also remove much of the interlock and dispatch logic overhead. These concepts are illustrated on Figure 1.6 by showing an example of the reduced instruction count compared to an equivalent execution on a scalar processor.

- **Register file access.** Operations within a vector instruction access the vector register file in a regular pattern, so a high-bandwidth vector register file can be built from smaller, fewer-ported banks. In contrast, a superscalar architecture with its flexibility to access any combination of registers for any operation requires full multiported access to the entire register file storage.

- **Datapath data.** Vector instructions group similar operations, therefore, it is likely that there is much greater bit-level correlation between successive elements in a vector than between successive instructions executed in a scalar processor. This reduces power dissipation (see Section 4.3.1).

[1] Vector processors are SIMD architectures in Flynn's taxonomy [40], although by SIMD we refer to such type of multimedia extensions.

```
for (i=0; i<8; i++) v2{i}=v0{i}+v1[i];
```

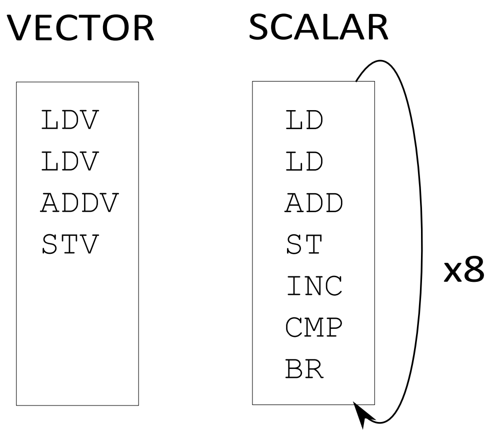


Figure 1.6: Vector vs. scalar processor instruction count using array addition as an example. In the scalar case we need to execute 8 times the listed 7 instructions, while in the vector case we need just 4 vector instructions.

- **Datapath control lines.** A VFU executes the same operation on a set of elements, so datapath control signals are only switched once per vector. This should reduce switching activity compared to a scalar architecture where different types of operation are time multiplexed over the same FUs, and hence datapath control lines are toggled more frequently.

- **Memory accesses.** Vector memory operations present regular data access patterns to the memory system, which enables further energy savings. For example, unit-stride vector memory accesses may only require one, or at most two, TLB accesses per vector of operands.

The main sources of potential increases in switching activity per operation are in structures that provide inter-lane communication, including control broadcast and the memory system crossbar. This inter-lane cost can be reduced by using highly vectorizable algorithms that avoid inter-lane communication, and by adding caches adapted for vector processors that reduce lane interactions with memory.

1.3 Clock-Gating Background

Clock-gating is a well known method to reduce switching power in synchronous pipelines [85]. It is a widely used power saving optimization method and practically a standard in low-power design. It is the most efficient power reduction technique for active operating mode, especially because dynamic power remains dominant over the static power for logic. The early unpopularity of clock gating was due to the inability of the tools of that time to deal with the timing implications of the gated clock signals and by the reduced fault coverage achieved by logic testers [81]. Clock gating was originally conceived as a system level power optimization technique aiming to reduce the power dissipated on the clock network by deactivating parts of the system that are idle. Its applicability has been extended to the register level as a power efficient implementation of registers on a hold condition.

The goal is to "gate" the clock of any component whenever it does not perform useful work. The underlying circuit mechanism is presented on Figure 1.7. A register candidate for clock-gating is shown on the upper part of Figure 1.7. During a hold condition, the register preserves its previous value at a high power cost. Unnecessary power is dissipated on: (1) the clock tree with its buffers and clock lines, (2) the register itself, (3) logic between the registers, and (4) the multiplexors on the feedback paths. By controlling the clock driving the clock input of the register (lower part of Figure 1.7), reloading is only conditionally performed resulting in both reduced power consumption and area overhead. Due to the high potential savings at low cost, clock-gating is fully automated in most commercial synthesis tools [32, 106].

Since it can reduce power significantly, the conditions under which clock-gating can be applied should be extensively studied and identified. A widely used approach is to clock-gate a whole FU when it is idle. An advanced approach is to use a finer granularity (than the whole FU) and to clock-gate FUs on the internal pipeline stage level i.e. only stages where instructions are present are clocked. We call this approach idle clock-gating technique - *IdleCG* (see Section 6.4.5). A complementary, more advanced and challenging approach is to clock-gate the FU or its sub-blocks when it is active, i.e. operating at peak performance. We call these

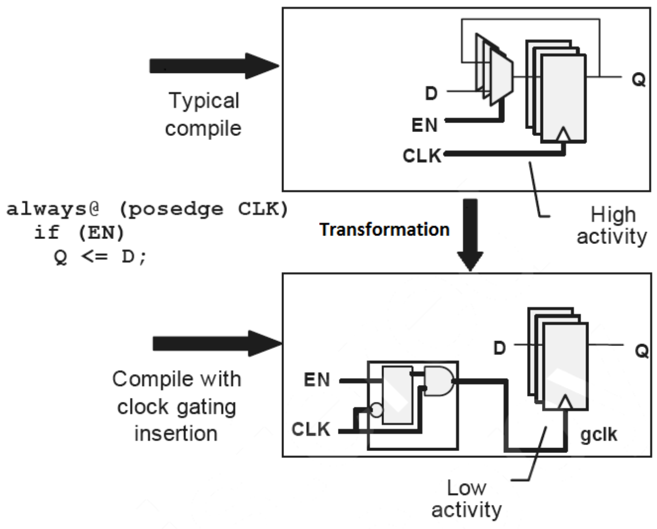


Figure 1.7: Clock-gating transformation. Adapted from [39].

techniques active clock-gating techniques - *ActiveCG* (see sections 6.4.1, 6.4.2, 6.4.3, and 6.4.4).

While on the circuit level clock-gating is standardized and in most cases automated with library, the real challenges are at the architectural level as an extensive investigation has to be done in order to explore the conditions under which clock-gating can be applied (i.e. *Enable* signal generation). In order to prevent timing overheads, enable signal generation has to be done on-time, i.e. it must not be on the critical path.

Regarding alternative gating low power techniques, an important technique is power gating [56, 3, 94]. It is especially important in lowering the leakage power. Reasons why we studied clock-gating rather than power-gating are:

- The main goal of clock-gating is dynamic power suppression, while the main goal of power gating is leakage power suppression. In our experiments, the leakage is practically negligible, and the reasons are explained in sections 3.2.1 and 6.6.

- Clock-gating has much finer granularity than power-gating, thus allowing us to gate not only the whole VFU but its components as well.

- Clock-gating has low or no timing overhead which is not a case with power gating [56, 62]. Turning power gating mechanism on and off incurs timing overhead, thus, it is more suitable for lowering power in idle mode (long idleness) than in active operating mode (short idleness). During active operating mode (the one that this research targets) gate/ungate frequently happens (often per cycle), thus the overhead of the mechanism can be unacceptable.

- Power-gating physical implementation is more complex and require dealing with power and signal integrity issues [1, 96, 115].

1.4 Thesis contributions and overview

In this thesis we try to solve the problems listed in the Motivation (Section 1.1) with the following main thesis' contributions:

- A deep multi-level investigation on achieving high power and energy efficiency of VFUs, in particular of: vector adders (Chapter 4), multipliers (Chapter 5) and IEEE 754-2008 compliant floating point FMA (Chapter 6). We have decoupled 32-bit vector machine with support for 64-bit floating point and TSMC40LP as target architecture and technology, respectively. We achieve these goals mainly in two complementary manners.

  – We identify, propose, and evaluate in-depth the most suitable clock-gating techniques for VFU running at peak performance periods without jeopardizing performance (Chapter 6). We look beyond classical clock-gating: we examine additional opportunities to avoid unnecessary activity in vector workload executions. We present the first proposal of active clock-gating techniques for VFU. Using only one of these techniques can achieve up to 52% of power reductions. Savings of up to 80% are available when the techniques are applied together on a VFU running a variety of benchmarks. This is a consequence of the fact that clock-gating efficiency (percentage of time that clock-gating technique(s) is used) is quite high, often close to 100%.

  - A comprehensive (power, delay, energy, area, power density) joint circuit-system design space exploration of vector adders (Chapter 4) and multipliers (Chapter 5) for vector processors. The exploration includes circuit-level parameters like the number of pipeline stages, clock-gating support, and arithmetic family, as well as architectural parameters such as MV_L and benchmarking. We discover that vector multi-lane is useful for achieving low *Energy-Delay* products, and beats increasing frequency as a measure to achieve energy- and thermal-efficient speed-up, especially for long vectors lengths. Finally, we provide guidelines on optimal vector adder unit (VA) and vector multiplier unit (VMU) configuration selection for different low power vector architectures.

- A novel methodology that includes exploration frameworks. In order to join architectural-level information (e.g. microbenchmarks) with circuit-level outputs (e.g. power measurements), and accomplish in that way all aforementioned explorations, we developed automated and integrated architecture-circuit exploration frameworks (different variants of frameworks are present in all the chapters) that consist of several generators and simulators at different levels. The frameworks assume both synthetic and real application-based benchmarking (SPEC [101], SDVB [112], and STAMP [72]) and include a variety of parameters at all the abstraction levels from Figure 1.4. Additionally, it includes both hardware (Chisel-based) and software generators. The use of generators facilitates design space explorations through sweeping the parameters of the design. The basis of the frameworks is explained in Chapter 3.

- Development of physical layout aware synthesis (PAS) and PnR estimation flows and their comparative analysis in terms of power, timing, area, and the flow completion time using adders as a case study (Chapter 3). We analyze side-effects of clock-gating, pipelining, and extensive timing optimizations. We perform this analysis using various design parameters, including switching activity factor input switching activity factor (α_I). The estimation flows serve as a basis for the above mentioned frameworks.

The aforementioned chapters that present these contributions are complemented by Chapter 2 that describes reference vector architecture used in the rest of the chapters, as well as vector simulator that we use. Additionally, chapters 8 and 7 list the publication of the thesis and conclude the thesis, respectively.

2 Reference Vector Architecture

In this chapter, we present vector microarchitecture that we use as an assumed reference in the rest of the thesis[1]. For vector architecture modeling we use vector simulator *VectorSim*. The main configuration parameters are listed in Table 2.1 and explained in subsequent chapters.

2.1 VectorSim

We built VectorSim based on the vector architecture library (*VALib*) and the *SimpleVector* simulator (both developed in our research group [102]). VALib is a library that implements vector instructions and allows rapid manual vectorization and characterization of applications. SimpleVector is a simple and very fast trace-based simulator which helps to estimate the performance of a vector processor. We took advantage of the fact that both tools have been designed to be easily extended with new instructions or implementation alternatives. Therefore, we modify them to satisfy our research goals and to enable its integration in our exploration frameworks. High-level *VectorSim* configuration is presented in Table 2.1.

[1]Vector processing background is provided in Section 1.2.

[1]VFU and ALU latencies are variable as they are design parameters in our research (see sections 3.3.2, 4.2.2, 5.2.2, and 6.5.3)

Table 2.1: High-level *VectorSim* configuration.

Execution	32-bit in-order vector core; decoupled vector and scalar core; vector chaining.
Vector Register File	MV_L: 16, 64, or 128 elements; Number of registers: 8.
VFU	n_L: 1, 2, or 4; 1 32-bit arithmetic logic unit (ALU), 1 64-bit VFMA[1].
Memory System	L2 (direct access to L2, shared with scalar core): 1MB, 4-way, 128b cache line, hit latency: 7ns, miss latency: 70ns; 1 load unit, 1 store unit.

2.2 Execution

We setup *VectorSim* to model a decoupled 32-bit vector machine with support for 64-bit floating point. The decoupled execution model assumes separated in-order vector and scalar execution units [36, 2, 65, 118]. They share instruction fetch and decode, and they separate issue logic and functional units, allowing in that way independent scalar execution. In-order execution is common in low power processors and performs more efficient in vector than in scalar processing as in vector architectures the drawbacks of in-order execution are diminished, especially if the vectors are long. As a result, the FUs are kept busy. Also, many modern low power processors are in-order due to its simplicity (e.g. Intel Bonnell (Atom) [59] and some of ARM Cortex-A architectures [27, 28, 24, 25, 26]). Additionally, we model chaining and dead time elimination. For MV_L, in our experiments, we choose the following values: 16, 64, and 128.

2.3 Instruction Set Architecture

We extend the ISA of *VaLib*, which is inspired by traditional register-based vector machines, e.g. CRAY and CONVEX. Among other upgrades, we added a set of vector FMA instructions. The instructions can be grouped into the following classes:

- Arithmetic and logical. These instructions are common operations such as addition, multiplication, logical bitwise operations, etc. They can operate in vector-vector or vector-scalar mode. The exception is the FMA instruction that has 3 operands and more combinations of vector and scalar operands (see Table 6.4). Most instructions support masking.

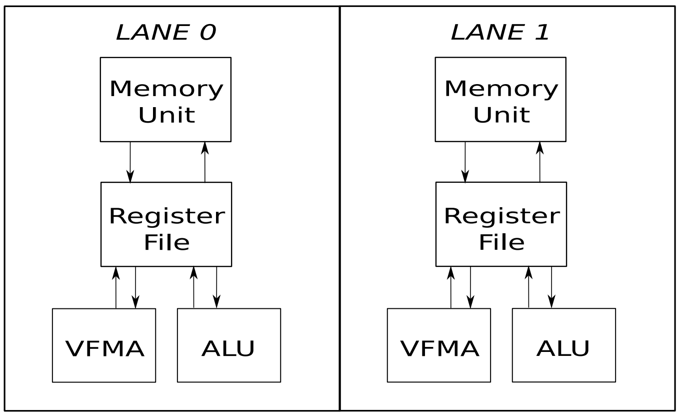


Figure 2.1: A 2-lane vector datapath.

- Memory. There are instructions for unit-stride, strided and indexed access. Some variants support masking, which is useful to access elements conditionally.

- Reduction. This class includes sum, max, and min.

- Bit/element manipulation. Examples of this class include instructions to read or write individual elements of a vector register, or to compress a register according to a vector mask.

2.4 Vector Register File

Our configuration assumes 8 vector registers. We find this number is sufficient for all the workloads we examined. Each lane has its own slice of register file. We rely on the regular access pattern (a characteristic of vector architectures) in order to keep number of R/W ports small. For a low power vector processor it is important to keep the usage of vector register file low. In order to achieve that, the compiler plays a big role. The vector part of the code that we use is compiled by hand, so we avoid unnecessary usage of vector registers.

A 2-lane vector data path is depicted on Figure 2.1, where we can see a 2-lane partitioning of the vector register file.

2.5 Memory System

Vector load-store unit. The vector memory unit handles the memory instructions and loads or stores a vector to or from cache memory. The vector loads and stores are fully pipelined, so that words can be moved between the vector registers and cache memory with a bandwidth of 1 word per clock cycle (if accesses hit the cache), after an initial latency. We have 1 load and 1 store unit. More than that would exceed a low power processor budget.

Cache memory.

Cache configuration (and memory system in general) has significant impact on the execution time for vector processors. In our model, the vector unit directly accesses the L2 cache (shared with the scalar unit). This technique has been proposed to facilitate the inclusion of vector units in microprocessor designs [84, 36, 2, 65, 118]. Accessing the L2 instead of the L1 allows not modifying the design of the L1 interface in the scalar core, which is a critical component of the design, typically in the critical path. By leaving this interface unmodified, we can safely focus on the vector component of the design. Vector code benefits from the large cache lines that L2 can provide. By interfacing the cache hierarchy we can provide enough bandwidth and exploit data locality. As it is larger than L1, L2 is more suitable for multi-banking implementation. Vector processors benefit from using cache banks rather than interleaving mainly due to efficient access to words that are not sequential.

2.6 High-Level VFU Configuration

The vector execution engine is organized as n_L identical vector lanes (Figure 2.1). Possible values of the number of lanes (n_L) are 1, 2, and 4. In our experiments, we do not examine more lanes as it would not satisfy well a low power core budget[1]. Moreover, values that we choose are typical in vector processor design [49]. Each lane has a slice of the vector register file, a slice of the vector mask file, 1 vector integer ALU, 1 VFMA, and a private TLB. There is no communication across lanes,

[1]The total number of functional unit per core is in accordance with many other low power processors [27, 28, 24, 25, 26, 59, 60, 83]

except for gather/scatter, reduction, and compress instructions. In addition to the vector ALU, each lane also includes 1 logic unit that handles logic operations, shifting and rotating. We assume the division is done in software, since it is rare in vectorizable applications and the hardware support is costly. Additionally, a control unit is needed to detect hazards, both from conflicts for the functional units (structural hazards) and from conflicts for register accesses (data hazards).

3 Estimation Flows

3.1 Introduction

In this chapter, we perform a comparative analysis of PAS and PnR flows using design space exploration of low power 32-bit adders as a case study. Additionally, this chapter serves as a foundation for the rest of chapters. These estimation flows are very important parts of the methodology that we propose and use across the thesis.

We study and compare post-PAS and post-PnR estimations of the metrics of interest and the impact of various design parameters and input switching activity factor (α_I). Specifically, we compare estimations in terms of area, timing, and power for several design parameters: clock cycle, adder family, number of pipeline stages, and clock-gating. We study how adder area, timing and power differ with respect to the applied estimation flow. Adders are particularly interesting for this study because they are fundamental microprocessor units, and their design involves many parameters that create a vast design space. A design space exploration with a large number of design points allows observing advantages and drawbacks of estimation flows in a comprehensive way. We show cases in which the post-PAS and post-PnR estimations could lead to different design decisions, especially from a low-power designer point of view. We examine and explain in which cases post-PAS results are (or are not) reliable, compared to post-PnR (since PnR ones are more accurate and close to post-fabrication results).

Since power has become the primary design constraint for the majority of computer systems, we pay special attention to power dissipation. In contrast to area and timing estimation of a VLSI design that does not require knowledge of the inputs, we cannot have a solid total power estimation without information about the inputs. Therefore, we evaluate how the power numbers obtained in post-PAS and post-PnR estimations vary with input switching activity factor (α_I).

3.2 Metrics

This section defines the metrics of interest that are used in the rest of the chapter and the thesis. We can define two types of metrics which are used in digital design - basic and derived metrics.

3.2.1 Basic Metrics

Switching activity factor (α) of a circuit node is the probability the given node will change its state from 1 to 0 or opposite at a given clock tick. Activity factor is a function of the circuit topology and the activity of the input signals. The activity factor is necessary in order to analytically compute (estimate) dynamic power of a circuit.

Capacitance (C) is the ability of a body to hold an electrical charge, and its SI unit is the *farad* (F). Capacitance can also be defined as a measure of the amount of electrical energy stored (or separated) for a given electric potential.

Energy (E) is defined as the ability that a physical system has to do work on other physical systems and its SI unit is the *joule* (J). The total energy consumption of a digital circuit can be expressed as the sum of dynamic and static energy:

$$E_{tot} = E_{dyn} + E_{stat}. \tag{3.1}$$

Dynamic energy has three components that are the result of the next three sources: charging/discharging capacitances (switching), short-circuit currents, and glitches. For processor designers the most relevant energy is the switching one, as the others components are parasitic, thus are not directly affected with architectural level low power techniques. According to the general energy definition, dynamic energy in digital circuits can be interpreted as: When a transition in a digital circuit occurs (a node change its state 0→1 or 1→0) some amount of electrical work is done; thus, some amount of electrical energy is spent. In order to obtain an analytical expression of dynamic energy, a network node can be modeled as a capacitor C_L which is charged by voltage source V_{DD} through a circuit with resistance R. In this case the total energy consumed to charge the capacitor C_L is $E = C_L V_{DD}^2$, where the half of the energy is dissipated on R, and half is saved in C_L, $E_C = E_R = \frac{C_L V_{DD}^2}{2}$.

The total static energy consumption of a digital circuit is the result of leakage and static currents. Leakage current I_{leak} consists of drain leakage, junction leakage, and gate leakage current, while static current I_{DC} is bias which is needed by some circuits for their correct work. Static energy at a time moment t $(t > 0)$ is given as follows:

$$E(t) = \int_0^t V_{DD}(I_{leak} + I_{DC})d\tau = V_{DD}(I_{DC} + I_{leak})t. \tag{3.2}$$

Delay (D). In processor design, delay has two definitions, depending on the abstraction level (layers from Figure 1.4) that we consider. At the circuit level, it is propagation delay, and is defined as the difference between the time when the input to a logic gate becomes stable and valid and the time when the output of that logic gate is stable and valid. It usually refers to the time required for the output to reach 50% of its final output level when the input changes. However, at the architectural level, by delay we assume the execution time of an application kernel ($\boldsymbol{t_e}$). In this thesis, we assume the latter definition.

Power (P) is the rate at which work is performed or energy is converted, and its SI unit is the *watt* (W). Average power is given with $P = \frac{\Delta E}{\Delta t}$, in which ΔE is amount of energy consumed in time period Δt. From architectural perspective, average is more important than instantaneous power. Like energy, power dissipation sources in digital circuits can be divided into two major classes: dynamic and static.

Dynamic power dissipation, similar to dynamic energy consumption, consists of switching (P_{switch}), short-circuit (P_{sc}) and glitching (P_{glitch}) power:

$$P_{dyn} = P_{switch} + P_{sc} + P_{glitch}. \tag{3.3}$$

From references [111, 38, 86] we have:

$$P_{switch} = \frac{1}{2}\alpha f C_L V_{DD}^2, \quad P_{sc} \propto \alpha f V_{DD}^3/C_L, \text{ and } \quad P_{glitch} \propto \alpha f V_{DD}^2. \tag{3.4}$$

Static power in CMOS digital circuits is a result of leakage and static currents (the same sources which cause static energy):

$$P_{stat} = V_{DD}(I_{DC} + I_{leak}). \tag{3.5}$$

Further, P is typically structurally divided into power of adder's clock tree and of the rest of the adder: $P = P_{ct} + P_{rest}$

Generally, in our research, we observe from the results that P_{stat} is typically two orders of magnitude less than P_{dyn}. The leakage is negligible due to the following reasons: (1) arithmetic' topologies produce high switching (high P_{dyn}), (2) the technology that we use has low leakage, and (3) we optimize non-critical path logic for leakage using high-V_{TH} cells. Therefore, the power of an arithmetic unit (P) is practically equal to its dynamic component ($P \approx P_{dyn}$).

3.2.2 Derived Metrics

Energy-Delay Product (***EDP***) - While low power often used to be viewed as synonymous with lower performance that is no longer the case. In many cases, application runtime is of significant relevance even in energy- or power-constrained environments. With the dual goals of low energy and fast runtime in mind, energy-delay product (***EDP***) was proposed as a useful metric [46]. ***EDP*** offers equal "weight" to either energy or performance degradation. If either energy or delay increases, the ***EDP*** increases. Thus, lower ***EDP*** values are desirable.

Energy^i-Delay^j product (ED^{i}P^j) - ***EDP*** of a design tells how close the design is to a perfect balance between performance and energy efficiency. Sometimes, in real designs, achieving that balance may not necessarily be of interest. Typically, one metric is assigned greater weight, for example, energy is minimized for a given maximum delay or delay is minimized for a given maximum energy. In the high-performance arena, where performance improvements matter more than energy savings we need a metric which has $i < j$, while in low-power design we need one with $i > j$ $(i, j > 0)$. For example, a commonly used performance oriented metric is $ED2P$ [71].

Surface power density (P_d) - Another related metric which is defined as power per unit area:

$$P_d = \frac{P}{A}, \tag{3.6}$$

and its SI unit is $\frac{W}{m^2}$. This metric is the crucial one for thermal studies and cooling system selection and design, as it is related with the temperature of the given surface by *Stefan-Boltzmann law* [104].

3.3 Methodology

This section describes the methodology, framework, and flow parameters used to perform this extensive comparison of PnR and PAS estimation flows. By flow parameters, we assume design parameters and α_I (explained in sections 3.3.2 and 3.3.4 respectively).

3.3.1 Framework

A simplified block diagram of the framework is depicted on Figure 3.1. As a PAS tool and post-PAS results estimator we use Cadence RTL compiler [32] in Physical Layout Estimation mode while for PnR and post-PnR estimations we use Cadence Encounter Digital Implementation System [31]. We developed several scripts for automation and interfacing between tools. PAS is the first common part of both PAS and PnR estimation flows. Each design point is defined by four parameters, adder family (*AF*), n_S, *CGable* and T_{CLK} (described in Section 3.3.2). We wrote all the hardware description language (HDL) code by hand for the adders of each valid combination of the *AF*, n_S, and *CGable* parameters. For each design point, we supply the HDL code of the adder together with a target clock cycle (the T_{CLK} parameter) to the PAS tool to produce adder's mapped netlists. We perform PAS for all combinations of our design parameters in an automated way. The PAS tool tries to meet timing constraints (T_{CLK}) while prioritizing power over area

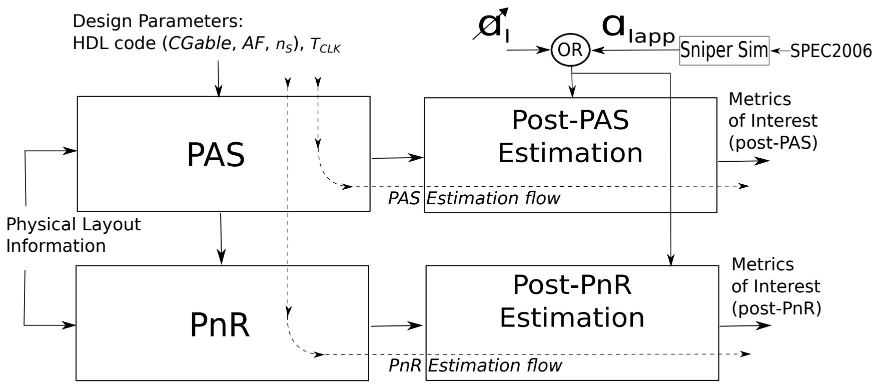


Figure 3.1: Block diagram of the framework used to perform the comparative analysis of PnR and PAS estimation flows in terms of area, timing and power (metrics of interest).

optimization. Once we have all the netlists produced using PAS, we perform post-PAS estimation of the metrics of interest (area, timing and power), and complete the PAS estimation flow. Both post-PAS and post-PnR α_I-based power estimations are explained in Section 3.3.4. In order to complete the PnR estimation flow we further process our designs and perform PnR. The last stage of the PnR estimation flow is post-PnR estimation of the metrics of interest. Additionally, the most critical paths are verified with Cadence Spectre [97]. This step is not shown in Fig. 3.1 for the sake of simplicity.

All designs are implemented using the mentioned TSMC40LP library for *typical* operating conditions. Since practically all existing vector processors (and vector processors exploration is the final goal of the research presented in this chapter) are developed using standard cells [49], we selected this approach in our research. We chose effective current based model (ECSM)[18] as it delivers accuracy to within 2% of SPICE [19]. It is more accurate than conventional models like non-linear delay model (NLDM) which differ as much as 20%.

We perform PAS and PnR under the same conditions. PAS tool requires basic physical layout information in order to perform its physical layout aware synthesis. This is a subset of the complete set of physical layout parameters required by the PnR tool. For example, core density is a fundamental and basic layout parameter required by both tools. An initial core density of 70% is selected as a sensible balance between timing improvement and shrinking area for the wide set of adder designs parameters that we use. We experimentally found that, in PnR stage in general, density below 70% sometimes provides negligible faster timing for a non-negligible area overhead while densities higher than 70% can spoil timings noticeably as the tool suffers from the lack of free space for optimizations and routing. Additionally, the initial densities below 70% sometimes even cause DRC errors and density violations. Regarding another important parameter used by both flows, number of routing layers, we use 4 since we experimentally found that using more would be a waste of resources as the additional layers do not improve the quality of results (QoR).

The optimization techniques that are applied in the PAS stage are: adding buffers, resizing gates, remapping logic, swapping pins, boundary optimizations (removing undriven or unloaded logic connected, propagating constants, collapsing equal pins, and rewiring of equivalent signals across hierarchy), and deleting buffers. However, we prevent restructuring optimization techniques that could change the adder structure, in order to maintain the original topology of each adder family. The optimization techniques done in the PAS stage are applied again in PnR since there is more optimization space in PnR than in PAS. Additionally, two more techniques are applied in PnR: moving instances and applying useful skew. PAS optimizations are run with high effort, while PnR ones with medium effort rather than high in order to keep PnR runtime affordable for a large number of design points. More details about the optimizations are available in the documentation of the aforementioned Cadence tools.

The necessary times for the complete PAS and PnR estimation flow for a single design are around 70 and 400 seconds in average, respectively. Thus, PAS is around 5.7 times faster than PnR estimation flow. The time needed to perform PnR is fairly reasonable for a small number of designs. However, the difference in execution time of the estimation flows becomes an important issue when we have experiments with a large number of design points. For example, in this design space exploration the number of design points is over 2000, which puts the difference of time needed between the flows completion into days. Moreover, for larger and faster designs this ratio increases as the PnR tool makes more effort on routing and optimizations. Therefore, we can expect the difference between these flows to be higher for large and more complex designs (e.g. accelerator or processor's core).

The framework is built for adders, but it can be easily accommodated for any other kind of arithmetic block. For example, in order to adapt the current framework to perform the same kind of comparative analysis for multipliers, practically the only change would be to supply the HDL code of the multipliers instead of current HDL code. To make it fully compatible with this framework, the new HDL code should, of course, incorporate the same design parameters (*AF*, n_S, and *CGable*).

3.3.2 Design Parameters

In this section we present the design parameters used in this comparative analysis of estimation flows:

- ***CGable*** indicates whether a VFU (in this case an adder) is implemented with or without support for clock-gating (*CG* or *noCG*, respectively). Clock-gating prevents useless switching activity in circuits and makes pipelining more effective. We assume the clock-gating decision is done in the issue stage. It is done at the stage level, i.e. each stage is activated with its *Enable* signals only when the data on its inputs is valid. We implement clock-gating in the HDL code and use latch-based clock-gating cells from TSMC40LP library. This clock-gating technique is called *IdleCG* and it is additionally explained and evaluated in Section 6.4.5. For simple structures like 32-bit integer adders and multipliers, there is no enough room for fruitful application of more advanced techniques like the ones presented in sections 6.4.1, 6.4.2, 6.4.3, and 6.4.4.

- *AF* is the adder family (algorithm). We choose five adder families: Brent-Kung (*bk*), Kogge-Stone (*ks*), carry-lookahead (*cla*), ripple-carry adder (*rca*), and conditional-sum adder (*cosa*), all implemented in HDL code. We choose *rca* as it is a basic algorithm and it is the simplest to implement (as for each bit it practically requires only one full adder cell), *cla* as a basic model of fast adders, and prefix (*bk* and *ks*) and *cosa* adders as they are fast and suitable for pipelining. All the adders that we use in this thesis are explained in Section 3.3.3. We obtain the results for static CMOS. However, as it is observed in [80], the topology that is the most energy efficient in one logic style, is also the most energy-efficient in the other styles.

- $\boldsymbol{T_{CLK}} = (1/f)$ is the clock period of the adder (and the system), while f is its clock frequency. We define 0.1-5.0 GHz as a frequency range in order to explore a vast design space.

- $\boldsymbol{n_S}$ indicates the number of pipeline stages in the adder (pipeline depth), and it ranges from 1 to 8. We only increase n_S for a particular *AF* if it provides shorter T_{CLK} with PAS. In particular, we implement *bk*, *ks*, *rca* for 1-8, *cosa* for 1-7, and *cla* for 1-4 stages. The structures of all the observed adders except *cla* are regular, so pipelining these adders is done in a fairly obvious way. *cla* does not have a structure that allows such obvious way to pipeline it, so in this case we need to apply *ad-hoc* pipelining. Since the adders are pipelined and have a clocked input register (first pipeline stage) they can be easily incorporated into a datapath. The input register of an *n-stage* adder consists of 2x32 DFlipFlops (*DFF*) for the input operands and one DFF for the carry-in bit, and in case of *CG* we have *n* additional DFFs for the *Enable* signal. Additionally, each *n-stage* adder has *n-1* pipeline clocked registers (consisting of DFFs) which are used to save intermediate results. The size of each register depends on the *AF* structure and the place inside the adder where the register is inserted (i.e. on the way the adder is pipelined). We include in this study high numbers of pipeline stages because we need high frequencies for vast design space exploration, and because we want to study the impact of deep pipelining.

3.3.3 Adder Families

This section presents the adder families used in this chapter and Chapters 4 and 6. Even more detailed explanations of each *AF* are available in [14, 33].

Ripple-Carry Adder (*rca*)

The *rca* is $O(n)$ for both time and area, where n is the width of the operands. In the worst case, a carry can propagate from the least significant bit position to the most

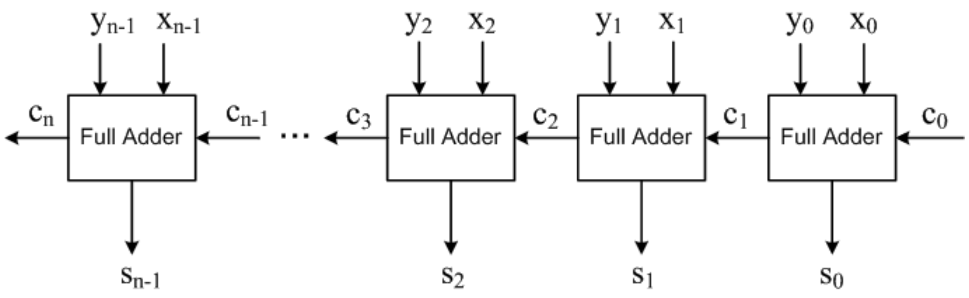


Figure 3.2: n-bit ripple carry adder (*rca*).

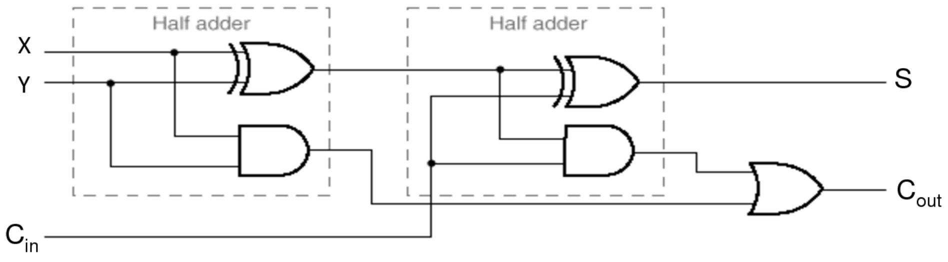


Figure 3.3: Full adder (*FA*) and half adder (*HA*).

significant bit position. A block diagram of the RCA is shown in Figure 3.2.

The basic unit is full adder (FA), also referred to as a (3,2) counter, computes a sum bit and a carry bit, $s_i = x_i \oplus y_i \oplus cin_i$, $c_{i+1} = x_i y_i + x_i c_i + y_i c_i$. That is, it adds two operand bits with the incoming carry bit to produce a sum bit and an outgoing carry bit. Its block diagram is shown on Figure 3.3

When there is no incoming carry bit, i.e. when it is assumed to be '0', half adder (HA) is used: $s_i = x_i \oplus y_i$, $c_{i+1} = x_i y_i$. It is also referred to as (2,2) counter and its block diagram is shown on Figure 3.3

The delay (T_{CLK} assuming $n_S = 1$) of *rca*, expressed in logic levels, is $T_{rca} = 2n$ ($O(n)$), while area, expressed in number of simple gates, is $A_{rca} = 7n + 2$ ($O(n)$).

Carry-Lookahead Adder (cla)

The bottleneck for ripple carry addition is the calculation of c_i, which takes linear time proportional to n. To improve it, we define Generate ($g_i = x_i y_i$) and Propagate ($p_i = x_i + y_i$) functions. Both g_i and p_i can be generated for all n bits in constant

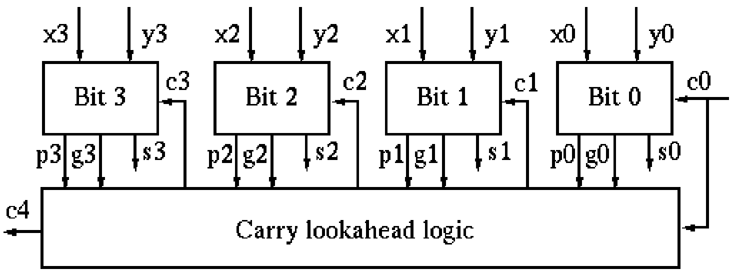


Figure 3.4: 4-bit Carry-Lookahead Adder.

time (1 gate delay). g_i and p_i are used for carry generation:

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i = x_i y_i + (x_i + y_i)c_i = g_i + p_i c_i$$

These equations allow us to calculate all the carries in parallel from the operands. For example, the carries for a 4-bit *cla* adder (Figure 3.4) are given as:

$$\begin{aligned}
c_0 &= c_0, \\
c_1 &= g_0 + c_0 p_0, \\
c_2 &= g_1 + g_0 p_1 + c_0 p_0 p_1, \\
c_3 &= g_2 + g_1 p_2 + g_0 p_1 p_2 + c_0 p_0 p_1 p_2, \\
c_4 &= g_3 + g_2 p_3 + g_1 p_2 p_3 + g_0 p_1 p_2 p_3 + c_0 p_0 p_1 p_2 p_3,
\end{aligned}$$

The delay is $T_{cla} = 4 \log n$ ($O(\log n)$), while area is $A_{cla} = 14n - 20$ ($O(n)$).

Conditional Sum Adder (cosa)

The basic idea in the *cosa* is to generate two sets of outputs for a group of k operand bits. One set is valid if input carry is one, while another is valid if it is zero. Depending on the value of incoming carry bit, only one of the sets is selected. Both of the sets have k sum bits and an outgoing carry bit. An example of *cosa* for 4-bit operands is shown on Figure 3.5

The delay is $T_{cosa} = 2 \log n + 2$ ($O(\log n)$), while area is $A_{cosa} = 3n \log n + 7n - 2 \log n - 7$ ($O(n)$).

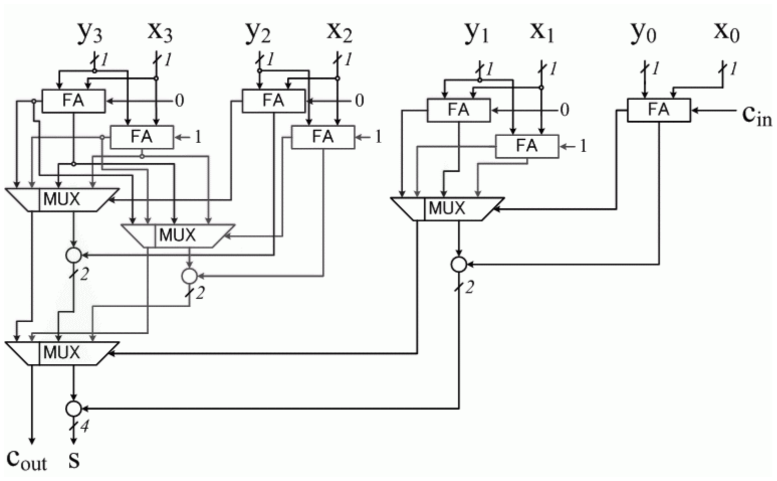


Figure 3.5: 4-bit Conditional Sum Adder (*cosa*).

Parallel-Prefix Adders

Parallel-prefix networks are widely used in high performance adders. They use direct parallel-prefix scheme for fast carry computation. They all have the initial generate and propagate signal generation and the final sum bit generation and differ only in the arrangement of the intermediate carry generation levels.

Parallel-prefix adders are constructed out of fundamental dot operators denoted by $\bullet$ as follows:

$$(g'', p'') \bullet (g', p') = (g'' + g'p'', p'p''),$$

where p'' and p' indicate the propagations while g'' and g' indicate the generations. A parallel-prefix adder can be represented as a parallel-prefix graph consisting of dot operator nodes. In the graphs, the black nodes $\bullet$ depict nodes performing the binary associative dot operation $\bullet$ on its two inputs, while the white nodes $\circ$ represent feed-through nodes with no logic i.e. cells are empty or contain buffers (Figure 3.6).

In this research, we focus on *bk* and *ks* parallel-prefix algorithms. They all have delay and area at the same order of magnitude: the delay is $O(\log n)$, while area is $O(n \log n)$.

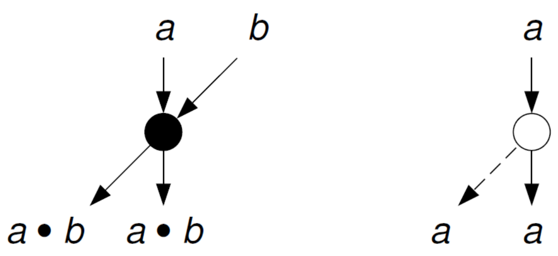


Figure 3.6: Black and white nodes.

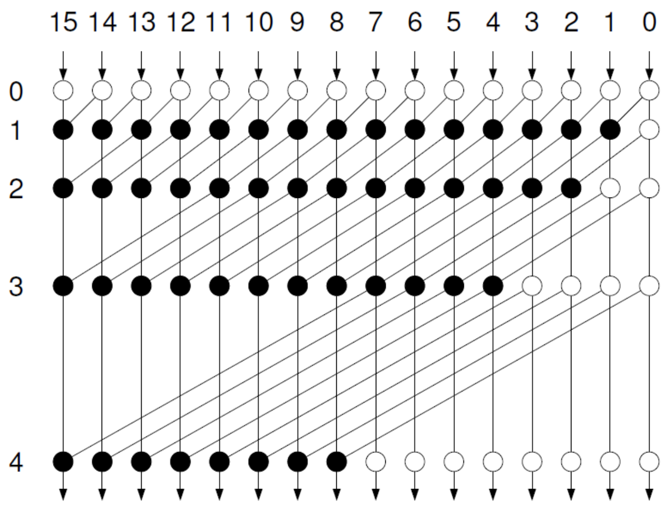


Figure 3.7: 16-bit Kogge-Stone (*ks*) Adder.

Kogge-Stone (*ks*) On Figure 3.7 is shown the parallel-prefix graph of a *ks* adder [64]. This adder structure is characterized by minimum logic depth, and full binary tree with minimum fan-out (maximum fan-out is 2), resulting in a fast adder (due to smaller capacitive loads) but with a large area, i.e. increased number of black nodes and interconnections. The delay is $T_{ks} = 4 \log n + 4$, while area is $A_{ks} = 3n \log n + n + 8$.

Brent-Kung (*bk*) On Figure 3.8 is depicted the parallel-prefix graph of a *bk* adder [17]. The tree has a low number of dot operators, while at the same time there are few cascaded dot operators. This adder is characterized by high logic depth and small area. The delay is $T_{bk} = 4 \log n$, while area is $A_{bk} = 14n - 3 \log n - 1$.

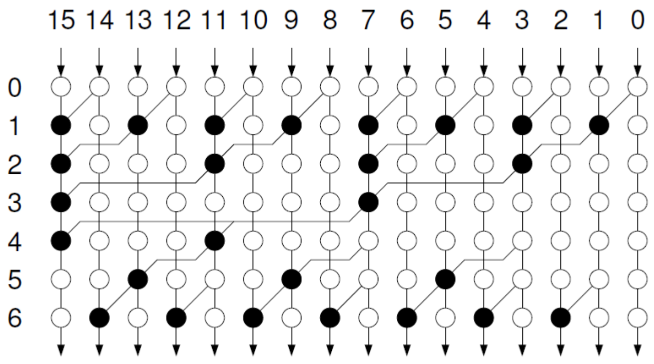


Figure 3.8: 16-bit Brent-Kung (*bk*) Adder.

3.3.4 Power Estimation

Power estimation is specially analyzed as it depends on data inputs. Formulas 3.4 implies that all P_{dyn} components are proportional to α. The probability of changing the state from 0 to 1 at a given clock tick for circuit input and node is denoted α_I and α respectively. α is a function of the circuit topology and aforementioned α_I. A probability of 0 means there is no activity (or negligible activity), 0.5 means signal changes its state every cycle while 1 means the signal has the same frequency as the reference clock.

However, for power estimation purposes, both PAS and PnR tools divide total power into net, internal, and leakage power. Net power is the power dissipated when charging or discharging the capacitive load, which includes the net capacitance and the capacitance of the input pins, and is calculated using the first equation in Formula 3.4. The internal power consists of two components: (*i*) the power dissipated in an instantaneous short-circuit connection between the voltage supply and the ground when the gate transitions (P_{sc}) and (*ii*) the internal net power dissipated during charging or discharging internal capacitance (P_{switch}). It is calculated by using internal power tables provided in the library, which are generated as a result of SPICE simulations for a range of input slew rates and external loadings, the effective frequency of a given circuit input (α_I/T_{CLK}), and the probability that the input signal is high. P_{glitch} is modeled inherently by propagating switching activity through the circuit. The leakage power computation depends on the state of the gate, and is calculated using the power tables provided in the library.

We perform post-PAS and post-PnR power estimation by providing the outcome of PAS and PnR stages to post-PAS and post-PnR estimators respectively and changing the α_I of the input signals of the adders. This statistically-based power estimation method is suitable in cases in which a large number of design parameters are used, i.e. a large number of design points. We perform power experiments in two ways:

- By varying α_I (shown as $\cancelto{}{\alpha}_I$ in Figure 3.1) from 0 to 0.5, with increments of 0.025. Since we study pipelined adders with an input register on its inputs, the maximum possible α_I of our input signals is 0.5. We assign the input signals the same probability (p) of being '1' or '0', as we assume they have independent and uniformly distributed random values.

- By assigning the input ports of the adder utilizing signal statistics ($\alpha_I = \alpha_{Iapp}$ and p) obtained using Sniper simulator [21]. Using Sniper we collect the data from input registers of the adder and information about adder usage. We further process these data to calculate α_I (α_{Iapp}) and p of each input port of the adder separately (*A[31:0]*, *B[31:0]*, *CarryIn*, and *Enable* signals). We run Sniper for four SPEC CPU2006 benchmarks: mcf, bzip2, libquantum, and h264ref [50]. Sniper implements an Intel's x86 microarchitecture, and we configure it to use a single adder in our experiments.

3.4 Results

3.4.1 Area

Table 3.1 shows the statistics of the ratio of area for all designs obtained using PnR and PAS. Area estimated using PnR and PAS estimation flows is compared for all combinations of the design parameters. For both flows, the effective area occupied by cells is considered. The geometrical mean of the ratios is high (1.64). The main reason is that clock tree synthesis with PnR is more realistic (but still imperfect) so with all its buffers and clock routing, it occupies more area than in PAS case. Clock tree in *CG* designs is more complicated, and they suffer from higher area increases than *noCG* ones. We observe, for a given *AF* and T_{CLK}, the ratio decreases with

Table 3.1: PnR vs. PAS ratio for area, timing, and power

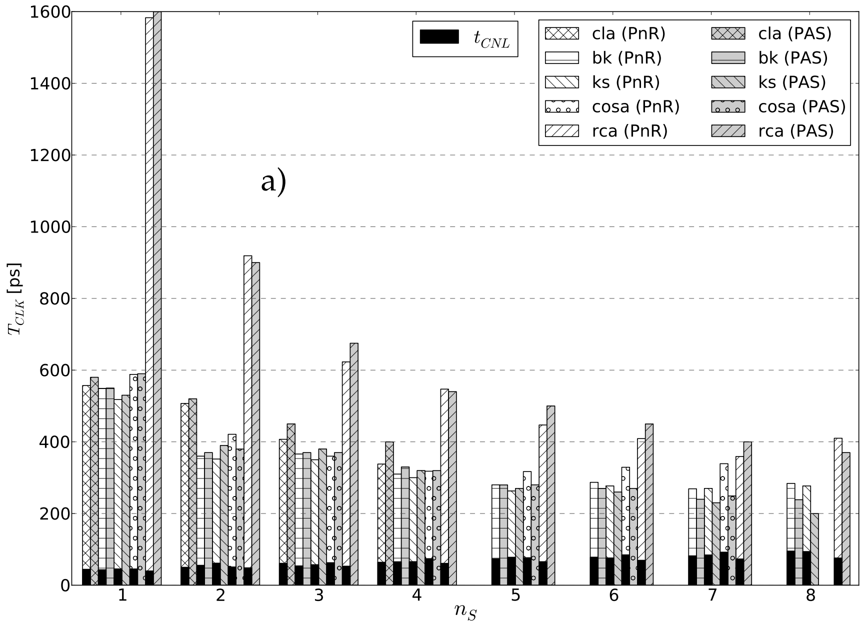
	Area			Timing			Power		
	All	*noCG*	*CG*	All	*noCG*	*CG*	All	*noCG*	*CG*
Min	1.43	1.43	1.45	0.84	0.84	0.88	0.49	0.56	0.49
Max	1.94	1.73	1.94	2.07	1.38	2.07	7.52	2.07	7.52
GEOM	1.63	1.58	1.69	1.18	1.02	1.38	1.12	1.02	1.27
GStDev	1.06	1.04	1.06	1.25	1.12	1.21	1.35	1.21	1.45

n_S. This happens as the PnR tool does more effort in timing closure (that occupies extra area) for adders with fewer stages as they are inherently slower. Regarding *AF*, *bk* has the highest ratio, as due to its high max-fanout, additional circuit sizing is done in PnR. For any given T_{CLK} and n_S, *AF*s have the same relative area order in both flows: *Area(cosa)>Area(ks)>Area(bk)>Area(cla)>Area(rca)*.

3.4.2 Timing

Table 3.1 provides statistics of ratios of T_{CLK} for all designs. In the rest of this section we focus on the minimal achievable clock period (T_{min}) for a given n_S for all *AF* according to post-PAS (T_{minPnR}) and post-PnR (T_{minPAS}) timing results. In other words, T_{min} represents the critical path of the fastest achievable adder structure for given n_S and *AF*. The comparison is shown on Figures 3.9(a) and 3.9(b) for *noCG* and *CG* respectively. Additionally, Figure 3.9 indicates the clock network latency (t_{CNL}) of post-PnR adders. In PAS, for timing analysis purpose, the clock tree is not modeled, so clock network latency is not considered properly. Therefore we present clock latency only for post-PnR designs. The ratios of T_{min} for PnR and PAS (T_{minPnR}/T_{minPAS}) are presented in Table 3.2. A ratio below 1 means that for the same n_S and *AF*, PnR adder can achieve higher speed (lower clock period) than PAS adder.

From Figure 3.9(a) we notice that post-PnR estimations report faster timings for designs without clock-gating support (*NoCG* case) in most cases. However, both post-PnR and post-PAS provide very similar estimations on average. Except for highly pipelined designs, the impact in overall timing of clock network latency in post-PnR is not very high. This is because the PnR tool is able to overcome the



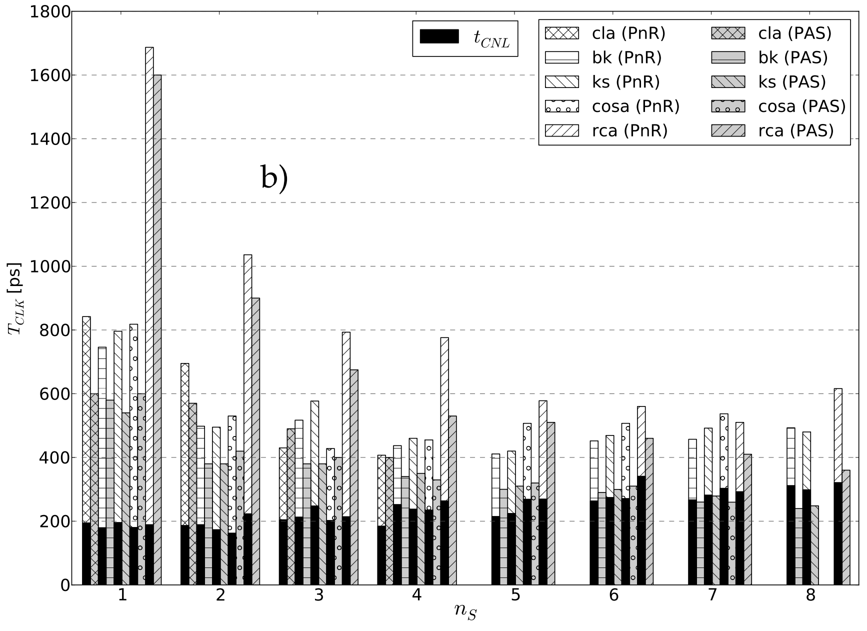


Figure 3.9: PnR vs. PAS: minimal achievable clock periods (T_{min}) for given n_S and *AF*, for (a) *no*CG and (b) CG. Black part of PnR bars indicates the clock network latency (t_{CNL}).

clock tree network latency by applying optimization techniques. As a conclusion, timing results obtained with PAS are reliable in most cases for designs without

Table 3.2: Ratios of minimal achievable clock periods for PnR and PAS (T_{minPnR}/T_{minPAS}). In the upper part of the table are presented results without clock-gating, while in the bottom are the results with clock-gating support.

$AF \setminus n_S$	1	2	3	4	5	6	7	8	
bk	1	0.97	0.99	0.94	1	1.06	1.12	1.19	*noCG*
ks	0.97	0.90	0.92	0.94	0.97	1.06	1.17	1.38	
cla	0.96	0.98	0.90	0.84	N/A	N/A	N/A	N/A	
cosa	1	1.11	0.97	0.99	1.13	1.22	1.36	N/A	
rca	0.99	1.02	0.92	1.01	0.89	0.91	0.9	1.11	
bk	1.29	1.31	1.36	1.28	1.37	1.56	1.76	2.06	*CG*
ks	1.47	1.30	1.52	1.31	1.35	1.56	1.76	1.93	
cla	1.40	1.22	0.88	1.02	N/A	N/A	N/A	N/A	
cosa	1.11	1.26	1.07	1.38	1.58	1.64	2.07	N/A	
rca	1.05	1.15	1.17	1.46	1.13	1.22	1.24	1.71	

Table 3.3: n_S of the fastest design per *AF* for PnR. The maximum n_S for a given *AF* is in parenthesis

CGable \ *AF*	*bk*	*ks*	*cla*	*cosa*	*rca*
noCG	7 (8)	5 (8)	4 (4)	5 (7)	7 (8)
CG	5 (8)	5 (8)	4 (4)	3 (7)	7 (8)

clock-gating (*NoCG*).

The impact of the clock tree for *CG* (Figure 3.9(b)) is higher since the clock tree is significantly larger than in designs without clock-gating. This results in an increase of clock network latency which causes timing degradation since with high clock network latency, register-to-output critical paths become hard to satisfy. The final result is that post-PnR estimations report consistently slower timings than post-PAS estimations. Therefore, we can say that PAS flow underestimates the negative effect of including clock-gating logic. We also notice that the ratio (T_{minPnR}/T_{minPAS}) is higher when n_S increases in case of high area *AFs* (*bk*, *ks*, *cosa*). For low area *AFs* (*cla*, *rca*) this trend does not exist but still the 8-stage design (*rca*) has the highest ratio. Additionally, we observe that, according to post-PnR estimations, *cla* performs faster for *CG* than for *NoCG* compared to other *AF*. Due to its small area and simple structure, the delays introduced with routing and clock tree synthesis are smaller than for high area parallel adder structures.

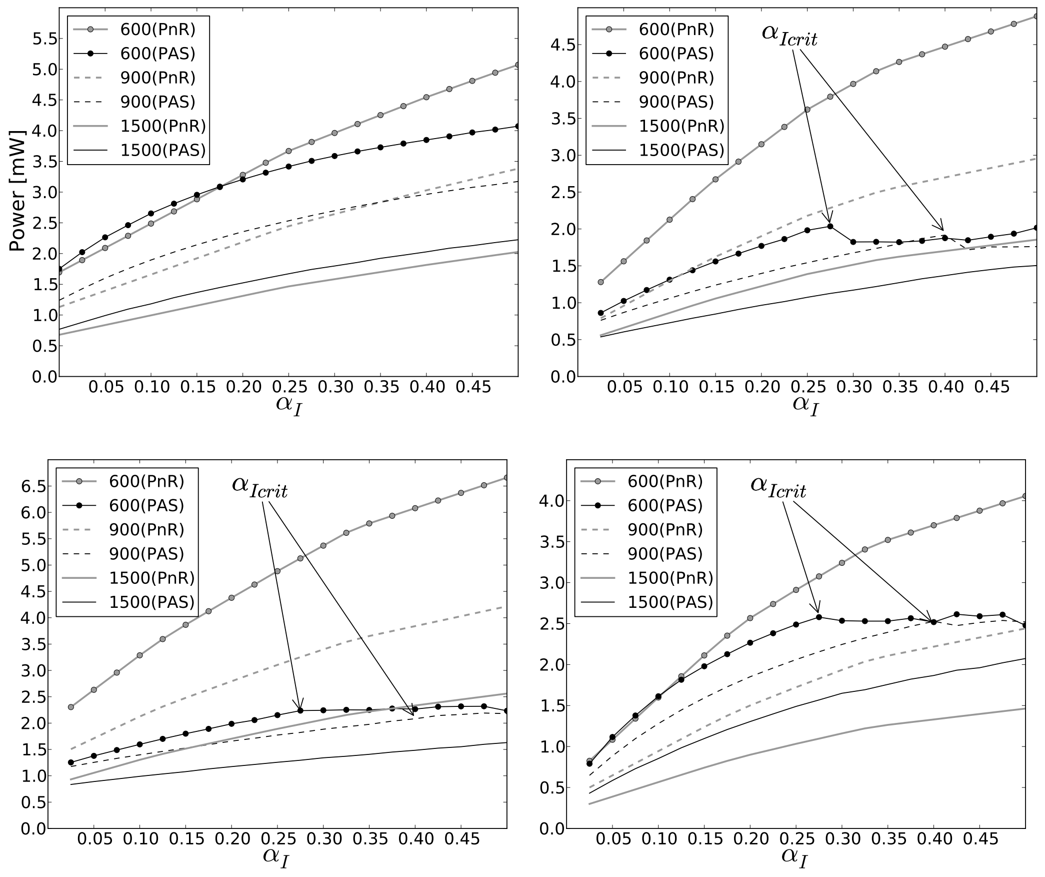


Figure 3.10: (α_I, P) graph for post-PnR and post-PAS estimations. From left to right: (a) *NoCG*, ks, *3-stage*, (b) *CG*, cla, *4-stage*, (c) *CG*, rca, *6-stage* and (d) *CG*, cosa, *2-stage* adders.

From Figure 3.9 we observe that for PnR, there are cases in which designs with fewer stages are faster than designs with more stages. For example, the 2-stage *bk* adder with *CG* can achieve shorter T_{CLK} than the 3-stage one. These counterintuitive results happen due to two reasons. The first and the main reason is that with more pipeline stages, the clock tree is larger, so clock network latency is higher. In this case, the clock network latency increase is higher than timing shortening achieved with additional stages. The second, less dominant, but also less obvious reason can be observed in the case of *NoCG* 2- and 3-stage *bk* PnR adders. The combinational path delay $(T_{PnR} - t_{CNL})$ in the 2-stage case is shorter than in the 3-stage case (even if we allow more space for clock and signal routing). Most likely, the reason why the PnR tool routes less efficiently the 3-stage case is that it gets stuck in a local minima. Finding optimal routing for a given placement is an NP-complete problem, so tools typically have to resort to use heuristics, which cannot always guarantee to find the optimal solution. The mentioned counterintuitive result is not an isolated case. For example, lower clock tree latency for a higher n_S, also can be seen in 4- and 5-stage *CG*, bk adders. We suppose this is a consequence of the irregularity of adder structures, and that for regular structures like caches the results would be more expectable. Reducing clock network latency can be achieved by manual clock-tree optimizations, but for a design space exploration it is not practical to optimize by hand a large number of design points. If we optimized by hand only some of them, the comparison would not be fair.

We observe two important consequences of aforementioned facts. The first one is that the fastest adders produced using PnR and PAS flows have different design parameters. The fastest adders are 5-stage *ks* (263ps, PnR) and 8-stage *ks* (200ps, PAS) for *NoCG*, while 4-stage *cla* (407ps, PnR) and 8-stage *bk* (240ps, PAS) for *CG*. Accordingly, when clock-gating and PnR are assumed, *cla* outperforms parallel-prefix AF (*bk* and *ks*), that are traditionally known to be fast. The second observation is that according to post-PnR estimations, in terms of T_{min} there is an optimal point beyond which pipelining stops to give the desired speedup but rather slowdowns for all *AF* except *cla*. On the contrary, in post-PAS estimations more pipeline stages always result in shorter T_{CLK}. Table 3.3 shows the optimal number of stages per *AF*, for *NoCG* and *CG*.

3.4.3 Power

As explained in Section 3.2.1, we have that the power of an adder (P) is practically equal to its dynamic component ($P \approx P_{dyn}$). Therefore, for the sake of simplicity, we only present the aggregated power P.

Table 3.1 shows the statistics of the ratio of power for designs generated for all combinations of design parameters and α_I obtained using PnR and PAS flows. Regarding *AF*, we observe that post-PAS power estimations are the most similar to the post-PnR ones for *rca* (simple topology) while the highest deviations are found for parallel-prefix adder structures. For *CG*, we exclude the $\alpha_I = 0$ case from the statistics and further analysis as it is a special case when clock-gating logic is always enabled. Results for this case are fairly different, and the ratios range from 0.4 to 15.4. Therefore, we can deduce that this case is modeled differently in post-PnR and post-PAS estimations.

In order to examine how post-PAS and post-PnR power change with α_I, we present (α_I, P) graphs for different T_{CLK} (Figure 3.10). As a representative *NoCG* case, *ks*, *3-stage* adder is selected (Figure 3.10(a)). The graph shows that post-PAS overestimates power dissipation, comparing to post-PnR results, for low α_I. For high α_I this remains true for longer T_{CLK} designs while for designs with shorter T_{CLK} post-PAS power results are lower than post-PnR. We observe that, for higher n_S, the post-PnR and post-PAS power ratio is higher. The trends are similar for all *AF* except for *rca*. In this case, post-PAS power is typically lower than post-PnR power.

Figures 3.10(b)-(d) show graphs for three representative *CG* cases. For *CG*, the post-PAS power estimations are usually below post-PnR ones. The exceptions are designs for low n_S and long T_{CLK} where post-PAS results are typically above post-PnR ones (Figure 3.10(d)). As can be expected, post-PnR power estimations increase with α_I. On the contrary, for post-PAS, we observe that there is a critical α_I (α_{Icrit}) from which estimation starts decreasing and contradicting the formulas for power dissipation (Formulas 3.4). This α_{Icrit} does not depend on *AF* but increases with T_{CLK}. It starts at 0.175 for the shortest T_{CLK} (around 400ps), and it disappears when T_{CLK} gets around 1200ps. The trend is the same for other *AF*, but for *rca* it is much less noticeable as it can be seen from Figure 3.10(c). For *CG*, n_S has the same impact on post-PnR and post-PAS power estimations as for *NoCG*. The unexpected effect of α_I ($\alpha_I > \alpha_{Icrit}$) is related to the way how the post-PAS tool estimates clock-gating power reduction. Since we do not have access to the source code of the tools, we are not able to further investigate the problem.

From a low-power processor's designer point of view, the most relevant adders are those that have the lowest power for a given frequency. Figure 3.11(a) shows two graphs, (f, P) and (f, n_S), of the lowest power adder design for a given frequency for both post-PnR and post-PAS. The graph shows the *AF* and the n_S of the PnR adders and the *AF* of PAS adders. Bar colors indicate whether the PnR and PAS adders have the same n_S. The graph includes only *CG*, as we target low power, while $\alpha_I = 0.5$ is chosen, as it is the worst case from low-power design point of view. We can observe from the graph that post-PAS power drops for high T_{CLK}, and this is a consequence of the aforementioned effect of α_{Icrit}. For f below 0.5 GHz according to both post-PnR and post-PAS estimations *rca* is selected. In the next range (0.5-1 GHz) in post-PnR power estimations, *cla* has the lowest power, while in post-PAS case it varies between *cla* and *bk*. For medium f range (1-2 GHz) in both cases *bk* is typically the lowest power family. Adder *bk* continues to be the lowest power *AF* over 2 GHz for post-PAS, while for post-PnR the most power efficient *AF* are typically *cosa* and *cla*. Regarding n_S, it is different for PAS and PnR for around 30% of f range, and when it is different, usually PnR adders have a higher n_S as follows from Section 3.4.2.

Figure 3.11(b) shows the results for α_I extracted from the benchmarks ($\alpha_I = \alpha_{Iapp}$). There are important differences compared to the experiment with $\alpha_I = 0.5$. For the sake of simplicity, we do not present results for α_I of each benchmark separately but we present results for averaged α_{Iavg}. In this averaged case, we can notice that post-PAS power drops, observed for $\alpha_I = 0.5$, are not visible for $\alpha_I = \alpha_{Iavg}$. This means that α_{Iavg} of the input bits does not go above α_{Icrit}. Still, when benchmarks for which the data supplied on adders have high α_I ($\alpha_I > \alpha_{Icrit}$) are considered (e.g. bzip2), the aforementioned post-PAS power drops are observable. Although for α_{Iavg} there are not post-PAS power drops as for $\alpha_I = 0.5$, post-PAS power estimations are still significantly below post-PnR ones for high frequencies (around 3 times). Additionally, the lowest power adders according to post-PnR and post-PAS are fairly different. The most interesting observation is that *cla* is the lowest power *AF* for the high frequency range (> 2 GHz) according to post-PnR. As it is presented in Section 3.4.2, *cla* is able to achieve high frequencies with less n_S than other *AF* for CG, being in that way more power-efficient. The lowest power designs for post-PAS are similar in both experiments. Therefore, we can observe that PAS estimation flow is less dependent on α_I

3.4.4 Related Work

In the context of adder characteristic evaluation, there is considerable prior research [119, 8, 105, 117, 116]. Our research differs in that we perform evaluation of adders from the estimation flow perspective. Additionally, in the context of comparison of estimation flows there is some published work [29, 48] that targets comparison between proposed physically aware behavioral level estimators and other state-of-the-art synthesis tools, using digital arithmetic as a case study. The main difference to our work is that we perform a design space exploration with a large number of design points, which allows more general conclusions to be drawn. Some of the previous research efforts aim to connect statistics of the input signals with power dissipation [12, 4]; however, their goals are different as they do not include design space exploration nor a comparison of estimation flows.

3.5 Summary

In this chapter, we examined and analyzed the differences between results obtained using PnR and PAS estimations flows. Results obtained for PnR are closer to the post-fabrication ones, thus, we took them as a reference in the discussions. As a case study, we use a wide design space exploration of low-power adders that include various design parameters and α_I. We examined both regular and corner cases.

For both timing and power, we found that differences between post-PAS and post-PnR flows can be significant for *CG* and high n_S. Nevertheless, for *NoCG* and low n_S the differences are not substantial. We found that for simple topologies (e.g. *rca*) the mentioned estimation flows have the most similar results.

We concluded that timing accuracy strongly depends on clock-tree modeling. Idealistic modeling is responsible for optimistic post-PAS results. Additionally, we found that due to PnR clock-tree modeling imperfectness, counterintuitive results sometimes occur. As a consequence, there are cases in which designs with less stages are faster than designs with fewer stages.

We observed that accuracy of post-PAS power results is inversely proportional to α_I and frequency. Power estimations increase less with α_I and frequency than post-PAS ones. The differences between power estimations are especially high for *CG*, where post-PAS power is typically less than post-PnR. Moreover, post-PAS power estimations can decrease for $\alpha_I > \alpha_{Icrit}$, contradicting the formulas for power. Fortunately, α_I of most real-world application is typically low, so the mentioned unreliable and contradicting results are not common to happen.

Regarding area, we observe that post-PAS practically always underestimates area compared to post-PnR.

When accuracy is wanted and computational resources are not a bottleneck, PnR flow is preferred choice. However, when a large number of design point (and restricted computation resources) and simple structures are considered, we have observed that PAS flow is an acceptable choice. Still, while using any of the flows, the limitations should be kept in mind.

We followed these guidelines in the rest of the thesis and use the PAS flow in Chapter 4, where we perform a very wide exploration of integer adders, while in Chapters 5 and 6 we use the PnR flow to study more complex circuits like an integer multiplier and a floating point unit.

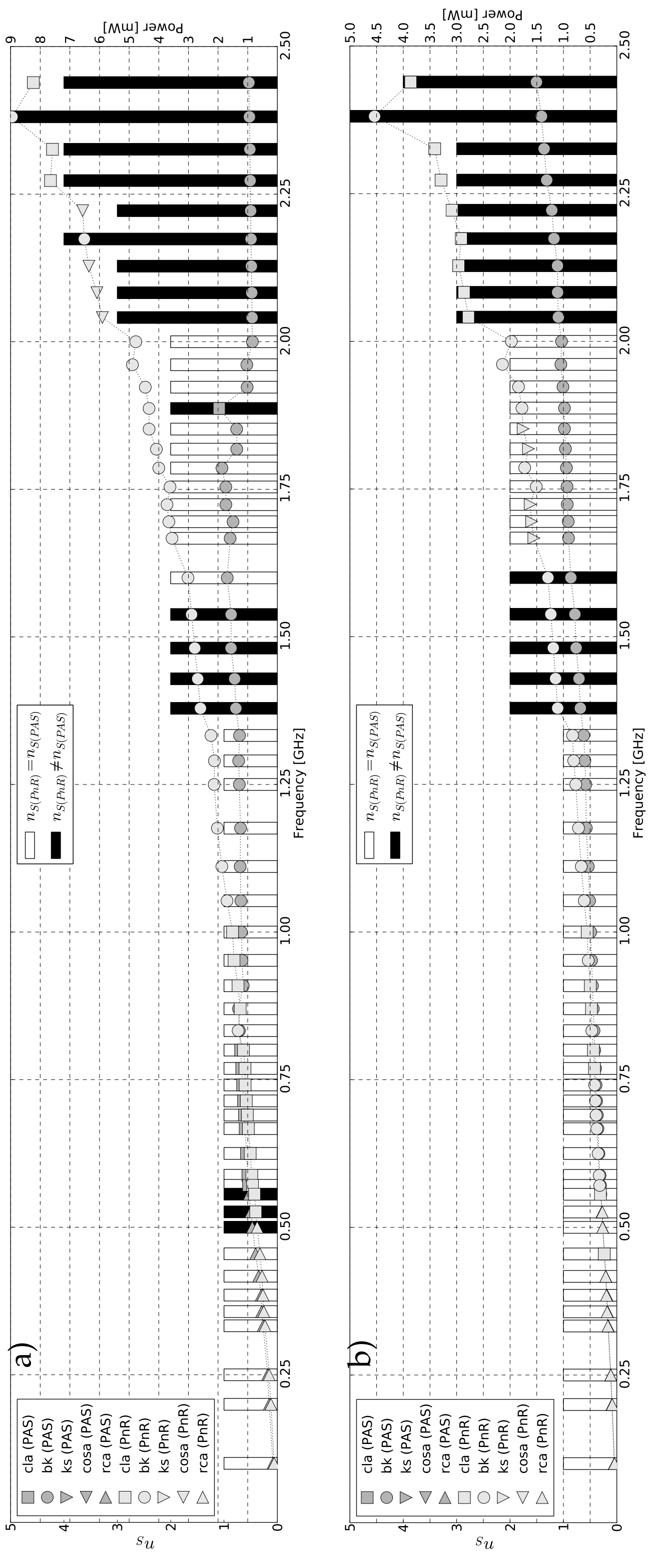


Figure 3.11: Post-PnR and post-PnR power estimations graph of the lowest power adder designs with clock-gating support (CG) for given frequency, for (a) $\alpha_I = 0.5$, and (b) $\alpha_I = \alpha_{Iapp}$ extracted from the benchmarks. Bars represent n_S of lowest power PnR adder, while their color indicate whether the PnR and PAS adders have the same n_S.

4 Exploration of Adder Unit Structure for Energy Efficient Vector Processors

4.1 Introduction

In this chapter, we perform a comprehensive (power, delay, energy, area) design space exploration of the adder unit for vector processors VA. The final goal of this design space exploration is to provide guidelines for designers of low power and energy efficient vector processors. We propose a complete framework that consists of several simulators at different levels, and we do this exploration mostly with real microbenchmarks (consisting of application data) obtained from vectorized SPEC applications [50]. It is necessary for this exploration as we need to observe how architectural-level parameters (e.g. vector length) affect the circuit-level metrics (e.g. adder's power dissipation) and how circuit-level parameters (e.g. adder's clock cycle) impact the execution time of a microbenchmark. We discover that these observations are not possible with a framework based on random value input. Due to a large number of design points and simulations, as a basis for the exploration framework we use a PAS estimation flow (see Chapter 3).

First, we present a discussion of adders' power characteristics considering clock-gating and timing stage analysis for various kinds of testbenchs and modern technology (Section 4.3.1). Afterwards, we study vector multi-lane usefulness in case of VA (Section 4.3.2). We also discuss the advantages of using application-based testbenchs over traditional random-data ones (Section 4.3.3). Finally, we provide guidelines on VA configuration selection for different low power vector processors (Section 4.4).

4.2 Methodology

This section describes the framework used to perform the design space exploration of VA as well as the framework parameters and the test benchmarks that we use.

4.2.1 Framework

The framework is depicted on Figure 4.1. The basis for this framework is PAS estimation flow (Chapter 3), so the corresponding explanations and details are valid here. Therefore, here we focus on the added features, i.e. differences, characteristic for this framework.

The framework includes architectural- (*VectorSim*) and circuit-level (*RC*, *NCsim*) simulators and tools, as well as an interfacing tool (*tBenchGenerator*). For various circuit- and architectural-level parameters (explained in Section 4.2.2) we obtain the metrics of interest of our VA (described in Table 4.1).

The first step in the framework is to feed *VectorSim* (introduced in Chapter 2) with the vector parameters (microbenchmark (*uBench*), maximum vector length (MV_L), number of lanes (n_L)) as well as some design parameters (number of stages (n_S), clock period (T_{CLK})). This stage generates data and timing traces. The traces include information about vector additions only, excluding scalar additions.

The next step is transforming this architectural-level information into Verilog test benchmarks (*tBenchs*) using *tBenchGen* (introduced in Section 4.2.3). We generate independent *tBenchs* files for each lane of the VA. Section 4.2.4 describes the different *tBenchs* that we generate. Apart from the traces, we include here one more parameter (*CGable*) which indicates if the design has clock-gating support.

We incorporate design parameters (*AF*, T_{CLK}, n_S, *CGable*) into handcrafted HDL codes of adders which are supplied to Cadence RTL-Compiler [32] (*RC PLE Synthesis*) to produce different adder's synthesized mapped netlists and to perform static timing analysis (STA). For multiple lane VA configurations, adders in all lanes are identical.

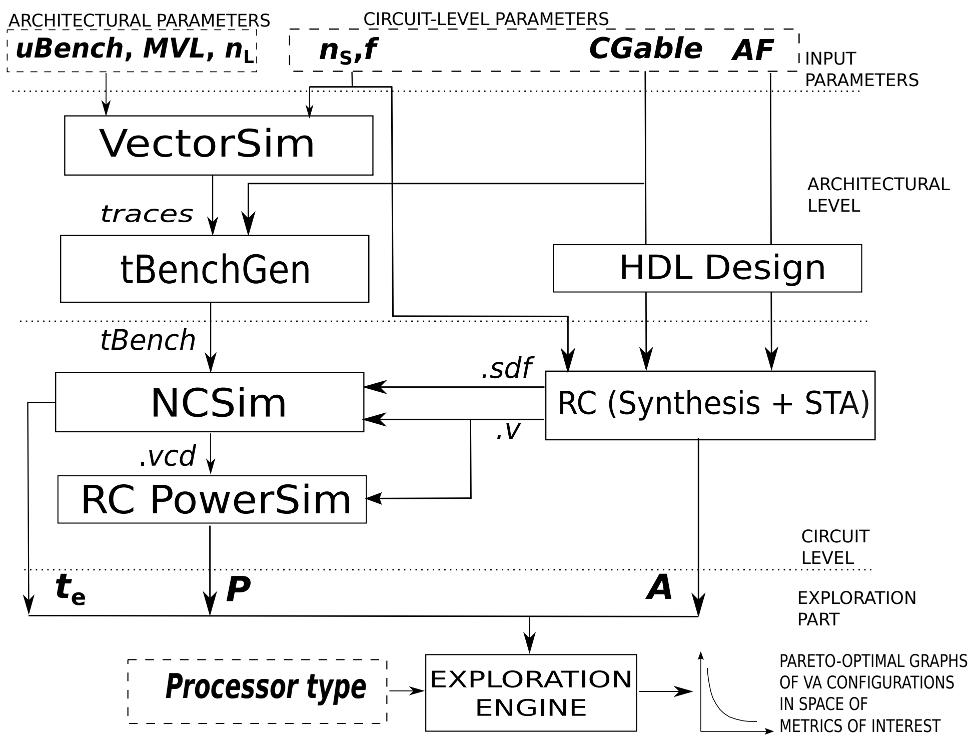


Figure 4.1: Block diagram of the framework's steps, parameters, and metrics.

The next step is to simulate each synthesized adder in *NCSim* [58] for each matching *tBench* with back-annotated delays using standard delay format (sdf) files [100]. This is done in order to obtain the execution time t_e, verify the synthesized designs and extract resulting switching activity information using Value Change Dump (vcd) files [110]. The final step of the framework is a precise computation of power metrics (P and P_{rest}) using *RC Power Simulation*. The inputs are synthesized designs in Verilog and vcd files.

4.2.2 Framework Parameters

We first present the vector processor specific parameters:

- ***uBench*** is a vectorized microbenchmark (kernel) extracted from an application, and it consists of integer data. It is a representative part of the application and small enough (between 100k and 150k test vectors) to keep circuit simulation time reasonable. We use three different *uBenchs* extracted from

Table 4.1: Metrics of Interest. Explained in detail in Section 3.2

Measured Metrics
P and P_{rest}. Average power of FU (or FUs if we have more than one lane) including and excluding the clock tree respectively.
t_e. The execution time of a test benchmark *tBench*, also referred as Delay (*D*).
A. The area of FU.
Derived Metrics
P_d. Surface power density = *P/A*. It is proportional to the fourth power of temperature of the given surface by Stefan-Boltzmann law ($P_d \propto T^4$) [104].
E=PDP. Power-Delay product is total energy spent in FU during a *tBench*.
P_dDP, EDP, E^2DP, and E^3DP are commonly used Power density and Energy-Delay products [75].

three vectorized SPEC applications (described in Table 4.2) that are used in mobile devices and can also be found in server workloads. In the *uBenchs*, there are long and short vector data, so our application set is comprehensive for our needs. They are addition intensive (in average 27.14% of total instructions executed), so adder's impact on the *uBenchs* t_e is significant.

- MV_L is the already mentioned maximum vector length of the vector processor. Possible values are 16 and 128 to represent both extremes of short and long maximum vector lengths. We chose 16 as a short vector length that is close to SIMD extensions while 128 represents long maximum vector lengths.

- n_L is the number of vector lanes. Possible values are 1, 2, and 4.

The design parameters, which are primarily needed for the circuit-level part of the framework, are *CGable*, *AF*, T_{CLK}*(=1/f)*, and n_S. The explanation for *CGable*, *AF*, T_{CLK}, and n_S in Section 3.3.2 are valid here as well.

4.2.3 Test Benchmarks Generator - tBenchGen

To transform architectural-level information to Verilog *tBench*s we develop a tool written in Perl named *tBenchGen*. The most important inputs are: data and time

Table 4.2: Vectorized microbenchmarks (*uBench*)

Hmmer (SPEC2006) applies profile Hidden Markov Models (HMMs) and is useful in many areas such as speech synthesis, handwriting, gesture recognition, part-of-speech recognition, machine translation, bioinformatics, etc.
Facerec (SPEC2000) is an implementation of a face recognition system.
H264ref (SPEC2006) is the reference implementation of H.264/AVC standard for video compression. and it is currently one of the most commonly used formats for the recording, compression, and distribution of high definition video.

traces from *VectorSim*, all circuit- and architectural-parameters, CG_{type} (explained in Section 6.5.3), data bit-width, and *tBench* length expressed in the number of Verilog test vectors. As output, it provides *tBench* for each lane separately, and its profiling report.

4.2.4 Test Benchmarks

We generate two kinds of *tBenchs*: *app-tBench* are obtained from real *uBenchs* and *synth-tBench* are synthetic. *app-tBench* are a function of all the parameters used in *VectorSim*. On the contrary, *synth-tBenchs* are not related with the vector simulator and are a function of T_{CLK} only. There are three types of *app-tBenchs* (*noCG*, *CG* and *100%*) and two types of *synth-tBenchs* (*rnd* and *0%*).

- ***noCG*** is used to evaluate designs without clock-gating. The input values of the VA are provided for each cycle.
- ***CG*** enables evaluating designs with clock-gating support. Clock-gating is enabled when we have idle cycles. Here we need additional clock-gating signals, one per stage.
- ***100%*** represents a case where the VA is always busy, assuming that memory is fast enough to provide data on time and that consecutive vector add instructions (`ADDV`) are independent. Therefore, the execution time depends only on the characteristics of the VA.

- ***rnd*** is a *tBench* with random values that are supplied each cycle to the adder. This is the traditional methodology of testing digital designs.

- ***0%*** is used to evaluate the case when there is no vector additions in the *uBench*, i.e. clock-gating is always active.

rnd and *100%* have the highest and the second highest activity factors respectively. *0%* has the lowest activity factor among the *tBenchs*. *CG* and *0%* are supplied to the adders with clock-gating, while the rest are supplied to adders without clock-gating logic integrated.

4.3 Design Space Exploration

In this section we first analyze the adders' characteristics and then the effectiveness of multiple lanes, showing their behavior on the measured frequency range using averaged results of all mentioned *uBenchs*. However, we comment on *uBenchs'* results separately when there are interesting differences between them. The execution times (t_e) for all *uBenchs* are normalized before averaging to assure the correctness of averaged results. At the end of the section, we present the major drawbacks of using a synthetic benchmark of random data for our research.

4.3.1 Adders Characteristics Discussion

We compare the power characteristics of the examined adder families by comparing their power dissipation for the measured frequency range. For a given frequency and adder family we plot the lowest power configuration (Figure 4.2). The goal here is to examine power dissipation of adders alone, so we test configurations with one lane and do not include the clock tree power. Results for MV_L=16 and 128 are almost the same, so we do not comment on them separately. The only exception is *CG* where for MV_L=128 we have slightly higher power dissipation. It is due to better exploitation of VA, so the clock-gating mechanism is active less percentage of time than for MV_L=16.

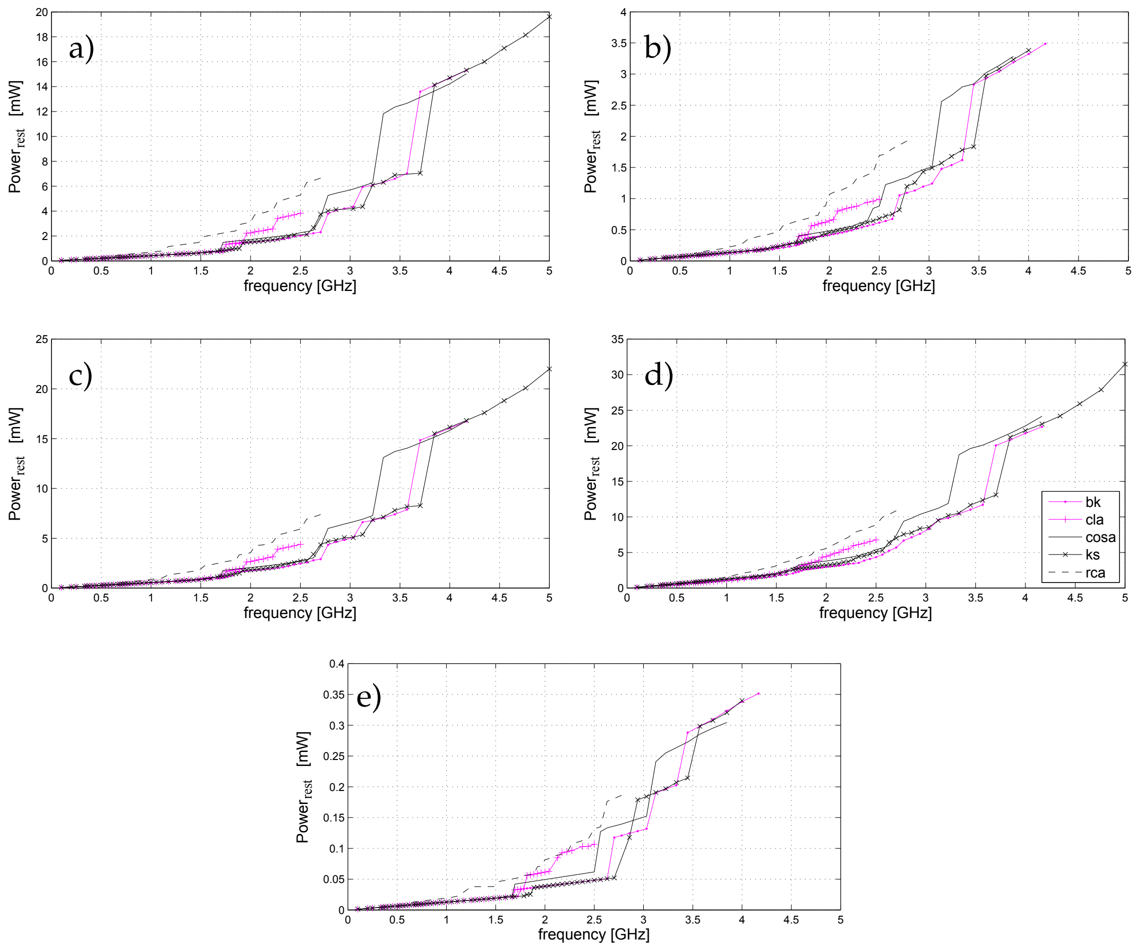


Figure 4.2: Power dissipation of analyzed adder families for the following test benchmarks: a) *noCG*, b) *CG*, c) *100%*, d) *rnd*, and e) *0%* for MV_L=128.

noCG: As we can see in Figure 4.2(a), power dissipation is highly dependent on the number of pipeline stages. For a given frequency, the lowest power configuration is always the one with the fewest stages. On the graph lines we can observe steps, that occur when the number of stages increases for a given adder family. *bk* adder family is the most efficient one except for the lowest and highest frequency ranges. For the lowest frequencies, until 0.57GHz, *rca* is slightly better than others due to its simplicity. However, it requires more stages to achieve a given frequency than other adder families because of its intrinsically poor timing characteristics. Therefore, it is worse than the other designs for higher frequencies. For the highest frequency range, *ks* is the best as it is the only one that can satisfy the frequency range from 4.17GHz to 5GHz. There is a short range (3.85GHz-4.17GHz) where *cosa* is the lowest power adder.

CG: Clock-gating reduces power dissipation roughly 5x (Figure 4.2(b)). Designs are slightly different in this case as clock-gating logic is incorporated in baseline designs used in the previous case. As a side effect, timings are a bit worse, especially for *ks* adder family, which is not the fastest adder as in the previous case. In fact, it can achieve "only" 4GHz, causing *bk* to perform the best in both mid- and high-range.

100%: The graph (Figure 4.2(c)) is fairly similar to *noCG* one, but power consumption, in general, is around 10% higher than for *noCG* case as the activity factor here is higher.

rnd: As we can see in Figure 4.2(d), power dissipation is significantly higher than for any *app-tBench* case, even for *100%*, which is the most similar to *rnd*, as in both cases we have data constantly supplied to the adder inputs. Here, the results are closer to the worst case ones, they are up to 2x higher than ones obtained with *noCG*, and up to 10x higher that *CG* ones. The reason for this difference is that random data is not correlated, which makes the activity factor higher. This is particularly emphasized in *cosa* as its conditional logic suffers from excessive power dissipation when it is driven by uncorrelated values. A higher activity factor implies that logic uses relatively more power than in *app-tBenchs* (diminishing pipelining relative overhead) and the slope of the (f, P_{rest}) graph is higher. As a result, power gaps between stages are smoother than in previous cases.

0%: This case (Figure 4.2(e)) actually shows the maximal possible power reduction using clock-gating. The greatest consumer in this case is clock-gating logic. The size of this logic is directly proportional to the number of stages - number of FFs to be gated. As a result, power dissipation is practically defined by the number of pipeline stages. Therefore, for a given frequency and number of stages two adders have quite similar power dissipation. Power is around 10x lower than for *CG*.

General Observations: Despite the glitch reduction achieved with pipelining [98], the overhead of adding registers is too high. Even with clock-gating presence, it is not possible to compensate it. Thus, for all five *tBenchs* we find that, for a given frequency, the adder with the lowest number of stages has the lowest power. *bk* is generally the most power efficient one. Although *ks* and *cosa* can provide high frequency, due to its high area, they suffer from long wires that cause excessive net power dissipation. *rca* is the most power and area efficient for low frequencies, and this makes it suitable for low power and low performance requirements where we can hide its long carry propagation delay.

From an implementation point of view, the most relevant results are for *CG* (Fig 4.2b). Power dissipation of the adders with clock-gating ranges from 11.68μW for 0.1GHz and 3.49mW for 4.2GHz, which is adequate for the low power design demands.

4.3.2 Multi-lane Effectiveness

In order to examine the fruitfulness of vector multi-lane over the measured frequency range, we look for P, E, EDP, and ED^2P characteristics of *1-*, *2-*, and *4-lane* VAs. For a given combination of MV_L, *app-tBench* and f, we pick the configuration with the lowest power (including clock tree) for each n_L.

Table 4.3 shows ratios of power dissipated by *2-* and *4-lane* configurations over single lane power for 1GHz and 4GHz. For *noCG* having twice n_L causes doubling power almost independently from MV_L and f. On the contrary, for *CG*, doubling the n_L increases power by less than 2x. The reason is that addition is faster while the rest of the hardware is the same, so a higher percentage of time the VA is clock-gated. Due to the same reason, the ratio decreases when frequency increases. For *CG* and *100%* multi-lane is more power efficient for longer MV_L, especially for 4 lanes. There is a correlation between data inside a vector, and with multi-lane we actually slice a vector into n_L shorter vectors, which causes decrease of the correlation and increase of the activity factor. Due to this reason having n_L lanes causes a power increase grater than n_L times in *100%*. This trend decreases with frequency as for higher frequencies we need to use adders with more stages. By having more stages a design has fewer glitches, which means that a decrease of data correlation has less effect in terms of glitching increase for *8-* than for *1-stage* adder.

Table 4.3: Power Dissipation Ratio of 2- and 4- over 1-lane VA

f[GHz]	1.0				4.0			
n_L	2		4		2		4	
MV_L	16	128	16	128	16	128	16	128
noCG	1.99	1.98	3.95	3.88	1.98	1.99	3.94	3.96
CG	1.79	1.74	3.1	2.9	1.68	1.65	2.69	2.62
100%	2.12	2.06	4.63	4.17	2.06	2.02	4.29	4.08

Table 4.4: Speed-up of 2- and 4- over 1-lane VA

f[GHz]	1.0				4.0			
n_L	2		4		2		4	
MV_L	16	128	16	128	16	128	16	128
CG, noCG	1.41	1.52	1.74	1.97	1.29	1.42	1.48	1.73
100%	2	2	4	4	2	2	4	4

Table 4.4 shows the speed-up of the multi-lane configurations over single lane both for 1GHz and 4GHz. As it consists of additions only, in *100%* doubling the number of lanes halves the t_e. For *noCG* and *CG* we have the same result, as clock-gating does not affect t_e for a given frequency. For these *app-tBenchs*, the speed-up linearly decreases with frequency, so adding more lanes is more productive for lower frequencies (low power region). This happens because the parallelism offered by multiple lanes has lower effect, as the *app-tBench* does not have only arithmetic instructions (*Amdahl's law*), and cache and main memory latencies become more noticeable at higher frequencies. Cache latency is more difficult to be hidden with short MV_L, so speed-up decreases with frequency more in this case.

Results for PD^nP are functions of P and t_e, thus they are directly explained with the discussion above. We just highlight the most interesting observations. While for *100%* all lane configurations have the same energy independently of f and MV_L, for other *app-tBenchs* the single lane configuration is the most energy efficient one. Adding more lanes becomes more fruitful if we look for performance, so 2- and *4-lane* VAs are in most cases better solutions than *1-lane* for *EDP*, especially

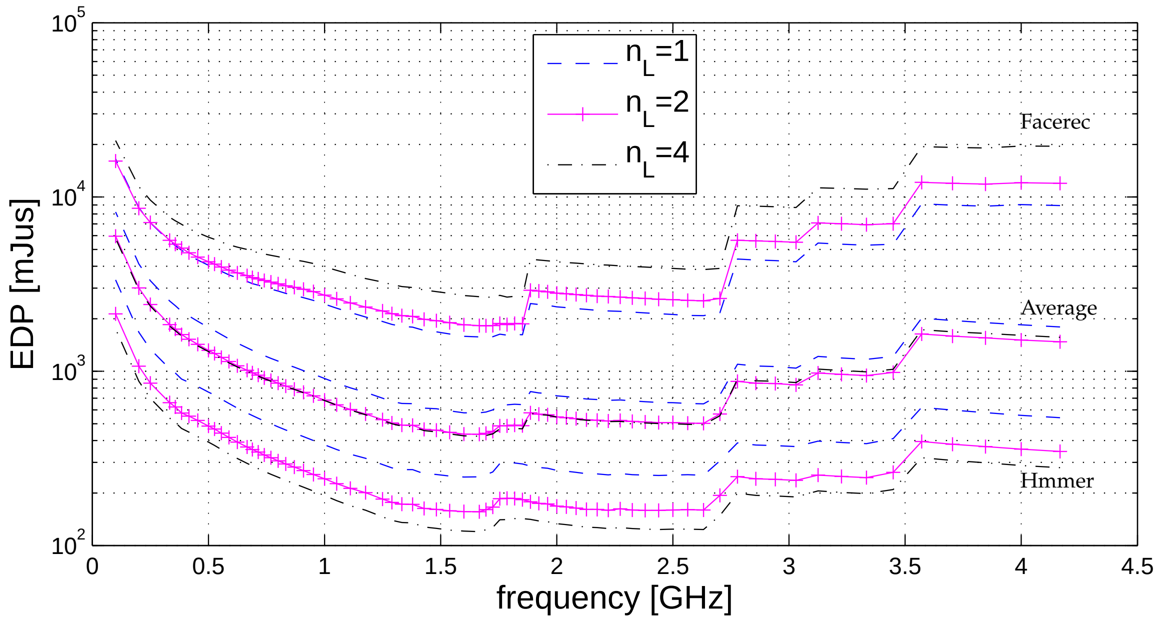


Figure 4.3: *EDP* of *1-,2-*, and *4-lane* VA for *CG* and *MV_L*=128.

for *CG* and long *MV_L*. While for other metrics averaged results lead to the same conclusions as individual *uBenchs'* results, we observe that adding lanes improves *EDP* for long-vector *uBenchs* (Hmmer), while it is the opposite for short-vector ones (Facerec) (Figure 4.3). Results for H264ref are similar to the averaged ones. For *ED^2P*, *4-lane* VA is better than *1-* or *2-lane* for all *MV_L* and *app-tBenchs*. The *(f, ED^nP)*(n>1) graphs have the minimum around 1.67GHz and, as it is shown in the next section, this is actually the most energy-efficient operating region.

4.3.3 Application-based vs. Random Data Benchmarking

We found the following problems with the traditional random data-based benchmarking:

- The main problem is that we cannot measure t_e, so we do not know how some design and architectural parameters affect t_e of *uBenchs*. Therefore, we cannot speak about the energy spent (or *EDP and ED^2P*) during a uBench execution.
- We cannot incorporate architectural-level parameters (*MV_L* and n_L) into power measurement, so we cannot analyze their effects on P and how this changes

with frequency.

- By supplying random data instead of real ones, we overestimate power dissipation as it is proved by comparing adders' power of *rnd* and *100%* cases. This happens because the value range of real data is "narrow", especially inside a vector, so activity factor is lower.

- Additionally, there are not idle cycles, so we are not able to analyze power of designs with clock-gating.

4.4 VA Selection Guidelines

In this section we provide some guidelines, derived from our results, on the choice of the best VA for the following types of low power vector processor:

- **Embedded.** A vector processor of this type would resemble "classic" embedded processors from TI and ARM that could be used in embedded signal processing solutions, including portable devices in audio, voice, communications, medicine, etc. Assumed frequency range is (0.1-0.4)GHz. The requirements are long battery life (E), as we need autonomous work and battery often is not rechargeable, and small area (A). Therefore, the desired VA should have a low product of E and A. Here power is indirectly limited with energy and frequency requirements, so there is a low risk of temperature violation.

- **Low-End Mobile.** The reference in this case is low-end ARM-like processors used in various handheld devices like GPS navigations and low-end smartphones. Targeted frequency range is (0.33-0.8)GHz. This kind of processors needs fast response (t_e) as it is a soft real time system, and long battery life (E). Therefore *EDP* is the right metric here as it usually is for this kind of low power devices[75]. Additionally, we need to maintain low P and A to satisfy processor's power/area budget.

- **High-End Mobile.** Here we assume a vector processor that resembles current processors like high-end ARM, Intel Atom, or AMD Bobcat which can be found in high-end smartphones, tablets, and notebooks. Assumed frequency range is (0.8-1.8)GHz. We have similar requirements like for low-end mobile processors with the difference that we stress performance more, so the desired metric is ED^2P [75].

- **Low Power - High Throughput.** Targeted vector processor in this case would have similar design goals as low power processors for servers like ThunderX ARM processors [108]. The frequency range of interest is (1.6-4.2)GHz. We took a pretty wide frequency range as from one side we target low frequency processors like the mentioned ones, but from the other side, we also want to explore possibilities and trends of high frequency range. In general, when we consider server design we always need to care about performance (t_e). Additionally, we distinguish and consider two cases. In the first case we assume that the server is always busy so we need to care about power dissipation P as it is constantly on, so the targeted metric is E. However, if the server is supposed to have idle periods, during which it goes to low power states, then we usually want to finish the job as soon as possible and go to one of these states. Therefore, in this case, we are interested in ED^2P.

We only consider designs with clock-gating support as today it is standard in any low power and energy efficient design. Results for each processor type are observed separately for short (16) and long (128) MV_L.

Figure 4.4 shows the best design points according to the metric of interest for three types of processor, via Pareto-optimal trade-off graphs. For example, if EDP is the metric of interest, for each t_e observed, we select the configuration with the least E. We discard slower designs that consume more energy than faster ones. For other metrics of interest, configurations are selected in a similar manner. The figure actually shows the t_e vs. P, E, or EDP trade-offs and the black curves indicate the best observed result for the metric of interest. The embedded case is not shown as there is a single solution. There are graphs for both cases of server processors.

Table 4.5 presents detailed information of the most relevant design points according to the metrics of interest discussed above. The best design for a given metric is highlighted using a bold font in the metric column. The table also includes some alternative design points that trade off the desired metric with significant gainings in other metrics. This is indicated with italic fonts in the metrics involved. Effects of parameters on a particular metric are explained in Section 4.3.

As it was observed in Section 4.3.1, *1-stage bk* adders are usually the best design choice. However, when speed is not critical, like in embedded vector processors, *1-stage*, *1-lane*, *rca* VA is the best choice due to its simplicity. It confirms that where the performance requirements are pretty low, *rca* is the most area and energy efficient choice [44]. When we stress performance, *2-stage* adders are more desirable, as we cannot achieve high frequencies with single stage. For high speed designs, *ks* adders are also an interesting solution.

When we consider low-end mobile vector processors, we can confirm *1-stage bk* as the most efficient solution. The VA configurations shown in Fig 4.4 a appear in series that correspond to different n_L values. In terms of *EDP*, *2-lane* configurations perform the best. For MV_L=128, configurations with 4 lanes are more desirable as we can exploit them better than for MV_L=16. When we deal with high-end mobile vector processors, *2-* and *4-lane* configurations provide lower ED^2P than single lane ones. With long MV_L, *4-lane* configurations are the only ones to be considered.

For power-constrained servers, where demands are conservative in terms of t_e, single lane configurations are usually the most desirable due to their low power. This remains true for long MV_L. However, when we consider VA design for fast, low power servers, multi-lane is as effective as in high-end mobile case. In this case, *2-stage* adders running around 2.6GHz perform the best in terms of ED^2P for both long and short MV_L.

In terms of the frequency, we can identify two efficient operating ranges. The first one is around 1.67GHz while the second one lies around 2.6GHz. Until these points, we can consider increasing frequency as a way to achieve energy efficient speed-up, but after that adding more lanes becomes a more fruitful solution.

4.5 Related Work

In the context of adders, there is considerable research on their comparison in the Energy-Delay space [80, 105, 79]. However, our analysis differs as we include architecture-related parameters and additional ones like clock-gating. Apart from that, our analysis targets vector architectures and the adders for these architectures lead to distinct conclusions. While there is very interesting research on the selection of adder units for particular architectures such as DSPs [69] and crypto-processors [52], their exploration demands are different, so they do not utilize the same kind of architectural parameters as we do. Moreover, they do not test the adders with architecture-driven benchmarks and assume adders are always busy.

4.6 Summary

The presented design space exploration addresses the issue of optimal VA selection in vector processor design. A novel, multi-level (circuit and architectural) methodology, on identifying the best VA is presented. We showed how design (e.g. frequency) and architectural (e.g. maximum vector length or number of lanes) parameters can direct the selection of the VA design. We reevaluate various adder's families characteristic with this novel methodology and showed that multi-lane approach is efficient in terms of *EDP* and ED^2P, especially with clock-gating and longer vector lengths. Additionally, we showed advantages of our benchmarking method over the traditional random values-based one for this exploration. Finally, we propose suitable VA configurations for four vector processor types according to several metrics of interests. We found that 2- and *4-lane* configurations with single stage *bk* adders are optimal for mobile, while *1-lane*, *rca* configurations perform the best for embedded vector processors. However, *4-lane* configurations with *2-stage bk* adders running around 2.6GHz are optimal solutions for VA if we target high-performance, low power server vector processors.

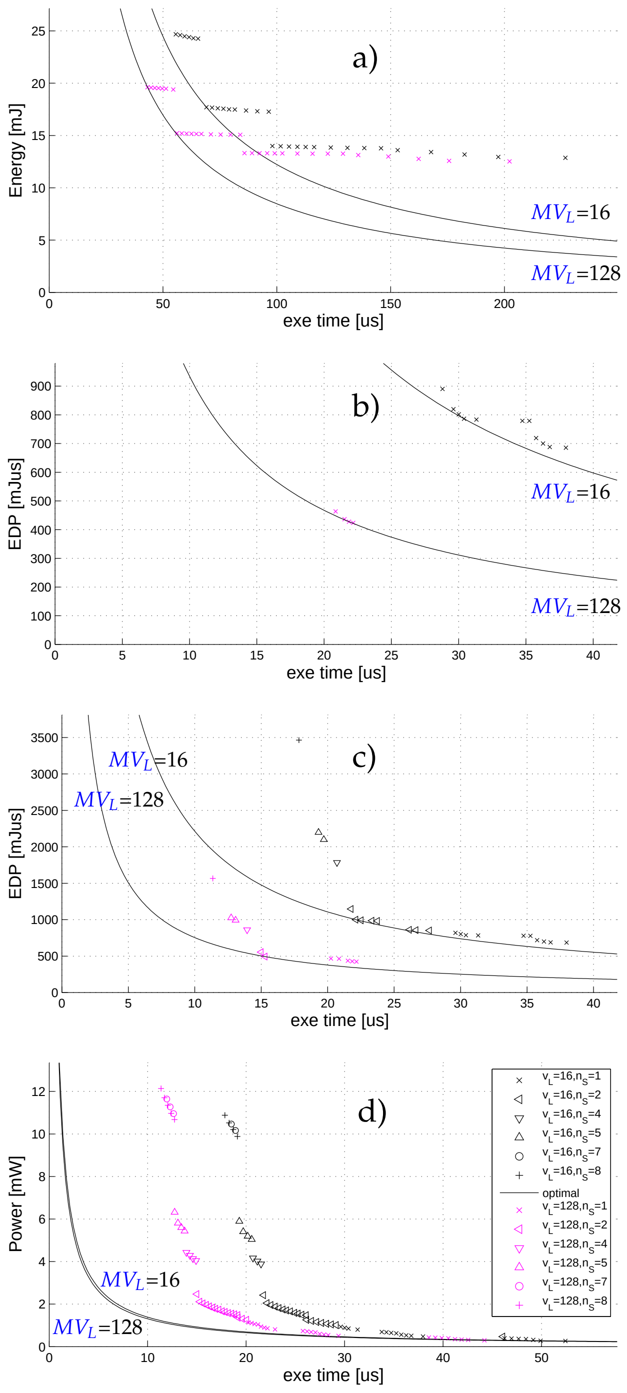


Figure 4.4: Pareto-optimal trade-off graphs of E (a), EDP (b), EDP (c), and P (d) vs. t_e for low-end mobile, high-end mobile, fast low power servers, and power-constrained servers respectively. Hyperbolic curves ("optimal" in the legend) represent EDP (a), ED^2P (b), ED^2P (c), and E (d) of the best design points for both 16 and 128 MV_L.

Table 4.5: Optimal VA Unit Configurations

Vector Processor	Input Parameters					Output Metrics						
Type	MV_L	n_L	*AF*	n_S	f[GHz]	$t_e[\mu s]$	P[mW]	E[mJ]	EDP[mJμs]	ED^2P[mJμs^2]	$A[\mu m^2]$	P_d[mW/m^2]
Embedded (0.1-0.4)GHz	16	1	*rca*	1	0.1	749.1	16.79x10^{-3}	**12.58**	9422	7.06x10^6	**281.2**	36.0
	128	1	*rca*	1	0.1	668.6	18.35x10^{-3}	**12.27**	8204	5.48x10^6	**281.2**	41.53
Low-End Mobile (0.33-0.8)GHz	16	2	*bk*	1	0.8	69.08	0.2562	17.7	**1223**	84.47x10^3	792.6	187.5
		4	*bk*	1	0.8	**55.62**	0.4436	24.67	1372	76.33x10^3	1585.0	144.1
		1	*rca*	1	0.33	226.8	56.76x10^{-3}	**12.87**	2919	66.18x10^6	309.4	111.0
		1	*bk*	1	0.8	*98.05*	0.1427	***13.99***	1371	0.134x10^6	396.3	224.2
	128	4	*bk*	1	0.8	**43.21**	0.45369	19.61	**847.5**	36.62x10^3	1585.0	150.6
		2	*bk*	1	0.8	56.12	0.2711	15.21	***1143***	47.91x10^3	*792.6*	206.3
		1	*rca*	1	0.33	202.2	61.9x10^{-3}	**12.52**	2532	51.2x10^6	309.4	127.6
		1	*bk*	1	0.8	*85.8*	0.1552	***13.32***	1143	98.03x10^3	396.3	255.9
High-End Mobile (0.8-1.8)GHz	16	4	*bk*	1	1.67	30.4	0.851	25.87	786.5	**23.91x10^3**	2244.0	255.5
		4	*ks*	1	1.8	**28.8**	1.073	30.91	890.3	25.64x10^3	3847.0	205.8
		2	*bk*	1	1.6	37.97	0.4753	18.05	**685.3**	26.02x10^6	1064.0	317.5
	128	4	*bk*	1	1.69	21.85	0.8958	19.57	427.7	**9.34x10^3**	2477.0	249.2
		4	*ks*	1	1.8	**20.86**	1.065	22.22	463.5	9.67x10^3	3847.0	203.7
		4	*bk*	1	1.67	22.17	0.8623	19.12	**423.8**	9.40x10^6	2244.0	260.6
Power-Constrained Servers (1.6-4.2)GHz	16	1	*bk*	1	1.69	49.9	0.2779	**13.87**	692	34.53x10^3	619.2	373.8
		4	*bk*	8	4.2	**17.84**	10.88	194.1	3464	61.79x10^3	8327.0	747.9
		4	*ks*	5	3.45	***19.71***	*5.405*	*106.5*	2100	41.39x10^3	5195.0	621.3
		1	*bk*	1	1.6	52.42	**0.2686**	14.08	738	38.69x10^6	531.9	375.6
	128	1	*bk*	1	1.6	44.2	**0.2957**	**13.07**	577.6	25.53x10^3	531.9	426.6
		4	*bk*	8	4.2	**11.36**	12.14	137.9	1566	17.79x10^3	8327.0	898.5
Fast Low Power Servers (1.6-4.2)GHz	16	4	*bk*	2	2.63	22.12	2.045	45.24	1001	**22.14x10^3**	3877	372.9
		4	*bk*	1	1.6	37.97	0.4753	18.05	**685.3**	26.02x10^6	1064.0	317.5
	128	4	*bk*	2	2.63	15.29	2.11	132.26	493.3	**7.54x10^3**	3877.0	389.5
		4	*bk*	1	1.67	22.17	0.8623	19.12	**423.8**	9.40x10^6	2244.0	260.6

5 Exploration of Multiplier Unit Structure for Energy Efficient Vector Processors

5.1 Introduction

In this chapter, we perform comprehensive (power, delay, energy, area, power density) design space exploration of multipliers for vector processors. The exploration includes circuit-level parameters like the number of pipeline stages and clock-gating support, as well as architectural parameters such as the number of lanes. In order to join architectural-level information (e.g. microbenchmarks) with circuit-level outputs (e.g. VMU power measurements) we use an automated and integrated architecture-circuit exploration framework (explained in detail in Section 5.2.1) that consists of several simulators at different levels, and we perform this exploration with microbenchmarks generated with data obtained from vectorized applications of the San Diego Vision Benchmarks [112]. To achieve high accuracy and estimate properly the interconnect and routing overhead inside the multiplier we use PnR estimation flow (explained in detail in Chapter 3) as a basis for the framework.

We observe how architectural-level parameters (e.g. vector length) affect the circuit-level metrics (e.g. multiplier's power dissipation) and how circuit-level parameters (e.g. multiplier's clock cycle) impact the execution time of a microbenchmark. We present a discussion of multipliers' power, timing and area characteristics, considering clock-gating, for various kinds of testbenchmarks and verified physical results for a recent technology (Section 5.3). Additionally, we present a study of vector multi-lane advantages and drawbacks for VMU. The final goal of this design space exploration is to provide guidelines for designers of low power and energy efficient vector processors. Guidelines are based on Pareto-Optimal trade-off graphs of VMU design space that we get as result of our framework.

5.2 Methodology

This section describes the exploration framework used to perform the design space exploration of VMU, the framework parameters and the test benchmarks that we use.

5.2.1 Exploration Framework

The framework is depicted on Figure 5.1. Basis for this framework is PnR estimation flow (Chapter 3), i.e. corresponding explanations and details are valid here. Additionally, it is upgraded with the features explained in Section 4.2.1. Therefore, here we focus here on the added features, i.e. differences, characteristic for this framework.

The framework includes architectural- (*VectorSim*) and circuit-level (*RC, Encounter, NCsim*) simulators and tools, as well as an interfacing tool (*tBenchGen*) and an exploration engine. For various circuit- and architectural-level parameters (explained in detail in section 5.2.2) we obtain the metrics of interest of our VMU (described in Table 4.1).

We use *VectorSim* (explained in Chapter 2) and we feed it with the architectural and some circuit parameters. Architectural stage generates data and timing traces for the vector multiplications. The interfacing part transforms this architectural-level information into Verilog test benchmarks (*tBench*s) using *tBenchGen* (explained in Section 4.2.3).

At the circuit level, we incorporate all the circuit parameters into handcrafted HDL codes of multipliers which are supplied to Cadence RTL Compiler (*RC Synthesis*) to produce different multiplier's synthesized mapped netlists and to perform

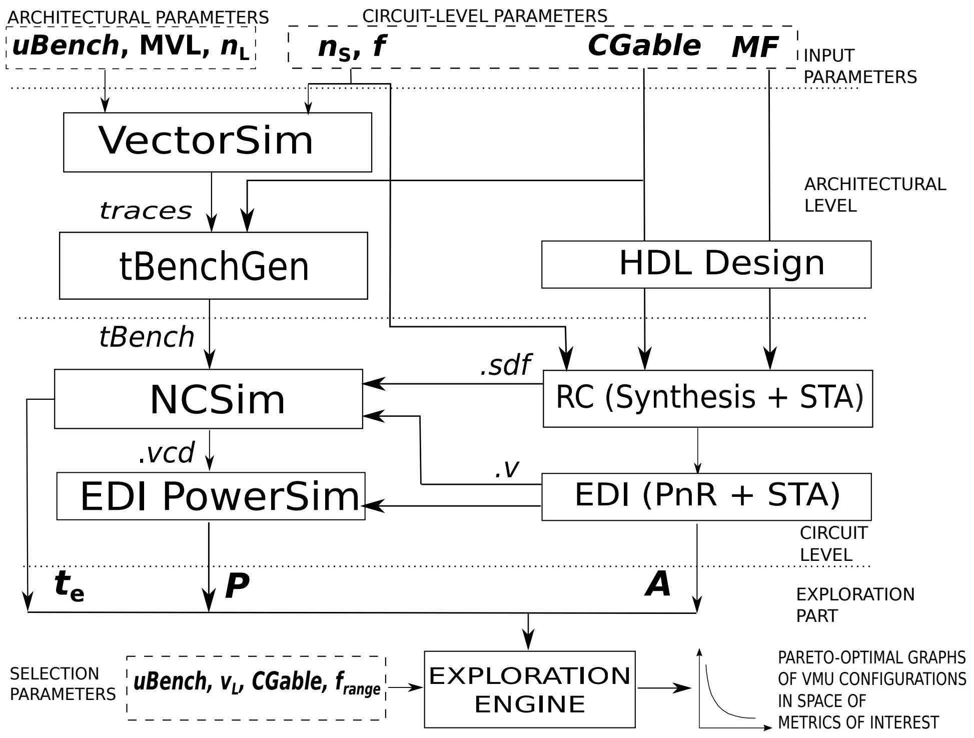


Figure 5.1: Block diagram of the framework's steps, parameters, and metrics.

STA. For VMU configurations with multiple lanes, multipliers in all lanes are identical. We provide synthesized Verilog netlists together with the physical layout information to Cadence Encounter Digital Implementation System to get placed and routed designs (*EDI P'n'R*) and to again perform STA.

The next step of the framework is a precise estimation of power metrics (*P*) using *EDI Power Simulation*. The inputs are placed and routed designs and corresponding vcd files.

The final step is the exploration part, where the design points (VMU configurations) are supplied to the exploration engine (a combination of MATLAB and Perl scripts). The design points are first filtered with selection parameters: *uBench*, *CGable*, v_L, and frequency range (f_{range}). and then depicted using Pareto-optimal trade-off graphs in the space of desired metrics of interest (e.g. E-D space).

Table 5.1: Vectorized microbenchmarks (*uBench*). All the microbenchmarks are run for *fullHD* inputs.

Disparity Map (computeSAD) computes depth information using dense stereo. It is used for robot vision for stereo vision.
Maximally Stable Regions (FitEllipses) does Blob detection in images. It is used for stereo matching and object recognition
Feature Tracking (ImageBlur) extract motion from a sequence of images. It is used for Robot vision for tracking.

5.2.2 Framework Parameters

We first present the vector architecture specific parameters.

uBench is a microbenchmark (kernel) vectorized by hand. We use three different integer *uBench*s from San Diego Vision Benchmark [112] run for *fullHD* inputs (described in Table 5.1). These *uBench*s are used in mobile devices and can also be found in server workloads. *uBench*s are multiplication intensive, so multiplier's impact on the *uBench*s' execution time is significant. We keep their size between 100k and 150k test vectors to keep circuit simulation time reasonable.

MV_L and $\boldsymbol{n}_L$ are explained in Section 4.2.2

We now present the circuit-level parameters which are primarily needed for the circuit-level part of the framework:

- ***f*=(1/*T*$_{CLK}$)** is the clock frequency of the multiplier and of the whole processor, while T_{CLK} is the clock cycle. We use (0.1-2.05) GHz as a practical frequency range for low power vector processors. Vector processors are usually able to afford high frequency due to two reasons: (1) control logic is simple, so it is not a timing bottleneck and (2) they can afford deep pipelining because latency is hidden. f_{range} is a frequency range used as a selection parameter.

- ***MF*** is the multiplier family (algorithm). We consider only multiplier families that are suitable to be fully-pipelined. Structures that are not fully-pipelinable produce pipeline bubbles and this is inefficient for vector processors, as vector operands are streamlined to the vector ALU at the rate of one per cycle. For this reason, we do not include serial (with a feedback loop), but only parallel (fully-pipelined) structures. Therefore we consider array and tree *MFs* (explained in Section 5.2.3). carry-save array (*ar*) and Wallace (*wl*) [113] multipliers are chosen as representative examples for each of the mentioned *MFs*. Array multiplying algorithm is selected as it has a regular layout with simple interconnects (which became important in deep submicron design). Tree multiplier is chosen as it is supposed to be both timing- and area-efficient (delay proportional to the logarithm of the operand width). However, it has irregular layout with complicated interconnects which can spoil their power, timing and area results in recent technology nodes. For partial product generation a simple AND network is chosen rather than Booth one, as we experimentally found that when it is attached to multiplier, it performs better in terms of area, timing, and power (a previous study confirmed these findings [13]). As the final stage adder, for both considered multiplier families we choose Brent-Kung adder, as it is typically the optimal one when clock-gating and pipelining are considered 4.3. We implement our multipliers so they support 32-bit 2's complement integers.

- ***CGable*** is explained in Section 3.3.2.

- $\boldsymbol{n_S}$ indicates the number of pipeline stages in the multiplier (pipeline depth). We only increase n_S for a particular multiplier family (*MF*) if further pipelining provides noticeable f increase. In particular, we implement *ar* for 1-9 and *wl* for 1-5 stages. We consider a wide range of pipeline stages as the vector arithmetic units are typically fully and deeply pipelined, as in vector processing the throughput of an arithmetic units is more important than its latency. Additionally, pipelining is quite necessary for multipliers due to two reasons: (1) to achieve higher frequency, and (2) to reduce glitching. In parallel multipliers (especially tree ones), glitches or spurious transitions are typically generated because partial product bits arrive at the same time but are added serially, and the input-to-output paths in adder cells have different delays. The input register (first pipeline stage) of an *n-stage* multiplier consists of 2x32 DFlipFlops for the input operands. Additionally, each *n-stage* multiplier has *n-1* pipeline clocked registers which are used to save intermediate results. The size of each register depends on the *MF* structure and the place inside the multiplier where the register is inserted (i.e. on the way the multiplier is pipelined).

5.2.3 Multiplier Families

Parallel, fully pipelinable, multipliers consists of three phases: (1) partial product generation (PPG), (2) partial product reduction (PPR), and (3) final carry-propagate addition (CPA) (final stage adder). The PPR is the main part of these multipliers so they are divided by PPR algorithm. In this research, we focus on *ar* and *wl* algorithms. Even more detailed explanations of each *MF* are available in [14, 33].

Array (*ar*)

This is the most regular and simple multiplier. The algorithm is similar to multiplication "by hand" as each partial product is being added. This concept is illustrated on Figure 5.2. A diagonal in *ar* multiplier is equivalent to a column in multiplication matrix.

As shown on Figure 5.2, the multiplier consist of FA, modified full adder (MFA), HA, and modified half adder (MHA) cells. Implementation of FA and HA is explained in Section 3.3.3, while MFA/MHA consists of an AND gate that creates a partial product bit and FA/HA cell. The partial product is being added with a partial product from the previous row in case of MHA, while for MFA it is being added with sum and carry bits from the previous row. A block diagram of MFA is shown in Figure 5.3.

As it was mentioned above *ar* multiplier has simple and efficient physical implementation (layout). The reason is its highly regular structure and using only short wires that go from one FA to another horizontally, vertically, or diagonally. For this reason, *ar* is done in a square.

The delay (T_{CLK} assuming $n_S = 1$) is $O(n)$ and area is $= O(n^2)$

Wallace (*wl*)

Tree multipliers are an efficient way to reduce multiplier's delay from $O(n)$ to $log(n)$. The main idea is to is to reduce the partial product tree down enough

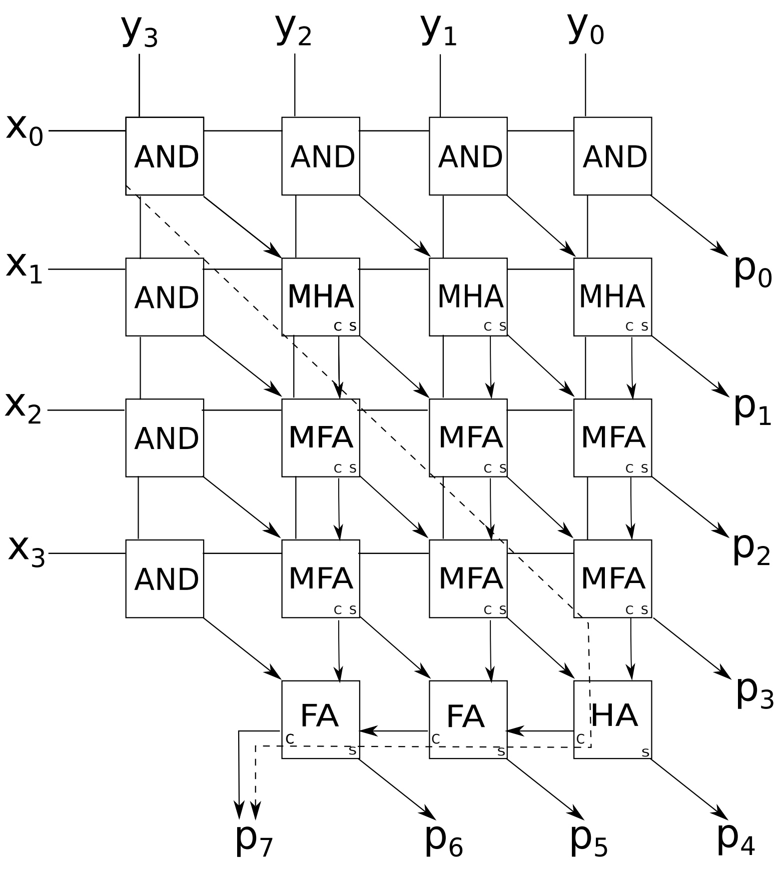


Figure 5.2: A 4x4 unsigned carry-save array (*ar*) multiplier.

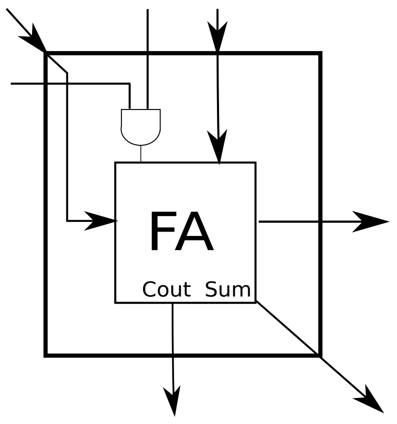


Figure 5.3: A modified full adder (MFA) cell.

so that a fast adder (e.g. Brent-Kung adder) can be employed in CPA stage. The reduction is achieved by combining three bits at time by using carry-save adder

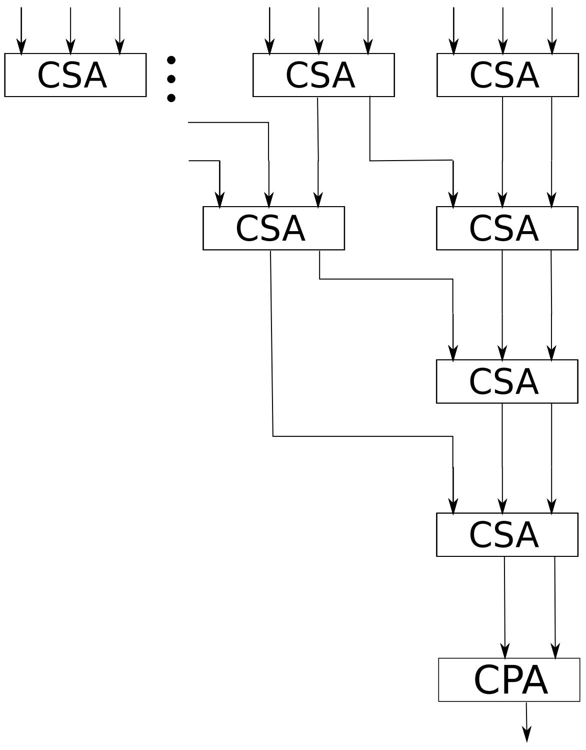


Figure 5.4: Wallace tree reduction using CSAs.

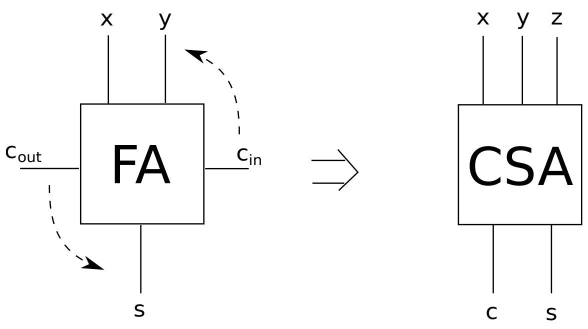


Figure 5.5: CSA -> FA transformation.

(CSA) cells as shown on Figure 5.4. CSA cells are FA cells used in a different manner, as it shown in Figure 5.5.

As a side-effect of these reductions that provide timing improvement, there are physical implementation issues: layout is irregular, wasted area are is increased, complexity is higher, and interconnect is more complicated, thus incurs some routing overhead.

Table 5.2: Wallace vs. Array ratio for area and power

	Area (*CGable*)		Power (*tBench*)			
	noCG	*CG*	*100%*	*noCG*	*CG*	*0%*
Min	0.47	0.50	0.09	0.12	0.13	0.55
Max	1.08	1.08	0.98	0.99	1.03	1.52
GeomMean	0.64	0.66	0.21	0.24	0.29	0.80
GeomStdDev	1.28	1.22	1.75	1.65	1.70	1.30

5.3 Multipliers' Characteristics

We analyze the power, timing, and area characteristics of the examined multiplier families by observing the lowest power configurations for each pair of frequency and multiplier family. Table 5.2 provides statistics for area and power ratios of *wl* over *ar* multipliers for all four *tBench*s of the lowest power configurations, with and without clock-gating mechanism integrated (*CGable*=*CG* and *noCG*). Power graphs of the lowest power configurations are shown on Figure 5.6. Since the goal here is to analyze characteristics of multipliers alone, we test configurations with one lane.

5.3.1 Area

Except for slow designs ($f < 0.3\,\text{GHz}$), *wl* is more area efficient than *ar*, as it can be observed from Table 5.2. For higher frequencies *ar* needs more pipeline stages (higher n_S to achieve the same timing as *wl*, which increases area.

5.3.2 Timing

Maximal achievable frequencies for *CGable*=*noCG* are 2.05 GHz and 1.85 GHz for *wl* with 5 stages and *ar* with 9 stages respectively. For *CG* both 5-stage *wl* and 9-stage *ar* achieve the same max frequency - 1.7 GHz. As *wl* has an irregular structure, its clock tree is also irregular, so the insertion of a clock gating mechanism affects overall timing results to a higher extent than in *ar*. Maximum frequency of multipliers would be higher if a high-performance technology is used instead of low power one.

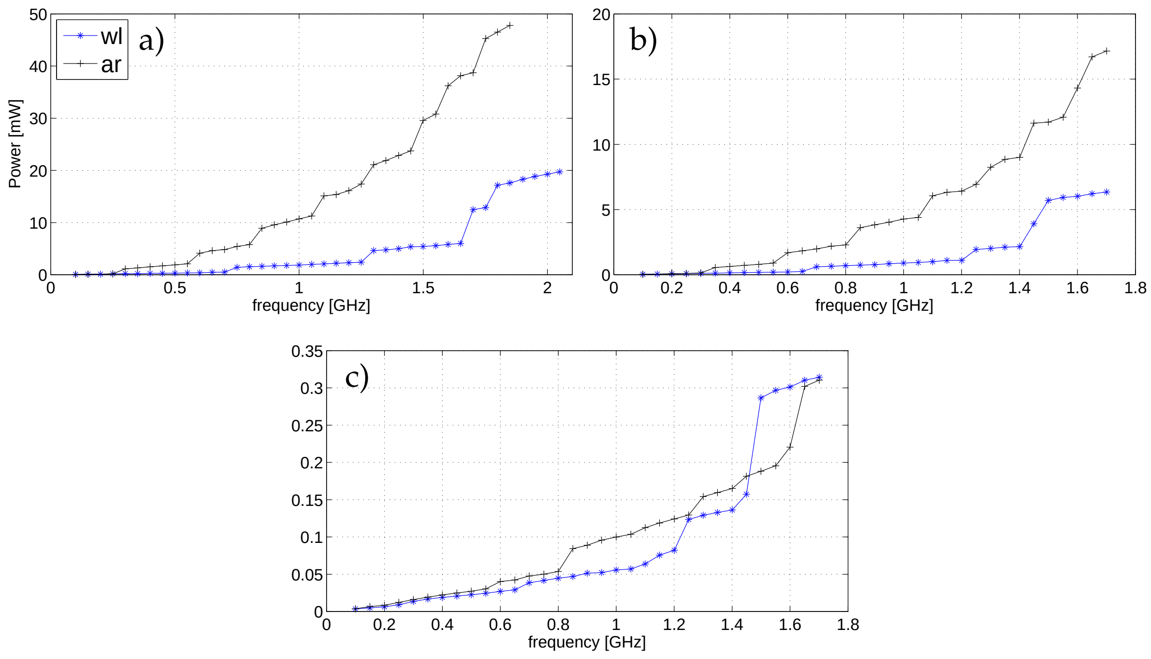


Figure 5.6: Power dissipation of the lowest power configurations of analyzed multiplier families for (a) *noCG*, (b) *CG*, and (c) *0%*, for MV_L=128.

5.3.3 Power

Similarly as in the previous chapters we observe that P_{stat} is almost negligible, it is an order of magnitude less than P_{dyn}, even for *CG*. This applies to all the *tBench*s except for *0%*, where P_{stat} contributes with 20%. Therefore, the power of a multiplier (P) is practically equal to its dynamic component ($P \approx P_{dyn}$).

Figure 5.6 shows the lowest power configurations for *noCG*, *CG*, and *0%* for MV_L=128. Qualitatively, power graphs for MV_L=16 and 128 have similar shapes for all *tBench*s. Absolute numbers indicate that power is slightly higher for MV_L=16 in case of *noCG* and *100%* as in case of MV_L=128 there is less correlation between data being executed (slightly higher α_I). However in case of *CG*, the difference is drastic, power dissipation for MV_L=128 is almost 3 x higher than for 16. It is due to much better exploitation of the multiplier, so the clock-gating signal is active less often than for MV_L=16. The graph for 100% is not shown for the sake of simplicity. It has the same shape as *noCG* one, but it is on average 5-10% higher.

The reduction of power achieved with clock-gating (*CG* vs. *noCG*) is in average around 5 x and 2 x for MV_L=16 and 128 respectively. The reduction strongly depends on the multiplier usage, so they vary across the *uBench*s. They are the highest for *Feature Tracking*, the lowest for *Disparity map*, while the savings for *Maximally Stable Regions* are in between and they are close to the average savings for all the three *uBench*s.

Power dissipation depends largely on n_S. Despite the glitch reduction achieved with pipelining, the overhead of adding registers is high. Therefore, for a given frequency, the lowest power configuration is almost always the one with the fewer stages. As a result of their topologies and critical paths, *ar* requires higher n_S than *wl* to achieve the same f. This is the main reason why *wl* in most of the cases has significantly lower power than *ar*.

The staircase shape of $P(f)$ graphs (Figure 5.6) is a result of pipelining. When we want to make a multiplier able to work on a higher frequency, we first try to achieve this with circuit optimization techniques (buffering, resizing, etc.). In this case, power increases approximately linearly with frequency. However, when it is not possible to further increase frequency with circuit optimization techniques, we apply pipelining (increase n_S). As a result, we obtain these high gradient increases.

While for low frequencies *wl* and *ar* perform similarly, on the rest of the explored f scale, *wl* in most cases performs better. The only exception is *0%* where *ar* has lower power for $f > 1.45$. Thus, we can consider *ar* as a suitable choice for highly memory bound workloads when clock-gating is active most of the time, while for almost all other scenarios *wl* has lower power.

From Table 5.2 we can observe that geometrical mean (GM) of *wl*/*ar* power ratio differs among *tBench*s: GM(*100%*) < GM(*noCG*) < GM(*CG*) < GM(*0%*). There are two reasons for this: (1) Switching activity of *tBench*s. When switching activity of a *tBench* is high, P_{rest} is the dominant power component. P_{rest} is proportional to the area of the multiplier. (2) Clock-gating. The more clock-gating is active, the more significant P_{ct} is. When clock-gating is active most of the time, total P is practically defined by P_{ct}. In this case, designs with regular layout (thus less complicated clock-tree) perform better in terms of power. Thus, area inefficient and regular layout designs (*ar*) can compete in power efficiency with low area and irregular layout ones (*wl*) only for low activity *tBench*s, when clock-gating is active most of the time.

5.4 Multi-Lane Effectiveness

In order to fully understand the results expressed via Pareto-optimal trade-off graphs (Section 5.5) we analyze the effectiveness of using multiple lanes, showing their behavior on the measured frequency range using averaged results of all *uBench*s. For a given combination of MV_L, *tBench*, and f, we pick the configuration with the lowest power for each n_L. The execution times for all *uBench*s are normalized before averaging to assure the correctness of averaged results. We comment on *uBench*s' results separately when there are interesting differences between them.

Table 5.3 presents the ratios of power dissipated by *2-* and *4-lane* configurations over *1-lane* power. For the sake of simplicity, we choose two points in the frequency range explored, close to the extremes: 0.4 GHz and 1.6 GHz. Generally, there are three factors that define VMU power, and power ratios in case of vector multi-lane: (1) the amount of parallel multipliers (n_L), (2) switching activity factor of data at the VMU inputs (α_I), and (3) the usage of VMU. While the effect of the first one is obvious, the last two factors affect VMU power in less obvious ways, thus, require deeper analysis.

As data of consecutive elements inside a vector is typically correlated, α_I plays a more important role in vector than in scalar processing. The effect of α_I is clearly observable for *100%* where VMU usage is always the same (no VMU idleness). With multi-laning we actually slice a vector into n_L shorter vectors decreasing in that way correlation between data inside the vector (i.e. increasing α_I). Therefore, although we increase the amount of hardware 4 times, we actually increase power 5 times. Shorter vector lengths also produce some increase of α_I, therefore ratios are slightly higher for MV_L=16 than for 128. We observe that power dependency on α_I decreases with frequency. The reason is reduced glitching. For high frequencies multipliers with more pipeline stages are used, and pipelining reduces glitching [98].

The notable increase of α_I with the increase of n_L is a consequence of the characteristics of image data. Neighboring pixels often have similar values. When processing them as a vector, this results in high correlation between consecutive values seen by the VMU, which translates into low α_I, even for short vectors. However, if data is processed on the multiple lanes, elements seen by one lane are interleaved,

Table 5.3: Power Dissipation Ratio of 2- and 4- over 1-lane VMU

f[GHz]	0.4				1.6			
n_L	2		4		2		4	
MV_L	16	128	16	128	16	128	16	128
noCG	1.93	2.19	3.73	4.5	2.19	2.27	4.36	4.5
CG	1.53	2.52	2.31	5.32	1.36	1.69	1.81	3.25
100%	2.24	2.24	5.02	5.01	2.40	2.31	4.96	4.77

i.e. they are not neighbors anymore. For this reason, there is the aforementioned notable increase of α_I (and consequently power), in designs with multiple lanes.

The usage effects are most visible comparing the ratios of power dissipated for low and high frequencies in case of *CG*. The ratios are significantly lower for higher frequencies as for high frequencies the memory system becomes a bottleneck. Adding more lanes results in more idle cycles which causes clock-gating to be more active, thus average power is lower.

Power ratios for CG and noCG are notably higher for MV_L=128 than for 16. As it has been already mentioned in Section 5.3, in case of short MV_L (16) we have lower average power as the VMU is more time idle. Adding more lanes means that the "effective" vector length is actually MV_L/n_L which augments this effect. This is especially visible for *CG*, during idle cycles, the VMU is clock-gated.

Table 5.4 shows the speed-up obtained with 2 and 4 lanes over a *1-lane* VMU. We have chosen the same frequency points as in the previous example: 0.4 GHz and 1.6 GHz. As can be expected from Amdahl's law, doubling the number of lanes provides exactly 2 x speedup for the *100%*, since there are multiplications every cycle. *CG* and *noCG* have practically the same results since the presence or absence of clock gating at a given frequency does not affect the execution time as long as they have the same n_S. In these two cases, the speed-up decreases pretty linearly with frequency. As a consequence, increasing the number of lanes is more effective for VMUs operating at lower frequencies (low power region). The reason is that in these *tBench*s memory access time is modeled (unlike in *100%*), and main memory access latency translates into more cycles when the processor operates at higher frequencies. Moreover, cache misses are more difficult to hide when the v_L is short, so speed-up degrades with frequency to a higher extent in this case.

Table 5.4: Speed-up of 2- and 4- over 1-lane VMU

f[GHz]	0.4				1.6			
n_L	2		4		2		4	
MV_L	16	128	16	128	16	128	16	128
CG, noCG	1.23	1.74	1.43	2.59	1.16	1.48	1.31	1.80
100%	2	2	4	4	2	2	4	4

EDP characteristics of multiple-lane VMUs are shown on Figure 5.7 for *CG* and both MV_L=16 and 128. While for MV_L=128 2- and *4-lane* VMUs are better solutions than *1-lane* in terms of EDP, for MV_L=16 *1*- and *2-lane* VMUs are the most efficient in terms of EDP. Regarding the rest of PD^nP metrics, for the sake of simplicity, we do not present the figures but we highlight the most interesting observations. When considering energy (PDP), 1-lane outperforms 2- and 4-lane configurations for all *tBench*s. However, when we put the accent on the performance and consider ED^2P, the *4-lane* VMU performs the best for all MV_L and *tBench*s. The *(f,* ED^nP*)*(n>1) graphs have the minimum around 0.63 GHz and, as can be seen in Figure 5.7, this is actually the most energy-efficient operating region. Results for PD^nP are functions of P and t_e, thus they are directly explained with the discussion above. Regarding per *uBench* results, we have observed that the results for *Disparity map* are close to the average ones, while other two *uBench*s differ to some extent. *Feature Tracking* takes the most benefit of vector multi-lane being *4-lane* VMU more often the most efficient choice, while for *Maximally Stable Regions* the situation is the opposite and 1-lane performs better than in averaged results.

5.5 VMU Design Guidelines

This section presents guidelines on efficient VMU design for two groups of the targeted low power processors. The first group represent low power embedded processors with short MV_L (16) that operate inside the lower frequency range (0.1 GHz-1.0 GHz). Embedded processors typically have restricted power budgets, therefore, they cannot afford long vector registers neither high frequencies. Examples of this market are ARM Cortex-M*x* processors. The second group assumes low power mobile and low power throughput-oriented server processors with long

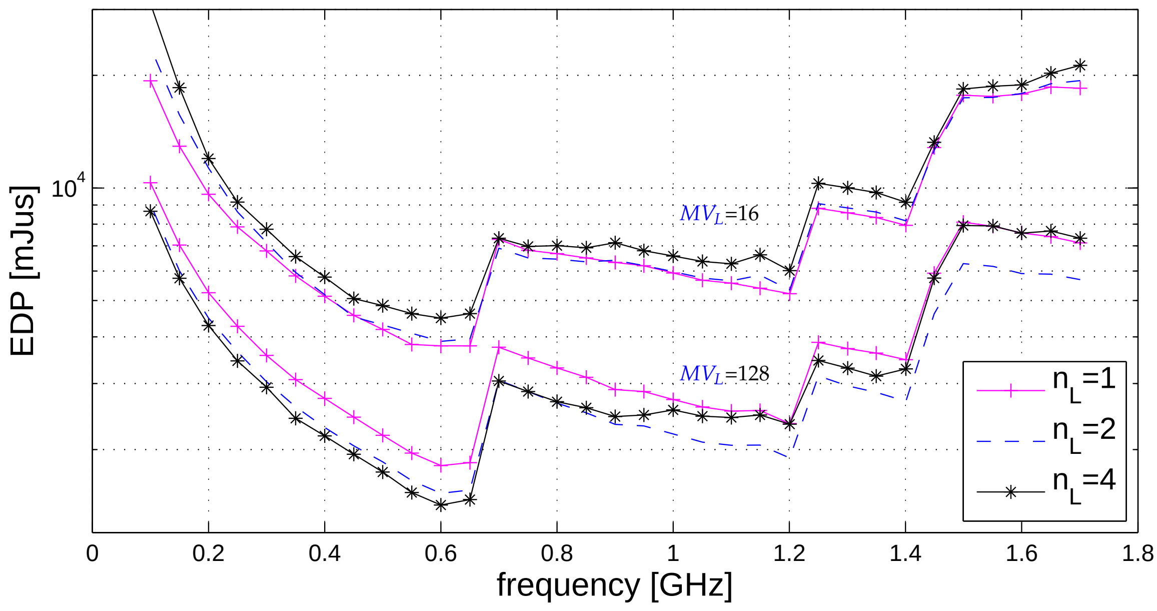


Figure 5.7: *EDP* of *1-*,*2-*, and *4-lane* VMU for *CG*.

MV_L (128) that operate inside the higher frequency range (0.7 GHz-1.7 GHz). Mobile and server processors are expected to take full benefit of their workloads (that exhibit much DLP), thus they can afford long vector registers to achieve energy efficiency. Typical examples of this group are ARM Cortex-A*x* and Hwacha [65] for mobile, and ARM-based servers [37] for server market. In this section, we consider only VMUs with clock gating support as it is the most relevant from a low power implementation perspective. Additionally, we assume averaged results of all *uBench*s.

Pareto optimal design points are shown on Figure 5.8, for both groups of processors in the Power−Execution time ($P - D$) trade off design space. The most efficient designs for each metric of interest are listed in Table 5.5. Design points that perform the best for particular metrics of interest are highlighted in both Table 5.5 and Figure 5.8 using bold font in the output metric column and arrows respectively. Pareto optimal design points for a given processor group are selected by filtering all design points with the selection parameters.

When we emphasize performance (ED^nP, n>1) vector multi-lane turns to be the most adequate choice. We observe it is an effective way to achieve energy-efficient speed-up, especially for v_L=128 due to its fair DLP exploitation. Vector multi-

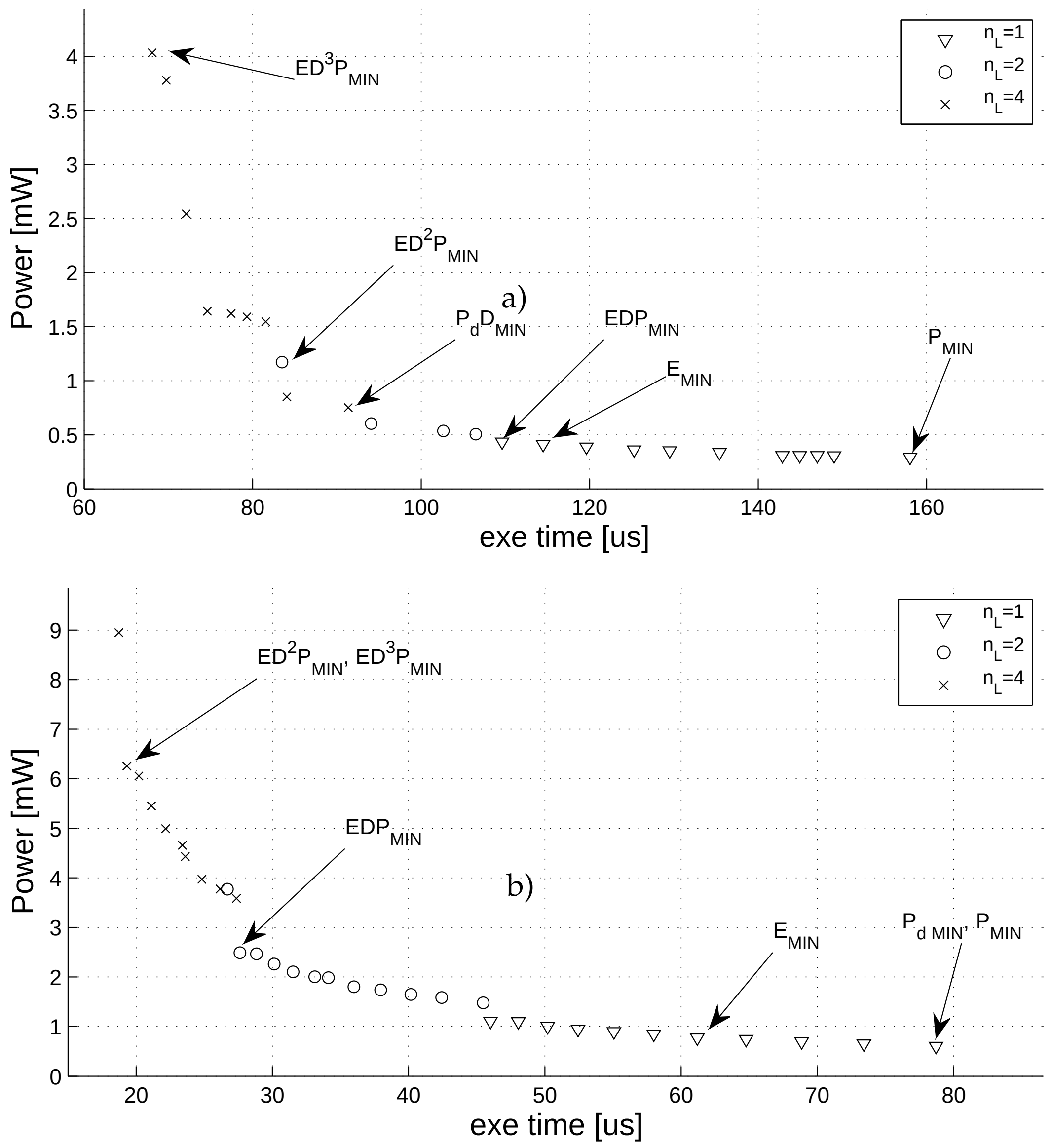


Figure 5.8: Pareto optimal trade-off graphs in $P - D$ space for (a) $v_L = 16$, $f_{range} = (0.1\,GHz - 1.0\,GHz)$ and (b) $v_L = 128$, $f_{range} = (0.7\,GHz - 1.7\,GHz)$.

lane turns to be a more effective approach for fast and low power VMU design than frequency increase. Additionally, it also an efficient choice for achieving a thermal-efficient speed-up (low P_dDP). Consequently, high n_S and high frequency design points do not appear as optimal design points for a given metric of interest. Therefore, *1-* and *2-stage* designs are always the best choices.

As it is expected from Figure 5.7, there are two efficient frequency points: 0.65 GHz and 1.2 GHz. After these break-even points, adding more lanes becomes a more effective approach than what has been increasing frequency. Regarding particular *MF*, as a consequence of discussed in Section 5.3, *wl* is practically always more efficient than *ar* except when we look for ultra low power embedded processors market.

5.6 Related Work

There is considerable prior circuit-level research on comparison of multipliers in the Energy-Delay space [20, 109, 34] However, our analysis differs as we include architecture-related parameters, additional parameters like clock-gating and more recent technology nodes, thus, some of our conclusions are different. For example, we found that *ar* multipliers sometimes beat *wl* multipliers in terms of power when highly memory-bound workloads are considered (Section 5.3). While there is very interesting research on the design of multiplier units for particular architectures such as DSPs [41, 34], their exploration demands are different, they do not target vector architectures neither utilize the same kind of architectural and circuit parameters as we do, thus, their analyses lead to distinct conclusions (Section 5.5).

5.7 Summary

We presented a joint circuit-system analysis, consisting of various simulators and tools, which tackles the problem of design of optimal VMU. We explored VMU design space in a comprehensive way including both circuit (e.g. clock-gating, multiplier family) and architectural parameters (e.g. number of vector lanes, maximum vector length) and showed how they direct optimal VMU configuration. All VMU design points were implemented and verified using PnR flow and simulated using the San Diego Vision Benchmark suite as input benchmark set. We found that Wallace multiplier topology performs the best for almost all the cases due to its area and timing efficiency. The exceptions are very low power design and memory-bound systems where Carry-Save Array (*ar*) outperform Wallace (*wl*) multipliers thanks to its regular interconnects. We showed that vector multi-lane approach beats increasing frequency as a measure to achieve energy- and thermal-efficient speed-up, especially for long vector lengths. Additionally, we analyzed the importance of considering the correlation between vector elements and VMU usage when using multiple vector lanes. Finally, we provided guidelines on optimal VMU design by highlighting the most efficient design points for all the metrics of interest for both low- and high-end low power vector processors. For instance, we observe that *2-lane*, *2-stage* *wl* VMU running at 1.2 GHz performs the best in terms of *EDP* for the low power mobile market.

Table 5.5: Summary of optimal VMU configurations

Selection Parameters			VMU Configuration				Output Metrics						
f_{range}[GHz]	MV_L	*Metric*	n_L	*AF*	n_S	f[GHz]	t_e[μs]	P[mW]	E[mJ]	EDP[mJμs]	ED^2P[mJμs^2]	A[μm^2]	P_d[mW/m^2]
(0.1-1.0)	16	$P_{d\ min}$	1	*wl*	1	0.1	920.1	23.22x10^{-3}	21.36	19.66x10^3	18.09x10^6	6544	**3.55**
		P_dDP_{min}	4	*wl*	1	0.65	**125.6**	0.2924	36.73	4612	57.92x10^3	31.85x10^3	**9.18**
		P_{min}	1	*ar*	1	0.1	920.1	**22.83x10^{-3}**	21.01	19.33x10^3	17.78x10^6	6034	3.78
		E_{min}	1	*wl*	1	0.55	189.9	0.1058	**20.09**	3815	724.5x10^3	6707	15.77
		EDP_{min}	1	*wl*	1	0.6	181.1	0.1154	20.9	**3785**	685.5x10^3	7076	16.31
		ED^2P_{min}	2	*wl*	1	0.65	140.8	0.1993	28.05	3949	**555.9x10^3**	15.93x10^3	12.51
		ED^3P_{min}	4	*wl*	2	1	**98.83**	0.6736	66.57	6579	**650.2x10^3**	29.58x10^3	22.77
(0.7-1.7)	128	$P_{d\ min}$ P_{min}	1	*wl*	2	0.7	78.7	**0.6059**	47.69	3753	295.4x10^3	7224	**83.88**
		$P_dD_{min}P$	4	*wl*	2	0.7	**31.59**	3.055	96.5	3048	96.29x10^3	28.9x10^3	**105.7**
		E_{min}	1	*wl*	2	0.9	61.19	0.772	**47.24**	2891	176.9x10^3	7269	106.2
		EDP_{min}	2	*wl*	2	1.2	27.61	2.489	68.74	**1898**	52.42x10^3	17.29x10^3	144
		ED^2P_{min} ED^3P_{min}	4	*wl*	2	1.2	**19.32**	6.257	120.9	2336	**45.13x10^3**	34.58x10^3	180.9

6 Low Power Vector FP FMA Design Using Advanced Clock-Gating Techniques

6.1 Introduction

In this chapter, we investigate the design of low power fully pipelined double precision IEEE754-2008 compliant FMA unit for vector processors (VFMA). We comprehensively identify, propose, and evaluate the most suitable clock-gating techniques for VFMA running at peak performance periods. We present three kinds of techniques: (1) novel ideas to exploit unique characteristics of vector architectures for clock-gating during active periods of execution (e.g. vector instructions with a scalar operand or vector masking), (2) novel ideas for clock-gating during active periods of execution that are also applicable to scalar architectures but which application is especially beneficial to vector processors (gating internal blocks depending on the values of input data), and (3) ideas that are widely used in other architectures and that we present as its application is especially beneficial to vector processors and for the sake of completeness (idle VFMA). Regarding the second and the third group of ideas, an advantage of vector processing that extends the applicability of clock-gating, is that vector instructions last many cycles, so the state of the clock-gating and bypassing logic remains the same during the whole instruction. As a result, power savings typically overcome the switching overhead of the added hardware (which is often not a case in scalar processors). The proposed techniques are explained in Section 6.4.

We present a fully automated, parameterizable, and scalable exploration framework including our hardware and software (benchmark) generators (Section 6.5). Our methodology is completely aligned with current trends in digital design that promote building generators rather than instances [95, 10, 78]. To achieve high accuracy and estimate properly the interconnect and routing overhead, we integrate PnR estimation flow (Chapter 3) into our framework. We evaluate the presented techniques using both synthetic and real application based benchmarks.

6.2 Related Work

We present the related work for each of our clock-gating techniques together with the description of the technique in Section 6.4.

In the context of alternative low power techniques for floating-point units, interesting approaches have been proposed: memoing (caching results that can be reused) and byte encoding (computation performed over significant bytes). However, detailed and accurate evaluation reveals that the actual savings are often low and with an unaffordable area overhead [43].

Regarding the automatic generation of FMA units, there is influential research [42]. While their framework focuses on the internal implementation on the mantissa multiplier and general microprocessors, it lacks the following features, needed for our research, that we implemented: (1) incorporation of different low power techniques, (2) a complete testing infrastructure including architectural simulators as well, and (3) energy-efficient vector processing aware FMA design. Additionally, their method employs a high-level language to serve as a macro processing language to generate predefined blocks written in the underlying HDL. The disadvantages of these methods lie in the lack of connection between the higher level language and the underlying HDL.

6.3 Floating-Point Arithmetic Background

This section introduces FP representation and FP FMA. Additional details about floating-point arithmetic are available in [45, 14, 33].

Table 6.1: IEEE754 single and double precision formats. The number of bits for each field is shown (bit ranges are in square brackets, 0 = least-significant bit)

	Sign	Exponent	Fraction
Single Precision (32 bits)	1 [31]	8 [30-23]	23 [22-0]
Double Precision (64 bits)	1 [63]	11 [62-52]	52 [51-0]

6.3.1 Floating-Point Representation

Floating-point (FP) representation is the most common way to represent real numbers in computers. It is based on scientific notation to encode numbers, $M * 10^E$, where M and E are the mantissa (base) and the exponent, respectively. For example, 123.4 could be represented as $1.234 * 10^2$. In the same way in binary format, we have that the number 10100.1_2 could be represented as $1.010001_2 * 2^4$.

IEEE754 floating-point numbers have three basic components: the sign (S), the exponent (E), and the fraction (F). IEEE754 double and single precision floating-point formats are shown in Table 6.1. The sign bit '1' indicates negative, while '0' indicates positive numbers. The mantissa is composed of the fraction and an implicit (hidden) leading '1'[1]. The exponent base (2) is implicit and need not be stored. The exponent field contains sum of bias (B) and true exponent (E_T). The bias is 127 for single and 1023 for double precision numbers. Therefore the value represented by an FP IEEE754 FP number is: $(1-2S) * M * 2^{E_T} = (1-2S) * (1+F) * 2^{E-B}$. The most important features of IEEE754 floating point format are shown in Table 6.2.

Special value NaN is used for representing undefined values. This happens when:

- one (or more) operand is NaN,
- the operation is $0 * \infty$,
- the operation is $\infty - \infty$,
- the operations $0/0$, ∞/∞,

[1]An exception are subnormal numbers where the implicit bit is '0'

Table 6.2: IEEE754 single and double precision features.

Feature	*Single*	*Double*
Zero ($\pm$)	$E = 0, F = 0$	
Subnormal	$E = 0, F \neq 0$	
Infinity ($\pm\infty$)	$E = 255, F = 0$	$E = 2047, F = 0$
Not-a-number (*NaN*)	$E = 255, F \neq 0$	$E = 2047, F \neq 0$
Ordinary number	$E \in [1, 254], E_T \in [-126, 127]$	$E \in [1, 2046], E_T \in [-1022, 1023]$
MIN	$2^{-126} \approx 1.2 * 10^{-38}$	$2^{-1022} \approx 1.2 * 10^{-308}$
MAX	$\approx 2^{128} \approx 3.4 * 10^{38}$	$\approx 2^{1024} \approx 1.8 * 10^{308}$

- the operations $x \; mod \; 0$, $\infty \; mod \; y$,
- square roots of negative numbers $\sqrt{x}$, $x < 0$.

Another important special value is infinity ($\pm\infty$). This happens when either input is ∞ and the result is *NaN*. *NaN* and infinity handling are explained in [45].

Since the result of floating point operation is a real number, the exact representation could need more bits than available (F). In these situations a representation that is close to the exact result is necessary. Therefore a rounding operation is needed. While several rounding modes exist, IEEE754 specifies the following ones:

- Round to nearest (tie to even),
- Towards zero,
- Round toward plus infinity, and
- Round toward minus infinity.

6.3.2 Fused Multiply-Add (FMA)

The FMA unit executes the FMA instruction (`FMA R <- A, B, C`) that implements $R = A * B + C$. In contrast to a multiplication followed by an addition, FMA instruction assumes processing all three operands at the same time. It was introduced for the first time in IBM's RS/600 in 1990 [53]. IEEE754-2008 standard defines FMA instruction to be computed initially with unbounded range and precision, rounding only once to the destination format. For this reason, FMA is faster and more precise than a multiplication followed by an addition. The FMA unit performs operand alignment in parallel with the multiplication. This leads to shorter latency (n_S) compared to a multiplication followed by an addition. Additionally, the FMA operation reduces the number of interconnections between floating-point units and the number of adders and normalizers. The FMA instructions help compilers to produce more efficient code. Potential drawbacks are increased latency of FP adder (if executed on the FMA) and a complex normalizer.

A simplified list of steps of FMA flow are:

1. Mantissa multiplication ($M_A * M_B$), exponents addition ($E_A * E_B$), alignment of the addend's mantissa (M_C), and calculation of the intermediate result exponent $E_R = \max(E_A + E_B, E_C)$.

2. Addition of the product ($M_A * M_B$) and aligned M_C.

3. Normalization of the addition result and exponent update.

4. Rounding.

5. Determination of the exception flags and special values.

A simplified implementation block diagram of the FMA unit used in our research is shown on Figure 6.1. As we assume double precision we need an 162-bit adder and a 53x53 multiplier. For the adder and the multiplier, we choose Brent-Kung and Wallace algorithms, respectively, as it is aligned with our findings in Chapters 4 and 5. The aligner performs shifting of the addend based on the exponent difference in order to align it with the product ($M_A * M_B$).

Floating point addition using FMA unit is implemented by setting the first operand to 1 ($A = 1.0$), while floating point multiplication is implemented by setting the third operand to 0 ($C = 0.0$).

Table 6.3: A classification of the proposed techniques using two criteria: (1) Vector Processing-*Specific* or -*Beneficial* and (2) operating mode (*Active* or *Idle*).

	VP-specific	VP-beneficial
Active	*MaskCG, ScalarCG, ImplCG*	*InputCG*
Idle	n/a	*IdleCG*

Table 6.4: Types of instructions where *ScalarCG* is applicable. Capital letters indicate vector operands.

Operation	Vector Instruction
$A * b + C$	`FPFMAVSV` - a multiplicand is a scalar
$A * B + c$	`FPFMAVVS` - the addend is a scalar
$A * b + c$	`FPFMAVSS` - a multiplicand and the addend are scalars
$A +/* B$	`FPADDV/FPMULV` - a multiplicand/the addend is a scalar[1]
$A +/* b$	`FPADDVS/FPMULVS` - 2 out of 3 operands are scalars

6.4 Proposed Techniques

This section presents the proposed clock-gating techniques for VFMA. The classification is presented in Table 6.3.

6.4.1 Scalar Operand Clock-Gating (ScalarCG)

We propose this technique to tackle the cases when one or two operands do not change during the vector instruction. Table 6.4 lists the types of instructions during which *ScalarCG* is active. As there is only one of all the supported vector instructions that has all three operands vectors, most of the execution time at least one operand is scalar.

During these instructions the corresponding input register(s) of scalar operand(s) should latch a new value only on the first clock edge of the execution of the instruction, while during the rest of the instruction, they can be clock-gated. To implement this, we introduce the signals $VS[2..0]$ (Figure 6.1), where $VS[i] = 0$ means that the i-th operand is gated. VS signals are derived from the instruction *OPCODE*. Only the `FPFMAV` instruction, in which all operands are vectors, does not benefit from this technique.

6.4.2 Implicit Scalar Operand Clock-Gating (ImplCG)

This technique is an additional optimization of *ScalarCG* and aims to exploit further the information given through the instruction *OPCODE* for clock-gating, operand isolation, and computation bypassing. In the case of addition and subtraction instructions, such as `FPADDV` and `FPSUBV`, the 53x53 mantissa multiplier is not needed as it is known that one of the multiplicands is '1', thus, we can bypass, isolate, and clock-gate it providing the value of the other multiplicand directly to the adder. There is an analogous situation when we have an `FPMULV` where the addend is known to be '0'. In this case, the 162-bit wide adder and the aligning part are not needed.

To control bypassing, isolation, and clock-gating of the mention modules, we introduce signal *INSTYP* (Figure 6.1), generated from the instruction *OPCODE*, which indicates whether an `FPFMAV` or an `FPADDV/FPSUBV/FPMULV` instruction is executed. *INSTYP* together with VS signals provide information of the instruction type. For example, *INSTYP*=1 and *VS*[0]=1 and *VS*[2]=0 indicate that we have an `FPMULV` instruction while *INSTYP*=1, and *VS*[0]=0 indicate that we have an `FPADDV/FPSUBV` instruction. Circuitry added for implementing *ImplCG* mostly consists of clock gating cells and MUXs.

In the context of instruction-dependent techniques, there is interesting research done in the past for scalar processors [82]. The main advantages of our *ImplCG* proposal over the mentioned research are: (1) we apply the technique for a variable number of pipeline stages, (2) we evaluate power, timing, and area, and (3) we propose the technique for vector processors.

The advantage of the vector over other models (e.g. scalar) is that vector instructions last many cycles, so the state of the related hardware (clock-gating logic and MUXs) maintains the same state during the whole instruction. Thus, there will be less switching overhead than in the scalar case.

[1]When executing an addition/multiplication the "unused" operand is constant during the whole instruction: $A = 1/C = 0$.

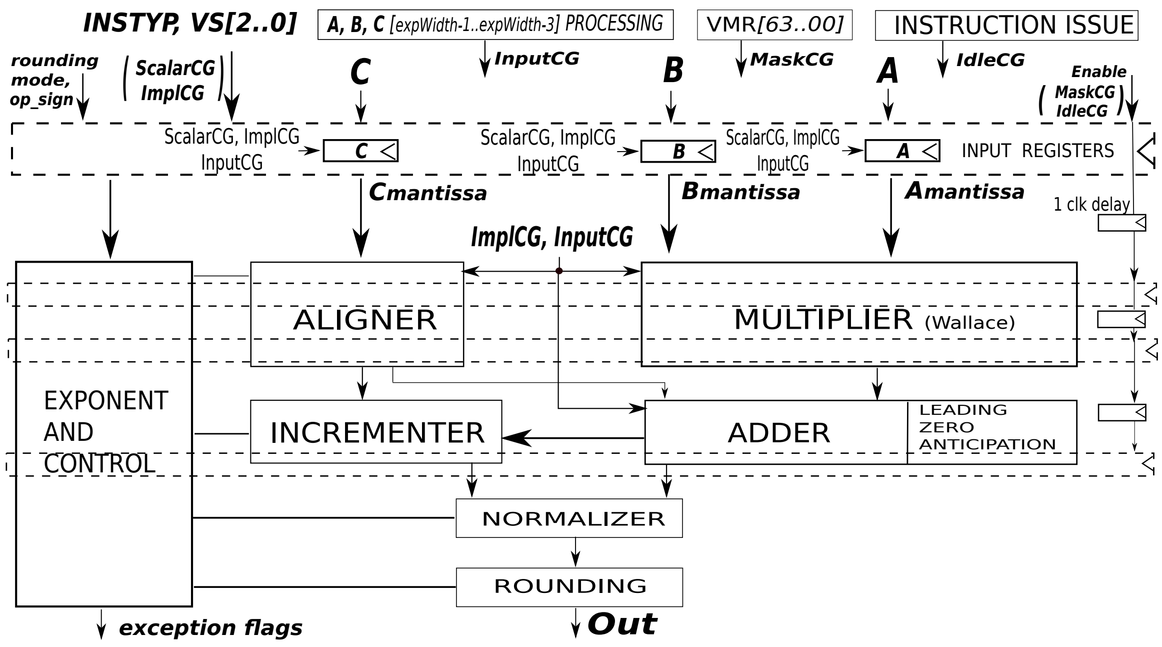


Figure 6.1: A simplified block diagram of a 1-lane, 4-stage, VFMA with all clock-gating techniques applied (*AllCG* technique). Input signals for the baseline without clock-gating are multiplicands (*A*, *B*), addend (*C*), *rounding mode*, and operation sign (*op_sign*) while output signals are result (*Out*) and *exception flags*. Details regarding applied clock-gating techniques are explained in Section 6.4.

6.4.3 Vector Masking and Vector Multi-Lane-Aware Clock-Gating (MaskCG)

Here we target cases when there are idle cycles during the vector mask instructions (e.g. `FPFMAV_MASK`). Common cases where vector mask control is used are: (1) sparse matrix operations and (2) conditional statements inside a vectorized loop.[1] Additionally, we assume the same mechanism is also used to reduce the EV_L to less than the MV_L. With n_L lanes, in the last cycle of the operation, there will be `mod(`EV_L, n_L`)` idle lanes.

The VMR directly controls the clock-gating of the whole arithmetic unit during these idle cycles (Figure 6.1). Regarding the internal implementation, we perform

[1]Vector mask instruction are common. For example, in Facerec (SPEC2000-ref [50]) and Sphinx3 (SPEC2006-ref [50]) applications we measured that 4% and 2% of all the vector instructions are vector mask instructions.

Table 6.5: *InputCG* - conditions under which a hardware block of mantissa arithmetic computations and corresponding input registers can be bypassed, isolated and clock-gated.

Hardware block	*Condition*	*Subtechnique*
Full computation	The result is *NaN*.	$InputCG_{NaN}$
Full computation	The result is ∞.	$InputCG_{inf}$
Multiplier	Multiplicand is ‘0’.	$InputCG_{mul0}$
Adder and aligner	Addend is ‘0’.	$InputCG_{add0}$

clock-gating at pipeline stage granularity [114], so we prevent useless cycles inside the unit i.e. the data is latched in subsequent stages only if necessary. Once the *Enable* signal of the first pipeline stage's register gets the value ‘1’, this *Enable* signal propagates to the end of the pipeline, one stage per cycle (Figure 6.1). In other words, *Enable* signal of n-th stage is actually the first stage's *Enable* signal delayed by $n-1$ cycles. This is implemented by adding a 1-bit wide, $n_S - 1$ long shift register that drives clock-gating cells.

To the best of our knowledge, there is no related work that aims to exploit vector conditional execution with VMR to lower power of vector processors.

6.4.4 Input Data Aware Clock-Gating (InputCG)

Here we identify the scenarios in which, depending on the input data, a part of mantissa processing is not needed for the correct result, thus, can be bypassed. We use a recoded format for internal representation [15], that allows us detecting special cases and zeros with low hardware overhead: it requires inspection of only the three most significant bits of the exponent. Therefore, we can detect the scenarios by adding simple circuitry at the inputs of the VFMA (Figure 6.1). Table 6.5 presents identified scenarios (conditions) when a hardware block of mantissa arithmetic computations and corresponding input registers can be bypassed, isolated and clock-gated. Similar as for *ImplCG*, the hardware required for bypassing, isolation and clock-gating mostly consist of clock gating cells and MUXs.

There are many workloads whose data contain a lot of zero values [11, 30], thus can fairly benefit from the last two subtechniques presented in Table 6.5. Although these techniques are applicable to other architectures as well, their application to vector processors is more efficient since the recurrent values are common within the vector data, thus lowering the switching overhead in added hardware (clock-gating logic and MUXs).

While both *ImplCG* and *InputCG* techniques aim to exploit cases when the addend is '0', in this case, there is no external information of '0' existence via *VS* signals, but is has to be detected, and the gating has to be done on time.

As in the case of *ImplCG*, the research done in [82] presents a related data-driven technique for scalar processors. The main advantages (that enable additional savings) of our *InputCG* technique over the mentioned research are (1) detection of zero operands, (2) distinction between ∞ and *NaN*, and (3) gating mantissa multiplier when processing *NaN*s.

6.4.5 Idle Unit Clock-Gating (IdleCG)

This technique clock-gates the VFMA when no data is supplied to it. The clock (un)gate decision is made in the instruction issue pipeline stage, where it is known if an instruction will be sent to the VFMA in the next cycle (Figure 6.1). As indicated on Figure 6.1, this technique uses the same internal clock-gating circuitry as *MaskCG*. A similar approach is widely used in scalar processors and is known as *Deterministic Clock-Gating* [68, 73]. Nonetheless, this technique has more potential for power savings than its scalar equivalent as it can benefit from the following vector specific advantages:

- VFMAs are used in burst fashion (with idle periods between bursts), since a single `FMA/ADD/SUB/MUL` instruction processes all vector elements in consecutive cycles. This makes clock-gating more efficient as the overhead of its buffers is minimized.
- For very high frequency designs, the issue stage may need an additional cycle to determine if a unit will be used in the next cycle. In a scalar processor, we would need to waste that cycle once per each scalar `FMA/ADD/SUB/MUL` instruction. In a vector processor, we waste this cycle $EV_L - 1$ times less.

Although here we focus lowering power of VFMA when it is active, we present this technique for the sake of completeness as it was used in Chapters 3, 4, and 5.

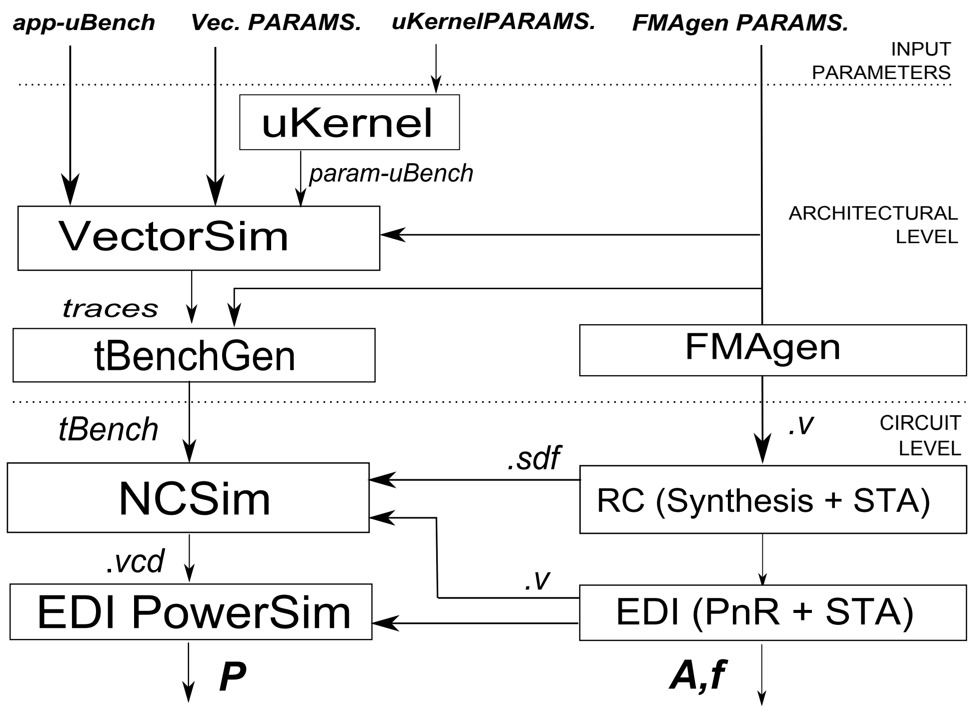


Figure 6.2: A simplified block diagram of the framework.

6.5 Methodology

6.5.1 Exploration Framework

A simplified block diagram of the framework is depicted in Figure 6.2. It includes architectural- (*uKernel*, *VectorSim*, *FMAgen*) and circuit-level (*RC, EDI, NCsim*) simulators and tools, as well as an interfacing tool (*tBenchGen*). For various parameters we obtain power (*P*) and area (*A*) of the VFMA. The basis for power and area measurements is the PnR estimation flow (Chapter 3), thus, corresponding explanations and details are valid here.

The first step is feeding *VectorSim* with vectorized microbenchmarks (*uBench*) and vector parameters (MV_L and number of vector lanes (n_L)). Using these inputs *VectorSim* generates data and timing traces for the vector floating point operations. We use two kinds of *uBenchs* (both explained in Section 6.5.2):

- ***param-uBenchs*** are generated by feeding the parameterizable microkernel *uKernel* with its parameters, and

- ***app-uBenchs*** are extracted from vectorized applications.

The synthesizable Verilog netlists are generated using *FMAgen* (described in Section 6.5.3). The output of architectural-level simulations together with *FMAgen* parameters are transformed into Verilog test benchmarks (*tBench*s) using *tBenchGen* (explained in Section 4.2.3). Afterwards, we use Cadence RTL Compiler (*RC*) to obtain synthesized mapped netlists and to perform STA of the VFMA. All the designs are synthesized for the minimum clock period that provides a safe slack on the critical path. We provide synthesized Verilog netlists together with the physical layout information to the Cadence Encounter Digital Implementation System (*EDI*) to get placed and routed designs and to perform again STA.

In order to verify the designs and extract resulting switching activity information (written into vcd files) we simulate each VFMA in Cadence *NCSim* for each matching *tBench* with back-annotated delays using sdf files. Afterwards, we perform precise power estimation using *EDI PowerSim*.

6.5.2 Benchmarking

This section explains the two benchmarking methods that we employ for an in-dept evaluation of the proposed techniques. The first method has as goal to stress each of the techniques separately, while the second tests all the techniques simultaneously and provides the results for "real world" applications.

Fully Parameterizable Kernel - *uKernel*

We generate different *param-uBench* using the same *uKernel*. It is a variant of the *DAXPY* loop: $D = A * B + C$. The inputs are random values unless specified otherwise. *uKernel* parameters (Table 6.6) are used to determine the characteristics of the generated *param-uBench*. There are parameters that modify the code (*INSTYP*, *ADD/MUL*, *MULS*, *ADDS*), execution (*IR*, p_m), and data (p_{inf}, p_{NaN}, p_0). Listing 6.1 shows an example of uBench pseudocode generated with the uKernel. We iterate *param-uBench* until we reach 10000 test vectors to assure a representative sample, using a uniform distribution. The aforementioned *VS* signals are derived from *INSTYP*, *ADD/MUL*, *MULS* and *ADDS* parameters.

Table 6.6: *uKernel* parameters.

Parameter	Description
INSTYP	It indicates whether we have (0) `FPFMAV` or (1) `FPADDV` and `FPMULV` types of vector instructions.
ADD/MUL	It indicates whether the instruction is addition (0) or multiplication (1) if *INSTYP*=1.
MULS, ADDS	They indicate whether one of the multiplicands and the addend are scalar values, respectively.
p_m	The probability that a bit in the VMR during a vector operation is '0'. The probability that one lane is idle in the last cycle is included in p_M.
p_{inf}, p_{NaN}, p_0	The probabilities that an operand is ∞, not a number (*NaN*), and 0, respectively.[1].
IR	Idleness Ratio, $IR = \frac{T_{IR}}{T_{EX} + T_{IR}}$, where T_{IR} is the average pause length between two subsequent vector instructions and T_{EX} is the average execution time of vector instructions.

Listing 6.1: A simplified *param-uBench* pseudocode generated with *INSTYP*=0, *MULVS*=0, *ADDS*=1, and *pm>0*

```
for (i=0; i< length; i+=MVL) {
  LDV V0 <- A[i+0..63]
  LDV V1 <- B[i+0..63]
  LD R2 <- c
  VMR <- MASK[i+0..63]
  FPFMAVVS_MASK V2 <- V0, V1, R2
  STV D[i+0..63] <- V2
}
```

Application-Based Microbenchmarks - *app-uBench*

app-uBench is a vectorized and floating-point intensive microbenchmark (kernel) extracted from an application. It is a representative part of the application and small enough (between 100k and 150k test vectors) to keep circuit simulation time rea-

[1]To explore potential savings we use the whole range of probabilities, including values that not represent a realistic case (e.g. 100% *NaN*s)

Table 6.7: Vectorized application-based microbenchmarks (*app-uBench*). In brackets are given names of corresponding benchmark suites

Sphinx3 (SPEC2006-ref [50]) is a widely known speech recognition system that includes both an acoustic trainer and various decoders, i.e., text recognition, phoneme recognition, N-best list generation, etc.
Facerec (SPEC2000-ref [50]), explained in Table 4.2.
K-means (modified STAMP [92]) is one of the oldest and most commonly used clustering algorithms. It is a prototype based clustering technique defining the prototype in terms of a centroid which is considered to be the mean of a group of points and is applicable to objects in a continuous n-dimensional space.
Disparity Map - computeSAD (SDVB), explained in Table 5.1.

sonable. We use four different *app-uBenchs* extracted from vectorized applications described in Table 6.7. We selected application with different type and behavior to make the results more general. These are the applications that are used in mobile devices and can also be found in server workloads.

6.5.3 A Fully Parameterizable FMA Generator

We developed FMAgen as a hardware generator written in Constructing Hardware in Scala Embedded Language (Chisel), a hardware construction language aimed at designing hardware by using parameterized generators [10]. Chisel is based on the Scala programming language, and it supports a combination object-oriented and functional programming and good software engineering techniques. We find it as an optimal way to design and test parameterizable FUs. On one side it provides the possibility to design and connect hardware blocks in the same way as in other HDLs (Verilog or VHDL), while on another side it is significantly more flexible (parameterizable) than existing HDLs and provides significantly faster testing. Chisel allows users to code their designs in one source description and target multiple backends without rewriting their designs. The Chisel code is compact, due to its higher level of description than traditional HDLs. Not surprisingly, as a general problem of high-level design approaches, a disadvantage of Chisel-based digital design is that it sometimes has worse quality of results than hand-crafted Verilog [5].

As a base for FMAgen, we take an open source floating-point library - Berkeley Hardware Floating-Point Units (BHFPU) [15]. This open source library internally uses a recoded format (the exponent has an additional bit) to detect and handle special cases, such as subnormal numbers, more efficiently[1]. BHFPU can produce FMAs for an arbitrary floating-point format, i.e. arbitrary number of mantissa and exponent bits.

FMAgen generates synthesizable Verilog code of 1-lane VFMAs according to the input parameters (FMAgen parameters): Clock-Gating technique type (CG_{type}), latency - number of pipeline stages (n_S), and the input floating-point format. Possible values for CG_{type} are any combination of the aforementioned clock-gating techniques (*IdleCG*, *MaskCG*, *ScalarCG*, *ImplCG*, and *InputCG*), including all of them together (*AllCG*) or none of them (*NoCG*). A combination of clock-techniques that will be discussed later is *ActiveCG* which contains all active clock-gating techniques from Table 6.3 (*MaskCG, ScalarCG, ImplCG, InputCG*). n_S can be an arbitrary number. Presented advanced clock-gating techniques are compatible with each other and can be arbitrarily combined. In this study, we put 4 stages as a reasonable limit for a low power processor. Additionally, we set the VFMA input floating-point format to double precision. Apart from the mentioned features that we added to BHFPU (support for all the clock-gating techniques as well as support for combining them arbitrarily, pipelining, and different pipelining styles), we also added full IEEE754-2008 compliance [23] (which introduces some timing overhead). A simplified block diagram of modeled VFMA is shown on Figure 6.1.

We paid special attention to ensure that clock-gating logic does not create a critical timing path. Since we target low power, we do not incorporate any speculative hardware for improving latency, thus, no energy is wasted on precomputed results that get discarded.

We would also like to stress that the FMAgen in combination with high-effort PnR flow is a powerful tool capable of producing a variety of designs with Verilog-like QoR in a highly elegant, scalable and flexible manner. We especially noticed

[1]We assume recoding is done when loading and storing to memory.

highly efficient pruning which is important in case of complex generators with a variety of parameters like FMAgen.

6.6 Evaluation

This section presents an evaluation of the presented vector processing aware clock-gating proposals in terms of power savings (S) and area efficiency. Regarding power measurements, first we evaluate each technique separately using benchmarking method from Section 6.5.2, and afterwards we evaluate combined scenarios using the method explained in Section 6.5.2.

VFMA designs with 1, 2, 3, and 4 stages are synthesized and run for 0.45, 0.85, 1.1, and 1.3 GHz respectively. We assume a *NoCG* 2-lane VFMA as a baseline and its power is 15.6, 30.9, 44.9, and 59.2 mW for 1, 2, 3, and 4 stages respectively. As in the previous chapters, we observe that the static (leakage) power is practically negligible compared to dynamic power. Although it is negligible when considering active operating modes (i.e. the execution inside a vector kernel), when the execution is outside a vector kernel (i.e. when the vector core is inactive), the leakage might be additionally suppressed via power gating.[1] However, power gating is out of the scope of this research since we target lowering power during active operating modes with no performance loss.

We focus on 4-stage results as they are the most important from the processor design perspective. Nonetheless, the 1-stage results are presented as a reference and in most cases it has the highest overhead in terms of power and area across all n_S. For the sake of simplicity, in the rest of this section we do not present further results for 2- and 3-stage designs, but we observe these results regularly scale between results for 1 and 4 stages.

6.6.1 Area Efficiency

Table 6.9 reveals the area efficiency of the proposed techniques. Area for a *NoCG* 2-lane VFMA configuration is 36191, 38060, 40693, and 43419 μm^2 for 1, 2, 3, and 4

[1] The gate signal in this case could be generated from vector kill instruction `KILLV` (similar to `VRIP` instruction in Cray X1 instruction set [93]).

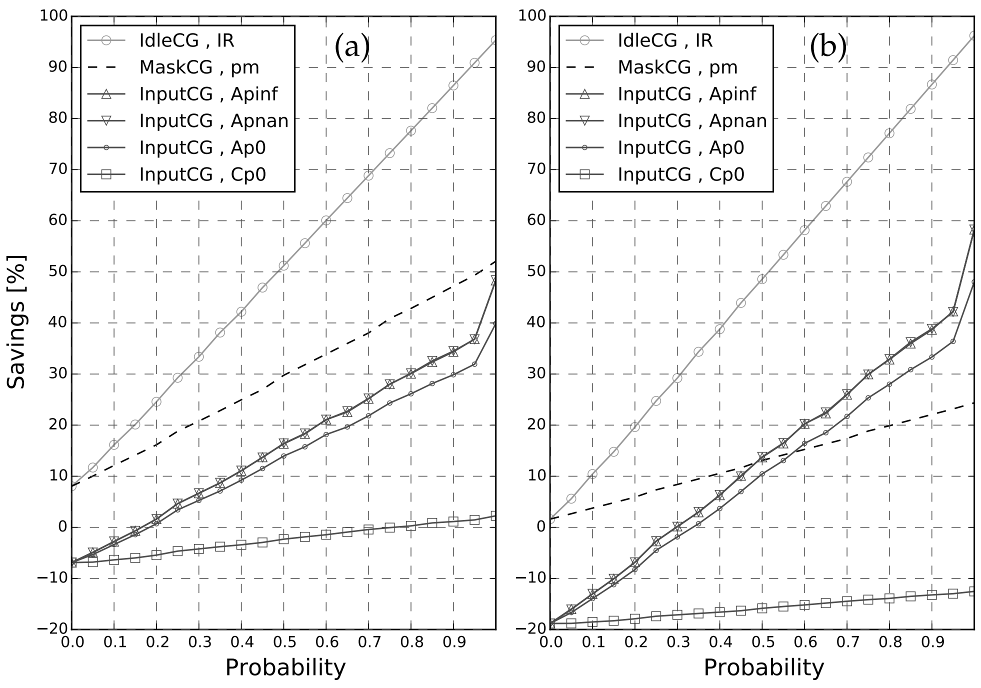


Figure 6.3: Evaluation of power savings over the baseline (*NoCG*), as a function of *uKernel* parameters for *IdleCG*, *MaskCG*, and *InputCG* for 4-stage (a) and 1-stage (b) 2-lane VFMA. For each graph, only one *uKernel* probability parameter from Table 6.6 is assumed to be variable while other parameters are zero. This is indicated in the legend with *technique*, *probability* pairs.

stages respectively. Area overhead is in some cases higher than expected because: (1) during synthesis, we prioritized timing and power over area to assure power savings without spoiling timing and (2) Chisel generated Verilog code is sometimes less area efficient than equivalent manually written Verilog [5]. However, we observe this overhead has a strong decreasing trend as the n_S increases.

6.6.2 Per Technique Power Analysis

Figure 6.3 and Table 6.8 reveal results for each of the presented vector processing aware clock-gating proposals separately, in terms of power savings for 4- and 1-stage VFMA. In these experiments we set MV_L and the number of vector lanes (n_L) to 64 and 2 respectively.

Table 6.8: Evaluation of power savings for *ScalarCG* and *ImplCG* depending on the instruction type against the baseline (*NoCG*). *INSTYP, ADD/MUL, MULS,* and *ADDS uKernel* parameters are variable in these experiments to test all the instructions separately, while the rest are zero.

Vector Instruction	n_S	$S_{ScalarCG}(\%)$	$S_{ImplCG}(\%)$
`FPFMAV`	1	-34.35	-38.27
	4	1.87	0.39
`FPFMAVSV`	1	-10.11	-12.73
	4	16.47	14.55
`FPFMAVVS`	1	-31.47	-33.19
	4	4.07	4.74
`FPFMAVSS`	1	-6.77	-7.77
	4	18.9	18.83
`FPADDV`	1	-10.7	65.07
	4	16.54	42.42
`FPADDVS`	1	-6.72	70.30
	4	18.92	46.16
`FPMULV`	1	-31.36	-22.81
	4	4.12	9.0
`FPMULVS`	1	-6.69	2.52
	4	18.95	23.04

Table 6.9: Area efficiency of the proposed clock-gating techniques against the baseline (*NoCG*) for n_S =1 and 4.

CG_{type}	n_S	*IdleCG/MaskCG*	*ScalarCG*	*ImplCG*	*StageCG*	*InputCG*	*ALLCG*
Ratio(%)	1	98.1	153.8	184.6	98.1	129.0	185.4
	4	96.3	104.6	125.4	96.9	107.3	148.0

We observe that in most of the cases the savings grow with n_S, as more pipeline stages enable finer granularity of clock-gating. Due to its higher practical importance, in the rest of the discussion we focus on 4-stage results.

Figure 6.3 shows results for *MaskCG*, *InputCG*, and *IdleCG*:

✦ ***MaskCG***. Due to its simplicity, this technique comes with practically no overhead and the savings are between 8% and 52% depending on the p_m. The saving attainable when p_m=1 (S=52%) is the maximum possible power reduction for active VFMA.

✦ ***InputCG***. In order to isolate savings for each of the mentioned subtechniques (Table 6.5), we test all them separately by asserting the probabilities p_{inf}, p_{NaN}, and p_0 (Table 6.6) to operands. In *InputCG*$_\infty$, *InputCG*$_{NaN}$, and *InputCG*$_{mul0}$, the corresponding probability affects operand A, while in *InputCG*$_{add0}$ it affects operand C, the addend.

The maximum saving of 48.3% is available when p_{inf} or p_{NaN} is 1 (*InputCG*$_\infty$ and *InputCG*$_{NaN}$). The same savings are available when an operand is *NaN* or ∞, as in both cases the same hardware is clock-gated. The minimum probability p_{NaN} or p_{inf} (of any operand) necessary for saving power is the spot where the savings graph crosses the probability axis (16%).

When considering *InputCG*$_{mul0}$, the maximum saving is 40%, and the minimum probability p_0 (of any multiplicand) necessary for saving power is 18.5%.

Much lower maximum saving (2.3%) is available when the addend is a zero (*InputCG*$_{add0}$), as the adder consumes much less power than the multiplier (around 5 times in average). However, by combining these scenarios at the same time (which is reasonable to assume in a real application case), higher savings would be available. Therefore, even though detecting zero addend and clock-gating the adder and the corresponding aligner and input registers are not enough to justify its existence by itself, it improves overall savings of the complete *InputCG* technique when a real, combined scenario is considered. Since it shares some hardware with other *InputCG* subtechniques, the overhead of adding it is less than the saving it can achieve.

The power overhead of the added hardware can be identified in the case when the probability is 0, i.e. when *InputCG* is never active. The cost is a bit higher than expected taking into account the amount of additional logic that we include (detecting, bypassing, and clock-gating logic). In line with our discussion of area results, Chisel generated designs sometimes suffer from unexpected overhead, and our initial experiments confirm it. However, we observe it significantly decreases as n_S increases, thus, we expect this to be negligible for high n_S.

✦ ***IdleCG***. This technique provides savings between 8% and 95.3%. Although it uses the same hardware as *MaskCG*, the savings are higher in average because here the input data is stable, while in *MaskCG* it is not, thus incurring some switching albeit the first pipeline stage is gated. The saving attainable when *IR*=1 (95.3%) is the maximum possible power reduction when the VFMA is idle.

The results for *ScalarCG* and *ImplCG* are shown in Table 6.8:

✦ ***ScalarCG***. Savings are available for all the combinations of *INSTYP, ADD/MUL, MULS,* and *ADDS uKernel* parameters, i.e. for all vector floating point instructions. Not surprisingly, as the internal multiplier dissipates more power than the adder, latching multiplicand's mantissas provides more savings than latching addend's mantissa (`FPFMAVSV` and `FPADDV` vs. `FPFMAVVS` and `FPMULV`).

✦ ***ImplCG***. This technique significantly improves savings for `FPADDV`/`FPMULV` instructions compared to *ScalarCG*. The highest savings, and the largest improvements compared to *ScalarCG*, are available for `FPADDV` and `FPADDVS` instructions as in these cases the multiplier is gated. As for *ScalarCG*, there are savings for all vector floating-point instructions. Incidentally, we can observe that for `FPADDV` and `FPADDVS` power savings are higher for 1- than for 4-stage VFMA. The reason is that the mantissa multiplier contributes with more percentage to the total power when only one pipeline stage is present.

6.7 Real Application-Based Combined Power Evaluation

Table 6.10 reveals the evaluation of the presented clock-gating proposals as well as workload profiling results for all combinations of *app-uBench* and vector parameters (MV_L and n_L), while figures 6.4, 6.5, and 6.6 visualize the key metrics from the Table. For the reasons explained above, we focus on VFMAs with 4 stages. The meaning of the results shown in table and the figures is:

- *S* - power savings,
- *Active Execution / Idle Execution* - Active/Idle VFMA execution, i.e percentage of *app-uBench* execution time that VFMA is active/idle,
- *IdleCG, ScalarCG, ImplCG, InputCG* and *InputCG* subtechniques efficiency - percentage of execution time that a technique is used,

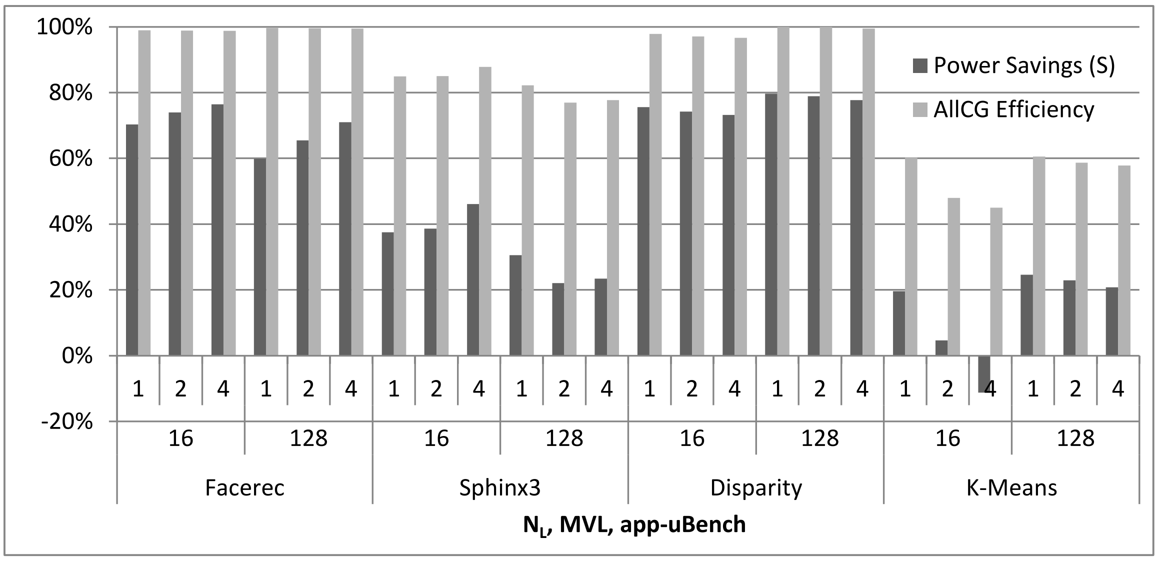


Figure 6.4: Power savings (*S*) and *AllCG* efficiency.

- *AllCG* efficiency - percentage of execution time that any clock-gating technique is used,
- *ActiveCG* efficiency - percentage of execution time that any active clock-gating technique is used, and
- *ActiveExeCG* efficiency - percentage of active VFMA execution that any active clock-gating technique is used,

We show the profiling results as well to understand the VFMA behavior and where the saving come from. Data for MaskCG, InputCG$_{NaN}$, and InputCG$_{inf}$ is not present in the Table 6.10 as they are 0%. The reason is that the selected *app-uBenchs* do not have vector mask instructions. Also, none of the input values are *NaNs* nor infinities. However, abundant vector mask instructions could be found in any vector workload that has conditional execution, so in this kind of workloads we can expect fair savings as the result of *MaskCG* technique. Regrading *NaNs* and infinities, for some other applications and/or input data sets their occurrence might be more common, thus, the benefit of *InputCG*$_{NaN}$ and *InputCG*$_{inf}$ subtechniques will be visible. Common cases of *NaN* and infinity processing are explained in [45].

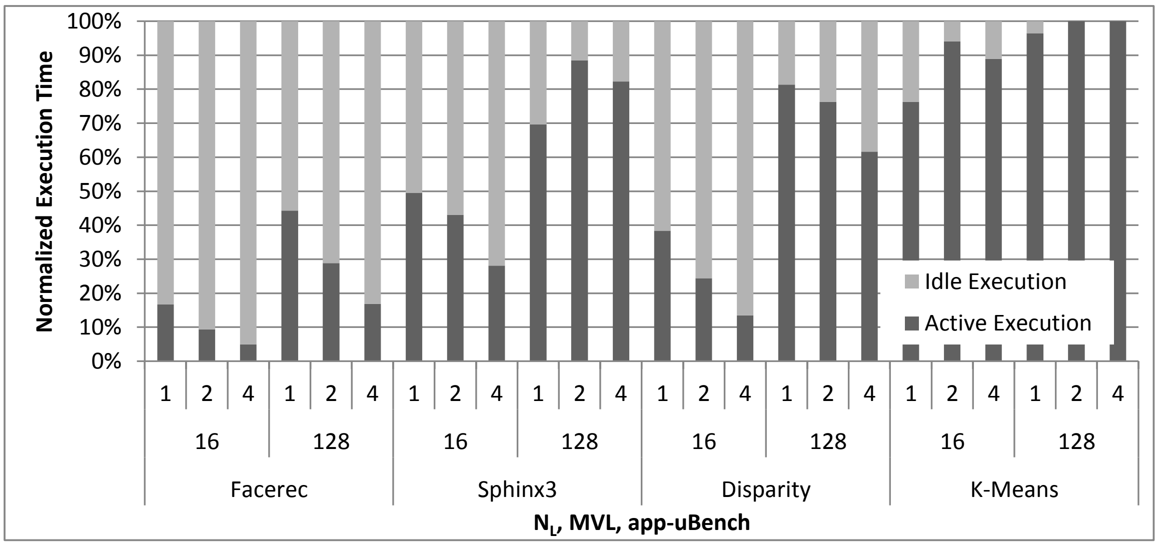


Figure 6.5: *ActiveCG/IdleCG* - Active/Idle VFMA execution.

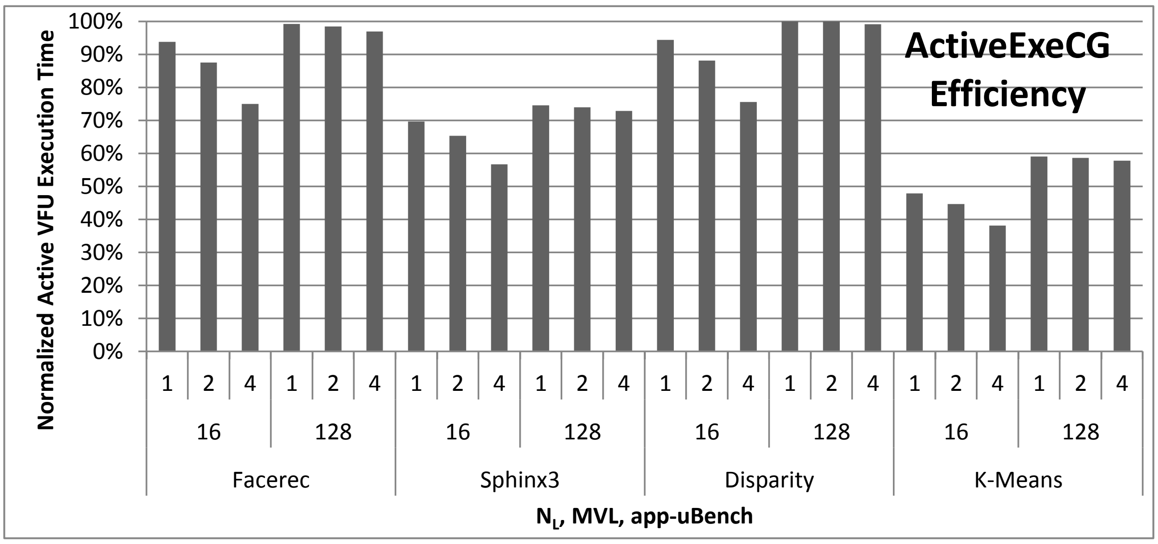


Figure 6.6: *ActiveExeCG* efficiency.

Figure 6.4 and Table 6.10 reveals that *AllCG* efficiency is very high, i.e. clock-gating is often used almost 100% of total execution time. This is a consequence of the fact that proposed clock-gating techniques are used during both idle and active VFMA execution. Due to this very high *AllCG* efficiency, the power savings are also fairly high. We observe that power savings are available for practically all the combinations of *app-uBenchs* and vector parameters. The most savings are available for computer vision *app-uBenchs* (Facerec and Disparity) and are between 60% and 80%. The only case when the savings are not available is for K-Means, MV_L=16 and n_L=4. There are two reasons for that: (1) clock-gating efficiency (percentage of execution time that any clock-gating technique is used) is not high and (2) with n_L=4 and MV_L=16 the effective vector length per lane is 4 which makes *ImplCG* less fruitful since it is used only 3 consecutive cycles per vector on each lane (which as a result has more switching activity in clock-gating and bypassing logic).

Figure 6.5 shows that ratio of active and idle VFMA execution varies across *app-uBenchs* and vector parameters, and explains the nature for each combination of parameters. There are situations when the VFMA is most of the time active and vice versa. However, we can notice there is a trend that vector processors with MV_L of 128 have its VFMA most of the time active (busy) as with longer vectors the effects of cache latency are diminished. *IdleCG* is used whenever VFMA is idle, thus, *IdleCG* efficiency inside these idle periods is 100%. When considering active VFMA execution, the efficiency (*ActiveExeCG*) varies across the *app-uBenchs* and vector parameters and is shown on Figure 6.6. As we can observe from the figure, a very high percentage of the time at least one of the active clock-gating techniques is used and depending on the benchmark it goes up to 100%. Table 6.10 shows that in all these cases the used techniques are some variants of *ImplCG*[1] and *InputCG*. Also, there are cases when these techniques overlap.

ActiveCG techniques can arbitrarily overlap and there are two kinds of overlaps. The first group of overlaps assumes cases when the techniques jointly produce higher savings than each technique separately. This happens when the techniques target different hardware. For example, when we have a zero addend inside a `FPADDV` instruction, $InputCG_{add0}$ gates the mantissa adder and the aligner, while *ImplCG* gates input register of *A* operand and the mantissa multiplier. The second group of overlaps happens when one technique gates just part of the hardware that another technique gates. In these cases, the savings are equal to the savings of the technique that have larger scope. For example, if the corresponding bit in VMR is '0' and current instruction is `FPFMAVVS`, the savings are going to be equal to savings achieved by *MaskCG* alone.

[1]Explained before, *ScalarCG* is integrated in *ImplCG*

6.8 Summary

In this research, we extensively identify, propose, and evaluate the most suitable clock-gating techniques for vector FMA (VFMA) considering peak performance, and focusing on the active operating mode. We propose techniques that are either (1) completely novel ideas to lower the power of VFMA using active clock-gating (e.g. vector instructions with scalar operand or vector masking) or (2) ideas that exist in some form in scalar architectures and that we extend to achieve more savings by taking advantage of vector processing characteristics. We find that each of the proposed optimizations achieves power reductions while maintaining the performance. As a consequence of this fact, sometimes an area increase is observed.

An in-depth evaluation is performed, and each of the techniques is evaluated separately as well as combined with other techniques. For this evaluation, both synthetic and real application-based benchmarks are employed. We considered a variety of benchmarks with different behavio to assure a fair evaluation and general conclusions. In the case of active 4-stage VFMA with 2 lanes actively operating at the peak performance, power savings are up to 52% are available when using a single technique. Regarding the vector instruction-dependent techniques that we propose, we observe savings for all floating-point vector instructions. Testing all the techniques together and using real application benchmarks (especially computer vision ones) reveals fairly high power reductions that go up to 80%. Clock-gating efficiency (percentage of time that some of the proposed techniques are used) is quite high, often close to 100%. When considering the efficiency of only active clock-gating techniques, this number is usually between 70% and 100%. Additionally, we notice the trend that savings for the proposed techniques rise with the number of pipeline stages.

We performed this research in a fully parameterizable, scalable and automated manner using simulators and tools at many levels. Although targeting floating-point FMA, as the major consumer among all functional units, similar low power techniques as well as the framework could be re-tailored for other vector functional units as well. We would also like to stress that the combination of Chisel-based generators and state-of-the-art synthesis and PnR tools is a powerful tool for flexible hardware generation with Verilog-like QoR.

Table 6.10: Power savings and clock-gating statistics. All the results are expressed in percentages.

app-uBench	MVL	nL	S	AllCG	Active Exe	ActiveCG	ActiveExeCG	Idle Exe, IdleCG	ScalarCG/ ImplCG	InputCG	InputCG$_{mul0}$	Input$_{add0}$
Facerec	16	1	70.32	98.96	16.68	15.63	93.74	83.32	15.63	7.50	0.00	7.50
		2	73.99	98.83	9.36	8.19	87.49	90.64	8.19	4.21	0.00	4.21
		4	76.48	98.77	4.91	3.68	74.99	95.09	3.68	2.21	0.00	2.21
	128	1	59.84	99.65	44.33	43.98	99.21	55.67	43.98	19.90	0.00	19.90
		2	65.50	99.55	28.81	28.35	98.43	71.19	28.35	12.93	0.00	12.93
		4	71.04	99.47	16.83	16.30	96.87	83.17	16.30	7.55	0.00	7.55
Sphinx3	16	1	37.54	84.96	49.48	34.43	69.59	50.52	32.06	17.67	2.37	15.29
		2	38.61	85.05	43.04	28.10	65.27	56.96	26.03	15.37	2.07	13.30
		4	46.13	87.83	28.07	15.90	56.63	71.93	14.55	10.02	1.35	8.68
	128	1	30.55	82.25	69.67	51.92	74.53	30.33	48.31	25.27	3.61	21.66
		2	22.09	76.98	88.49	65.47	73.98	11.51	60.89	32.10	4.58	27.52
		4	23.43	77.70	82.26	59.96	72.89	17.74	55.70	29.84	4.26	25.58
Disparity	16	1	75.55	97.85	38.41	36.26	94.41	61.59	36.26	25.90	0.00	25.90
		2	74.20	97.10	24.38	21.49	88.11	75.62	21.49	16.44	0.00	16.44
		4	73.17	96.69	13.51	10.20	75.52	86.49	10.20	9.11	0.00	9.11
	128	1	79.69	100.00	81.38	81.38	100.00	18.62	82.60	60.81	0.00	60.81
		2	78.90	100.00	76.24	76.24	100.00	23.76	76.77	56.97	0.00	56.97
		4	77.71	99.45	61.60	61.05	99.10	38.40	61.05	46.03	0.00	46.03
K-Means	16	1	19.56	60.22	76.26	36.49	47.84	23.74	35.87	0.61	0.00	0.61
		2	4.61	47.97	94.05	42.02	44.68	5.95	41.27	0.76	0.00	0.76
		4	-11.24	45.00	88.93	33.93	38.15	11.07	33.21	0.71	0.00	0.71
	128	1	24.62	60.53	96.44	56.97	59.07	3.56	50.46	6.51	0.00	6.51
		2	22.91	58.65	100.00	58.65	58.65	0.00	51.90	6.75	0.00	6.75
		4	20.77	57.81	100.00	57.81	57.81	0.00	51.05	6.75	0.00	6.75

7 Conclusions

In this work we have achieved the following main goals:

- Optimizing power and energy efficiency of VFU by applying various clock-gating techniques and by optimal structure selection through extensive design space exploration.

- A methodology that enables achieving the first goal by using fully parameterizable multi-level exploration frameworks that consist of various tools, simulators and generators (including several that we developed) at both circuit and architecture levels.

The following paragraphs summarize and conclude the achievements of this thesis.

We performed extensive design-space exploration of VA and VMU to find the optimal structure from energy-efficient and low power perspective when considering various types of vector processors. The exploration consists of both architectural- and circuit-level parameters: MV_L, number of vector lanes, benchmark, frequency, adder/multiplier family, and clock-gating. Among other findings we concluded that the vector multi-lane method is an energy- and thermal-efficient way to achieve speed-up and it outperforms frequency scaling in that sense. We analyzed the importance of considering the correlation between vector elements and VFU usage when using multiple vector lanes. Additionally, we discovered that 1-stage *bk* and 2-stage *wl* multipliers are often good matches for mobile vector processors.

We comprehensively identify, propose, and evaluate the most suitable clock-gating techniques for VFMA. We went a step further into power reductions using classic clock-gating, we looked at unexplored opportunities for clock-gating application to vector processors, especially when considering active operating mode and peak performance. VFMA is large enough so that significant power savings that we report diminish the overheads of additional logic insertion. The savings have a rising trend when increasing the number of pipeline stages. Applying one of the proposed techniques (*MaskCG*) in isolation brings down power savings of up to 52%. When combined all together, the techniques achieve reductions of up to 80%. Large savings are the result of high clock-gating efficiency: the percentage of time that clock-gating technique(s) is used. It is quite high, often close to 100%. When considering the efficiency of only active clock-gating techniques, this number is usually between 70% and 100%. Generally, we observe that computer vision applications (Facerec and Disparity) are the ones that take the most benefit of the proposed techniques. Additionally, the lessons learned from above mentioned explorations helped us to further optimize VFMA through optimal mantissa arithmetic selection and understanding the effects of vector architecture concepts (e.g. number of vector lanes or maximal vector length) on arithmetic units.

To perform the complete mentioned research we propose a novel methodology that is based on multi-level, fully parameterizable and automated frameworks. We developed frameworks that consists of various tools and generators (some of which we developed) and parameters at both circuit and architectural levels. Joint circuit-architectural research helps to understand how architectural level parameters (e.g. vector length) affect the circuit-level metrics (e.g. VMU power dissipation) and how circuit level parameters (e.g. multipliers clock cycle) impact the execution time of a microbenchmark. Generators that we use enable design space exploration through sweeping the parameters of the design. We use two kinds benchmarking, synthetic and "real world" application-based ones. Synthetic ones are necessary when stressing and isolating some functionalities and features of the observed system. This approach gives an opportunity to precisely understand underlying mechanisms. However, for an overall evaluation and verification of the results, and getting more general conclusions, application-based benchmarking is a must. Two key advantages are: (1) knowledge of how architectural and circuit parameters affects execution time, (2) knowledge of how architectural level parameters, such as vector multi-lane, affect power dissipation and how it changes with some circuit-level parameters (e.g. clock period). In this thesis, we used applications from SPEC, SDVB and STAMP benchmarks.

Additionally, we developed two different estimation flows: PAS and PnR. We performed a comparative analysis of both of them in terms of power, timing, area, and completion time using a design space exploration as a case study. Among other experiments we observed the flows' estimations accuracy change with input switching activity factor. We showed in which cases the results are reliable and in which cases they are not. In all the experiments presented in this thesis the technology that we use is a 40nm low power one (TSMC40LP).

This thesis could be important not only for vector processor designers but for a wider microprocessor design community, especially in this era when the power envelope is stricter than ever, and the demand for performance has not waned.

The following three directions seem very promising for future work that builds from the research of this thesis:

- Adapt the proposed optimizations and methodologies for low power FP and reduced precision FP computation in artificial neural networks (ANN). In the training part, there is abundant FP FMA computation [47, 54, 63, 70, 22] and it is usually done in server processors. We expect our clock-gating techniques that lower power without performance loss to be quite effective for these ANN computations. However, in the inference part there are reduced precision FP FMA operations [55]. Thus, a combination of our integer and FMA arithmetic optimizations has potential for success. As the inference part is often done in battery operated devices, here the optimizations would focus on the energy consumption. In both training and inference parts, abundant DLP is present.

- Application of similar methodology and ideas to other architectures for exploiting DLP such as graphics processing unit (GPU). GPUs are prominent in the last decade and have increasing popularity, not only for graphics and gaming, but for accelerating workloads from many domains.. Additionally, due to abundant computation (mostly FP), they are known as significant power consumers. Vector processors and GPUs, although both targeting efficient DLP exploitation, work in, to some extent, different way. Therefore, the proposed ideas would sometimes require an adaptation to the GPU paradigm. For example, the vector masking technique would need to adapt to GPU predication, while it still can exploit multi-lane slack in the same way.

- Extending the research to the front end of the vector pipeline, memory unit, and the register file. A large vector register file could be implemented in 3D DRAM and connected to VFUs via through-silicon via (TSV). In that way, we would be able to significantly increase the vector register file while keeping its latency small.

8 Publications on the topic

8.1 Publications from the thesis:

- Ivan Ratković, Oscar Palomar, Milan Stanić, Osman Unsal, Adrian Cristal, and Mateo Valero, "On the Selection of Adder Unit in Energy Efficient Vector Processing", Proceedings of the 2013 The International Symposium on Quality Electronic Design (ISQED), March 2013, Santa Clara, USA. [87]

- Ivan Ratković, Oscar Palomar, Milan Stanić, Osman Unsal, Adrian Cristal, and Mateo Valero, "Physically vs. Physically-Aware Estimation Flow: Case Study of Design Space Exploration of Adders", In Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2014, Tampa, USA. [88]

- Ivan Ratković, Nikola Bežanić, Osman Unsal, Adrian Cristal, and Veljko Milutinović, "An Overview of Architecture Level Power- and Energy-Efficient Design Techniques", Elsevier Advances in Computers, 2015. [89]

- Ivan Ratković, Oscar Palomar, Milan Stanić, Milovan Duric, Djordje Pešić, Osman Unsal, Adrian Cristal, and Mateo Valero, "Joint Circuit-System Design Space Exploration of Multiplier Unit Structure for Energy-Efficient Vector Processors", In Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI), July 2015, Montpelier, France. [90]

- Ivan Ratković, Oscar Palomar, Milan Stanić, Osman Unsal, Adrian Cristal, and Mateo Valero, "A Fully Parameterizable Low Power Design of Vector

Fused Multiply-Add Using Active Clock-Gating Techniques", 2016 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), August 2016, San Francisco, USA.

8.2 Related publications not included in the thesis:

- Milan Stanić, Oscar Palomar, Ivan Ratković, Milovan Duric, Osman Unsal, Adrian Cristal, and Mateo Valero, "VALib and SimpleVector: Tools for Rapid Initial Research on Vector Architectures", ACM International Conference on Computing Frontiers, May 2014, Cagliari, Italy.

- Milan Stanić, Oscar Palomar, Ivan Ratković, Milovan Duric, Osman Unsal, Adrian Cristal, and Mateo Valero, "Evaluation of Vectorization Potential of Graph500 on Intel Xeon Phi", International Conference on High Performance Computing & Simulation (HPSC), July 2014, Bologna, Italy.

- Milovan Duric, Milan Stanić, Ivan Ratković, Oscar Palomar, Osman Unsal, Adrian Cristal, and Mateo Valero, "Imposing Coarse-Grain Reconfiguration to General Purpose Processors", International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), July 2015, Samos, Greece.

- Milovan Duric, Milan Stanić, Ivan Ratković, Oscar Palomar, Osman Unsal, Adrian Cristal, and Mateo Valero, "Dynamic Specialization of Mobile Cores for Data-Parallel Applications", In submission at International Journal of Parallel Programing, Springer

- Milan Stanić, Oscar Palomar, Timothy Hayes, Ivan Ratković, Osman Unsal, Adrian Cristal, and Mateo Valero, "Towards Low-Power Embedded Vector Processor", ACM International Conference on Computing Frontiers, May 2016, Como, Italy.

- Milan Stanić, Oscar Palomar, Timothy Hayes, Ivan Ratković, Osman Unsal, Adrian Cristal, and Mateo Valero, "POSTER: An Integrated Vector-Scalar Design on an In-order ARM Core”, International Conference on Parallel Architectures and Compilation, September 2016, Haifa, Israel.

References

[1] Abdollahi, A., Fallah, F. & Pedram, M. (2005). An effective power mode transition technique in mtcmos circuits. In *Proceedings of the 42nd annual Design Automation Conference*, 37–42. 14

[2] Abts, D., Bataineh, A., Scott, S., Faanes, G., Schwarzmeier, J., Lundberg, E., Johnson, T., Bye, M. & Schwoerer, G. (2007). The cray blackwidow: a highly scalable vector multiprocessor. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 17, ACM. 18, 20

[3] Agarwal, K., Nowka, K., Deogun, H. & Sylvester, D. (2006). Power gating with multiple sleep modes. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, 633–637. 13

[4] Alipour, S., Hidaji, B. & Pour, A.S. (2010). Circuit level, static power, and logic level power analyses. In *EIT*, 1–4. 45

[5] Arcas-Abella, O., Ndu, G., Sonmez, N., Ghasempour, M., Armejach, A., Navaridas, J., Song, W., Mawer, J., Cristal, A. & Luján, M. (2014). An empirical evaluation of high-level synthesis languages and tools for database acceleration. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, 1–8. 101, 103

[6] ARM (2016). `http://arm.com`. 2

[7] Asanović, K. (1998). Vector microprocessor. *PhD Thesis*, UC Berkeley. 2, 10

[8] Aslund, A., Gustafsson, O., OhIsson, H. & Wanhammar, L. (2004). Power analysis of high throughput pipelined carry-propagation adders. In *Norchip Conference, 2004. Proceedings*, 139–142, IEEE. 44

[9] AVX-512 instructions (2016). https://software.intel.com/en-us/blogs/2013/avx-512-instructions/. 9

[10] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J. & Asanović, K. (2012). Chisel: constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, 1216–1225. 88, 100

[11] Balakrishnan, S. & Sohi, G.S. (2003). Exploiting value locality in physical register files. In *Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on*, 265–276, IEEE. 95

[12] Baran, D., Aktan, M., Karimiyan, H. & Oklobdzija, V. (2009). Exploration of switching activity behavior of addition algorithms. In *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*, 523–526. 45

[13] Baran, D., Aktan, M. & Oklobdzija, V.G. (2010). Energy efficient implementation of parallel cmos multipliers with improved compressors. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, 147–152, ACM. 71

[14] Behrooz, P. (2000). Computer arithmetic: Algorithms and hardware designs. *Oxford University Press*. 31, 72, 88

[15] Berkeley Hardware Floating-Point Units (2016). https://github.com/ucb-bar/berkeley-hardfloat/. 95, 101

[16] Borkar, S. & Chien, A.A. (2011). The future of microprocessors. *Commun. ACM*, **54**, 67–77. 2

[17] Brent, R. & Kung, H. (1982). A regular layout for parallel adders. *IEEE Trans.Comput.*, **31**, 260–264. 35

[18] Cadence Design Systems (2016). `http://www.cadence.com/`. 28

[19] Cadence Design Systems (2016). Encounter library characterizer datasheet. 28

[20] Callaway, T.K. & Swartzlander Jr, E.E. (1997). Power-delay characteristics of cmos multipliers. In *Computer Arithmetic, 1997. Proceedings., 13th IEEE Symposium on*, 26–32. 83

[21] Carlson, T.E., Heirman, W. & Eeckhout, L. (2013). Sampled simulation of multi-threaded applications. In *ISPASS'13*, 2–12. 37

[22] Chung, I.H., Sainath, T.N., Ramabhadran, B., Pichen, M., Gunnels, J., Austel, V., Chauhari, U. & Kingsbury, B. (2014). Parallel deep neural network training for big data on blue gene/q. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, 745–753. 115

[23] Committee, I.S. *et al.* (2008). 754-2008 ieee standard for floating-point arithmetic. *IEEE Computer Society Std*, **2008**. 101

[24] Cortex-A32 Processor (2016). `https://www.arm.com/products/processors/cortex-a/cortex-a32.php`. 18, 20

[25] Cortex-A35 Processor (2016). `https://www.arm.com/products/processors/cortex-a/cortex-a35.php`. 18, 20

[26] Cortex-A53 Processor (2016). `https://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php`. 18, 20

[27] Cortex-A7 Processor (2016). `https://www.arm.com/products/processors/cortex-a/cortex-a7.php`. 18, 20

[28] Cortex-A8 Processor (2016). `https://www.arm.com/products/processors/cortex-a/cortex-a8.php`. 18, 20

[29] Dougherty, W.E. & Thomas, D.E. (2000). Unifying behavioral synthesis and physical design. In *Proceedings of the 37th Annual Design Automation Conference*, 756–761. 45

[30] Ekman, M. & Stenstrom, P. (2005). A robust main-memory compression scheme. In *ACM SIGARCH Computer Architecture News*, vol. 33, 74–85. 95

[31] Encounter Digital Implementation System (2016). `http://www.cadence.com/products/di/edi_system/pages/default.aspx/`. 27

[32] Encounter RTL Compiler (2016). `http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx/`. 12, 27, 50

[33] Ercegovac, M. & Lang, T. (2003). *Digital Arithmetic*. Morgan Kaufmann. 31, 72, 88

[34] Erdogan, A., Zwyssig, E. & Arslan, T. (2004). Architectural trade-offs in the design of low power fir filtering cores. In *Circuits, Devices and Systems, IEE Proceedings-*, vol. 151, 10–17. 83

[35] Esmaeilzadeh, H., Blem, E., Amant, R.S., Sankaralingam, K. & Burger, D. (2011). Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, 365–376. 2

[36] Espasa, R. *et al.* (2002). Tarantula: a vector extension to the Alpha architecture. In *ISCA 29*, 281–292. 5, 18, 20

[37] Euro Server (2016). `http:www.euroserver-project.eu/`. 81

[38] Favalli, M. & Benini, L. (1995). Analysis of glitch power dissipation in cmos ics. In *Proceedings of the 1995 international symposium on Low power design*, 123–128. 26

[39] Flynn, D., Aitken, R., Gibbons, A. & Shi, K. (2007). *Low Power Methodology Manual: For System-on-Chip Design*. Springer Science & Business Media. 13

[40] Flynn, M.J. (1966). Very high-speed computing systems. *Proceedings of the IEEE*, **54**, 1901–1909. 10

[41] Gailhard, S., Julien, N., Diguet, J.P. & Martin, E. (1998). How to transform an architectural synthesis tool for low power vlsi designs. In *VLSI, 1998. Proceedings of the 8th Great Lakes Symposium on*, 426–431, IEEE. 83

[42] Galal, S., Shacham, O., Brunhaver, J., Pu, J., Vassiliev, A. & Horowitz, M. (2013). Fpu generator for design space exploration. In *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, 25–34. 88

[43] Gandhi, K.R. & Mahapatra, N.R. (2003). A study of hardware techniques that dynamically exploit frequent operands to reduce power consumption in integer function units. In *Computer Design, 2003. Proceedings. 21st International Conference on*, 426–428. 88

[44] Ghosh, S. & Roy, K. (2008). Exploring high-speed low-power hybrid arithmetic units at scaled supply and adaptive clock-stretching. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, ASP-DAC '08, 635–640. 62

[45] Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, **23**, 5–48. 88, 90, 107

[46] Gonzalez, R. & Horowitz, M. (1996). Energy dissipation in general purpose microprocessors. *IEEE J. Solid-State Circuits*, **31**, 1277–1284. 26

[47] Goodfellow, I., Bengio, Y. & Courville, A. (2016). Deep learning, book in preparation for MIT Press. 115

[48] Gu, Z., Wang, J., Dick, R.P. & Zhou, H. (2007). Unified incremental physical-level and high-level synthesis. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, **26**, 1576. 45

[49] Hennessy, J., Patterson, D. & Asanović, K. (2011). *Computer Architecture, Appendix G*. MK. 8, 20, 28

[50] Henning, J.L. (2006). Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, **34**, 1–17. 37, 49, 94, 100

[51] Heo, S., Barr, K. & Asanović, K. (2003). Reducing power density through activity migration. In *Low Power Electronics and Design, 2003. ISLPED'03. Proceedings of the 2003 International Symposium on*, 217–222. 5

[52] Heo, S.W. *et al.* (2003). Study of optimized adder selection. In *ASIC, 2003. Proceedings. 5th International Conference on*, vol. 2, 1265 – 1268. 63

[53] Hokenek, E., Montoye, R.K. & Cook, P.W. (1990). Second-generation risc floating point with multiply-add fused. *Solid-State Circuits, IEEE Journal of*, **25**, 1207–1213. 90

[54] Holley, L.H. & Karplus, M. (1989). Protein secondary structure prediction with a neural network. *Proceedings of the National Academy of Sciences*, **86**, 152–156. 115

[55] How to Quantize Neural Networks with TensorFlow (2016). https://petewarden.com/2016/05/03/how-to-quantize-neural-networks-with-tensorflow/. 115

[56] Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H. & Bose, P. (2004). Microarchitectural techniques for power gating of execution units. In *ISLPED '04*, 32–37. 13, 14

[57] Huang, Z. & Ercegovac, M.D. (2000). Effect of wire delay on the design of prefix adders in deep-submicron technology. In *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, vol. 2, 1713–1717, IEEE. 7

[58] Incisive Enterprise Simulator (2016). http://www.cadence.com/products/fv/enterprise_simulator/pages/default.aspx/. 51

[59] Intel Bonnell Microarchitecture (2016). https://en.wikipedia.org/wiki/Bonnell_%28microarchitecture%29. 18, 20

[60] Intel Silvermont Microarchitecture (2016). https://en.wikipedia.org/wiki/Silvermont. 20

[61] Jacobson, H., Bose, P., Hu, Z., Buyuktosunoglu, A., Zyuban, V., Eickemeyer, R., Eisen, L., Griswell, J., Logan, D., Sinharoy, B. *et al.* (2005). Stretching the limits of clock-gating efficiency in server-class processors. In *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, 238–242. 5

[62] Jiang, H., Marek-Sadowska, M. & Nassif, S.R. (2006). Benefits and costs of power-gating technique. In *2005 International Conference on Computer Design*, 559–566. 14

[63] Kim, J., Hopfield, J. & Winfree, E. (2004). Neural network computation by in vitro transcriptional circuits. In *Advances in neural information processing systems*, 681–688. 115

[64] Kogge, P. & Stone, H. (1973). A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans.Comput.*, **22**, 783–791. 35

[65] Lee, Y., Waterman, A., Avizienis, R., Cook, H., Sun, C., Stojanovic, V. & Asanovic, K. (2014). A 45nm 1.3 ghz 16.7 double-precision gflops/w risc-v processor with vector accelerators. In *European Solid State Circuits Conference (ESSCIRC), ESSCIRC 2014-40th*, 199–202. 2, 5, 18, 20, 81

[66] Lee, Y. *et al.* (2011). Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators. In *Proceedings of the 38th annual international symposium on Computer architecture*, 129–140. 2

[67] Lemuet, C. *et al.* (2006). The potential energy efficiency of vector acceleration. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, 1. 2, 3

[68] Li, H., Bhunia, S., Chen, Y., Vijaykumar, T.N. & Roy, K. (2003). Deterministic clock gating for microprocessor power reduction. In *HPCA '03*, 113–. 96

[69] Liu, Y. & Zhang, T. (2007). On the selection of arithmetic unit structure in voltage overscaled soft digital signal processing. In *ISLPED '07*, 250–255. 63

[70] Luo, Z., Liu, H. & Wu, X. (2005). Artificial neural network computation on graphic process unit. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 1, 622–626. 115

[71] Martin, A.J., Nystroem, M. & Penzes, P. (2001). Et2: A metric for time and energy efficiency of computation. *Tech. Rep. CaltechCSTR:2001.007, Caltech Computer Science*. 27

[72] Minh, C.C., Chung, J., Kozyrakis, C. & Olukotun, K. (2008). Stamp: Stanford transactional applications for multi-processing. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, 35–46, IEEE. 15

[73] Mohyuddin, N., Patel, K. & Pedram, M. (2009). Deterministic clock gating to eliminate wasteful activity due to wrong-path instructions in out-of-order superscalar processors. In *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, 166–172. 96

[74] Momose, S. (2015). Nec vector supercomputer: Its present and future. In *Sustained Simulation Performance*, 95–105, Springer. 5

[75] Monchiero, M. *et al.* (2006). Design space exploration for multicore architectures: a power/performance/thermal view. In *Proceedings of the 20th annual international conference on Supercomputing*, 177–186. 52, 60, 61

[76] Moore, G.E. (1965). Cramming more components onto integrated circuits. *Electronics*, **38**, 114–117. 2

[77] Moore, G.E. (2003). No exponential is forever: but" forever" can be delayed![semiconductor industry]. In *Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC. 2003 IEEE International*, 20–23. 2

[78] Nikolic, B. (2015). Simpler, more efficient design. In *European Solid-State Circuits Conference (ESSCIRC), ESSCIRC 2015-41st*, 20–25, IEEE. 2, 7, 88

[79] Oklobdzija, V. *et al.* (2005). Comparison of high-performance vlsi adders in the energy-delay space. *IEEE Trans. Very Large Scale Integr. Syst.*, **13**, 754–758. 62

[80] Patil, D. *et al.* (2007). Robust energy-efficient adder topologies. In *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, 16–28. 31, 62

[81] Plakaris, G. (2003). *Power efficient arithmetric circuits for application specific processors*. Ph.D. thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark. 12

[82] Preiss, J., Boersma, M. & Mueller, S.M. (2009). Advanced clockgating schemes for fused-multiply-add-type floating-point units. In *Computer Arithmetic, 2009. ARITH 2009. 19th IEEE Symposium on*, 48–56. 93, 96

[83] Qualcomm Krait (2016). https://en.wikipedia.org/wiki/Krait_(CPU). 20

[84] Quintana, F. *et al.* (1999). Adding a vector unit to a superscalar processor. In *Proceedings of the 13th international conference on Supercomputing*, 1–10. 20

[85] Rabaey, J. (2009). *Low power design essentials*. Springer Science & Business Media. 12

[86] Rabe, D. & Nebel, W. (1996). Short circuit power consumption of glitches. In *Proceedings of the 1996 international symposium on Low power electronics and design*, 125–128. 26

[87] Ratković, I., Palomar, O., Stanić, M., Unsal, O.S., Cristal, A. & Valero, M. (2013). On the selection of adder unit in energy efficient vector processing. In *Quality Electronic Design (ISQED), 2013 14th International Symposium on*, 143–150, IEEE. 117

[88] Ratković, I., Palomar, O., Stanic, M., Unsal, O., Cristal, A. & Valero, M. (2014). Physical vs. physically-aware estimation flow: Case study of design space exploration of adders. In *VLSI (ISVLSI), 2014 IEEE Computer Society Annual Symposium on*, 118–123, IEEE. 117

[89] Ratković, I., Bežanić, N., Ünsal, O.S., Cristal, A. & Milutinović, V. (2015). Chapter one-an overview of architecture-level power-and energy-efficient design techniques. *Advances in Computers*, **98**, 1–57. 117

[90] Ratković, I., Palomar, O., Stanić, M., Duric, M., Pešić, D., Unsal, O., Cristal, A. & Valero, M. (2015). Joint circuit-system design space exploration of multiplier unit structure for energy-efficient vector processors. In *VLSI (ISVLSI), 2015 IEEE Computer Society Annual Symposium on*, 19–26, IEEE. 117

[91] Reference Manual for ARM Architecture - ARMv7-A (2016). `http://arm.com/`. 5

[92] Rethinagiri, S.K., Palomar, O., Sobe, A., Yalcin, G., Knauth, T., Gil, R.T., Prieto, P., Schneegass, M., Cristal, A., Unsal, O. *et al.* (2015). Paradime: Parallel distributed infrastructure for minimization of energy for data centers. *Microprocessors and Microsystems*, **39**, 1174–1189. 100

[93] Russell, R.M. (1978). The cray-1 computer system. *Communications of the ACM*, **21**, 63–72. 102

[94] Sathanur, A., Pullini, A., Benini, L., Macii, A., Macii, E. & Poncino, M. (2007). Timing-driven row-based power gating. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, 104–109. 13

[95] Shacham, O., Azizi, O., Wachs, M., Qadeer, W., Asgar, Z., Kelley, K., Stevenson, J.P., Richardson, S., Horowitz, M., Lee, B.W. *et al.* (2010). Rethinking digital design: Why design must change. *Micro, IEEE*, **30**, 9–24. 7, 88

[96] Shin, Y., Seomun, J., Choi, K.M. & Sakurai, T. (2010). Power gating: Circuits, design methodologies, and best practice for standard-cell vlsi designs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, **15**, 28. 14

[97] Spectre Circuit Simulator (2016). `http://www.cadence.com/products/cic/spectre_circuit/pages/default.aspx/`. 28

[98] Srinivasan, V. *et al.* (2002). Optimizing pipelines for power and performance. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 35, 333–344. 56, 78

[99] Stan, M.R., Skadron, K., Barcella, M., Huang, W., Sankaranarayanan, K. & Velusamy, S. (2003). Hotspot: A dynamic compact thermal model at the processor-architecture level. *Microelectronics Journal*, **34**, 1153–1165. 3

[100] Standard Delay Format (2016). http://www.eda.org/sdf/. 51

[101] Standard Performance Evaluation Corporation (2016). https://www.spec.org/benchmarks.html/. 15

[102] Stanic, M., Palomar, O., Ratković, I., Duric, M., Unsal, O. & Cristal, A. (2014). Valib and simplevector: tools for rapid initial research on vector architectures. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, 7. 5, 17

[103] Stanic, M., Palomar, O., Hayes, T., Ratković, I., Unsal, O. & Cristal, A. (2016). Towards low-power embedded vector processor. In *Proceedings of the ACM International Conference on Computing Frontiers*, 339–342. 4

[104] Stefan-Boltzmann law (2016). http://hyperphysics.phy-astr.gsu.edu/. 27, 52

[105] Sun, S. & Sechen, C. (2007). Post-layout comparison of high performance 64b static adders in energy-delay space. In *Computer Design, 2007. ICCD 2007. 25th International Conference on*, 401 –408. 44, 62

[106] Synopsys Design Compiler (2016). http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx/. 12

[107] Tang, A., Yang, Y., Lee, C.Y. & Jha, N.K. (2015). Mcpat-pvt: Delay and power modeling framework for finfet processor architectures under pvt variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **23**, 1616–1627. 4

[108] ThunderX ARM Processors (2016). http://www.cavium.com/. 5, 61

[109] Townsend, W.J., Swartzlander Jr, E.E. & Abraham, J.A. (2003). A comparison of dadda and wallace multiplier delays. In *Optical Science and Technology, SPIE's 48th Annual Meeting*, 552–560, International Society for Optics and Photonics. 83

[110] Value Change Dump (2016). `https://en.wikipedia.org/wiki/Value_change_dump/`. 51

[111] Veendrick, H. (1984). Short-circuit dissipation of static cmos circuitry and its impact on the design of buffer circuits. *Solid-State Circuits, IEEE Journal of*, **19**, 468 – 473. 26

[112] Venkata, S.K., Ahn, I., Jeon, D., Gupta, A., Louie, C., Garcia, S., Belongie, S. & Taylor, M.B. (2009). Sd-vbs: The san diego vision benchmark suite. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, 55–64. 15, 67, 70

[113] Wallace, C.S. (1964). A suggestion for a fast multiplier. *Electronic Computers, IEEE Transactions on*, 14–17. 71

[114] Xanthopoulos, T. & Chandrakasan, A.P. (1999). A low-power idct macrocell for mpeg-2 mp@ ml exploiting data distribution properties for minimal activity. *Solid-State Circuits, IEEE Journal of*, **34**, 693–703. 95

[115] Xu, T., Li, P. & Yan, B. (2011). Decoupling for power gating: Sources of power noise and design strategies. In *Proceedings of the 48th Design Automation Conference*, 1002–1007. 14

[116] Zeydel, B., Baran, D. & Oklobdzija, V. (2010). Energy-efficient design methodologies: High-performance vlsi adders. *IEEE Solid-State Circuits*, **45**, 1220–1233. 44

[117] Zhou, C. *et al.* (2009). 64-bit prefix adders: Power-efficient topologies and design solutions. In *Custom Integrated Circuits Conference, 2009. CICC '09. IEEE*, 179 –182. 4, 44

[118] Zimmer, B., Lee, Y., Puggelli, A., Kwak, J., Jevtic, R., Keller, B., Bailey, S., Blagojevic, M., Chiu, P.F., Le, H.P. *et al.* (2015). A risc-v vector processor with tightly-integrated switched-capacitor dc-dc converters in 28nm fdsoi. In *VLSI Circuits (VLSI Circuits), 2015 Symposium on*, C316–C317, IEEE. 2, 5, 18, 20

[119] Zimmermann, R. (1997). Binary adder architectures for cell-based vlsi and their synthesis. *PhD Thesis*, ETH Zurich. 44

Abbreviations

EV_L effective vector length. 8, 9, 94

MV_L maximum vector length. 8, 9, 14, 18, 50, 52, 54, 55, 57–59, 61, 62, 64, 65, 70, 76–81, 85, 94, 97, 103, 106, 109, 113

AF adder family. 27, 30, 31, 38–44, 50, 65

MF multiplier family. 71, 72, 83

ar carry-save array. 71–73, 75, 77, 83–85

bk Brent-Kung. 30, 31, 35, 36, 38, 40–42, 44, 55–57, 62, 63, 65, 113

cla carry-lookahead. 30, 31, 33, 40–42, 44

cosa conditional-sum adder. 30, 31, 33, 34, 40, 41, 44, 56, 57

ks Kogge-Stone. 30, 31, 35, 40–43, 56, 57, 62, 65

rca ripple-carry adder. 30–32, 40, 41, 43–45, 55, 57, 62, 63, 65

wl Wallace. 71, 72, 75, 77, 83–85, 113

EDP Energy-Delay Product. 26

ALU arithmetic logic unit. 17, 18, 20, 21

Abbreviations

ANN artificial neural networks. 115

BHFPU Berkeley Hardware Floating-Point Units. 101

Chisel Constructing Hardware in Scala Embedded Language. 100, 101, 103, 105, 110

CPA final carry-propagate addition. 72, 73

CSA carry-save adder. 73, 74

DLP data-level parallelism. 2, 4, 8, 9, 115

EXEU execution unit. 4

FA full adder. 32, 72, 74

FMA fused multiply-add. 5, 8, 14, 18, 87, 88, 90, 91, 101, 110, 115

FP floating-point. 5, 88, 89, 91, 115

FU functional unit. 3, 5, 10, 12, 13, 18, 52, 100

GPU graphics processing unit. 115, 116

HA half adder. 32, 72

HDL hardware description language. 27, 30, 50, 68, 88, 100

IFU instruction fetch unit. 4

ISA instruction set architecture. 8, 18

LSU load-store unit. 4

MC integrated memory controller. 4

MFA modified full adder. 72, 73

MHA	modified half adder. 72
MMU	memory management unit. 4
PAS	physical layout aware synthesis. 15, 23, 24, 27–29, 31, 36–47, 49, 50, 115
PnR	place and route. 6, 15, 23, 24, 27–29, 36–47, 67, 68, 83, 88, 97, 101, 115
PPG	partial product generation. 72
PPR	partial product reduction. 72
QoR	quality of results. 29, 101, 110
RNU	renaming unit. 4
sdf	standard delay format. 51, 98
SIMD	single instruction multiple data. 9
STA	static timing analysis. 50, 69, 98
TSV	through-silicon via. 116
VA	vector adder unit. 14, 49, 50, 53, 54, 57–60, 62, 63, 65, 113
vcd	Value Change Dump. 51, 69, 98
VFMA	vector fused multiply-add unit. 8, 9, 18, 20, 87, 92, 94–98, 101–104, 106–110, 114
VFU	vector functional unit. 1, 3, 5, 8–10, 13, 14, 17, 18, 30, 113, 116
VLSI	very-large-scale integration. 5, 24
VMR	vector mask register. 9, 94, 95, 99, 109
VMU	vector multiplier unit. 14, 67–69, 78–85, 113, 114