TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Low power architectures for streaming applications

Citation for published version (APA):
He, Y. (2013). *Low power architectures for streaming applications*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR759534

DOI:
10.6100/IR759534

Document status and date:
Published: 01/01/2013

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 26. Aug. 2022

LOW POWER ARCHITECTURES FOR STREAMING APPLICATIONS

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 14 oktober 2013 om 16.00 uur

door

Yifan He

geboren te Hangzhou, China

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. H. Corporaal
en
prof.dr.ir. P.P. Jonker

Copromotor:
dr.ir. B. Mesman

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

He, Yifan

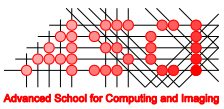
Low Power Architectures for Streaming Applications
/ by Yifan He. – Eindhoven : Technische Universiteit Eindhoven, 2013.
A catalogue record is available from the Eindhoven University of Technology Library
ISBN 978-90-386-3461-6
NUR 959
Subject headings: low power / processor architecture / streaming applications / embedded systems

LOW POWER ARCHITECTURES FOR STREAMING APPLICATIONS

Committee:

prof.dr. H. Corporaal (promotor, TU Eindhoven)
prof.dr.ir. P.P. Jonker (promotor, TU Eindhoven, TU Delft)
dr.ir. B. Mesman (copromotor, TU Eindhoven)
prof.dr. K. Bertels (TU Delft)
prof.dr. B.H.H. Juurlink (TU Berlin)
prof.dr. J. Pineda de Gyvez (TU Eindhoven, NXP Semiconductors)
prof.dr.ir. A.C.P.M. Backx (chairman, TU Eindhoven)

This work has been carried out as part of the EVA project PID-07121. The project is supported by the Ministry of Economic Affairs of the Netherlands.



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 290.

© Yifan He, 2013.

All rights are reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

Printing: Printservice Technische Universiteit Eindhoven

To my parents, my wife, and my daughter

ABSTRACT

LOW POWER ARCHITECTURES FOR STREAMING APPLICATIONS

Streaming applications are an important class in emerging embedded systems such as smart camera networks, unmanned vehicles, and industrial printing. On the one hand, these applications are usually very computationally intensive and have real-time constraints. On the other hand, embedded systems running these applications often have limited power sources like batteries or solar panels. Therefore, energy-awareness becomes increasingly important in architecture designs for these systems. To achieve high efficiency in such kind of systems, significant efforts at different hardware and software design levels are required. This thesis deals with part of the challenges in the design of energy-efficient architectures for streaming applications, and discusses the following four main topics.

In embedded processors, a substantial part of the energy is consumed by the register file (RF). As a first contribution of this thesis, *MOVE-Pro*, a new *transport triggered architecture* (TTA) based processor architecture is proposed to reduce energy consumption of the register file. With fine-grained control over datapath and optimization in different software/hardware levels, we achieve significant RF accesses reduction in MOVE-Pro: on average about 70% of RF accesses are eliminated, resulting in a dramatic reduction of RF energy. More importantly, with the proposed MOVE-Pro architecture, the RF energy saving is fully transferred to the total core energy saving. Compared to its RISC counterpart, a total core energy saving of up to 11.6% is achieved.

In application specific instruction set processor (ASIP) design, it is common to synthesize instruction sets that support patterns of operations in targeted applications to achieve better performance and energy efficiency. However, in an embedded generic processor with compact ISA, such instructions can lead to large overhead. As a second contribution of this thesis, an architecture that supports flexible operation pairs in processors with compact ISAs is proposed. It introduces a partially reconfigurable decoder and a software-controlled by-

pass network, allowing the processor to support operation pairs without increasing the instruction width or number of register file ports. Comprehensive and detailed experimental results demonstrate that the proposed architecture achieves an average of 26.0% reduction in dynamic cycle count and an average of 15.8% reduction in energy consumption compared to the reference processors.

In many embedded streaming applications, substantial amounts of data-level parallelism can be exploited. To deal with energy-efficiency in combination with high performance requirements, Xetal-Pro, a massively-parallel SIMD architecture, is proposed. The initial idea of combining massive parallelism and aggressive V_{dd} scaling is presented and discussed in detail. A hybrid memory system, which reduces the non-local memory traffic and enables further V_{dd} scaling, is also introduced. This work shows that it is possible to achieve 1 pJ/op core energy consumption for typical kernels of embedded streaming applications.

To meet the increasing demand for performance and efficiency in streaming applications, multi-processor system-on-chips (MPSoCs) are becoming a popular solution. As the fourth contribution of this thesis, an efficient and predictable *communication assist* (CA) for integrating generic IP cores into heterogeneous MPSoCs is proposed. The corresponding cycle-accurate *synchronous data flow* (SDF) model for the proposed communication assist is also presented. By integrating this SDF model into SDF analysis tools, worst-case system properties, such as throughput, latency, and buffer sizes can be conservatively analyzed at design time. As a case study, a vision processing pipeline of an industrial application, *Organic Light Emitting Diode* (OLED) screen printing, is mapped onto the proposed platform. This case study also demonstrates that the proposed design flow enables efficient integration of accelerator IPs into a heterogeneous MPSoC which targets streaming applications.

CONTENTS

CHAPTER 1

INTRODUCTION

There is an increasing demand for running applications with high performance requirements on embedded systems that have relatively limited resources. For example, a smart phone has to run high-definition video codecs, wireless signal processing, and 3D graphics processing. A smart camera may combine high resolution video sensing, low-level to high-level vision processing, and communication within a single embedded device. All these applications require an enormous amount of computation, and yet embedded system designers have to meet these requirements with a very small power budget. *Power efficiency* is thus becoming a dominant determinant in embedded system design, especially for those ones that run on restricted power sources like batteries or solar cells.

To achieve a highly power-efficient embedded system, many challenges at different design levels need to be overcome (Section 1.2). This thesis focuses on dealing with part of these challenges and contributes by providing solutions to the following four aspects, i.e., energy-efficient data movement in datapath (Chapter 2), flexible special instruction support (Chapter 3), ultra low-energy vector processing (Chapter 4), and efficient communication in heterogeneous MPSoCs (Chapter 5).

The remainder of this chapter is organized as follows. In Section 1.1, we take a closer look at the trends in embedded streaming applications. In Section 1.2, we look at the challenges in low-power architecture designs. Section 1.3 introduces design flows of streaming application systems. Section 1.4 clarifies the problems that this thesis mainly focuses on. Section 1.5 states the key contributions of the thesis. Finally, Section 1.6 gives a brief overview of the thesis.

1.1 Trends in Embedded Streaming Applications

An *embedded system* is an information processing system designed to perform one or a few dedicated functions often with real-time constraints. Embedded

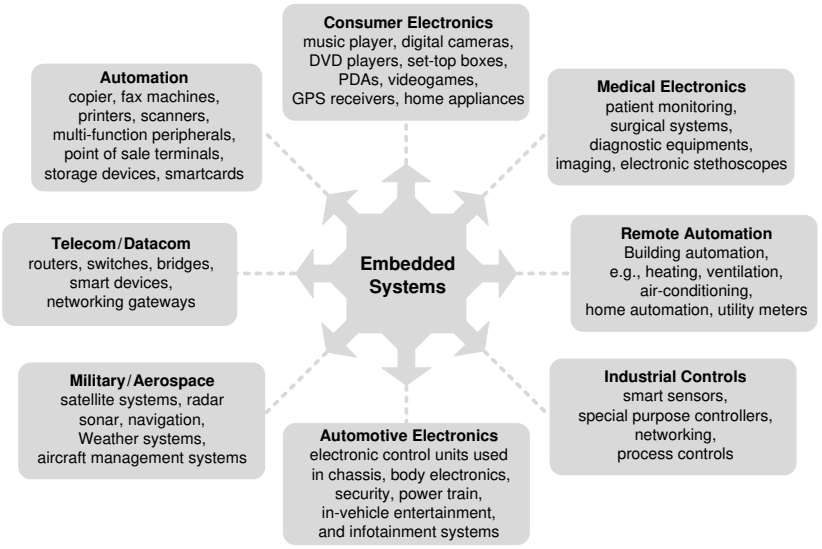


Figure 1.1 Application domains of embedded systems.

systems are widely used in our daily life, e.g., inside portable devices such as digital cameras and smart phones, or systems like televisions, printers, and electronic control unit (ECU) in automobiles. Figure 1.1 depicts the main application domains of embedded systems [5]. Compared to general computing devices, embedded systems perform a *limited* set of pre-defined functions and have *limited* field configuration capability. They are typically enclosed into a very compact packaging with *limited* power sources and cooling facilities. These features/constraints together mode embedded systems into the characteristics of domain specific, real time, low power, low cost, and small area. Among these characteristics, *low power* is usually a distinguishable feature and becoming one of the main challenges [6-8].

Many applications, such as image processing, computer vision, and multimedia applications, can be categorized as *streaming applications*, which periodically execute similar operations on a stream of data items (often called samples) [9]. Such applications are very common in embedded systems, e.g., 3G/4G [10, 11] wireless communication, H.263/H.264 [12, 13] video coding for mobile domain, MPEG-2/MPEG-4 [14, 15] for high resolution video streams, MP3/AAC [16] for digital audio compression and encoding, DAB/DAB+ [17] for digital radio coding, and various image processing applications such as image enhancement [18, 19], industrial inspection [20, 21], and 3D reconstruction [22, 23].

And their number grows very fast. For example, in the area of portable consumer systems, such as smart cameras and mobile phones, the number of applications doubles roughly every two years [24]. Many of these emerging applications fall into the category of streaming applications. The quick introduction of these new applications becomes an important driven for the development of new technologies [25].

A second trend in embedded streaming applications is the rapidly increasing computation intensity. This is caused by two factors: *i*) an increasing amount of computation must be applied on the data sets due to growing application complexity, e.g., more elaborate and complex coding schemes; *ii*) an increasing amount of data must be processed due to growing data sets, e.g., higher image resolution [26, 27]. As a result, the required processing power is expected to increase by an order of magnitude every five years [6]. Due to technology limitations and power/energy limitations, single processor systems can hardly keep up with the demand of emerging streaming applications [24]. A multi-processor system-on-chip (MPSoC) system seems to be the only solution to fulfill these requirements [6].

In most cases, embedded systems have very limited power sources, such as batteries, the capacity of which grows slowly. For example, in mobile devices, the energy density of Li-ion batteries only increased two to three times during the past fifteen years [28], which can hardly meet the power consumption requirement of modern embedded systems. And this gap is still enlarging. According to the recent research of Samsung's battery development team, the required power for mobile devices is predicted to increase about 20% per year, while the advancement in the energy density of batteries is expected to be only 10% per year [29]. This leads to the third trend: increasing emphasis on *low power*.

1.2 Challenges in Low-Power Architecture Designs

Modern mobile devices, such as smart phones, perform nearly 100 giga operations per second (GOPS) within a power budget of only 1 W [6]. In other words, no more than 10 pJ can be spent on the execution of one operation, which includes memory accesses, instruction decoding, data movements, and computation. As workload grows at a much faster rate than the improvement of battery capacity, the requirement is becoming even more demanding. We will soon enter into the 1 pJ/op era. Under such a critical constraint, many challenges

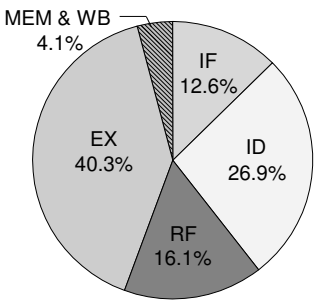


Figure 1.2 Energy breakdown of a classical 5-stage RISC processor (YUV2RGB).

need to be overcome to achieve an energy-efficient architecture for streaming applications.

Computation Overhead

In a typical reduced instruction set computing (RISC) processor, the energy consumed for executing an *add* instruction is 10× more than the energy spent in a 32-bit adder [30]. This basically means that most of the energy is not consumed by the computation we actually need, but "wasted" by the other overhead required to enable the computation. Among this overhead, the supplying of instructions and data to functional units dominates. This is the main reason why even low-power embedded processors consume significantly more energy than dedicated fixed-function hardware, which allows the communication of data between function units to be aggressively optimized [31]. The commonly used hierarchical memory subsystem can improve energy efficiency by exploiting locality. However, this is still not sufficient. Taking the register file (RF), which is usually at the top of a memory subsystem (i.e., most energy efficient), as an example, it alone can account for 16% of the energy consumed in an embedded processor [32, 33]. Our estimation also shows a similar result. Figure 1.2 presents the energy breakdown of a classical 5-stage RISC processor when executing a YUV to RGB color space conversion kernel [1]. This situation does

[1] The energy consumption is calculated as $P_{avg} \times T_{exe}$, where P_{avg} is the average power consumption during kernel execution while T_{exe} is the kernel execution time. These values are derived by post-synthesis simulation. TSMC 90nm low-power (LP) SV_t library is used here. The synthesis condition is set to typical case, 1.2V, and 25°C. Clock gating is also enabled. To improve the accuracy of power estimation, physical library (6 metal layers) is included. The interconnect information is estimated based on the physical library.

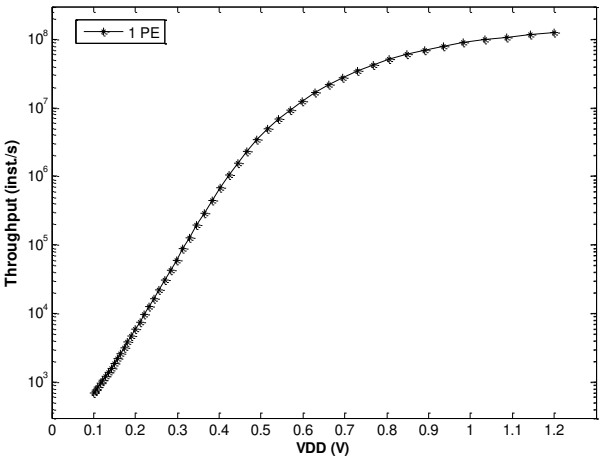


Figure 1.3 Impact of V_{dd} scaling on throughput of 1 processing element (PE) when executing a 5×5 non-separable filter kernel (refer to Chapter 4 for details).

not improve with advances in technology nodes as communication benefits less than computation from technology scaling. The interconnect-dominated register files and buses that deliver instructions and data to the function units will continuously increase the energy consumption portion [31].

Wide-Range Voltage Scaling

Voltage scaling is one of the most effective means to bring quadratic dynamic energy savings to standard-cell based logic, i.e., $E_{logic} \propto C_{load}V_{dd}^2$, where C_{load} is the loading capacitances including both gate and interconnection wire capacitances. However, the V_{dd} scaling range of commercial processors is normally limited to about 2/3 of nominal supply due to two fundamental problems at a low V_{dd}: *i)* severe throughput degradation (Figure 1.3); *ii)* high yield loss in the presence of process variations [34]. Compared to pure standard-cell based logic, wide-range V_{dd} scaling is even more difficult when applied to static random-access memories (SRAMs). First, the rapidly deteriorating read/hold static noise margin (SNM) of bit-cells causes severe reliability issues. A very small amount of injected noise can cause the bit-cell's state to flip [35]. Thus, commercial SRAMs with high density (6-T cells) strictly prohibit operating below 0.7~0.8 V. Second, SRAM's energy cannot scale quadratically with V_{dd}. This is because the bit-cells' swing voltage, which must exceed a minimum magnitude required by sense-amplifiers to perform correct decoding, cannot scale linearly with V_{dd}. Third, the speed of SRAMs degrades even faster with V_{dd} scaling, compared to that of

pure logic [36, 37]. This implies that SRAMs are more prone to be the system performance bottleneck when both SRAM and logic scale to the same low V_{dd}.

Variation

VLSI performance has increased by five orders of magnitude in the last three decades, made possibly by continued technology scaling [38]. This trend will continue, providing an integration capacity of billions of transistors. Advanced process technologies also lead to significant power reduction [24, 39]. However, a parallel trend is that process variation increases with each technology node, which becomes one of the main barriers to future scaling [40, 41]. Variation has many different causes, e.g., random dopant fluctuations [42], line-edge roughness [43], oxide thickness variations [44]. Transistor threshold voltage (V_{th}) is one of the parameters that are severely affected by variation, which causes both inter- and intra-die heterogeneity in transistor delay and power consumption [39]. A 30% variation in operating frequency and a five to ten times variation in leakage power can be seen in current technology nodes. And the variation in total power could be as much as 50% [38]. To compensate the impact of variation and to guarantee certain yield, more margins need to be reserved in circuit designs. As a consequence, both design cost and energy consumption are raised.

"Energy-Aware" Compiler

Improving energy efficiency is a joint effort of hardware and software. However, while hardware optimization has been the focus of low power design for many years, software approaches to optimize power are still relatively new. Most of the compiler techniques consider only delay and code size as their main performance metrics. With the growing demand for energy-aware software, there is an acute need for investigating low-power compilation techniques and their interaction and integration with performance-oriented compiler optimizations [45]. There are already some useful compilation techniques used in low-power embedded systems, such as giving priority to the lowest-energy-consuming (instead of the fastest) instruction sequences [46], or moving frequently used program codes or data blocks into more energy-efficient levels of the memory hierarchy [47]. However, to meet the rigorous requirement for low energy, a more "energy-aware" compiler, i.e., a compiler armed with more advanced energy-oriented optimization techniques and a more accurate energy model of the target machine, is needed for overall system energy optimization.

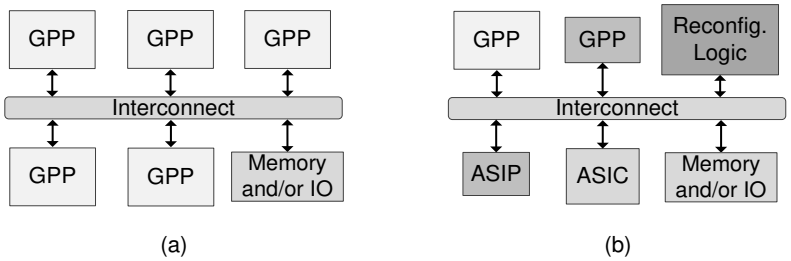


Figure 1.4 (a) a homogeneous MPSoC example; (b) a heterogeneous MPSoC example.

Heterogeneity

Since single core based systems alone can no longer keep up with the demand of emerging streaming applications, multi-processor system-on-chips (MPSoCs) are becoming a popular solution to fulfill the requirements. In general, an MPSoC can be categorized as one of the following two types. A *homogeneous MPSoC*, whose processing components are of the same type of programmable processor, can usually support a wide range of applications. Such a system provides high flexibility and is relatively easy to use. However, the overhead of supporting a wide range of applications in all cores may cause severe inefficiencies. Compared to a homogeneous MPSoC, a *heterogeneous MPSoC*, whose processing components can be general purpose processors (GPPs), application-specific instruction-set processors (ASIPs), dedicated application-specific integrated circuits (ASICs), or reconfigurable logic, can achieve much higher efficiency in both performance and power consumption by properly designing and configuring the system for the applications mapped onto it [48, 49]. Figure 1.4 shows examples of homogeneous and heterogeneous MPSoCs. Streaming applications are usually of high complexity and high diversity, so heterogeneous MPSoCs are commonly used when designing efficient systems for these applications [6]. However, heterogeneity also introduces extra complexity. Processing components of various types need to be designed and integrated. Modeling, mapping, scheduling, and debugging applications on a heterogeneous platform are also challenging tasks.

1.3 Design Flow of Streaming Application Systems

With the increasing complexity of embedded systems for streaming applications, it is hardly possible to evaluate system level trade-offs using back-of-the envelope calculations. Designers can neither handle all design aspects at the same time nor handle them by the same person. Therefore, design flows, which

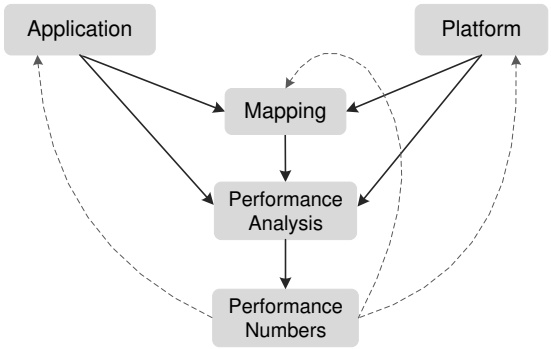


Figure 1.5 The Y-Chart.

can separate different design concerns, solve them efficiently with domain-specific knowledge, and then systematically integrate each component together with performance/cost evaluation, are required.

In this subsection, we first introduce the basic concepts of Y-chart and synchronous data-flow graph. Then we introduce MAMPS, an in-house design flow for mapping throughput constrained applications on an MPSoC.

Y-Chart

The Y-chart approach [50, 51] is a general paradigm used for designing embedded systems. Figure 1.5 illustrates the concept of the Y-chart, which involves the co-development of application(s), a platform, and the mapping of the application(s) onto the platform. To implement a system using this methodology, typically the designer studies the target applications with some initial calculations, and proposes an architecture. The performance of this architecture is then quantitatively evaluated and compared against alternative architectures. For such performance analysis, each application is mapped onto the architecture under investigation and the performance of each application-architecture combination is evaluated [52]. Subsequently, the resulting performance numbers may inspire the designer to improve the architecture, restructure the application(s) or modify the mapping of the application(s), which are shown as dotted lines in Figure 1.5. These feedback paths make the Y-chart an iterative design methodology that enables the co-evolution of hardware and software [53].

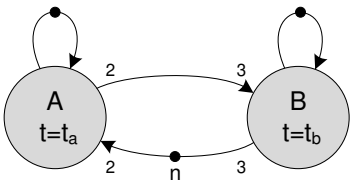


Figure 1.6 An SDF graph with two actors. Unmarked rates and buffer sizes are 1.

Synchronous Data-flow Graphs

In order to ensure that high performance or real-time requirement of an application can be met by the target platform, designers have to be able to model the application. In the absence of a good model, it is very difficult to know in advance whether the requirements can be met or not at all times, and extensive simulation and testing are needed [54].

The *Synchronous Data Flow* (SDF) graph is a very powerful model of computation (MoC) for analyzing streaming applications [55]. Both pipelined streaming and cyclic dependencies between tasks can be easily modeled with SDF graphs. There are many analysis algorithms for SDF that can be used to analyze at design time the throughput, latency, and buffer size requirements of applications modeled with an SDF graph [56-58].

An application modeled with an SDF graph consists of nodes called *actors* and edges, called *channels*, between these actors. Actors transfer data items called *tokens* to each other via channels. An example SDF graph containing a producer actor A and a consumer actor B is depicted in Figure 1.6. Actor A takes t_a time units to finish one firing (i.e., perform a computation) and it produces 2 tokens at the end of this firing. Actor B consumes 3 tokens in each firing and its execution time is equal to t_b time units. The channel from actor A to actor B models the communication channel. The channel from B to A is used to model the buffer between these actors. The amount of initial tokens n on this channel represents the buffer size of the channel from actor A to actor B. The number of concurrent firings of an actor, defined as its *auto-concurrency*, is constrained by the number of initial tokens on the actor's *self-edge*. In Figure 1.6, both actors have an auto-concurrency of 1. Hence, subsequent firings of actor A (or B) need to be serialized. This could for example be necessary when an actor firing produces state that is needed for a subsequent firing. Note that when an actor has no self-edge, it is possible to have multiple firings of an actor running in paral-

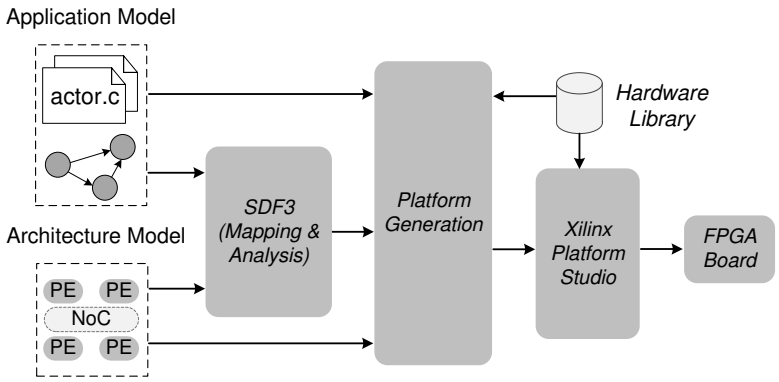


Figure 1.7 MAMPS design flow overview [1].

lel. This makes it possible to model data-parallel executions in the SDF graph. We use SDF graphs to model and map applications to MPSoCs in Chapter 5.

MAMPS Design Flow

MAMPS (Multi-Application and Multi-Processor Synthesis) [59] is an in-house design flow for mapping throughput constrained applications on an MPSoC, which is based on the Y-chart. It integrates several state-of-the-art analysis, mapping, and synthesis tools into an automated tool flow. Figure 1.7 shows an overview of this design flow.

The inputs of this tool flow are an application modeled as an SDF graph, a C-based implementation for each actor in the graph, and a template based architecture description. The application SDF graph and its corresponding implementation of each actor are joined into the *application model*. The application model specifies a set of metrics of the actor implementations. These metrics include the worst-case execution time (WCET), required memory sizes, and the size of communicated tokens. The template based architecture description, i.e., *architecture model*, describes the various components available in the hardware platform and how these components are connected [60].

The output of the design flow is an MPSoC platform tailored for the target applications, with C implementation of each actor mapped to general purpose processors (e.g., MicroBlaze) [1]. The analysis framework that MAMPS uses is called *SDF3* [61]. It consists of several tools that allow automatic mapping of an application described as a SDF graph to a given platform. SDF3 also verifies if

such a mapping is deadlock free and analyzes worst-case system properties, such as throughput, latency, and buffer sizes conservatively at design time. Besides these SDF facilities, MAMPS also provides platform generation tools and tightly integrated Xilinx EDA tools, which automate the processes of system instantiation, synthesis, and download.

1.4 Problem Statement

As discussed in the first section of this chapter, streaming applications are an important class in emerging embedded systems. On the one hand, these applications are usually computationally intensive and have real-time constraints. On the other hand, embedded systems running these applications often have limited power sources like batteries or solar panels. Therefore, energy-awareness becomes extremely important in architecture designs for these systems. Achieving such kind of highly energy-efficient systems is not an easy task. This thesis will deal with part of the challenges discussed in Section 1.2.

Energy-Efficient Data Movement

As a lot of energy is "wasted" in the supplying of data to functional units, the first problem handled in this thesis is to improve the energy-efficiency of data movement in embedded processors. Observing that a substantial part of the energy is consumed by the register file (RF) [33, 62], we particularly focus on reducing energy consumption of the RF, and converting this energy saving into the total core energy saving. In typical embedded streaming application kernels, most of the variables have very low use count (<3), i.e., the value of a variable is only used a very small number of times by the following instructions [63], which can be interpreted as a huge potential to eliminate RF accesses. In a pipelined processor, a lot of such variables can be accessed via the bypassing network instead of the RF. However, in conventional processor architectures, these RF accesses are usually performed regardless of the necessity, which result in a power hungry RF and a hotspot on the chip. A *transport triggered architecture* (TTA) can effectively reduce the RF pressure in both the number of accesses and the number of RF ports [64, 65]. However, conventional TTAs also have some disadvantages, such as relatively low code density and dynamic-power wasting. These disadvantages inhibit it being an energy-efficient architecture. In order to preserve the merit of conventional TTAs while solving these issues, new improvements in both architecture and compiler are required.

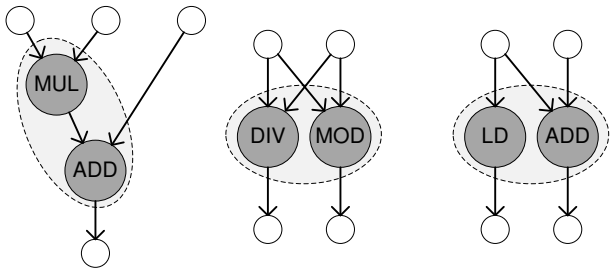


Figure 1.8 Operation patterns.

Flexible Special Instruction Support

Many streaming applications contain frequently executed operation patterns in the data-flow graphs (DFGs), like the ones shown in Figure 1.8. In application specific instruction set processor (ASIP) design, it is common to synthesize instruction sets that support such patterns in targeted applications to achieve better performance and energy efficiency [66-68]. However, extending this idea to a generic embedded processor and efficiently supporting flexible special instructions is challenging. In most mainstream processor architectures, only a few of such patterns are supported, as supporting arbitrary operation patterns in a generic processor incurs large overhead. From energy efficiency perspective, the overhead includes extra opcode space, extra RF ports and RF index space, as well as increased complexity in both control path and datapath. Therefore, efficient software and hardware solutions are in demand if we want to use flexible special instructions to improve the energy efficiency of a generic embedded processor.

Ultra Low-Energy Vector Processing

Massive parallelism in streaming applications typically shows up as data-level parallelism (DLP). Therefore, achieving low-energy vector processing is of great importance when designing an energy-efficient system for streaming applications. DLPs can be inherently exploited by a s*ingle instruction multiple data* (SIMD) processor, which is a low power architecture as it applies the same instructions to all *processing elements* (PEs) [69]. To further reduce energy consumption, V_{dd} scaling can be applied to SIMDs. However, issues such as severe throughput degradation, high yield loss in the presence of process variations, and scaling-unfriendly SRAM limit the V_{dd} scaling range, create a big obstacle to the energy efficiency improvement. So the third problem is how to mitigate these issues and achieve an ultra low-energy SIMD processor.

Efficient Communication Support

To meet the increasing demand for performance and efficiency in streaming applications, the use of energy-efficient programmable cores and accelerator cores in heterogeneous MPSoCs becomes inevitable [6, 48]. The problems discussed above aim at providing energy-efficient cores. These cores can be used as efficient computation components in a heterogeneous MPSoC. However, heterogeneity requires more than this. When mapping a streaming application onto a heterogeneous MPSoC that contains accelerators, several issues need to be solved: *i*) how to generate accelerator IPs for the application; *ii*) how to integrate these IPs into an MPSoC; and *iii*) how to predict performance/resource usage at design time. The first issue can be handled through the use of IP libraries or high level synthesis tools [70, 71]. Solving the other two issues requires an efficient and analyzable communication support.

1.5 Contributions

The main contributions of this work include:

- *MOVE-Pro*, a new TTA based processor architecture is proposed to reduce energy consumption of the register file, and convert this energy saving into the total core energy saving. With fine-grained control over datapath and optimization in different software/hardware levels, about 70% of RF accesses are eliminated on average, resulting in a dramatic reduction of RF energy. More importantly, with the proposed MOVE-Pro architecture, the RF energy saving is fully transferred to the total core energy saving. Compared to its RISC counterpart, an energy saving of up to 11.6% is achieved (Chapter 2).

- An architecture that supports flexible operation pairs in a processor with a compact 24-bit RISC-like ISA is proposed. It introduces a partially reconfigurable decoder and a software-controlled bypass network, allowing the processor to support operation pairs without increasing the instruction width or number of register file ports. Comprehensive and detailed experimental results demonstrate that the proposed architecture achieves an average of 26.0% reduction in dynamic cycle count and an average of 15.8% reduction in energy consumption compared to the reference processors (Chapter 3).

- Xetal-Pro, a massively-parallel SIMD architecture is proposed in this work. The idea of combining massive parallelism and aggressive V_{dd} scaling is presented and discussed in detail. A hybrid memory system, which reduces the non-local memory traffic and enables further V_{dd} scaling, is also introduced. This work shows that it is possible to achieve less than 1 pJ/op core energy consumption for typical kernels of embedded streaming applications (Chapter 4).

- An efficient and predictable *communication assist* (CA) for integrating generic IP cores into heterogeneous MPSoCs is proposed. The corresponding cycle-accurate *synchronous data flow* (SDF) model for the proposed communication assist is also presented. By integrating this SDF model onto SDF analysis tools, worst-case system properties, such as throughput, latency, and buffer sizes can be conservatively analyzed at design time. The existing design flow, MAMPS, is updated with the proposed CA-based hardware template. As a case study, a vision processing pipeline of an industrial application, *Organic Light Emitting Diode* (OLED) screen printing, is mapped onto the proposed platform. This case study demonstrates that the proposed design flow enables efficient integration of accelerator IPs into a heterogeneous MPSoC which targets streaming applications (Chapter 5).

1.6 Thesis Overview

The thesis is organized as follows. Chapter 2 presents the first design proposal, MOVE-Pro, which is a low-power and high code density processor framework based on *transport triggered architecture* (TTA). In Chapter 3, a novel approach for supporting flexible operation pair patterns in a generic processor with a compact ISA is proposed. This work improves energy efficiency by reducing the computation overhead in generic processors. Chapter 4 discusses the progress in developing Xetal-Pro, an ultra low-energy vector processor. The results show that with massively-parallel SIMD and aggressive V_{dd} scaling, it is possible to achieve computation efficiency of less than 1 pJ/op. Chapter 5 presents a *communication assist* (CA) to efficiently integrate generic IPs into an MPSoC with a predictable design flow. A case study is performed using an updated design flow called MAMPS+. Finally, Chapter 6 concludes this thesis and gives directions for future work.

CHAPTER 2

MOVE-PRO: A TTA-BASED PROCESSOR FRAMEWORK

Energy efficiency is becoming the dominant determinant of embedded system design: in most cases, embedded systems have limited power sources like batteries, which largely constrain the use of high performance processors. Moreover, high power dissipation also makes the chip's thermal design more difficult. A lot of work has been done to reduce the processor energy consumption in different ways [72-77]. We observed that a substantial part of the energy is consumed by the register file (RF), which mainly consists of data storage elements and (de)mux logic [62, 78]. Therefore in this chapter, we particularly focus on reducing energy consumption of the RF, and converting this energy saving into the total core energy saving.

In typical embedded application kernels, most of the variables have very low use count. Table 2.1 shows the local variable statistics of four representative kernels. In a pipelined processor, many of these variables can be accessed via the bypassing network instead of the RF. However, in conventional processor architectures, these RF accesses are usually performed regardless of the necessity, which results in a power hungry RF and a hotspot on the chip.

In this work, we propose MOVE-Pro, a new *transport triggered architecture* (TTA) based processor architecture, to solve this issue. With fine-grained control over datapath and optimization in different software/hardware levels, we achieve significant RF accesses reduction in MOVE-Pro: on average about 70% of RF accesses are eliminated, resulting in a dramatic reduction of RF energy. More importantly, with the proposed MOVE-Pro architecture, the RF energy saving is fully transferred to the total core energy saving. Compared to its RISC counterpart, a total core energy saving of up to 11.6% is achieved with no performance penalty. The experiment is carried out at 1.2V, 25°C, typical case, with TSMC 90nm low-power CMOS digital standard cell library.

Table 2.1 Kernel local variable statistics.

	FIR	Histogram	YUV2RGB	IDCT
Average reads per variable	1.71	1.59	1.77	1.55
Local variable with < 3 uses	94.10%	98.25%	90.31%	98.72%

The remainder of this chapter proceeds as follows: Section 2.1 introduces the basic idea behind register file access reduction. MOVE-Pro, the proposed TTA-based architecture is presented in Section 2.2. The energy-aware compilation flow for MOVE-Pro is briefly depicted in Section 2.3. In Section 2.4, the proposed design is verified with a detailed comparison between MOVE-Pro and its RISC/VLIW counterparts. Section 2.5 discusses the related work. Finally, Section 2.6 concludes our findings and discuses future work.

2.1 Register File Access Reduction

In processors designed for multi-media and/or high performance signal processing, the RF is one of the most power hungry components in the datapath, which could account for over 40% of the datapath power consumption [33]. For multi-issue architecture, the demand for RF with many ports is especially costly [32].

Figure 2.1 shows the datapath of a typical RISC processor with a 5-stage pipeline. To reduce pipeline stalls caused by data hazard, a bypass network (i.e., data wires which forward results to the previous pipeline stages) is usually employed to allow an instruction to use the results which are available in the pipeline but haven't yet been written back to the RF. With the bypass network, there are three situations where RF accesses are not necessary:

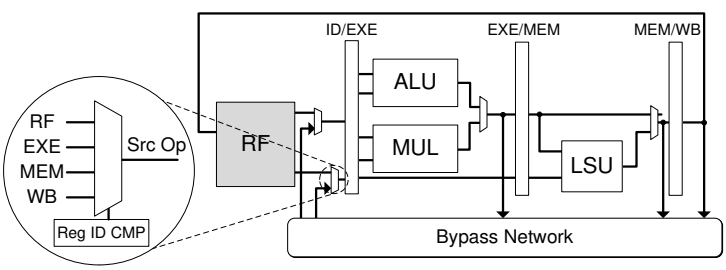


Figure 2.1 Operand bypassing in a typical processor datapath.

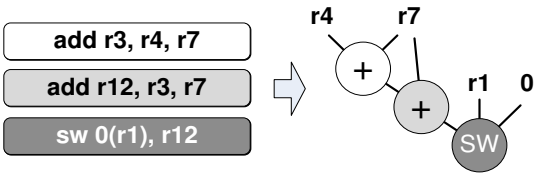


Figure 2.2 RF access elimination example.

- ***Bypassing***: the result of an operation can be read from the pipeline register before it is written back to RF.
- ***Dead result elimination***: if all uses of a variable are bypassed, writing it back to RF is not necessary.
- ***Operand sharing***: when an operand is used successively on the same port of a function unit (FU), it only needs to be read from RF once.

Figure 2.2 shows an example in which all three types of RF access eliminations are possible:

- Register r3 in the second addition (*add*) and r12 in the store instruction (*sw*) can be bypassed, i.e., we can directly get the required data values from the bypass network instead of reading them from RF.
- After bypassing, write-back of r3 and r12 in the first two instructions can be discarded.
- Register r7 is shared by the two additions, therefore only the first addition needs to actually read r7 from RF.

In Table 2.2, four representative streaming kernels are depicted, which are studied in this work. Table 2.1 shows the percentage of local variables with 3 uses in these kernels. Clearly, most variables have very small use counts, which can be interpreted as a huge potential to eliminate RF accesses.

Table 2.2 Kernel description.

Kernels	Description
FIR	5-tap 32-bit finite impulse response filter
Histogram	256-bin histogram for 8-bit gray-scale image
YUV2RGB	YUV to RGB color space conversion for 24-bit image
IDCT	Inverse cosine transformation in the JPEG decoder

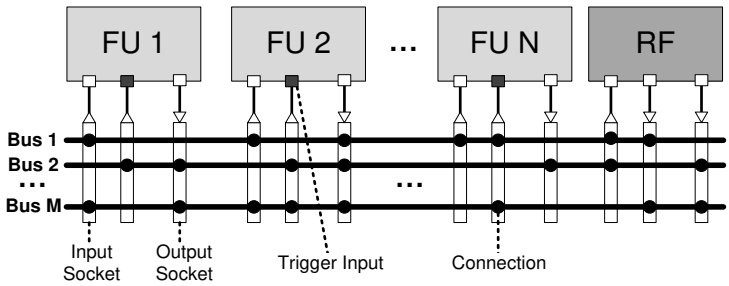


Figure 2.3 TTA architecture, with exposed inter FU and RF datapath.

2.2 Exposed Datapath Architecture

Architectures, such as RISC, Superscalar, and Very Long Instruction Word (VLIW), can be categorized as *operation-triggered architectures* (OTA). In such architectures, an instruction typically specifies what the operation is and what the source operands are. Usually, they cannot eliminate unnecessary RF accesses identified in Section 2.1. In a *transport triggered architecture* (TTA), however, an instruction directly controls the datapath by specifying the data transportation between different units, and operations are merely side-effect of the transportation [64]. As shown in Figure 2.3, by programming the communication network, data are explicitly transported from one unit to another.

In this work, we would like to explore TTA's low-power potential inherited from its *explicit datapath* nature, i.e., the capability of directly transferring an operand from the output of one unit to the input of another. With proper scheduling of the data transport, RF accesses can be dramatically reduced compared to its RISC/VLIW counterparts. As a result, the total processor energy consumption is expected to be reduced accordingly. However, it is challenging to achieve this goal due to some disadvantages of conventional TTAs.

2.2.1 Disadvantageous of Conventional TTAs

A TTA is well-known for its cost-effective trade-off between performance and flexibility. It is able to generate optimized cores for various domains, e.g., multimedia, telecommunication [64, 79, 80]. However, the conventional TTAs also have some evident disadvantages:

- Code density is lower than RISC/VLIW. This is because to perform a 3-address operation on a TTA in principle three *moves* are required. For

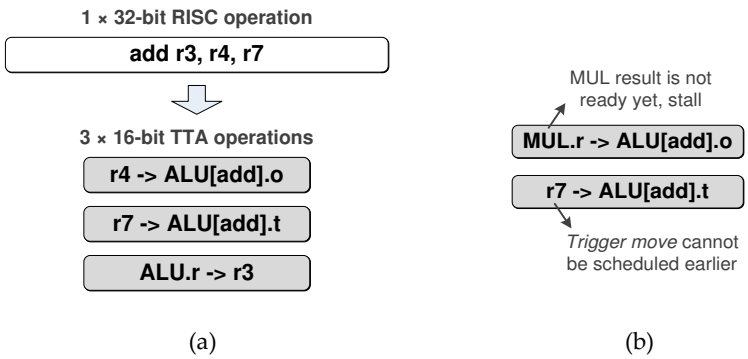


Figure 2.4 (a) Translation of a RISC operation onto TTA operations, where *o* is operand input port, *t* is trigger input port, and *r* is result output port. *r4 -> ALU[add].o*, for example, means that the value of register *r4* is *moved* to the operand input port of the function unit *ALU* for addition; (b) Extra scheduling constraint introduced by trigger port.

example, a direct translation of one 32-bit RISC *add* operation onto TTA results in three 16-bit TTA *move* operations (Figure 2.4 (a)).

- The separate scheduling of source operands increases circuit switching activity, causing more energy consumption. This will be explained in more detail in Section 2.2.3.
- Operations can only be triggered by moving data to the trigger port (shaded ports in Figure 2.3), so a trigger move has to be scheduled no earlier than the operand moves of the same operation, which introduces extra scheduling constraint. For example, in Figure 2.4 (b), if the operand move (the first one) is pending due to the dependency on result of MUL, the trigger move (the second one) cannot be scheduled earlier even though it has no unresolved dependency.[2]

2 In [64], a special latching discipline for the pipeline stages of function units, called *true virtual-time latching* (TVTL), is discussed. With TVTL, both operand moves and trigger moves can "trigger" operations. Since operand moves do not have opcodes, the opcode of the previous trigger move is used if an operand move triggers an operation. This latching discipline is not a very good choice in terms of power consumption, as many unnecessary operations may be triggered if an operand move is scheduled first. The TVTL also greatly reduces the scheduling freedom of result moves as the available time of useful results in FUs is reduced. Thus performance may also be degraded.

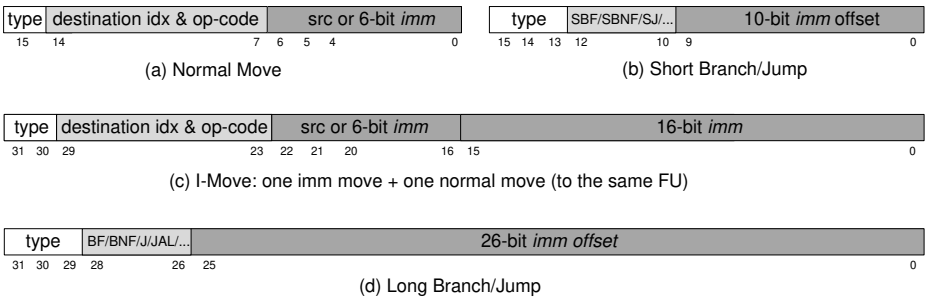


Figure 2.5 The MOVE-Pro instruction formats.

- Inefficient support for large immediates, which deteriorates code density and energy consumption. This is due to small immediate field in the conventional TTA instruction format and limited support for large immediate [64, 81].

Previous research on MOVE framework[3] [82] shows that comparing to a single-issue RISC processor (instr. width = 32 bit), the code density of a corresponding two-issue MOVE processor (instr. width = 32 bit) is 20% worse [83]. When using the TCE framework[4] [84], the overhead is about 30% based on our measurements. Poor code density becomes an obstacle for TTAs from being an energy efficient architecture.

In this work, in order to fully utilize TTA's low-power potential, more specifically, to successfully convert the potential energy saving in RF to energy saving of the complete processor core, we propose MOVE-Pro, a low-power and high code density TTA framework. In Section 2.2.2, we will introduce the instruction set architecture of the proposed MOVE-Pro processor. In Section 2.2.3, we will describe the details of the MOVE-Pro architecture.

2.2.2 Proposed MOVE-Pro Instruction Set Architecture

Code density not only affects processing performance, but also has a strong influence on the total energy consumption. Fewer dynamic instructions counts mean that fewer instructions need to be fetched from instruction memory and fewer instructions need to be executed, resulting less energy consumption. In order to improve the code density of conventional TTAs and to provide flexible

3 TTA implementation from Delft University of Technology
4 TTA implementation from Tampere University of Technology

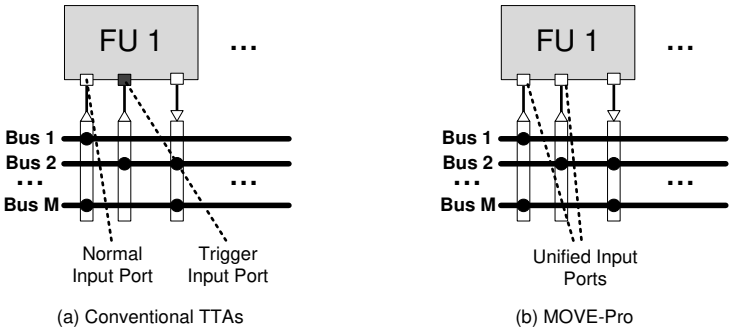


Figure 2.6 Binding between trigger move and a specific destination port is removed in MOVE-Pro, resulting in more scheduling flexibility for the compiler.

immediate support, we combine some helpful features of RISC *instruction set architecture* (ISA) with conventional TTA ISA, and propose a new one for our MOVE-Pro architecture.

The MOVE-Pro instruction formats are shown in Figure 2.5. The 16-bit *Normal Move* (Figure 2.5 (a)) is similar to the conventional TTA instruction format. The main difference is that in MOVE-Pro we removed the constraint that operations can only be triggered by moving data to a specific destination register/buffer, i.e., the trigger port [64, 79]. Thus, the binding between trigger move and a specific destination port is removed, resulting in unified function unit (FU) input ports (Figure 2.6). This provides more scheduling flexibility for the compiler. To implement this idea with minimum overhead, we did not introduce an extra instruction bit to distinguish a trigger move operation and a non-trigger move operation. Instead, all non-trigger moves are considered as a same class of operation, i.e., *Not-a-Trigger-Operation*, which is encoded into the *op-code* field of the instruction similar as other normal operations like *add*, *mul*.

A 32-bit *I-Move* (Figure 2.5 (c)) is added to support 16-bit immediate, which is similar as the *I-Type* of RISC instructions. Two moves are encoded in this format: one is a normal move and the other is a 16-bit immediate move. Since there is no extra space to encode the destination address for the 16-bit immediate move, these two moves must share a same FU as their destination. This constraint is not an issue as an immediate is always ready for issue (no data dependency). Compiler can directly pack it with a *Normal Move* to the same FU, forming a 32-bit *I-Move* instruction. Compared to the conventional TTA, the introduction of *I-Move* in MOVE-Pro efficiently encodes the 16-bit immediate.

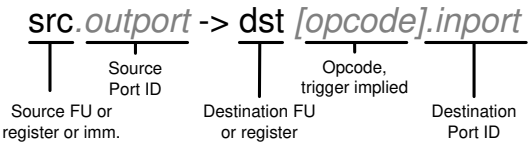


Figure 2.7 MOVE-Pro assembly format.

Two kinds of branch/jump instructions are supported, which differ in the range of branch offset. The purpose of introducing both short and long branch/jump formats is to achieve a better code density as well. A *Normal Move* with 6-bit immediate has an indexing range of only -32~31, which is often inadequate to handle local branches. However, the 16-bit *Short Branch/Jump* (Figure 2.5 (b)) introduced in the proposed MOVE-Pro ISA has an indexing rang of -512~511, which is usually sufficient to deal with most of branches within a function. Compared to its RISC counterpart, the 16-bit *Short Branch/Jump* is only half of the size. When even larger indexing range is required, the 32-bit *Long Branch/Jump* (Figure 2.5 (d)) can be used, which has a 26-bit indexing range.

Since 32-bit instructions are introduced in the proposed MOVE-Pro ISA, the minimum number of issue slots in our MOVE-Pro processors is two. Figure 2.7 depicts the MOVE-Pro assembly format. Note that, unlike the conventional TTA, which has three types of ports (i.e., operand input port, trigger input port, and result output port), we have only input and output ports in MOVE-Pro.

2.2.3 Proposed MOVE-Pro Organization

Section 2.2.2 described the main features of MOVE-Pro ISA. In this section the proposed MOVE-Pro architecture will be presented in detail.

Unlike *operation-triggered architectures* (OTAs), TTAs can schedule the source operands of the same operation in different cycles. On the one hand, it introduces an extra level of scheduling freedom. On the other hand, it causes more energy consumption. Take a multiplication, $a \times b = c$, as an example. In OTA, the multiplicand a and multiplicator b are scheduled together. They are latched into the two input operand registers of a multiplier FU at the same clock cycle. The multiplier circuit only toggles once before producing the product c. However, for a TTA-based processor, a and b can be scheduled separately, e.g., first a, then b. Thus, before producing the useful product c, the multiplier circuit

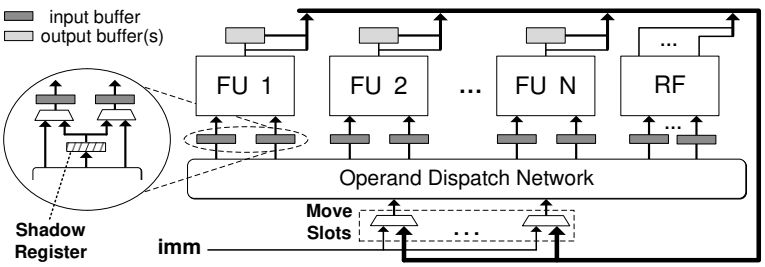


Figure 2.8 MOVE-Pro architecture.

undergoes extra toggles to produce a useless result, $a \times b'$, where b' is the previous value of b. This extra circuit switching causes dynamic power wasting, i.e., two multiplications are executed instead of one.

In the proposed MOVE-Pro architecture, we solve this issue by introducing *shadow register(s)* before the input buffers of FUs with multiple input operands (Figure 2.8). Let's look at the multiplication example above again. Suppose FU_1 in Figure 2.8 is a multiplier, multiplicand a is scheduled in *cycle t*, and multiplicator b is scheduled in *cycle t+n*. In the proposed MOVE-Pro architecture, a will be first buffered in the shadow register for n cycles until b is also issued. Both of them will then be loaded into the two input operand buffers at the same cycle, triggering the multiplier circuit only once. With this improvement, we avoid the dynamic power wasting in the conventional TTA, while still keeping TTA's flexible scheduling feature. For FUs with only a single input, clearly no such shadow buffer is needed.

Introducing shadow buffer(s) has another important advantage. Let's still use the multiplication as an example. Since the early arrived operand a is first buffered in the shadow register, the input buffers of the multiplier can still latch the valid multiplicand and multiplicator of the previous multiplication. A direct consequence is that the value of the previous multiplication is valid in the output of the multiplier for longer time (i.e., more cycles), which would otherwise have already been flushed by a garbage data. This results in three beneficial outcomes:

- Increased availability of previous results in the bypass network, as they can stay in FU outputs for longer time. This results in reduced RF read accesses.

- Increased possibility for previous results to be dead results as their available time in the bypass network is longer. This results in reduced RF write accesses.
- FUs can more effectively act as virtual output buffers, as useful outputs will not be flushed by garbage data. This results in reduced physical output buffer and/or register file requirement.

As discussed above, TTAs have finer scheduling granularity than their OTA counterparts. In OTAs such as a classical RISC datapath, the first and second operands usually correspond with FUs' different input buffers. For example, the first operand bus is only connected with the first input buffer of FU_1~FU_n, and the second operand bus is only connected with the second input buffer of FU_1~FU_n. So each operand bus has relatively limited load capacitance. However, in TTAs, the same operand move can be scheduled to different move slots (also called move buses), resulting in an *operand dispatch network* with higher load capacitance, i.e., higher power penalty to dispatch operand to the input buffers of FUs. In the most flexible case, each move slot can potentially send its operand to each destination location.

Reducing the *operand dispatch network* connectivity can mitigate this issue but at the cost of reduced scheduling flexibility [64]. So we would like to mitigate this issue from a different perspective. Viewing the fact that only part of the destination registers (input buffers of FUs) are enabled per cycle, we apply a circuit-level optimization to "isolate" the inactive destination registers. Figure 2.9 describes the basic idea. A 2-1 *AND* gate is inserted before the move slot selection *MUX*. Input *A* of the *AND* gate is the destination *write enable* signal, which is decoded in such a way that it arrives always earlier than the other input, *B*, which is the *move data*. When the destination register writing is not enabled, i.e., $A = 0$, the frequently changed move data, *B*, is isolated from propagating to the output, regardless of its value. With this method, the effective load capacitance of the *operand dispatch network* is reduced, resulting in lower data dispatch cost.

2.3 Energy-Aware Compilation For MOVE-Pro

For a TTA, a typical binary operation needs three moves: two for the source operands and one for the result. It typically takes 16 bits to encode a move. Therefore, a direct translation from *operation code* to *move code* would probably result in a very bad code density. Figure 2.10 shows an example where move

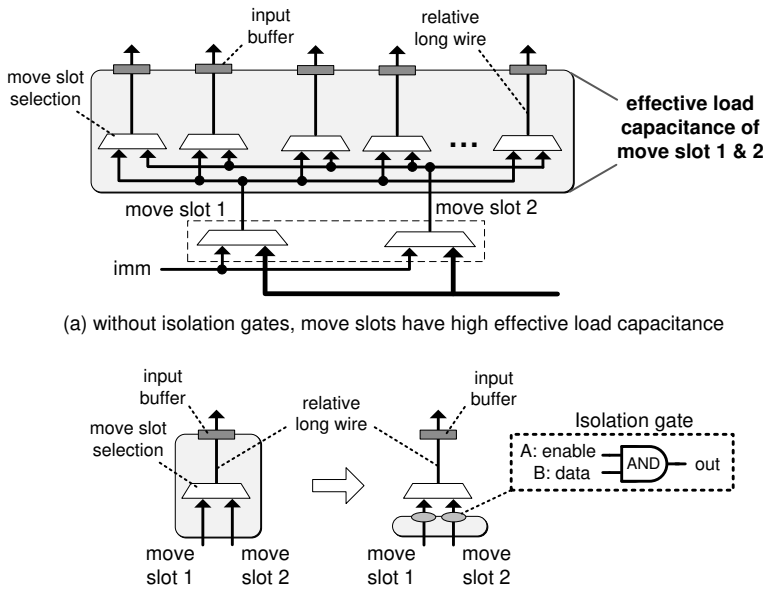


Figure 2.9 Reduce the effective load capacitance by *AND*-gate isolation. *enable* is the destination write enable signal (early arrival signal) and *data* is the move data.

code is 60% larger in a direct translation. The problem can be solved by the compiler. In the example, after performing software bypassing and instruction scheduling, the final move code has the same size as the operation code, with half of the RF reads and all of the RF writes eliminated. Obviously the compiler plays an important role in a TTA.

The compiler framework of MOVE-Pro is shown in Figure 2.11. The LLVM compiler framework is used as the front-end, which produces intermediate

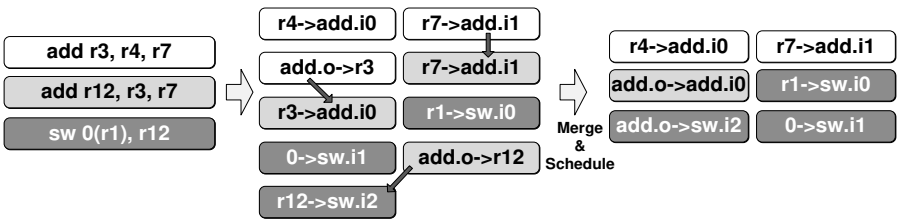


Figure 2.10 TTA code scheduling example.

representation (IR) for the backend. The instruction set of the IR is similar to a RISC ISA, with support of extra meta-data. The core of the backend is the instruction scheduler. In this work, the scheduler performs basic block level scheduling. It minimizes the energy consumption by eliminating unnecessary RF accesses. The number of instructions is another optimization target. Otherwise the scheduler may choose to serialize operation execution in order to increase the bypassing opportunity. A post-pass register allocator is used. Due to the nature of TTA, spilling rarely happens in MOVE-Pro. When there is any spill code that fails to fit into the current schedule, the scheduler simply performs a partial reschedule. Since the compiler work is reported in a different thesis and papers, it will not be elaborated in this thesis. For more details, please refer to [63, 78].

2.4 Experimental Results

To verify the low-power feature of the proposed MOVE-Pro architecture, two head-to-head comparisons are carried out: *i)* MOVE-Pro vs. RISC; *ii)* MOVE-Pro vs. VLIW.

2.4.1 MOVE-Pro vs. RISC

In this experiment, we implemented in RTL both a classical 5-stage RISC processor which is compatible with OpenRISC ISA [85] and its TTA counterpart, a two-issue MOVE-Pro processor (Figure 2.12). The instruction width of both designs is 32 bits. The same FUs are used in these two implementations, as well as the same low-power techniques, e.g., FU clustering and clock gating. To reach a solid conclusion, we further optimized the reference RISC processor for lower power consumption at different implementation levels. For example, to reduce unnecessary switching activity in both RF and pipeline registers, we only propagate write data/index through pipeline stages when the instruction under execution indeed requires updating RF.

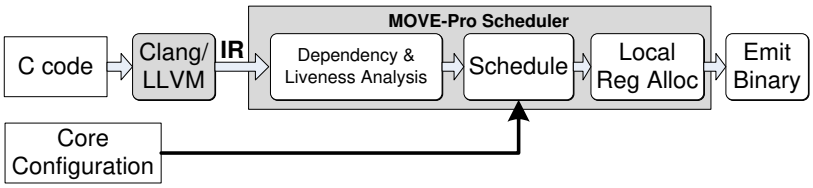


Figure 2.11 MOVE-Pro compiler framework.

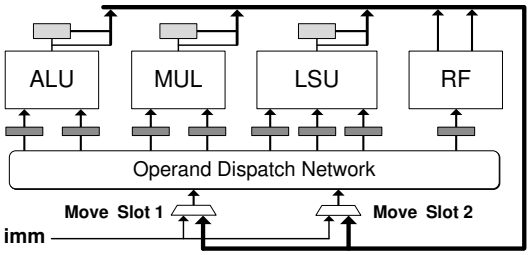


Figure 2.12 The block diagram of the two-issue MOVE-Pro Processor.

TSMC 90nm low-power library is used in the experiment. The synthesis condition is set to typical case, 1.2V, and 25°C. To improve the accuracy of power estimation, physical library (6 metal layers) is included. The interconnect information is estimated based on the physical library. The maximum clock frequency is 200 MHz with the critical path inside the multiplier. All four kernels listed in Table 2.1 are tested. Power consumptions are estimated with the post-synthesis simulation and toggle rate generated by running each kernel with test data of 2000 samples.

Figure 2.13 shows the energy breakdown of the reference RISC processor when executing a YUV to RGB color space conversion kernel. The energy consumption is calculated as $P_{avg} \times T_{exe}$, where P_{avg} is the average power consumption during kernel execution while T_{exe} is the kernel execution time. Energy consumption of the memory part is not shown in this graph as it highly depends on the memory size. The register file consumes about 16% of the total core energy in this application.

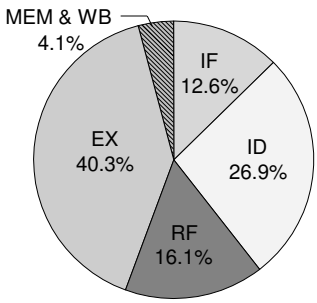


Figure 2.13 Energy breakdown of the reference RISC processor (YUV2RGB).

Table 2.3 RF access reduction in MOVE-Pro.

	FIR	Histogram	YUV2RGB	IDCT
Read reduction	65.5%	63.6%	63.6%	68.6%
Write reduction	82.3%	71.4%	67.7%	66.7%

Table 2.3 shows the dynamic RF access reduction statistics obtained by cycle accurate simulation. In all tested kernels, both the RF read and write access reduction are substantial. Clearly, RF energy consumption is reduced dramatically. However, it would be more interesting to find out whether or not the total core indeed benefits from this saving. Figure 2.14 shows the normalized energy breakdown of both the reference RISC processor and the two-issue MOVE-Pro processor when executing the four benchmark kernels. Both dynamic energy and leakage energy are taken into consideration. However, since the processors are busy executing instructions in every cycle and the low power library we use has optimized for the leakage power, the leakage energy consumption is negligible compared to the dynamic energy consumption in this comparison. We can see that the reduction in RF energy is directly transferred to the total core energy saving, which demonstrates the low-power feature of the proposed MOVE-Pro architecture. In all kernels, the reduction of RF energy is more or less proportional to the reduction of RF accesses. The saving of total core energy is up to 11.6%. The detailed comparisons of power consumption and cycle counts are shown in Table 2.4.

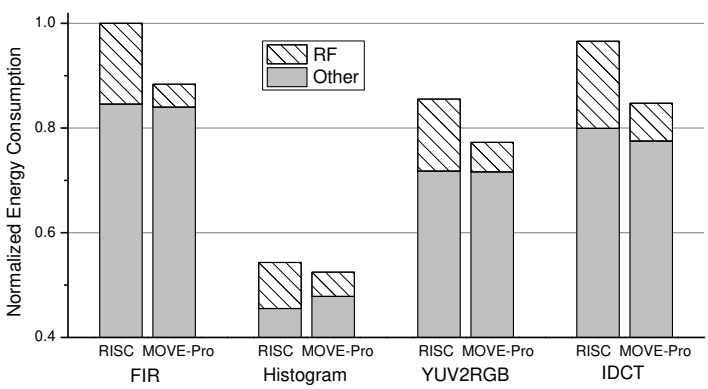


Figure 2.14 Energy consumption comparison (normalized to the total energy consumption). For the comparison of energy per cycle, the diagram is similar as the cycle counts are exactly the same for FIR, Histogram, and YUV2RGB. And for IDCT, MOVE-Pro requires slightly less cycles according to Table 2.4.

Table 2.4 Power comparison between MOVE-Pro and the RISC reference. Both using TSMC 90nm low-power library and at typical case, 1.2V, and 25°C.

		Reference RISC	Two-issue MOVE-Pro
FIR	Relative Cycle Count	1.00	1.00
	RF Power	0.84mW	0.24mW (-71.5%)
	Total Power	5.43mW	4.80mW (-11.6%)
Histogram	Relative Cycle Count	1.00	1.00
	RF Power	0.48mW	0.25mW (-47.6%)
	Total Power	2.95mW	2.85mW (-3.39%)
YUV2RGB	Relative Cycle Count	1.00	1.00
	RF Power	0.75mW	0.30mW (-59.1%)
	Total Power	4.64mW	4.19mW (-9.70%)
IDCT	Relative Cycle Count	1.00	0.95
	RF Power	0.90mW	0.41mW (-54.8%)
	Total Power	5.24mW	4.82mW (-8.58%)

According to Table 2.4 the dynamic instruction count of MOVE-Pro processor is similar as its RISC counterpart. Under the same clock frequency, our MOVE-Pro has no performance loss. In fact, for the IDCT kernel, the proposed MOVE-Pro processor even outperforms the reference RISC processor by 5%. With further compiler improvements, such as inter block scheduling, software pipelining, and if-conversion, we expect that the code density of MOVE-Pro can be even higher, resulting in further energy improvement.

2.4.2 MOVE-Pro vs. VLIW

In the previous subsection, we compared a two-issue MOVE-Pro processor with its RISC counterpart. The register file used here is small, 32b×32, and contains very few ports, two read and one write. It takes up only about 16% of the total RISC core energy consumption. However, even with such a less power-hungry RF, we still demonstrated a substantial energy saving.

It is more interesting and popular to use TTAs as Very Long Instruction Word (VLIW) architecture alternatives. In VLIW processors, register files are usually much larger and contain much more read/write ports [32]. The detailed analysis in [86] shows that the total power dissipation of the central register file organization grows as N^3, while the total power dissipation of the distributed register file organization with two-port register files grows as N^2. Here N is the number of arithmetic units of a VLIW processor. In order to quickly validate

Table 2.5 Energy comparison of different data accesses.

	32b×32 2R1W RF		32b×32 4R2W RF		8kB Memory		9kB Memory
	Read	Write	Read	Write	32-bit	64-bit	48-bit
Energy per access (pJ)	1.81	5.46	1.95	6.67	16.38	19.67	17.03

and illustrate again the efficiency of our MOVE-Pro framework before implementing a complete processor design, in this section we estimate the potential energy saving by only focusing on the major components whose energy consumption are affected. As already shown in Section 2.4.1, the datapath of conventional VLIW and MOVE-Pro processors are similar when they use the same amount/type of FUs and the same bypass network (e.g., fully connected). The difference of energy consumption is mainly caused by the number of RF accesses and the instruction size.

Table 2.5 lists the average energy consumption of RFs with different number of ports and SRAMs (used as instruction memories) with different widths and sizes. We did not take the energy consumption of the instruction memory into consideration in the previous subsection, this is because the instruction widths of the 2-issue MOVE-Pro and the reference RISC are the same, and the dynamic instruction counts of these two processors are also very similar. However, the processors compared in this subsection have different instruction widths, and the dynamic instruction counts are also different. Energy consumption of the instruction memory has considerable difference in different processors under comparison, thus cannot be ignored. The instruction memory size is set to be the same for different processors. For processors of 48-bit instruction width, since 8kB memory does not fit, we choose the closest practical size, i.e., 9kB. The library we used in the experiment is TSMC 90nm low-power technology. The RF access energy is derived by synthesizing the RTL design, extracting the physical information, and estimating the average toggle rate of each read/write port by performing 1024 random access. The access energy of the memory is estimated by CACTI [87].

The four kernels listed in Table 2.1 are again used in the evaluation. These kernels are compiled and executed on a cycle-accurate simulator to collect access statistics, from which the RF and memory access energies of each kernel are calculated. In all the experiments, we conservatively set the RF to the same size for all the processors, even though the RF pressure in the proposed

Table 2.6 Instruction counts of kernel loops on different processors (one iteration).

	RISC 2R1W (***R1***)	MOVE-Pro-2 2R1W (***M2***)	VLIW-2 4R2W (***V2***)	MOVE-Pro-4 4R2W (***M4***)	MOVE-Pro-4 2R1W (***M4'***)	MOVE-Pro-3 2R1W (***M3***)
Instr. Size	32-bit		64-bit			48-bit
Histogram	10	10	8	8	8	8
FIR	20	20	12	12	12	14
YUV2RGB	42	42	27	29	30	32
IDCT	87	83	48	49	51	60

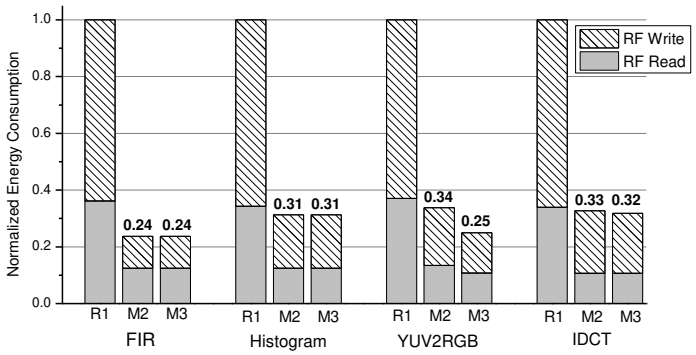
MOVE-Pro is much smaller. The access counts based power estimation is less accurate compared to the real implementation as what we did in Section 2.4.1. However, it can quickly validate multiple designs with reasonable accuracy.

Figure 2.15 (a) shows the RF (32b×32 2R1W) energy consumption comparison among a RISC (***R1***, 32-bit instr.), a 2-issue MOVE-Pro (***M2***, 32-bit instr.), and a 3-issue MOVE-Pro (***M3***, 48-bit instr.). The results are normalized to the RF energy consumption of the RISC processor for each kernel. We can see that the MOVE-Pro processors significantly reduce the RF access energy consumption. Compared to its RISC counterpart, ***M2*** saves an average of 72.6% write access energy and 65.6% read access energy. Increasing the issue width of MOVE-Pro can further decrease the traffic to RF. This is because variables' lifetime is reduced as they can be used earlier. For example, ***M3*** saves extra 26.0% of RF energy compared to ***M2*** for YUV2RGB.

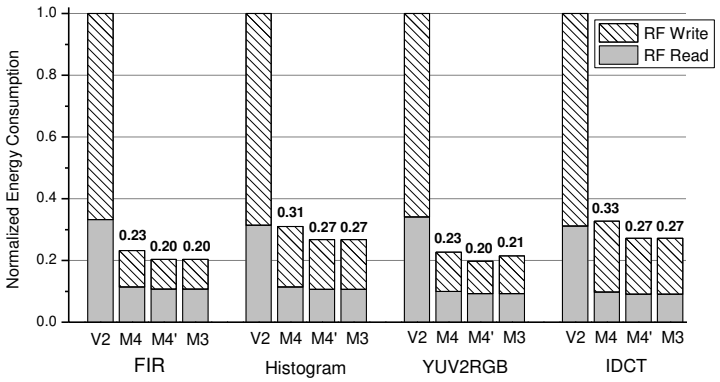
Similar results are observed in Figure 2.15 (b) when comparing VLIW processor and MOVE-Pro processors. Up to 80.3% of RF energy consumption is reduced. It is worth mentioning that since the RF traffic is greatly reduced in MOVE-Pro, the requirement on the number of RF's read/write ports alleviates accordingly. With reduced RF read/write ports, ***M4'*** (4-issue MOVE-Pro, 64-bit instr., 2R1W RF) shows an additional RF energy saving of 15.3% compared to ***M4*** (4-issue MOVE-Pro, 64-bit instr.), which has a 4R2W RF.

As discussed in the previous sections, the energy saving on RF access does not guarantee the energy saving in the whole processor. Conventional TTAs have a poorer code density compared to their RISC/VLIW counterparts, which can easily eat up the energy saving on RF. The proposed MOVE-Pro framework solves this issue. Table 2.6 shows the code sizes of our MOVE-Pro processors and their RISC/VLIW counterparts. We can see that for the cores with 64-bit instruction format, ***M4*** is only slightly worse in some kernels, while for

the cores with 32-bit instruction format, ***M2*** even has better code size than RISC. The comparison between ***M4*** and ***M4'*** shows that the number of RF ports has less impact on MOVE-Pro. It is interesting to mention that due to the finer scheduling grain (16-bit *move*), we can design MOVE-Pro processors like ***M3***, which uses 48-bit instruction format. These kinds of intermediate solutions introduce more flexibility to application-specific designs.



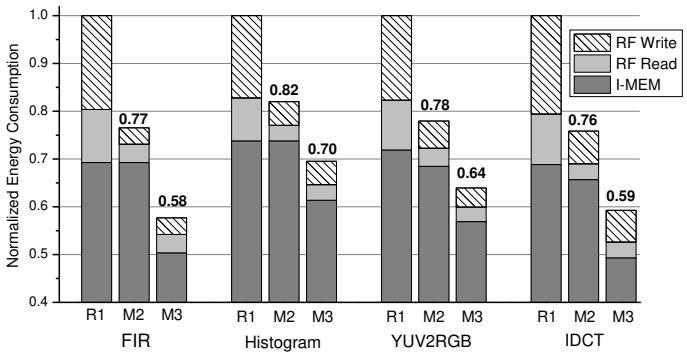
(a) RF energy consumption: RISC vs. MOVE-Pro.



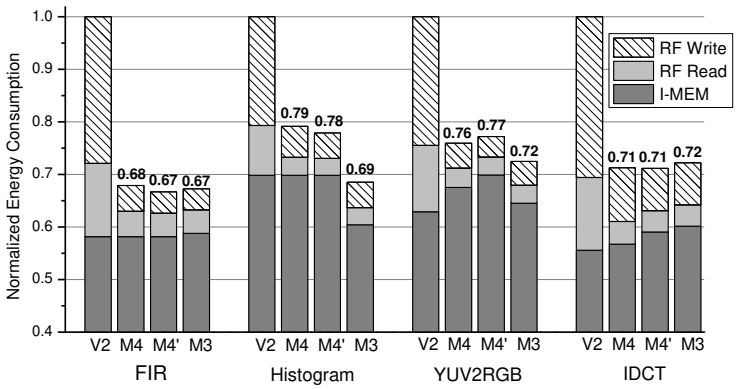
(b) RF energy consumption: VLIW vs. MOVE-Pro.

Figure 2.15 RF energy consumption results: *R1*: RISC with 2R1W-RF; *V2*: 2-issue VLIW with 4R2W-RF; *M2*: 2-issue MOVE-Pro with 2R1W-RF; *M3*: 3-issue MOVE-Pro with 2R1W-RF; *M4*: 4-issue MOVE-Pro with 4R2W-RF; *M4'*: 4-issue MOVE-Pro with 2R1W-RF.

We present the combined impact of the RF access energy and memory access energy in Figure 2.16. The result shows that even when taking the code size into consideration, the energy saving is still significant. Comparing to a RISC, the combined RF and I-Mem energy consumption is reduced by an average of 22.0% in ***M2***, while comparing to the VLIW, we see an average of 26.8% energy saving in ***M4'***.



(a) RF + I-Mem energy consumption: RISC vs. MOVE-Pro.



(b) RF + I-Mem energy consumption: VLIW vs. MOVE-Pro.

Figure 2.16 RF + I-Mem energy consumption results: *R1*: RISC with 2R1W-RF; *V2*: 2-issue VLIW with 4R2W-RF; *M2*: 2-issue MOVE-Pro with 2R1W-RF; *M3*: 3-issue MOVE-Pro with 2R1W-RF; *M4*: 4-issue MOVE-Pro with 4R2W-RF; *M4'*: 4-issue MOVE-Pro with 2R1W-RF.

2.5 Related Work

In microprocessors, the register file is one of the central components, which accounts for a considerable amount of energy consumption. A detailed analysis of RF power consumption is given by Zyuban and Kogge in [88]. In [86], Rixner et al. analyze the performance, area and power trade-offs in different register file designs for media processors.

The idea of using variable liveness information to reduce register accesses in conventional processor architectures has been exploited in a few previous studies [31, 89, 90]. Balfour et al. introduce the ELM architecture that reduces RF accesses by using a combination of software bypassing and hierarchical RFs [31]. Compared to ELM, MOVE-Pro has finer grained scheduling and a more explicit data path. In [74], Woh et al. address the similar problem in the SIMD context, and RF partitioning is used as the solution. As an alternative, TTAs, which is proposed by Corporaal [64, 65, 79], provide a more natural and efficient solution to reduce RF accesses due to its *transport-triggered* feature. MOVE32INT is the first TTA implementation [82], and the TTA-based co-design environment (TCE) is the latest implementation [84]. The MaxQ microcontroller designed by the Maxim is a commercial processor which exploits the concept of a TTA [81]. Guzma et al. discuss the performance impact of software bypassing in TTA [91], and the power consumption implication in [92]. However, the power aspects of TTAs are never studied in a detailed way. The *Synchronous Transfer Architecture* (STA) [93, 94] is very similar to TTA. The main differences between STA and TTA are: *i)* STA forces every function unit (FU) to use output buffers so that input buffers are not required; *ii)* Operands of the same FU are scheduled at the same cycle with explicit *opcode* field supplying the control signals. Comparing to TTA, STA creates a new bottleneck: it requires very large instruction memory and huge bandwidth for instruction fetch due to the very wide instruction width, so instruction compression becomes a must [93]. Like TTAs, the power aspects of STAs are never studied in a detailed and systematic way either. In this work we proposed a novel TTA architecture, evaluated the potential of building a TTA-based low power and high code density processor, and performed a detailed comparison between TTA and its RISC/VLIW alternatives.

2.6 Summary

Transport Triggered Architectures possess many advantages, such as modularity, flexibility, and scalability. As an exposed datapath architecture, TTAs can effectively reduce the number of RF accesses and the requirement for RF ports. However, conventional TTAs also have some evident disadvantages, such as relatively low code density, dynamic-power wasting due to separate scheduling of source operands, unequivalent normal data port and trigger data port, inefficient support for long immediate, etc. In order to preserve the merit of conventional TTAs, while solving these aforementioned issues, we proposed, MOVE-Pro, a novel low power and high code density TTA framework. With optimizations at ISA, architecture, circuit, and compiler levels, we showed that the low-power potential of TTAs is fully exploited. Moreover, with a much denser code size, thus less dynamic instruction counts, TTAs' performance is also improved accordingly.

In the head-to-head comparison, we showed that up to 80% of RF accesses can be reduced with the proposed MOVE-Pro framework. More importantly, we successfully transferred the reduction in RF energy to the total core energy saving. The comparison with RISC counterpart showed that up to 11.6% reduction of the total core energy is achieved. While compared to VLIWs, the advantage of the proposed MOVE-Pro architecture is even better.

Further optimization is still possible. For example, extending the compiler to support software pipelining and inter basic block scheduling would improve the efficiency. Extending this work to SIMD and multi-core architectures is also very interesting as more energy saving may be observed.

CHAPTER 3

AN ENERGY EFFICIENT METHOD OF SUPPORTING FLEXIBLE SPECIAL INSTRUCTIONS

Many applications contain frequently executed operation patterns in the data-flow graphs (DFGs), like the ones shown in Figure 3.1. In application specific instruction set processor (ASIP) design, it is common to synthesize instruction sets that support such patterns in targeted applications to achieve better performance and energy efficiency [66-68]. Due to its application specific characteristics, only a few *special instructions* are needed in an ASIP to cover these operation patterns appeared in its targeted applications.

However, extending this idea to a generic embedded processor and efficiently supporting flexible special instructions is challenging. In most mainstream processor architectures, only a few of such patterns are supported, as supporting arbitrary operation patterns in a generic processor incurs large overhead. From energy efficiency perspective, the overhead includes:

- More bits in the instruction to encode opcodes for all possible patterns and extra operands in the special instructions. In a compact instruction set architecture (ISA) like the ARM Thumb [95], the problem is even more serious as the number of bits in the instruction is very limited. If

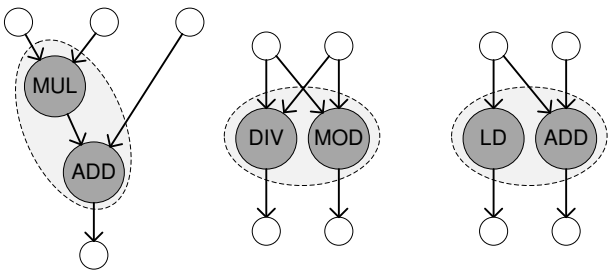


Figure 3.1 Special operation patterns.

the instruction is made wider, fetching an instruction consumes more energy.

- More ports on the register file (RF) to provide sufficient data bandwidth for the special function units. A RF with more ports is much less energy efficient. Also, even the normal instructions need to pay the extra cost. Methods like register file clustering [96] or FU internal registers [68] are able to partially solve the problem. However, such methods usually lack flexibility and often lead to very complex code generation.
- Increased complexity in many parts of the processor, including instruction decoder and bypassing network.

In this work, we tackle the problem of integrating flexible special instruction support in a generic embedded processor with a compact ISA. Apart from performance improvement, the main focus of this research is on energy efficiency of the processor for different types of streaming applications. To achieve high energy efficiency, the support for special instructions needs to have low overhead, while still being able to support applications from different domains.

We propose a scheme for integrating a special function unit (SFU) into a RISC-like embedded processor with 24-bit instruction width. The reason that we demonstrate this with a 24-bit architecture is to keep the baseline ISA orthogonal (i.e., we do not impose a limitation that requires a certain instruction to use a specific register). The ideas in this work also apply to more compact ISAs, e.g., a 16-bit Thumb-like ISA. The SFU supports flexible operation pair patterns. To integrate the SFU into the RISC datapath with minimum overhead, we introduce:

- A partially reconfigurable decoder that allows low overhead reconfiguration for each kernel to use its specific patterns. As a result, arbitrary pair patterns can be executed on the proposed architecture, while no extra bits are needed for the special instruction opcodes.
- A bypass network in the datapath that is exposed to software, thereby reducing the requirement, both for operand encoding and register file ports.

The use of a reconfigurable decoder and an explicit bypass network imposes some constraints on the special instructions the processor can execute, e.g., for a three-input special instruction, at least one of the operands has to come

from the bypass network. A compiler backend is designed to generate energy efficient code for the proposed architecture. The compiler selects patterns and performs energy-aware instruction scheduling to utilize the SFU and explicit bypass network. Experimental results show that for a set of benchmarks from different application domains, the proposed architecture achieves an average of 26.0% reduction in dynamic cycle count, which is only 1.5% worse than the architecture without constraints on the special instructions. As for energy consumption, the proposed architecture achieves an average reduction of 15.8%, while the unconstrained architecture only reduces energy by 2.2%. By introducing multi-cycle long-latency SFU operations, the proposed architecture is able to achieve a speed-up of 12.6% with 13.1% energy reduction compared to the baseline, which is useful when high performance is essential.

The remainder of this chapter proceeds as follows: Section 3.1 describes the DFG patterns we consider in this research and the design of the SFU that executes such patterns. The proposed integration of SFU into the processor datapath with explicit bypass is depicted in Section 3.2. Section 3.3 briefly introduces the compiler backend design for the proposed architecture. Detailed and comprehensive experimental results that demonstrate the effectiveness of the proposed design are given in Section 3.4. Section 3.5 discusses related work. Finally, Section 3.6 concludes our findings and discusses future work.

3.1 Operation Patterns and Special Function Unit

Each basic block of a program can be represented by a data-flow graph (DFG) $G(V, E_d, E_f)$, where:

- V is a set of nodes. Each node in V represents either an actual operation or a live-in variable (register file or immediate). In this work, we assume that the operations represented by nodes in V can be directly mapped to a function unit (FU) in a typical RISC processor. Such operations are defined as *basic operations*.
- E_d is a set of directed edges. If an edge $e = (u, v) \in E_d$, it represents that node v consumes the output of u, i.e., there is true data dependency between u and v.
- E_f is a set of directed edges. If an edge $e = (u, v) \in E_f$, there is false/output dependency between node v and u.

For a basic block, the DFG is a directed acyclic graph (DAG). A *special operation pattern* is defined as a subgraph of a DFG that contains more than one basic operation. Figure 3.1 shows some examples of these patterns. Compared to a sequence of basic operations that performs the same computation, executing a special operation pattern using a special instruction has a few advantages:

- Fewer instructions are needed to execute the operations, resulting in less control overhead.
- The communication between internal operations can be done within the FU, which is usually much more efficient. It may also save RF accesses.

For a certain application, some special operation patterns appear frequently [97]. In application specific instruction-set processor (ASIP) design, a common approach for improving performance as well as energy efficiency is to synthesize special function units that support these patterns [66-68, 98]. Different from ASIP design, the goal of this work is to support special operation patterns in a RISC-like generic processor, without introducing heavy modifications to existing architecture and code generation framework. Instead of trying to support arbitrary operation patterns, we focus on a specific type of operation pattern, namely, operation pairs. The definition of the operation pair pattern, as well as motivation of choosing such patterns, is given in Section 3.1.1. The design of a special function unit (SFU) that provides flexible support for these patterns is depicted in Section 3.1.2. In Section 3.1.3, we analyze a set of kernels based on the patterns supported by the proposed SFU.

3.1.1 Operation Pair Patterns

In this work, we want to integrate the support for special operation patterns without major modification to the original RISC architecture. A single-issue RISC processor typically has a register file (RF) with two read ports and one write port (2R1W). Though there are some other possible sources for input operands, like immediate field and bypass network, the number of source operands cannot grow dramatically without heavy modification to the instruction format. The same holds for the destination operand. In addition, the number of arbitrary operation patterns in different applications is huge. The FU that supports all these patterns becomes very complex and inefficient. So in this work, we focus on *operation pair* patterns, i.e., patterns with two basic operations a and b that meet the following criteria:

- There is true dependency between a and b, i.e., $(a, b) \in E_d$.

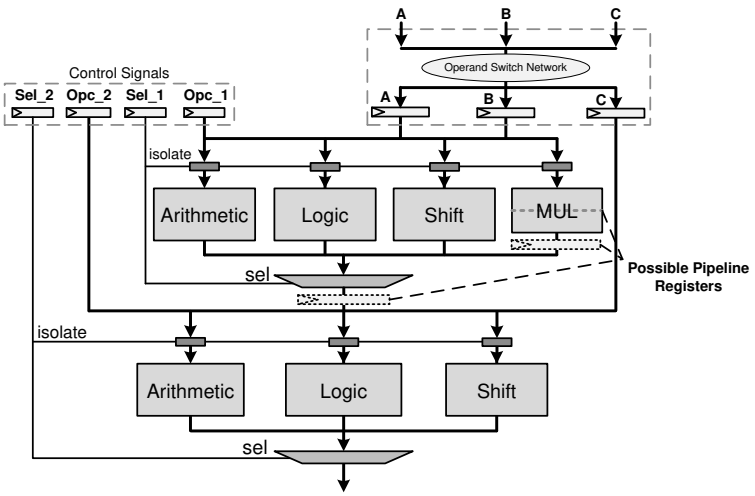


Figure 3.2 Special function unit.

- There are at most three input operands. More formally, for a set of edges P that contains all edges to a or b in E_d except (a, b), we have $|P| \leq 3$;
- At most only one of a and b has consumers outside the pattern, i.e., at least one of the following holds:
  - The result of a is only consumed by b: $\{(a, c) | (a, c) \in E_d, c \neq b\} = \emptyset$. If this constraint is met, only b may have consumers outside the pair pattern.
  - b has no consumer: $\{(b, c) | (b, c) \in E_d\} = \emptyset$. If this constraint is met, only a may have consumers outside the pair pattern.
- There is no path from a to b in G other than (a, b), which means combining a and b does not create cycles in G.

Integrating such patterns in a RISC processor is relatively easy: we only need to supply one more source operand than for a normal operation.

3.1.2 Special Function Unit Design

The design of our special function unit is shown in Figure 3.2. The SFU supports two levels of basic operations. To avoid introducing large area and timing overhead, only one multiplier is included in the SFU, which is put within the first level. An operand switch network is included in the SFU to allow more flexible operand encoding in a processor. The design of the SFU allows almost

Table 3.1 Kernel description.

Kernel	Description	Domain
FIR	5-tap finite impulse response filter	Image Processing
Histogram	256-bin histogramming	Image Processing
YUV2RGB	YUV to RGB color space conversion	Image Processing
IDCT	2D 8×8 Inverse cosine transformation	Codec
MatVec	Matrix vector multiplication	Linear algebra
CRC	Cyclic redundancy check code calculation	Network/Storage
DES	Data Encryption Standard algorithm	Security

arbitrary operation pairs that satisfy the constraints discussed in Section 3.1.1. When fully decoded, a 17-bit control signal is needed for the SFU to execute one special operation. To improve the energy efficiency of the SFU, *operand isolation* is used to isolate each sub-function-unit. So a unit only toggles when it actually needs to perform computation, thereby reducing unintended circuit activity.

Due to the extra level of sub-function-units, the proposed SFU is likely to increase the delay of the critical path of the processor. When a processor with the proposed SFU needs to run at high frequency, there are two ways to mitigate this delay effect: *i*) add a pipeline register inside the SFU, either between the first and the second level, or at the output (or in the middle) of long latency units (e.g., the multiplier); *ii*) allow the long latency operations to run in multiple cycles, thereby allowing the SFU to run at a higher frequency. By using either method, a processor with the proposed SFU is able to run at a frequency close to one without the SFU.

3.1.3 Application Analysis

We analyzed seven kernels listed in Table 3.1, which come from various domains. The DFGs of each kernel are scanned to find all possible pair patterns that can be executed by the proposed SFU. In total, 35 distinct pair patterns are needed. The number of patterns can grow much larger if more applications from different domains are included. In addition, to generate a valid special instruction from an operation pattern, more information needs to be encoded, e.g., whether or not there is an immediate and whether the immediate is for the first operation or for the second operation. Figure 3.3 shows an example of a three input operation pair that requires different control coding. In general, the number of operation pair patterns is $N_1 \times N_2 \times V$, where N_1 is the number of operations in layer 1, N_2 is the number of operations in layer 2, and V is the num-

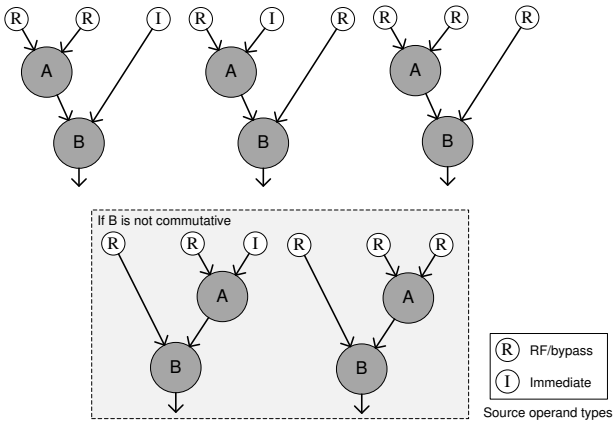


Figure 3.3 Cases for the same pair pattern that needs different coding.

Table 3.2 Kernel pattern statistics.

Total	FIR	Histogram	YUV2RGB	IDCT	MatVec	CRC	DES
35	3	2	9	9	11	9	12

ber of variants introduced by operand formats described above. The total number of different special instruction patterns is large when the SFU supports flexible operation pair patterns.

However, if we look into each individual kernel, we can see that the number of patterns used in one kernel is much smaller than the total number of patterns. Table 3.2 shows the statistics of pattern matches in the seven representative kernels from different domains. The statistics show that it is possible to exploit the *temporal locality of patterns* to reduce the number of patterns a processor needs to support during the execution of an application or a kernel. Findings in [74, 98] also lead to a similar conclusion. This observation can be used to guide the design of efficient special instruction support in processors, which is discussed in Section 3.2.

3.2 Integrating SFU into Processors with Compact ISA

In general, it is possible to integrate the proposed SFU design into any generic processor architecture. In this work, a 4-stage RISC processor with a 24-bit instruction set architecture (ISA) is used as the baseline architecture. The key features of the baseline architecture are described in Table 3.3. Figure 3.4 depicts the datapath of the baseline processor. The ISA of the baseline is similar to the

Table 3.3 Key features of the baseline ISA.

Instruction width	24 bits
Pipeline stages	4
Register file	32×32, 2R1W
Opcode	6 bits
Immediate	8 bits

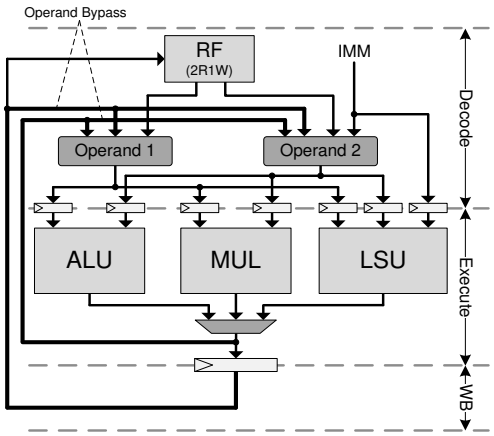


Figure 3.4 A typical RISC datapath.

one used in OpenRISC [85]. Most instructions are three-address instructions: two source operands and one destination are encoded. The baseline processor is representative of typical RISC processors used in embedded systems.

For the baseline architecture, the major limiting factors of integrating the SFU introduced in Section 3.1.2 are:

- After adding the basic integer and control operations, only less than 16 opcodes are left in the opcode space.
- At most 3 bits can be used for encoding the extra operand in three-input instructions, which are not enough for a register index.
- The 2R1W RF cannot provide enough operand bandwidth for the SFU.

A straightforward solution to these problems is to increase instruction width and the number of RF ports. To accommodate the extra opcodes and register index, at least additional 7 bits are needed (5 bits for the third register

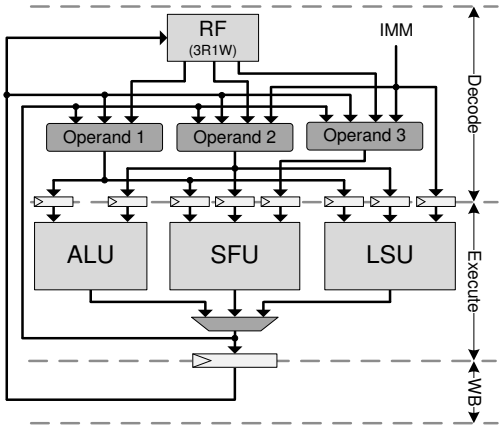


Figure 3.5 Datapath with unconstrained support for SFU.

operand, 2 bits for extra opcodes). As a result, the width of the instruction memory increases to 32 bits. In addition, the RF needs to have three read ports (3R1W) in order to provide sufficient bandwidth for the SFU. The resulting datapath is shown in Figure 3.5. To avoid high area overhead, the multiplier is absorbed into the SFU. Based on the estimation of CACTI [87], the energy consumption of each access to the instruction memory is increased by 10% to 30%, depending on the size and configuration. Based on the implementation result, the energy consumption of the RF is also increased by 12% due to the extra read port. Since both the instruction memory and the register file are among the most frequently used components in a processor, an architecture with such large overhead is unlikely to be energy efficient.

To improve the energy efficiency, this overhead has to be mitigated. In this work, we propose an energy efficient support for the SFU by using: *i*) a partially reconfigurable decoder that exploits the locality of the operation patterns to reduce the opcode encoding requirement; *ii*) a software-controlled bypass network that exploits the processor pipeline to reduce the operand encoding and RF port requirements. Section 3.2.1 and Section 3.2.2 describe the details of the partially reconfigurable decoder and the software-controlled bypass network, respectively. Section 3.2.3 shows how the SFU is integrated into the baseline processor.

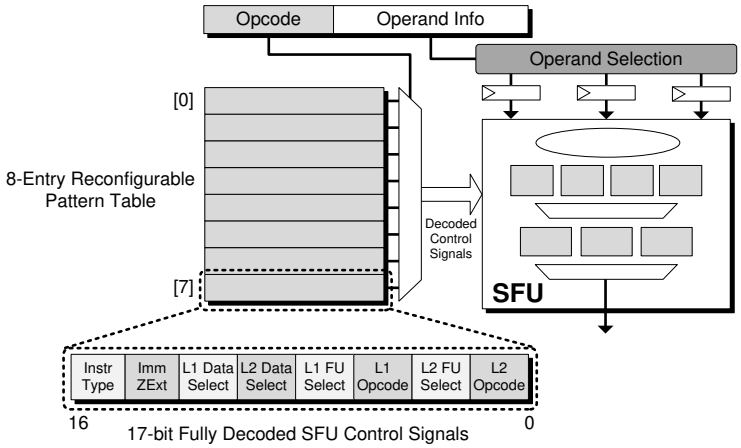


Figure 3.6 Partially reconfigurable SFU decoder.

3.2.1 Partially Reconfigurable Decoder for SFU

As discussed in Section 3.1.3, a key observation is that although a large number of patterns are needed to cover the operation patterns in different applications, only a small number of such patterns are active in one kernel, i.e., in most kernels, the operation patterns have good locality. To utilize such locality, this work introduces a partially reconfigurable decoder for the SFU.

Figure 3.6 depicts the structure of the reconfigurable decoder for the SFU. Central to the decoder is a look-up-table with eight entries, called *pattern table*. Each entry in the pattern table stores a 17-bit control signal required by a special instruction. Since the table only has eight entries, the free opcodes in the opcode space can be used to address it. When a special instruction is fetched, the decoder reads a pattern table entry and uses it to control the SFU; when a normal instruction is fetched, the decoder proceeds as a normal RISC decoder, and the pattern table is clock gated to eliminate unnecessary accesses.

The pattern table is visible to the software. So when different operation patterns are needed, the software can reconfigure the SFU decoder by writing extra control signal needed by these operations into the pattern table. By enabling the reconfiguration of the pattern table, the processor is able to use all the operation patterns supported by the SFU. And since in most cases the operation patterns have good locality, the overhead of reconfiguration is low.

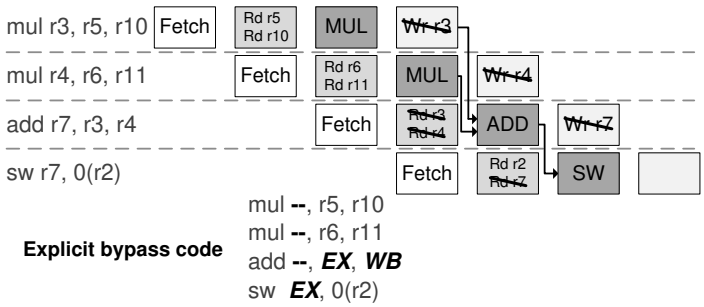


Figure 3.7 Reduce register file accesses via explicit bypassing.

3.2.2 Explicit Bypass

In a typical pipelined datapath of a processor, like the one in Figure 3.4, there is a bypass/forwarding network, whose primary function is to avoid pipeline stalls caused by true data dependencies. A side effect of such a network is that many operands can be read from the pipeline registers instead of the RF. There are two types of possible RF access elimination introduced by the bypass network:

- *Bypassing*: the result of an operation can be read from the pipeline register before it is written back to RF.
- *Dead writeback elimination*: if all uses of a variable are bypassed, its writeback to RF is no longer necessary.

However, in a conventional processor architecture, such a bypass network is invisible to software, which makes it difficult to eliminate unnecessary RF accesses: *i*) bypassing requires RF indexes to be checked before the decode stage, which may increase the critical path of fetch stage, or results in an extra pipeline stage; *ii*) dead writeback elimination is impossible unless the register liveness information is explicitly encoded in instructions. In this work, we propose to use a bypass network that is controlled by software, i.e., the bypassing information is statically encoded in the instructions. Figure 3.7 shows an example of reducing RF accesses via explicit bypassing. Apart from reducing the total number of register accesses, explicit bypassing helps integrating the SFU without increasing instruction width and RF ports in two ways:

- Encoding a bypass source uses fewer bits than an RF index, since the number of possible bypass sources is much smaller than the number of

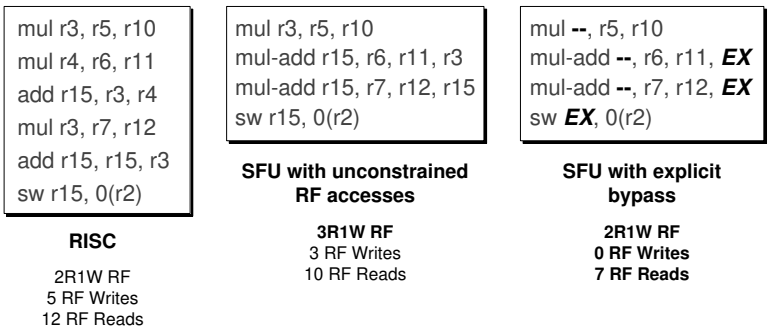


Figure 3.8 Special instruction example.

registers in a typical RF (4 vs. 32).

- Fewer RF ports are required when some operands are from the bypass network.

By imposing the constraint that at least one of the source operands in a three-input special instruction has to come from the bypass network, the special instructions can be easily encoded in the 24-bit instruction format, and there is no need to increase the number of register file ports. Figure 3.8 shows an example of using special instructions under this constraint. In a processor with an unconstrained SFU, the number of required instructions is reduced from 6 to 4, at the cost of increasing the number of read ports of the RF from 2 to 3. In a processor with explicit bypassing, the same code size improvement can be achieved even when there is a constraint that at least one of the operands comes from the bypass network. And with such a constraint, the requirements for extra instruction bits and RF port are removed.

3.2.3 Integrating SFU into Processor Datapath

We propose an architecture that is able to support all the operation pair patterns of the SFU described in Section 3.1.2, by employing the partially reconfigurable decoder and explicit bypass network introduced in previous subsections. Figure 3.9 shows the datapath of the proposed processor architecture. Note that because there are input registers for each FU, the result of an operation is stable at the output port of the FU until the next operation that uses the same FU starts. So it is possible to use the output of each FU as a separate bypass source, which increases the bypassing possibility.

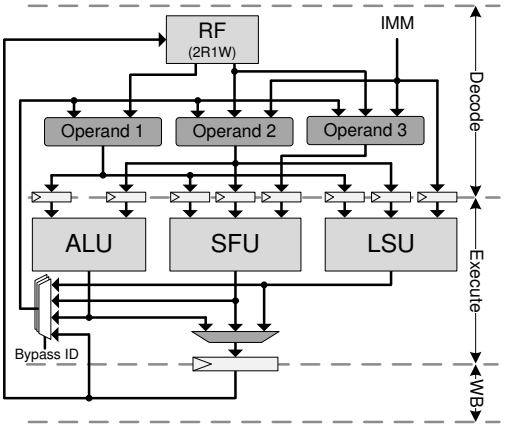


Figure 3.9 Datapath with constrained support for SFU.

Compared to the one with unconstrained SFU support (Figure 3.5), the proposed architecture imposes extra constraints on the special instructions it can execute:

- For a three-input special instruction, at least one of the source operands has to come from the bypass network.
- At most eight special instruction patterns are active at the same time. To support different patterns, the program needs to reconfigure the pattern table.

With these constraints, the proposed architecture is much more energy efficient: instruction width remains 24 bits instead of 32 bits and the RF is 2R1W instead of 3R1W. To use explicit bypassing without changing the normal instruction format, part of the RF address space is used for the bypass source. As a result, the number of registers in the RF reduces from 32 to 28. The effect of a smaller RF is mitigated by explicit bypassing, as it eliminates the necessity of allocating registers for short-lived variables in many cases.

The introduction of a pattern table and an explicit bypass results in extra context when an exception happens. The pattern table can be handled in a similar fashion as general purpose registers. For the explicit bypass, it is required that the processor saves the complete state for the *execute* and *writeback* stages of the pipeline. This can be done using a scan-chain that automatically

saves/restores the registers when an exception happens. Since the number of registers is small, the overhead in area and response time is also small.

As mentioned in Section 3.1.2, a pipelined or multi-cycle SFU is required if the processor needs to run at a higher frequency. In this work, when the target architecture needs to run at similar frequency as the baseline RISC processor, all special instructions that use the multiplier (e.g., multiply-add) are set to finish in two cycles. With such a configuration, there are two types of resource hazards need to be taken care of:

- *SFU hazard*: A special instruction that uses the multiplier occupies the SFU for two cycles. So the next instruction in the pipeline cannot use SFU.
- *Writeback hazard*: When a two-cycle instruction is followed by a single-cycle instruction, there is resource hazard in the writeback stage, since there is only one RF write port.

Both hazards can be resolved using hardware interlock. Only 2-bit extra information needs to be recorded when a special instruction is issued to the execute stage: *i*) whether it is a two-cycle instruction; *ii*) whether it requires to update the writeback stage. Based on these 2-bit data and the type of the following instruction, an interlock signal can be generated. Using hardware interlock makes the hazards transparent to software.

3.3 Code Generation for Special Instructions

The compiler in this work is implemented based on the open-source LLVM framework [99]. Figure 3.10 shows the flow of the compiler backend for the proposed architecture. The input of the backend is a low-level intermediate representation (IR), which is basically RISC assembly with virtual registers, embedded with control-flow and data-flow information. Most part of the compiler can simply reuse the same passes as a compiler for RISC architecture. However, the backend needs to be aware of the explicit bypass network and has to utilize the special function unit (SFU).

To use the SFU, the compiler first needs to choose pairs of DFG nodes that can be used to generate special instructions, i.e., the pair pattern selection. The first step of pair pattern selection is to find all the node pairs whose patterns are supported by the SFU in the data-flow graph (DFG) under the constraints described in Section 3.1.1. The next step is to choose a subset of the node pairs

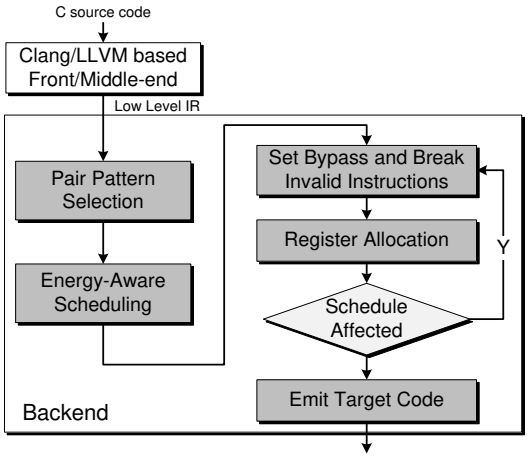


Figure 3.10 Compiler backend flow.

found in step 1 that will be used to generate special instructions. The objective is to find as many pairs as possible. *Minimum degree heuristic* is used to solve this issue. When a tie is met, pattern frequency is used to break the tie.

A list scheduler is used for basic block level scheduling. To increase energy awareness, the scheduler uses the following heuristics: *i*) the scheduler first tries to select a special operation node that constraints are guaranteed to be met; *ii*) if it fails to select a unique candidate, or no such special operation node exists, the scheduler chooses the node that benefits most from bypassing; *iii*) if there are still multiple candidates, the scheduler chooses the node with the same opcode as the previous operation; *iv*) if all tie breaking does not work, the scheduler chooses the first node it finds.

After scheduling, a scan through all instructions is preformed to check for invalid special instructions. If a special instruction is found to be invalid, the checker decomposes it into normal instructions. Due to the nature of explicit bypassing, this transformation does not increase register usage. The register allocation is done with a graph-coloring algorithm. It is similar as the one used for a normal RISC processor. Finally the compiler collects pattern information and decides where to insert the reconfiguration codes for the special instructions. Since the compiler work is carried out by another project member, it will not be elaborated in this thesis. For more details, please refer to [100].

Table 3.4 Configuration of different architectures.

Architecture	Baseline (Base)	Unconstrained SFU (SFU-I32)	Proposed (SFU-I24)	Proposed w/2-cycle SFU (SFU-I24-C2)
Instruction width	24-bit	32-bit	24-bit	24-bit
Instruction memory	12kB 24-bit	16kB 32-bit	12kB 24-bit	12kB 24-bit
	4k words			
Data memory	16kB 32-bit			
Register file	32b×32 2R1W	32b×32 3R1W	32b×28 2R1W	32b×28 2R1W
SFU patterns	0	128	8	8
Two-cycle special operations	None			Special operations with multiplication

3.4 Evaluation and Analysis

Table 3.4 presents the architectures used in the experiments. The proposed architecture, i.e., with partially reconfigurable decoder, explicit bypass network, and constrained special instruction patterns (see Section 3.2.3), is called *SFU-I24*. And the architecture that integrates SFU without the constraints introduced in *SFU-I24* is called *SFU-I32*. The *SFU-I24-C2* is an architecture that is almost identical to *SFU-I24*, except for the two-cycle SFU and interlock logic that help to achieve higher frequency. The datapaths of the *baseline*, *SFU-I32*, *SFU-I24*/*SFU-I24-C2* are shown in Figure 3.4, Figure 3.5 and Figure 3.9, respectively. All four cores are implemented in Verilog HDL and synthesized with TSMC 90nm low power library at 1.2V, 25°C, and typical case. Clock gating is used to minimize dynamic power consumption. The core energy consumption is estimated with the physical library and toggle rate generated by post-synthesis simulation. The area and energy consumptions of the memory are estimated with CACTI [87], using 90nm low power technology.

3.4.1 Area and Frequency

The implementation results of the four architectures are shown in Table 3.5. The increase in the core area is understandable and expected, as the SFU, as well as its decoding part, is much more complex compared to simple FUs in RISC. The core area of *SFU-I32* is slightly larger than *SFU-I24* as it needs to support more patterns in the decoder. *SFU-I24-C2* uses slightly more area than *SFU-I24* because of the interlock for multi-cycle operations. The difference in total memory area between *SFU-I32* and *SFU-I24* is significant. This is caused by the instruction memory since SFU-I32 uses 32-bit instructions, while *SFU-I24* uses 24-bit instructions. In all, the *SFU-I32* pays a very high price in terms

Table 3.5 Implementation result comparison.

Architecture	Base	SFU-I32	SFU-I24	SFU-I24-C2
Normalized core area	1.000	1.309	1.268	1.278
Normalized memory area	1.000	1.154	1.000	1.000
Maximum frequency	450MHz	325MHz	325MHz	385MHz

Table 3.6 Results of the baseline architecture.

Kernel	Simulated Cycles	Average Core Energy per Cycle	Average Memory Energy per cycle
Histogram	21547	11.05 pJ	16.10 pJ
FIR	40973	18.41 pJ	16.24 pJ
IDCT	2303	17.93 pJ	14.56 pJ
YUV2RGB	43032	17.88 pJ	13.82 pJ
MatVec	3729	13.27 pJ	14.00 pJ
CRC	162017	12.73 pJ	11.82 pJ
DES	857130	14.89 pJ	14.64 pJ

of area. In contrast, the proposed *SFU-I24* realizes the special instruction support with a relatively small overhead. In particular, it does not increase the memory area, which is the dominant part in many modern processors.

The reduced maximum frequency of *SFU-I32* and *SFU-I24* is mainly caused by the single-cycle SFU, which has two levels of sub-function-units. In *SFU-I24-C2*, this is mitigated by making the special operations that use the multiplier two-cycle operations. And as shown in Table 3.5, *SFU-I24-C2* only pays a small price for the high frequency. Compared to the baseline, there is still a 14.4% loss in frequency, which is primarily caused by the operand switch network in the SFU (see Figure 3.2).

3.4.2 Energy Consumption

Table 3.1 lists the benchmarks used in the experiments. These kernels are from various application domains. The code for the proposed *SFU-I24* and *SFU-I24-C2* is generated by the compiler described in Section 3.3. For *SFU-I32*, the code generation process is almost the same as *SFU-I24*, except that all the constraints on operand bypassing and opcode space are removed, and no reconfiguration code is generated. All benchmark programs are compiled with maximum optimization enabled (-O3). Table 3.6 shows the absolute results of the baseline processor. The memory energy in the table includes accesses to both instruc-

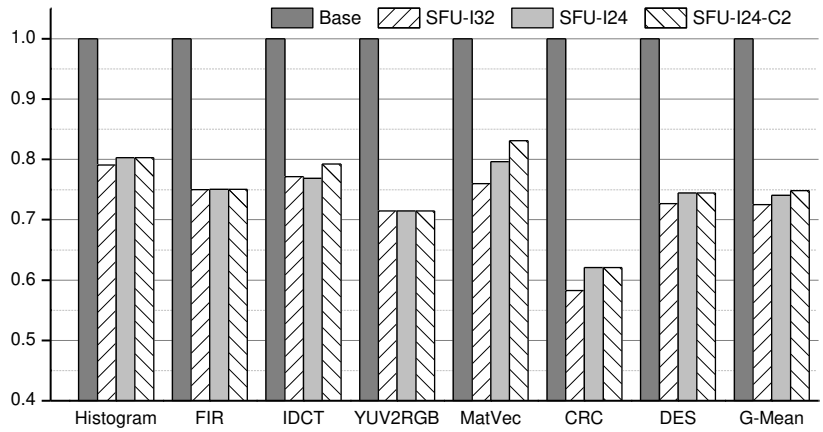


Figure 3.11 Dynamic cycle count (overhead included).

tion memory and data memory. The energy consumption of each kernel is calculated by multiplying the number of cycles with the average energy (i.e., core + memory) per cycle. In the remainder of this sub section, we normalize all results to the baseline.

Figure 3.11 shows the normalized cycle count of the four different cores. Including the overhead of reconfiguration, *SFU-I24* achieves a reduction of 26.0%, which is only 1.5% worse than *SFU-I32*. In some kernels that need to use special operations with many multiplications, *SFU-I24-C2* uses more cycles. But on average it still reduces 25.2% in cycle count compared to the baseline. When the instruction width is factored in, as shown in Figure 3.12, the total memory energy consumption of *SFU-I24* is much less than *SFU-I32*. Though the number of fetches is reduced dramatically, *SFU-I32* only achieves 4.0% average memory energy reduction due to increased instruction width. In 3 out of 7 benchmarks the energy consumption actually goes up. In contrast, the proposed *SFU-I24* is able to directly convert the reduction in instruction count into memory energy saving. An average of 21.7% saving is observed. For *SFU-I24-C2*, similar result is achieved: the average saving is 21.3%.

Figure 3.13 shows the normalized core energy consumption. Comparing to the baseline processor, the proposed *SFU-I24* reaches a maximal core energy reduction by 21.4% in the FIR case, and by 10.7% on average. The main contributions of energy reduction are from: 1) reduced RF access energy; 2) reduced datapath and control path overhead due to merged operations.

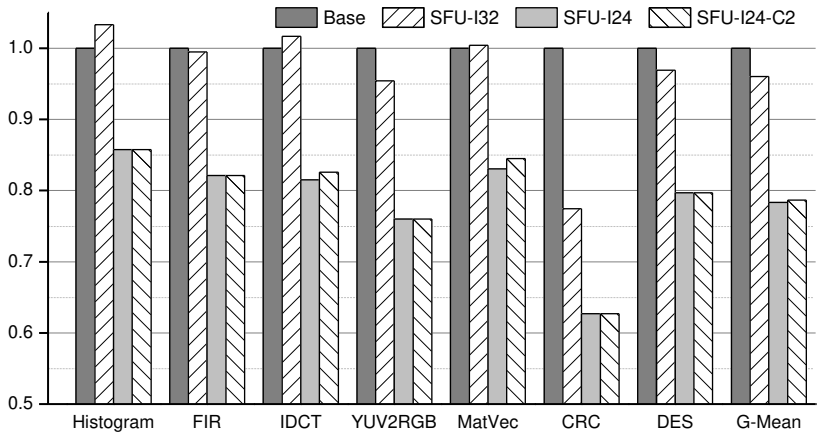


Figure 3.12 Normalized memory energy consumption.

On the other hand, *SFU-I32* only reduces the average core energy by 1.0%. The explicit bypass network is an important contributing factor to this huge difference. As shown in Figure 3.14, the number of accesses to the RF in *SFU-I24* is significantly reduced. In addition, the RF in *SFU-I24* has fewer ports than the one in *SFU-I32*. As a result, the core of *SFU-I24* consumes much less energy compared to *SFU-I32*.

In the case of *SFU-I24-C2*, the core energy consumption increases when a lot of special instructions with multiplication are used. The main reason is that the compiler is forced to use a less energy efficient schedule in order to fill the delay caused by multi-cycle operations. When no special instructions with multiplication are used, the result is similar to *SFU-I24*. On average, the core energy is reduced by 5.9% compared to the baseline.

Figure 3.15 shows the normalized total energy consumption. The proposed *SFU-I24* reduces both the memory and core energy, and it achieves an average saving of 15.8%. It reaches a maximal of 33.1% energy saving in CRC. In *SFU-I24-C2*, the total energy is reduced by an average of 13.1%, and the maximal saving occurs in CRC, which is 32.2%, while in the case of *SFU-I32*, the total energy saving is only 2.2%. The breakdown for the energy in different kernels is presented in Figure 3.16, which clearly shows that *SFU-I24* and *SFU-I24-C2* outperform *SFU-I32* in both core and memory energy in most cases.

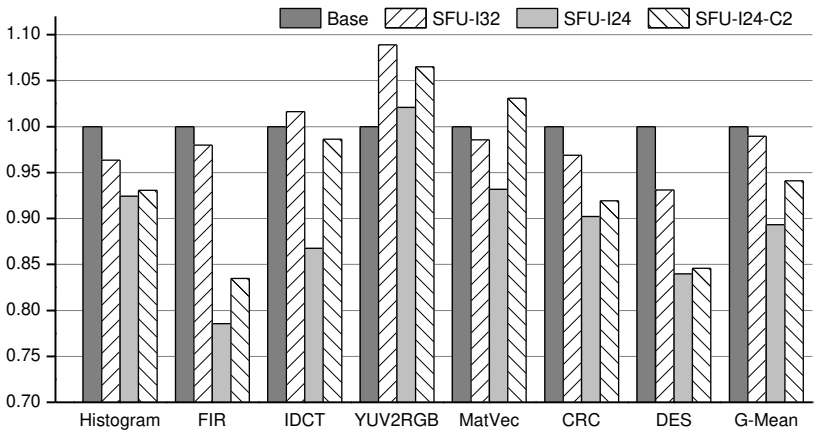


Figure 3.13 Normalized core energy consumption.

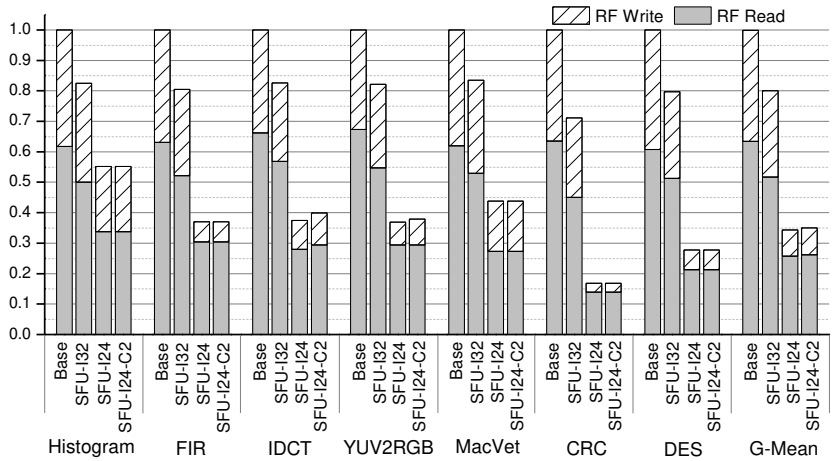


Figure 3.14 Normalized number of register file access.

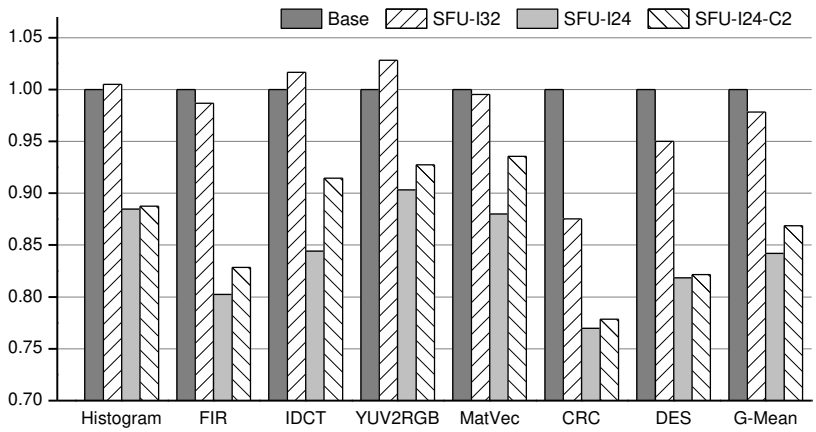


Figure 3.15 Normalized total energy consumption.

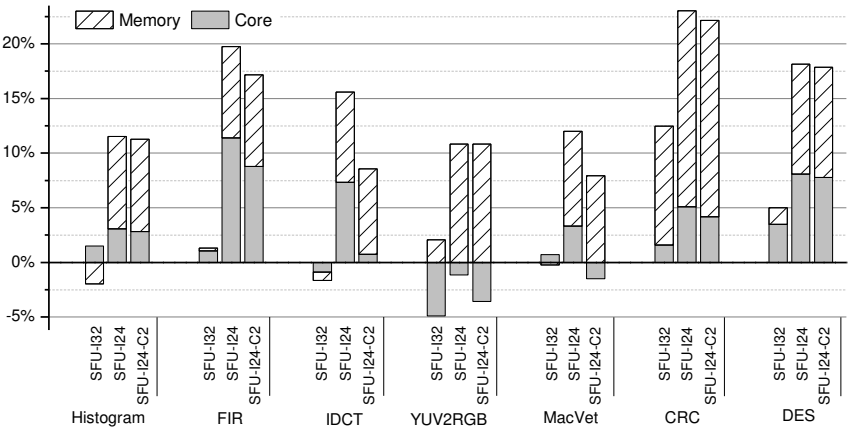


Figure 3.16 Energy gain breakdown.

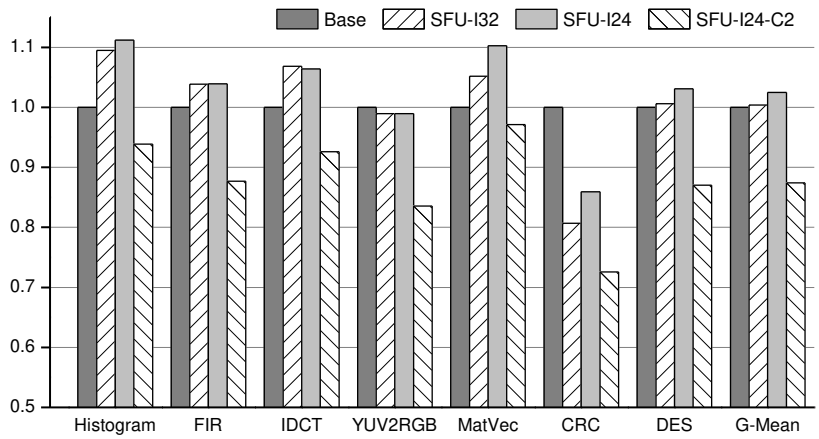


Figure 3.17 Normalized execution time (adjusted according to max. frequency).

These results show that although the use of SFU is able to significantly reduce the dynamic cycle count, directly putting the SFU into a generic processor without any constraint does not result in an energy efficient architecture.

3.4.3 Performance

The normalized execution time of the different cores used in the experiments is shown in Figure 3.17. The result is calculated based on the dynamic cycle count and the maximal frequency of each core. Due to the loss in frequency, both *SFU-I32* and *SFU-I24* suffer minor performance degradation, though they are able to reduce the cycle count by about 26%. However, in *SFU-I24-C2*, an average speed-up of 12.6% is observed. The *SFU-I24-C2* is able to achieve a good balance between performance and energy consumption, with relatively small overhead compared to *SFU-I24*.

The proposed architecture with a partially reconfigurable decoder and an explicit bypass network is able to reach a balance between the energy efficiency and the flexibility of the SFU, and it results in an architecture with high energy efficiency and good performance.

3.5 Related Work

The use of complex operation patterns, called instruction set extension (ISE), is common in instruction set synthesis for ASIP design [66-68]. There are also

studies trying to integrate such ISE in general purpose architectures [101-103], most of which focus on improving the performance.

The data bandwidth from the register file to the FUs is an important constraint in ISE design [104]. Leupers et al. introduced special register file called *internal registers* (IR) for the special instruction units [68]. The IR is an effective way of implementing application specific special instruction, but it lacks flexibility and complicates the code generation as the registers and FUs are no longer orthogonal, i.e., an FU cannot accesses arbitrary registers. Karuri et al. proposed RF clustering in single issue processor to mitigate the register file port pressure in ISE in ASIP design [96]. While reducing port pressure, the RF clustering, which is similar to what is used in clustered VLIW architectures, also makes the code generation much more complex. Pozzi and Ienne exploited the fact that pipelined SFUs do not need all operands in the same cycle to distribute register file accesses across multiple cycles [105]. This trick cannot be applied to the SFUs that are similar to the one used in this work. Utilizing the bypass network has been proven to be an efficient way to increase operand bandwidth and reduce register file energy in different types of architectures [30, 31, 106, 107]. Jayaseelan et al. proposed explicit forwarding to reduce register file port pressure and operand encoding cost for application specific ISE in a RISC-like datapath, which resembles somewhat the idea of explicit bypass in this work [103]. However the power model used in [103] only considers the consumption of the register file. The overall energy efficiency of the proposed architecture is not clear. Cong et al. proposed shadow registers to solve the operand bandwidth issue for supporting special instructions in a configurable processor [108]. The shadow registers are similar to explicit pipeline registers, but have more flexibility. To avoid dramatical increase of control bits, the shadow registers are hash-mapped, which may be less efficient in terms of energy. In this work, we explored the trade-offs in utilizing bypass network for energy efficient ISE in a generic processor with compact ISA and presented detailed and realistic results. The proposed solution achieved high energy efficiency while maintaining the generality of the baseline architecture.

In ASIP designs, dynamic instruction set configuration is often used to optimize the resource usage. The idea of using a dynamically reconfigurable decoder to support flexible ISA has been exploited in reconfigurable architectures like Montium [109] and MOLEN [110], but they are not tightly integrated into a general purpose processor. The ConCISe toolchain proposed by Kastrup et al. introduces accelerator called reconfigurable function unit (RFU) based on pro-

grammable logic device (CPLD/FPGA). The RFU is integrated into a MIPS datapath as a new FU [111]. In ConCISe, operation patterns are required to have no more than 2 input operands and one output. Reconfiguration of RFU is required for different applications. The rotating instruction set processing platform (RISPP) uses a runtime reconfigurable instruction set to enable the reuse of resources for special instructions in ASIP [112]. The Conservation Cores toolchain uses automatically synthesized accelerators called *c-cores* to improve energy efficiency in many core architectures [113]. Huynh et al. proposed dynamic instruction set configuration for a flexible reconfigurable custom instruction unit, addressing the trade-offs between area, performance and reconfiguration cost [114]. Some recent works proposed integration of special instructions for a relative wide range of applications. Clark et al. proposed integration of a configurable compute accelerator (CCA) into a general-purpose processor [101, 102]. The architecture of CCA is relatively complex and it requires up to 4 inputs and 2 outputs, as its main objective is to improve performance. The control part of CCA is designed to be transparent such that the code can be executed with or without CCA. Woh et al. proposed AnySP, a wide SIMD signal processor targeting wireless and multimedia applications [74]. In AnySP the idea of operation pairs is similar to the SFU design in this work, and the operand problem is partially solved by introducing an extra small RF. Venkatesh et al. proposed QsCores, a framework that automatically synthesizes accelerator for a wide range of applications from source code [115]. In PEPSC, an architecture designed for efficient scientific computing, Dasika et al. proposed a FPU that is capable of executing up to five back-to-back operation [116]. In this work we exploited the locality of the special operation patterns in designing a partially reconfigurable decoder to achieve energy efficient integration of SFU into a RISC processor with compact ISA, which allowed the proposed architecture to improve energy efficiency substantially in different application domains.

Selection and scheduling for special instructions is one of the most important parts in code generation for ASIP and many reconfigurable architectures. Kastner et al. proposed an algorithm for generating special instructions in a system with reconfigurable fabrics [117]. Guo et al. proposed a graph covering algorithm for code generation of Montium reconfigurable processor [118]. Park et al. presented a greedy algorithm for increasing the bypassing in a RISC processor [107]. In [103], integer linear programming (ILP) is used to perform bypass aware scheduling in a processor with application specific ISE. The proposed algorithm inserts register copying instructions in order to meet the constraints of the special instructions.

In this work, we proposed a novel architecture that uses special instructions to improve the energy efficiency of a generic processor with a compact ISA. Two major issues: *i*) opcode and operand encoding; *ii*) operand bandwidth to SFU are solved by using a partially reconfigurable decoder and explicit bypass network.

3.6 Summary

Integrating a special function unit (SFU) that executes complex operations into a generic processor for energy efficiency is not easy, as special instructions may incur large overhead, especially when the ISA is a compact one. This work introduced an architecture for integrating SFU that supports flexible operation pair patterns in a generic processor with a compact ISA. A partially reconfigurable decoder and a software-controlled explicit bypass network are used to: *i*) encode extra opcodes and operands in the limited instruction coding space; *ii*) supply sufficient data to the special instructions without increasing the number of register file ports. Results including benchmarks from different domains demonstrate that the proposed architecture is effective: average dynamic cycle count is reduced by over 25%. The total processor energy consumption is reduced by 15.8%. When high performance is required, the proposed architecture is able to achieve a speed-up of 12.6% with 13.1% energy reduction compared to the baseline, by introducing multi-cycle SFU operations. Future work includes supporting more complex patterns in the SFU, and exploring the further trade-offs between the complexity of the SFU and the energy efficiency of the processor architecture.

CHAPTER 4

TOWARDS AN ULTRA LOW-ENERGY SIMD PROCESSOR

The latest communication and multimedia standards, such as 4G wireless communication, H.264, and high-definition video, require ultra-high computational performance and ultra-high energy efficiency of end-user devices. While processors like Intel's Core i7 provide excellent computational performance, their energy consumption exceeds far beyond the energy budget of mobile terminals. Instead of these high-end processors, domain-specific streaming processors, particularly massively-parallel *Single Instruction Multiple Data* (SIMD) processors, are very popular candidates for SoCs within mobile devices. This is because: *i)* massive parallelism in streaming applications typically shows up as data-level parallelism (DLP) which can be inherently exploited by SIMD architectures, thus making SIMD the most common core execution engine on a stream platform; *ii)* SIMD is a low power architecture as it applies the same instructions to all *processing elements* (PEs). However, practice today is that power efficiency is still the main bottleneck in high performance embedded system design, especially for those ones that run on limited power sources like batteries. Moreover, the large power dissipation also worsens the SoCs' thermal and reliability issue, thus requiring expensive cooling techniques.

In this chapter, our progress in developing Xetal-Pro, an ultra low-energy SIMD processor, is presented. Xetal-Pro is a wide SIMD processor based on the previous Xetal-II processor from Philips, which ranks as one of the most computational-efficient (in terms of GOPS/Watt) processors available today [75]. Instead of power reduction, we focus on energy reduction, as energy/operation is the real metric for battery life. Xetal-Pro inherits many low-power peculiarities from the Xetal-II processor while removing serious shortcomings that may result in sub-optimal energy efficiency.

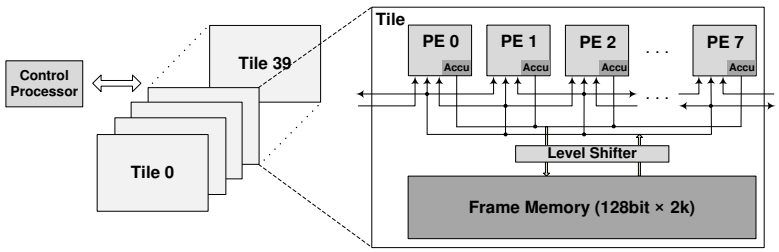


Figure 4.1 Block diagram of Xetal-II architecture.

The remainder of this chapter is organized as follows: Section 4.1 gives an overview of Xetal-II and analyzes its energy and performance by mapping different benchmark kernels. The energy breakdown of Xetal-II shows that the *frame memory* (FM) dominates the total energy consumption. Section 4.2 presents the challenge of applying ultra low V_{dd} scaling to the FM implemented with commercial SRAM. In Section 4.3, we explore alternative V_{dd} scalable FM. Unfortunately, these alternatives are also not effective and not efficient enough in lowering the energy of FM. To resolve this issue, the *hybrid memory system* (HMS) is introduced in Section 4.4. Section 4.5 discusses related work. Finally, Section 4.6 concludes our findings and discusses future work.

4.1 The Xetal-II Processor

The development of Xetal-Pro starts from exploring its predecessor Xetal-II [75, 119]. Xetal-II has been implemented in a 90nm CMOS process with 74 mm^2 die area. It consists of 320 PEs, and delivers a peak performance of 107 GOPS on 16-bit data at a running frequency of 84 MHz, with a power budget of 600 mW. Although Xetal-II already ranks as one of the most computational-efficient processors, it still cannot suffice the demanded computational efficiency for emerging mobile computing applications [6].

4.1.1 Overview of Xetal-II Processor Architecture

The block diagram of the Xetal-II processor is depicted in Figure 4.1. The *control processor* (CP) is a 16-bit, MIPS-like processor. The main task of the CP is to control the program flow, handle interrupts, communicate with the outside world and configure other blocks. The linear processor array contains 320 PEs and an integral 10M bit *frame memory* (FM). Layout and memory considerations necessitate partitioning of the linear processor array into *tiles* because: *i)* grouping all PEs and FM into one tile would result in a poor global layout with a

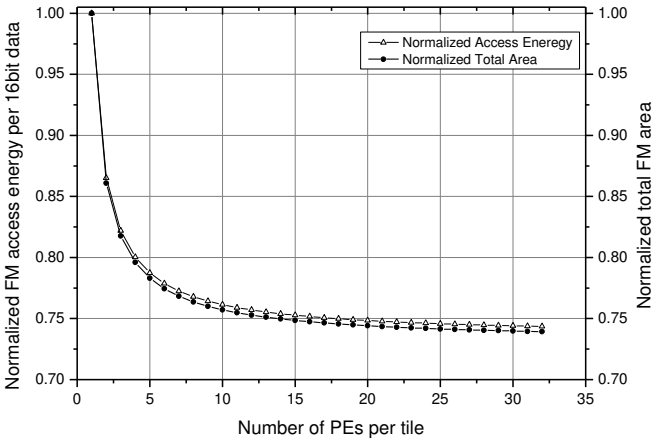


Figure 4.2 Number of PEs per tile vs. normalized FM access energy per 16-bit data and normalized total FM area.

very strange aspect-ratio and correspondingly large area; *ii)* commercial memory generators have limited maximum word width, which also disables this option. However, on the other extreme, using only one PE/FM per tile results in too many memories with corresponding addressing overhead and global/semi-global wiring overhead, in spite of the fact that it may provide advantages in programming flexibility for certain applications [120].

Apart from silicon area, our primary concern is energy consumption. The metric we used to decide the optimal number of PEs per tile is the energy/area efficiency of the shared FM. Different physical partitions affect both total area and energy consumption per unit data. Figure 4.2 shows the normalized FM energy per 16-bit data access and normalized total FM area under different partitions. We can see that including 8 PEs (power of two) per tile (thus, 40 tiles in total) is a good choice considering FM access energy efficiency, FM total area efficiency, and practical layout constraints.

Each PE has a two-stage pipeline and shares the instruction fetch and decode stage of the CP. Figure 4.3 shows the structure of the 16-bit PE, which is equipped with a local register (ACCU) for immediate result feedback and a flag register (FLAG) for guarded instruction execution. Each PE supports 16-bit ADD/SUB, MUL, MAC, logical operations, which can further be compounded with other operations (e.g., absolute, negative, etc.). All instructions are executed in a single cycle. The FM consists of 40 SRAM modules (each

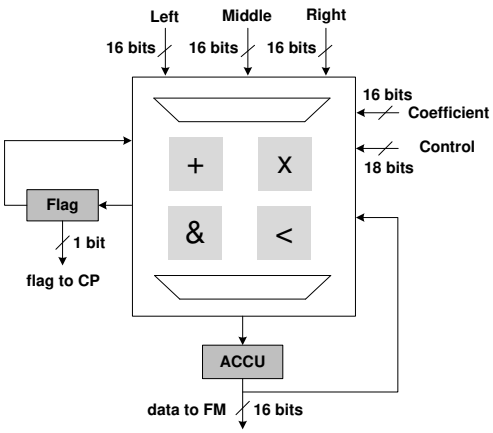


Figure 4.3 Structure of the 16-bit PE.

128bit×2048 [5]) with a pseudo-dual port interface to provide single cycle read and write accesses. This data memory can store both the frame data and the intermediate results. The relatively large capacity of the FM allows on-chip storage of multiple VGA frames or images with higher resolution, reducing in this way the off-chip traffic. The communication network between the FM and PEs enables PEs to directly access the memory (FM) data of its left and right neighbors. To provide better control of V_{dd} scaling, the tile is divided into logic and memory voltage domains, coupled with level-shifters. For simplicity, in the following sections of this chapter, PEs is used to refer to the logic part, including processing elements and communication network of the tile; FM is used to refer to the memory part of the tile.

4.1.2 Energy/Performance Analysis of Xetal-II

The proposed Xetal-Pro processor is designed in a 65nm CMOS process. As a reference for Xetal-Pro, we first migrate the Xetal-II processor from 90nm to 65nm technology. The logic part was synthesized with TSMC 65nm low-power (LP) SV_t CMOS digital standard cell library. LP process is superior over general-purpose (GP) process for medium/low end SoCs, because the LP feature can make leakage energy one to two orders of magnitude lower than with the GP feature. The V_t of our process is about 0.41 to 0.42 V. The SRAM was synthe-

[5] One SRAM module per tile. Since each tile consists of eight 16-bit PEs, the data width of the SRAM modules is 128 bit.

sized with a commercial low-power memory generator, which uses minimum size and HV_t devices of the same process technology for bit-cells to further constrain leakage energy without violating timing constraints. The impact of the long global wires for decoded instruction plus intermediate repeaters have been considered based on post-layout analog simulation results. The whole Xetal-II system can run at 125 MHz with 1.2 V voltage supply at 25°C room temperature at typical case, and offers 80 GOPS throughput (320 PEs in total, two operations per cycle per PE, and counting multiply and add operations only) with each PE processing one instr./cycle. The critical path is the FM read access plus the MAC operation within the PE.

To analyze the system energy breakdown, we use three representative application kernels which are typical benchmarks for SIMD processors. These kernels are: *i)* $N \times N$ non-separable filter; *ii)* $N \times N$ separable filter; and *iii)* YCbCr to RGB color-space conversion. Besides the popularity of these kernels, the other major reason for choosing them as our benchmark is that they have quite different data locality characteristics. The data in an $N \times N$ non-separable filter can be reused N^2 times while the data in an $N \times N$ separable filter is only reused $2N$ times. YCbCr-RGB conversion is a pixel-to-pixel operation, so there is no data sharing between pixels.

The mapping of three kernels on the reference Xetal-II processor is shown in Figure 4.4. Figure 4.5 describes the pseudo code of the non-separable filer kernel. The mapping of other two kernels can be described similarly. We take a VGA (640×480 pixels) image with interleaving factor of two as an example [6]. In the case of color conversion, the Y, Cb, and Cr values of a pixel are assumed to be stored in consecutive rows in the FM. Each PE can read the memory on its left (mem.l) and right (mem.r). The image height is represented by H (H is equal to 480 for VGA format).

Table 4.1 summarizes the energy breakdown of the reference Xetal-II processor when running the three benchmark kernels. In Table 4.1, the global wires do not include the intra-tile part, which is already included in the PEs. The summation of energy consumption percentage of both intra and inter-tile global wires takes around 5-7%.

[6] With interleaving factor of two, one image line is stored in two rows of the frame memory. Pixels at the odd (even) columns of the image line are stored in the odd (even) rows of the frame memory.

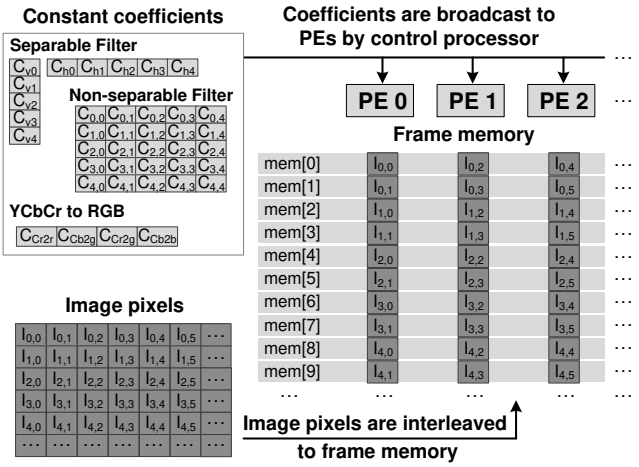


Figure 4.4 Mapping of YCbCr-to-RGB, non-separable filter, and separable filter on the baseline architecture.

Algorithm 1: The mem.l and mem.r represent the memory of a PE's left neighbor and right neighbor respectively. The image height, *H*, is equal to 480 for VGA format.

```
for i = 2 to (H - 3) do
    accu ← C0,0 × mem.l[2i-4];
    accu ← accu + C0,1 × mem.l[2i-3];
    accu ← accu + C0,2 × mem[2i-4];
    accu ← accu + C0,3 × mem[2i-3];
    accu ← accu + C0,4 × mem.r[2i-4];
    …  // other accu for output at mem[2H+2i]
    accu ← accu + C4,0 × mem.l[2i+4];
    accu ← accu + C4,1 × mem.l[2i+5];
    accu ← accu + C4,2 × mem[2i+4];
    accu ← accu + C4,3 × mem[2i+5];
    mem[2H+2i] ← accu + C4,4 × mem.r[2i+4];
    …  // accu for output at mem[2H+2i+1]
    mem[2H+2i+1] ← accu + C4,4 × mem.r[2i+5];
end
```

Figure 4.5 A 5×5 non-separable filter kernel mapped on the baseline architecture.

For the three kernels, the total energy is dominated by the energy of the tiles (i.e., PEs and FM). Compared with the 40 tiles, the CP and the global decoded instruction wires consume much less energy. Therefore, to effectively reduce the total energy, the tiles are the focus of our further exploration.

Table 4.1 Energy breakdown of the reference Xetal-II processor for three kernels. Using TSMC 65nm low-power library and at typical case, 1.2V, and 25°C.

Benchmark	PEs(%)	FM (%)	CP (%)	Global Wires (%)	Total (pJ/pixel)
5×5 non-separable filter	26.0	68.9	3.7	1.4	240.8
5×5 separable filter	23.5	71.9	3.3	1.3	106.5
YCbCr to RGB	14.7	81.3	2.9	1.1	109.9

4.2 Challenge of Ultra-Wide-Range V_{dd} Scaling

V_{dd} scaling is one of the most effective means to bring quadratic dynamic energy savings to standard-cell based logic, i.e., $E_{logic\_dynamic} \propto C_{load}V_{dd}^2$, where C_{load} is the loading capacitances including both gate and interconnection wire capacitances. The V_{dd} scaling range of commercial processors is normally limited to about 2/3 of nominal supply due to two fundamental problems at an ultra-low V_{dd}: *i)* severe throughput degradation; *ii)* high yield loss in the presence of process variations. To solve the first problem, the nature of the massive parallelism of the Xetal family can be used to compensate the throughput degradation, as will be discussed soon. To mitigate the second problem, techniques at different design levels can be used, such as structural duplication [121], Razor [122, 123], body biasing [124, 125], cell resizing [126].

Compared to pure logic, V_{dd} scaling is even more difficult when applied to SRAM. First, the rapidly deteriorating read/hold static noise margin (SNM) of bit-cells causes severe reliability issues. A very small amount of injected noise can cause the bit-cell's state to flip [35]. Thus, all commercial SRAMs achieving high density strictly prohibit operating below 0.7 V. Second, SRAM's energy cannot scale quadratically with V_{dd}. SRAM bit-cells' energy, which usually dominates total SRAM's energy, can be approximated as $E_{bitcell} \propto C_{bitline}V_{dd}V_{swing}$ in a single cycle. $C_{bitline}$ is the loading capacitance on a SRAM bitline. V_{swing} is the bitline swing, which must exceed a minimum magnitude required by sense-amplifiers to make correct decoding. V_{swing} cannot scale linearly with V_{dd}. Therefore, bit-cells' energy and total SRAM's energy only scale sub-quadratically with V_{dd}, while the energy of other SRAM components like sense-amplifiers, wordline and bitline drivers, address decoders can scale equally well as logic. Third, SRAM's speed degrades even faster with V_{dd} scaling, compared to that of pure logic [36, 37]. This implies that SRAM becomes the system performance bottleneck if both SRAM and logic scale to the same ultra-low V_{dd}.

While V_{dd} scaling lowers the dynamic energy, the leakage energy increases due to a prolonged cycle time. As a result, there is an energy-optimal V_{dd} point where the total energy is minimal. Pursuing a lower V_{dd} than this optimal V_{dd} point makes leakage energy dominate the total energy, hence rendering no additional energy benefits. To find out the optimal V_{dd} point and the impact of ultra-wide-range V_{dd} scaling on our design, three steps are carried out, which are listed below:

Step 1 *Standard-cell characterization*: we first characterized the TSMC 65nm low-power SV_t CMOS digital standard cell library under different supply voltage. By simulating an inverter in SPICE simulator with typical-typical case, 25°C room temperature, and different supply voltage (1.2V ~ 0.1V), the average drive current (I_{drive}) and the sub-threshold leakage current (I_{sub_leak}) at different supply voltages are obtained. This information, i.e., a set of tuples ($V(n)$, $I_{drive}(n)$, $I_{sub\_leak}(n)$), where $0.1V \le n \le 1.2V$, is used to calculate the scaling factors in *Step 3*.

Step 2 *Obtain latency (T), dynamic power (P_{dyn}), and leakage power (P_{leak}) of each component at nominal voltage*: We looked at three components, i.e., PEs (logic), memory tiles (SRAM), and long wires. The latency and dynamic/leakage power of PEs are obtained by post-synthesis simulation at nominal voltage, typical case and 25°C. For memory tiles, these data are obtained with a commercial low-power SRAM generator, which uses minimum size and HV_t devices of the same process technology for bit-cells. The impact of the long global wires for decoded instruction plus intermediate repeaters are considered based on post-layout analog simulation results. This nominal voltage information is also used in *Step 3*.

Step 3 *Estimate latency, dynamic power/energy, and leakage power/energy at different supply voltages*: With the data obtained from *Step 1* and *Step 2* as input, we can estimate latency, power/energy consumption of each component at different supply voltages with the following analytical models.

(1) Estimate the effective capacitor, C_{eff}, with the input of *Step 2*:

$$C_{eff} = \frac{P_{dyn}(1.2V)}{F(1.2V) * V_{dd}^2(1.2V)}, \text{ where } F = 1/T \text{ is the frequency} \quad \text{(eq. 1)}$$

(2) Estimate the leakage current at nominal voltage, $I_{leak}(1.2V)$, with the input of *Step 2*:

$$I_{leak}(1.2V) = \frac{P_{leak}(1.2V)}{V_{dd}(1.2V)} \qquad (eq.\ 2)$$

(3) Estimate $T(n)$, the latency under different supply voltage n, with the scaling factors obtained in *Step 1*:

$$T(n) = T(1.2V) * \frac{V(n) / I_{drive}(n)}{V(1.2V) / I_{drive}(1.2V)} \qquad (eq.\ 3)$$

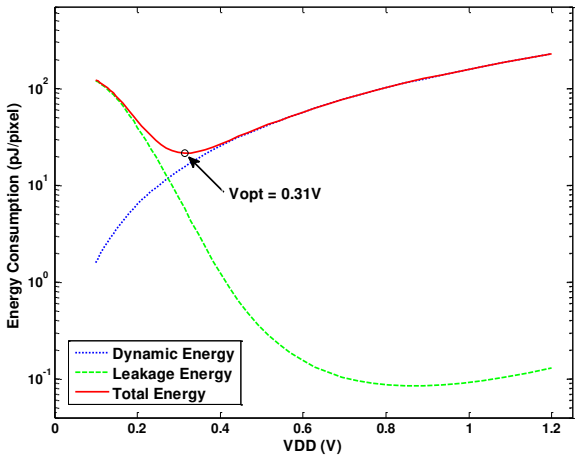
(4) Estimate $I_{leak}(n)$, the leakage current under different supply voltage n, with the scaling factors obtained in *Step 1*:

$$I_{leak}(n) = I_{leak}(1.2V) * \frac{I_{sub\_leak}(n)}{I_{sub\_leak}(1.2V)} \qquad (eq.\ 4)$$

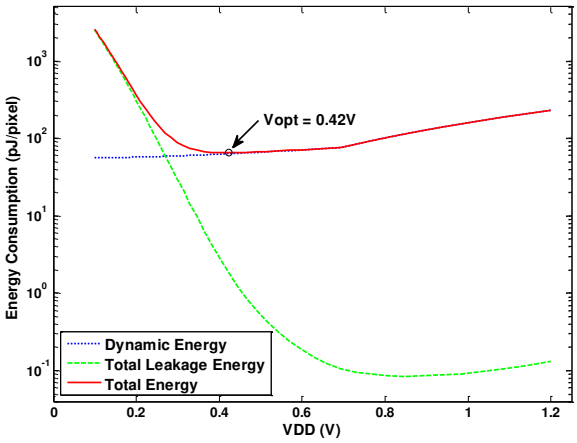
(5) Estimate $E_{dyn}(n)$, the dynamic energy under different supply voltage n:

$$E_{dyn}(n) = C_{eff} * V_{dd}^2(n) \qquad (eq.\ 5)$$

As shown in Table 4.1, the tiles (PEs + FM) are the most energy-consuming part of the design at the nominal voltage. Our further exploration then focuses on the tiles. The energy consumption of processing one pixel when applying the aforementioned 5×5 non-separable filter kernel (25 instr. in total) is used as an example. Figure 4.6 (a) depicts the energy consumption curve under different supply voltages. Note that here we unrealistically assume that the SRAM can be scaled to sub-threshold as well as the standard cells (i.e., scale SRAM together with PEs using the scaling factor obtained in *Step 1*), just to show the lower bound on energy reduction by V_{dd} scaling. The optimal point in this case occurs at V_{dd} = 0.31 V. At this point, the tile consumes 21.4 pJ/pixel, leading to a ten times reduction of the energy consumption ideally achievable, compared to operating at nominal 1.2 V.



(a)



(b)

Figure 4.6 V_{dd} versus tile (8 PEs + memory) energy consumption when processing one pixel with a 5×5 non-separable filter kernel: (a) assuming ideal SRAM voltage scaling; (b) SRAM only scales down to 0.7 V.

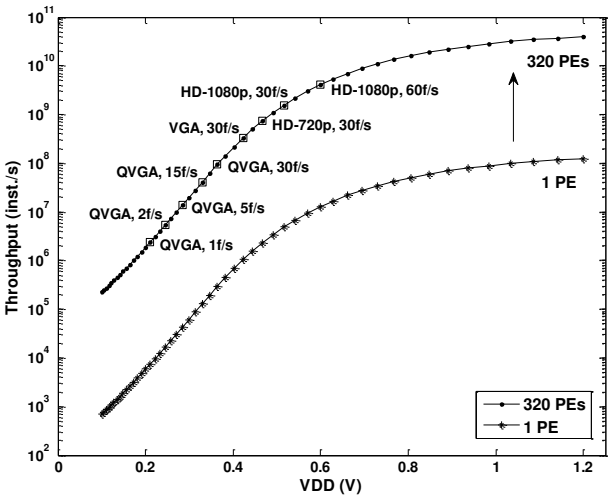


Figure 4.7 Impact of V_{dd} scaling (ideal) on system throughput of 1 PE (lower curve) and 320 PEs (upper curve). The squares on the upper curve indicate the supported resolution and frame rate with 320 PEs when executing a 5×5 non-separable filter kernel.

However, with V_{dd} scaling, the maximum frequency each PE can achieve also decreases dramatically, hence causing severe performance degradation, as shown by the lower curve of Figure 4.7. Fortunately, with 320 PEs processing in parallel, this performance loss can be largely compensated. This shows the unique advantage of massively-parallel SIMD architecture to outperform other processor architectures in energy efficiency and throughput. The upper curve of Figure 4.7 depicts the supported resolution and frame rate at different V_{dd} when running the 5×5 non-separable filter kernel by 320 PEs. Above 0.6 V and above 0.42 V, HD-1080p (1920×1080) 60 frames/s and VGA (640×480) 30 frames/s can be supported in real time respectively. Even when V_{dd} goes down to about 0.33 V, we can still run many low-end applications, such as QVGA (320×240) at 15 frames/s [7].

[7] As indicated in Figure 4.5, it requires 25 instructions to implement the 5×5 non-separable filter kernel on VGA resolution or higher (interleaving factor ≥ 2). However, QVGA format requires 5 additional instructions, as not all of the 5×5 pixels are directly accessible.

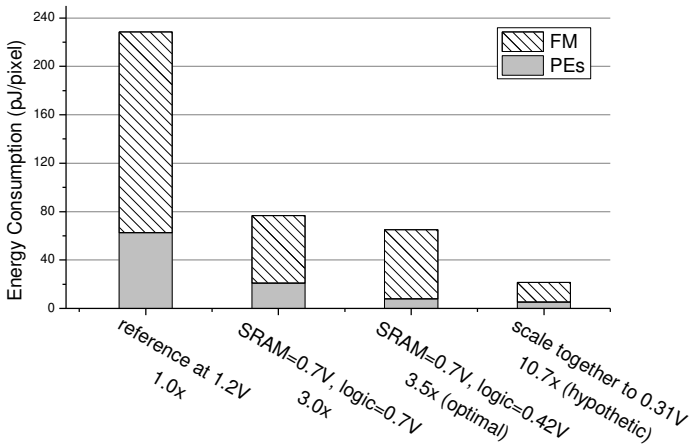


Figure 4.8 Tile (Xetal-II reference processor) energy consumption for different V_{dd}.

Figure 4.6 (a) presents only the ideal lower energy consumption bound of the reference processor. Because the V_{dd} of commercial 6T SRAM's cannot scale well below 0.7 V, Figure 4.6 (b) shows a more practical V_{dd} scaling result when SRAM is assumed to only scales to 0.7 V. In this scenario, the minimal energy consumption (65.1 pJ/pixel) is obtained when the logic part is scaled to 0.42 V (i.e., two voltage domains, SRAM at 0.7 V and standard logic at 0.42 V). Compared to operating at the nominal voltage supply, the energy reduction is only a factor of 3.5, far behind the ten times ideally achievable reduction. It should be noted that in this case about 88% of the total energy is consumed by the FM.

The tile energy consumption at different V_{dd} is compared in Figure 4.8. We can see that even when PEs' V_{dd} is aggressively scaled to sub/near threshold, it only reduces an extra 15% of the energy compared to that when both PEs and SRAM are supplied at 0.7 V. Thus, unless the FM can also scale further, it does not make too much sense to aggressively scale the V_{dd} of the standard-cell (PEs) part due to the low energy gain and high performance loss. This conclusion holds true for other kernels.

4.3 Exploration of V_{dd} Scalable FM

As shown from the above analysis, commercial SRAM module creates a big obstacle for V_{dd} scaling. To resolve this challenge and to further reduce the total energy consumption of the Xetal-II SIMD processor, one potential solution is to look for a V_{dd} scalable FM. Recent MIT low-power dual-port SRAM [127, 128]

Table 4.2 Tile energy consumption with MIT 10-T SRAM realization for FM.

Benchmark	pJ/pixel @ 1.2V	Compare [a]	pJ/pixel @ optimal V_{dd} [b]	Compare [c]
5×5 non-separable filter	265.0	16.0% ↑	49.6	1.3× ↓
5×5 separable filter	118.3	16.0% ↑	21.4	1.4× ↓
YCbCr to RGB	124.5	18.0% ↑	21.1	1.5× ↓

[a] Compare to the energy consumption with commercial SRAM realization for FM (at 1.2 V).
[b] FM and PEs are scaled to different sub/near threshold voltages, to reach an optimal combination for energy efficiency.
[c] Compare to the energy consumption with commercial SRAM realization for FM (after optimal scaling).

and the standard-cell synthesized memory are two possible choices. The MIT work achieves ultra-low V_{dd} operation by adding extra devices within the bit-cell. The standard-cell based memory can also approach ultra-low V_{dd} because: *i)* they are not limited by density constraints and composition style, so transistor up-sizing, buffer insertion and logic re-construction (which optimizes boolean expressions) can be used freely during synthesis; *ii)* they can employ hierarchical topology, which prevents high fan-out and relieves shared architecture.

The V_{dd} of MIT 10-T low-power SRAM can be scaled to below 0.4 V. However, it has several drawbacks in our case. First, it occupies 66% more cell area compared to the commercial differential 6-T SRAM [128]. When the FM is implemented with 6-T commercial SRAM, the ratio between SRAM bit-cell array area and SRAM total area is 7/10. If this FM is realized by the 10-T SRAM, more than 30% additional area is needed for each tile. Second, it consumes more access energy at nominal voltage. Besides, the high leakage power (about 100 μW at 1.2 V) also prevents it from scaling to very low V_{dd}, as the leakage energy increase will quickly counteract the reduction of the dynamic energy. Table 4.2 presents the energy consumption when FM is realized by the MIT 10-T SRAM, in comparison with commercial SRAM realization for FM. Third, the MIT SRAM is much slower. The reported maximal speed at nominal voltage is 2.5 times slower than the commercial 6-T SRAM with the same word width and depth that we are using. This severely degrades the performance at both nominal and scaled voltage. The maximum energy gain it can reach is rather small in contrast to its high area, performance and reliability overhead. So we conclude that, the MIT 10-T memory is not applicable in our case. These problems also exist for other sub-threshold SRAM works [127, 129].

The standard-cell realization of large on-chip SRAM is also not applicable. According to our synthesis result, although it can be faster than commercial SRAM, it consumes even more energy and area than the MIT 10-T SRAM at nominal voltage. This limits the standard-cell based memory designs to only very small arrays. Therefore, to reach our goals of ultra-low-energy, ultra-wide-voltage-range, and medium-to-high-throughput SIMD processor, architecture changes are required, in particular with respect to the memory hierarchy.

4.4 Hybrid Memory System (HMS)

Since V_{dd} scalable FM is not applicable in our Xetal-Pro design, in this section we propose a hybrid memory system (HMS) to exploit the often available data locality and reduce the non-local memory traffic and to enable further V_{dd} scaling. The "hybrid" implies two things: *i)* a hybrid memory architecture consisting of an ACCU register, a scratchpad memory (SM), and the FM; *ii)* a hybrid realization consisting of sub-threshold SM and super-threshold FM.

4.4.1 HMS Scheme

The HMS is shown in Figure 4.9. Within the proposed HMS, we have three types of characterized memories to preserve the data: *i)* ACCU register for short-term data storage; *ii)* SM for intermediate-term data storage; and *iii)* FM for long-term data storage. Both the FM and the SM are directly accessible by the PE as the source/destination operands, which means that they are at the same level of the memory hierarchy. This design choice not only increases the flexibility of memory access, but also reduces the penalty when little data locality can be exploited by the SM. Compared to FM, SM consumes much less energy per access due to its much smaller size. The SM supports all the addressing modes of FM, which makes it very friendly to access. For the low-level image/video processing (target domain of SIMD), most applications contain spatial data locality. When no data locality is exploitable, the SM can be bypassed and clock-gated with only a few μW leakage overhead. It should be noted that the critical path of the system is hardly changed, i.e., FM read access plus PE operation. In addition, when coupled with index addressing, the SM can also be used as a look-up table for index based algorithms.

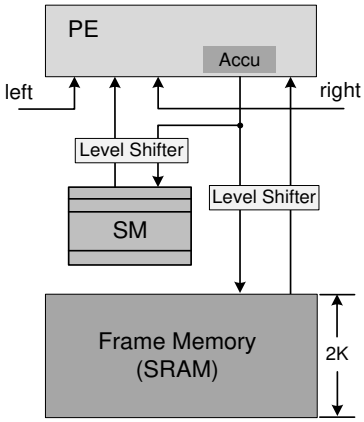


Figure 4.9 Proposed hybrid memory system.

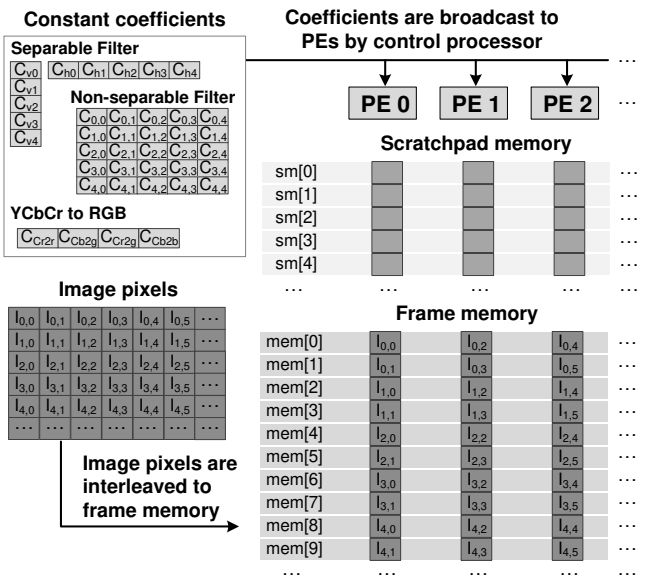


Figure 4.10 Mapping of YCbCr-to-RGB, non-separable filter, and separable filter on the proposed architecture.

Algorithm 2: The sm.l and sm.r represent the scratchpad memory of a PE's left neighbor and right neighbor.

```
for i = 4 to (H - 1) do
    // load one image row into scratchpad memory
    sm[8] ← mem[2i];
    sm[9] ← mem[2i+1];
    // apply 5 × 5 convolution
    accu ← C0,0 × sm.l[0];
    accu ← accu + C0,1 × sm.l[1];
    accu ← accu + C0,2 × sm[0];
    accu ← accu + C0,3 × sm[1];
    accu ← accu + C0,4 × sm.r[0];
    …  // other accu for output at mem[2H+2i]
    accu ← accu + C4,0 × sm.l[8];
    accu ← accu + C4,1 × sm.l[9];
    accu ← accu + C4,2 × sm[8];
    accu ← accu + C4,3 × sm[9];
    mem[2H+2i] ← accu + C4,4 × sm.r[8];
    …  // accu for output at mem[2H+2i+1]
    mem[2H+2i+1] ← accu + C4,4 × sm.r[9];
    …  // remaining 4 image rows in the unrolled code
end
```

Figure 4.11 A 5×5 non-separable filter kernel mapped on the proposed architecture (unrolling factor is 5).

The SM in Xetal-Pro is decided to be dual-ported with 128-bit word width and 32 entries. The reasons that we choose this relatively large number of entries are: *i)* to enable more applications with large working windows (e.g., motion estimation, etc.) or higher resolutions (>VGA) to fully exploit their data locality; *ii)* to demonstrate that even with such a (relatively) large size, we can still reach more than ten times energy gain. In Section 4.4.4, we will further justify this choice in details. The 32-entry SM (commercial SRAM realization) adds about 15% area to the tile. Fewer entries can slightly reduce the area overhead and energy consumption, but fewer applications can benefit from this HMA. The programming model of the proposed architecture is also slightly different since there is an extra memory (SM) to utilize. The mappings of the three kernels on Xetal-Pro are shown in Figure 4.10. The mapping of the non-separable filter kernel on the architecture with HMS is shown in Figure 4.11. The mapping of the other two kernels can be described similarly.

Table 4.3 Comparison of PE instructions of Xetal-II and Xetal-Pro (instruction format: *op dest, src 1, src 2*).

No.	Xetal-II				Xetal-Pro			
	op	*dest*	*src 1*	*src 2*	*op*	*dest*	*src 1*	*src 2*
1	*OP*	ACCU	FM	COEF/ACCU	*OP*	ACCU	SM	COEF/ACCU
2	*OP*	FM and ACCU	FM	COEF/ACCU	*OP*	SM and ACCU	SM	COEF/ACCU
3	*MV*	ACCU	FM	-	*OP*	FM and ACCU	SM	COEF/ACCU
4	*MV*	FM and ACCU	FM	-	*OP*	ACCU	FM	COEF/ACCU
5					*OP*	SM and ACCU	FM	COEF/ACCU
6					*OP*	FM and ACCU	FM	COEF/ACCU
7					*MV*	ACCU	SM	-
8					*MV*	SM and ACCU	SM	-
9					*MV*	FM and ACCU	SM	-
10					*MV*	ACCU	FM	-
11					*MV*	SM and ACCU	FM	-
12					*MV*	FM and ACCU	FM	-

4.4.2 Instruction Set Extension

Compared to Xetal-II, the instruction format of Xetal-Pro is almost the same because the SM has the same addressing modes as the FM and they are mapped to a continuous memory space. However, since the source operand can be read from and the result can be sent to one extra location (SM), the total number of instruction types increases from user's point of view. By categorizing the instructions based on: *i)* what the data source (*src*) and destination (*dest*) are; *ii)* if data is operated (*OP*) or only moved (*MV*) to a different location, the main differences of PE instructions between the two architectures are described in Table 4.3.

4.4.3 Exploration of HMS Implementation

ACCU, SM, and FM are the three components of the proposed HMS. Since the large on-chip FM cannot be implemented with V_{dd} scalable memory, it is therefore implemented with commercial low-power and high-density SRAM. Obviously, the ACCU register is most proper to be implemented by standard cells. The remaining question is how to implement the SM. In this section, we explore the implementation choices for the SM.

Taking the 5×5 non-separable filter as an example, Figure 4.12 (a) shows the energy breakdown of the proposed architecture at 1.2 V when the SM is realized by the commercial SRAM. Although the Xetal-Pro architecture requires

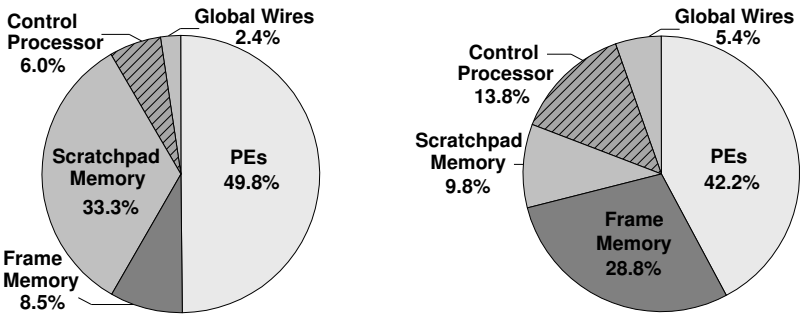


Figure 4.12 System energy breakdown of the proposed architecture: (a) at 1.2 V, and SM is realized by the commercial SRAM (151.9 pJ/pixel); (b) sub-threshold SM in combination with super-threshold FM (22.6 pJ/pixel), CP and global wires are only scaled to 0.7 V.

one extra instruction to implement this kernel compared to Xetal-II, the energy consumption per pixel (tile part) at nominal voltage is still 1.6 times less than that of the Xetal-II. Let us assume that commercial SRAM is used as the SM. Figure 4.13 (a) shows that, after V_{dd} scaling, a total of 6.8 times reduction can be reached at the optimal point where FM = 0.7 V, SM = 0.7 V, and PE = 0.42 V. At this point Xetal-Pro delivers a throughput of 0.88 GOPS. However, it should be noted that more than half of the energy is consumed by the SM at this point. Thus, further energy reduction needs an SM with better V_{dd} scalability.

Similar to the analysis we did for FM in Section 4.3, two other possible choices for the SM, i.e., the MIT low-power SRAM and the standard cells, are investigated, both of which have better V_{dd} scalability than commercial SRAM realization. According to our synthesis results, the standard-cell realization of the 128bit×32 dual-port memory is the best in terms of energy efficiency and speed. Thus, we propose a hybrid realization of our HMS, i.e., a sub-threshold standard-cell based SM in combination with super-threshold commercial SRAM based FM. Figure 4.13 (b) shows the energy consumption of this proposed architecture. After scaling, a total of 12.5 times energy saving (tile part) can be reached.

Figure 4.12 (b) shows the system energy breakdown when the minimal energy consumption is achieved. Note that we only conservatively scale CP and global wires (together they consume 5% of the total system energy at nominal) to 0.7 V. Compared to Xetal-II operating at nominal voltage, Xetal-Pro

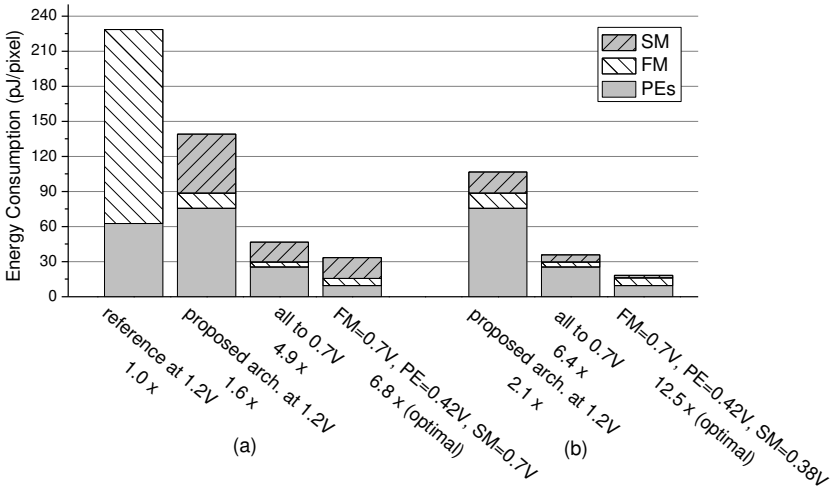


Figure 4.13 Tile (proposed architecture) energy consumption for different V_{dd}: (a) commercial SRAM realization for SM; (b) standard cell realization for SM.

gains more than 10 times energy reduction (240.8 pJ/pixel vs. 22.6 pJ/pixel), i.e., < 0.5pJ/16-bit op, while still delivering a throughput of 0.69 GOPS with 1.08 MHz frequency, sufficient to execute a 5×5 convolution kernel on VGA at 43 frames/s. Table 4.4 compares the tile part energy consumption between the reference Xetal-II processor and the Xetal-Pro processor. Even for the YCbCr to RGB conversion kernel which has little locality to be exploited, 1 pJ/16-bit op is achieved. In Figure 4.14, the *Intrinsic Computational Efficiency* (ICE) graph highlights the energy efficiency advantage of Xetal-Pro over that of earlier well-known works [8].

It is worth introducing the implementation of level-shifter (LS) in the HMS. Different from conventional LS which converts from 2/3 nominal supply to full nominal supply, the LS in Xetal-Pro should be capable of converting between signals from a sub-threshold VDDL supply domain (e.g., 0.4 V) to a super-threshold VDDH supply domain (e.g., 0.7 V). A two-stage LS, as shown in Figure 4.15 is proposed in this work to deliver robust, fast, and energy-efficient operation. Each stage uses a normal cross-coupled differential inverter. For low voltage inputs, the first stage handles the majority of the up-conversion, and the second stage is mainly to restore and re-shape the final output signal. To

[8] In the ICE curve, only programmable multiply and add operations are counted. Other operations, e.g., shift, dedicated adder tree, etc. are not counted. The energy of 8-bit and 16-bit operations are linearly scaled to 32-bit operations.

Table 4.4 Tile Energy Comparison between the Reference Xetal-II Processor and the Xetal-Pro Processor.

Benchmarks	Xetal-II (65nm, 125MHz@1.2V)			Xetal-Pro (65nm, 125MHz@1.2V)		
	instr./pixel	pJ/pixel @ 1.2V	pJ/pixel @ optimal V_{dd} [a]	instr./pixel	pJ/pixel @ 1.2V	pJ/pixel @ optimal V_{dd} [b]
5×5 non-separable filter	25	228.6 (1.0×)	65.1 (3.5×↓)	26	106.6 (1.0×)	18.3 (12.5×↓)
5×5 separable filter	10	101.6 (1.0×)	29.5 (3.4×↓)	11	51.7 (1.0×)	10.1 (10.1×↓)
YCbCr to RGB	9	105.4 (1.0×)	32.6 (3.2×↓)	9	63.9 (1.0×)	16.8 (6.3×↓)

[a] FM is scaled to 0.7 V, PEs are scaled to the sub-threshold region.

[b] FM is scaled to 0.7 V, PEs and SMs are scaled to the sub-threshold region.

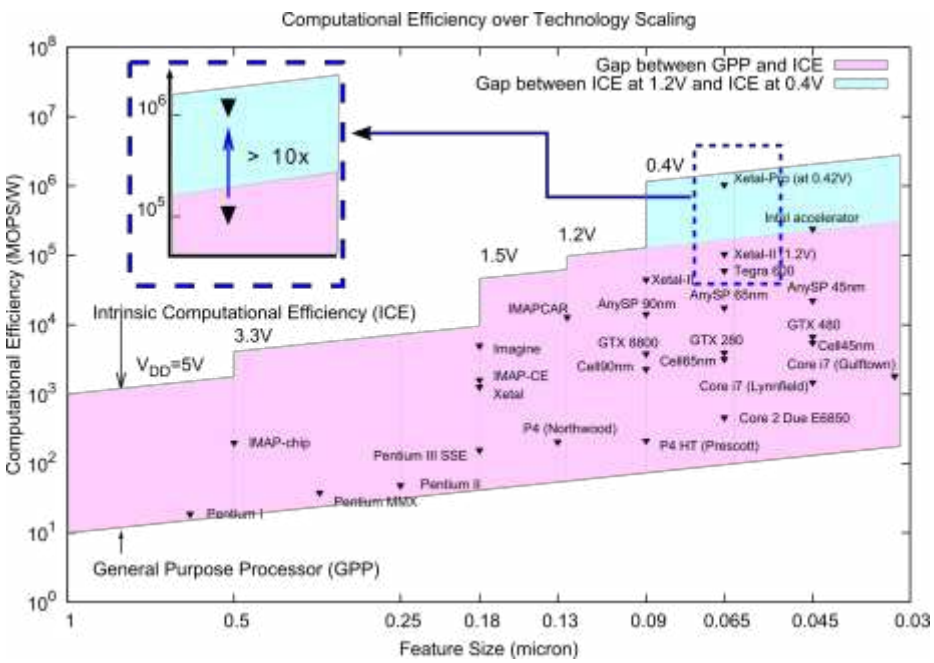


Figure 4.14 ICE graph annotates the energy efficiency of Xetal-Pro and other well-known works.

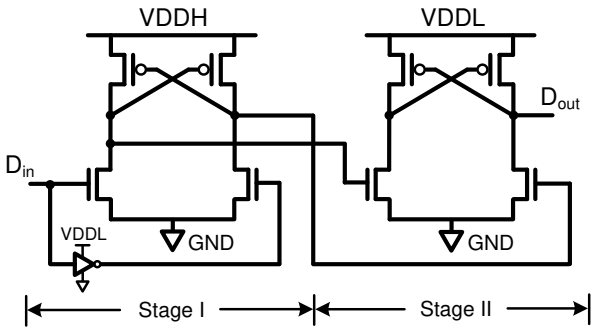


Figure 4.15 Two-stage level shifter.

ensure the correct functioning of the LS, the NMOS and PMOS devices are carefully sized so that the cross-coupled NMOS pull-down devices can overpower the PMOS pull-up devices in the presence of process variability. In addition, LV_t devices are used in the LS to enhance operation speed and reliability. LSs are small circuits in the overall system, so the impact of increased leakage by using LV_t devices is negligible in such a big system (the leakage power increase of each LS is in pW range, whereas the total system power is in mW range). The transient response to pull-up an input signal from 0.4 V to 0.7 V is 2 ns. When VDDL is in the super-threshold region, the transition delay can be less than 100 ps. Since the LS is designed by another project member, it will not be further elaborated in this thesis.

4.4.4 Data Locality Analysis for HMS

To achieve ultra low energy, domain specific processors often exploit the locality of typical kernels in their application domains. In the case of Xetal-Pro, the size of scratchpad memory is a crucial factor for energy consumption. Scratchpad memory of small size may not accommodate the potential locality of the kernels, which causes spilling to the energy consuming frame memory. On the other hand, oversized scratchpad memory, which is more than enough for the potential locality of the kernels, consumes more energy but cannot further reduce the access to frame memory. In order to choose the optimal size of scratchpad memory for energy consumption, experiments are performed to analyze the locality of the kernels.

In Figure 4.16, three kernels are mapped onto architectures with different sizes of scratchpad memory. The energy is obtained at nominal voltage (1.2 V).

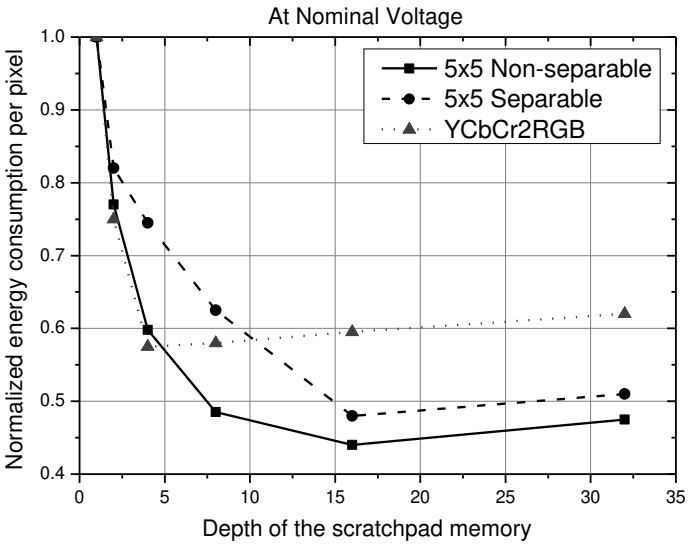


Figure 4.16 Normalized energy per pixel for different sizes of SM.

The energy per pixel of each benchmark is normalized to the energy per pixel of that benchmark on the baseline architecture, which has no scratchpad memory. For each size of scratchpad memory, the kernels are heavily optimized for energy consumption.

According to Figure 4.16, the optimal size of scratchpad memory for energy consumption of each benchmark matches the potential locality of the benchmark. The optimal sizes of scratchpad memory for the benchmarks are not the same. Scratchpad memory of 16 entries seems to be the optimal size on average for the three kernels. However, image processing applications often contain multi-pass filters, which has larger potential locality than a single filter. And applications with larger working windows or higher resolutions require a scratchpad memory of more entries too. To accommodate these cases, a scratchpad memory of 32 entries is used in Xetal-Pro. Based on our experiments, the energy consumption of using 32 entries scratchpad memory consumes less than 5% extra energy compared to that of using 16 entries. Therefore, this decision is justified.

4.5 Related Work

The related work is categorized into three subsections: *i)* sub-threshold designs; *ii)* scratchpad memory; and *iii)* SIMD processors.

4.5.1 Sub-threshold Designs

An emerging trend for lowering energy of digital processors is to scale V_{dd} to the sub/near threshold region, which brings not only quadratic dynamic power savings, but also super-linearly reduced leakage current. Many prototype chips, which can function in the sub-threshold region, have been presented in recent years. These chips include a 180 mV FFT processor in 180nm CMOS process [130], a 256 Kbit 10-T dual-port SRAM in 65nm CMOS process [128], which was later improved to 8-T dual-port SRAM [127]. In [129] a single-end sub-threshold SRAM has been developed for extremely low speed applications. A 130nm and a 180nm CMOS sensor node processors are presented in [131] and [132], respectively. A TI-MSP430 based DSP processor with integrated DC-DC converter in 65nm CMOS is presented in [133]. In the prototype chip called *SubJPEG* [134], a 65nm CMOS 8-bit JPEG co-processor is presented. *SubJPEG* is equipped with four parallel DCT-Quantization engines and delivers 15 *fps* VGA processing at about 0.4 V. Intel also announced its 45nm CMOS sub-300mV 4-Way sub-word parallel processors [135].

4.5.2 Scratchpad Memory

Using scratchpad memories can help reduce the traffic to higher memory levels significantly when applications show substantial locality [136]. For example, a stream register file (or memory) as used in the Imagine architecture [137] can provide high performance with low energy consumption for streaming applications.

4.5.3 SIMD Processors

Other than Xetal-II, IMAPCAR [138] from NEC is another very successful SIMD processor. It includes 128 PEs and each PE is a 4-way 16-bit VLIW with its own 2 KB on-chip memory. It achieves 100 GOPS within a power budget of 2 W. The IMAPCAR differs from Xetal-II in the VLIW PEs, the per-PE register files, and the index addressing to on-chip memory. Compared to Xetal-II, the indirect addressing capability of IMAPCAR [138] enables access to different frame memory locations by PEs within the same instruction. While this feature facilitates parallelization of some image tasks containing irregular memory access, it leads to increased energy consumption for most applications with predominantly regular memory accesses. Its successor, IMAPCAR-II [139], added the support of switching between SIMD and MIMD mode, which however is out of the scope of this work. The AnySP [74] architecture also proposes

a configurable SIMD datapath as the core execution engine in the stream platform. In the annotated ICE graph (Figure 4.14), GPUs like NVIDIA GTX8800 have different application domains. The sub-word parallel processors [135] also benefit from exploiting SIMD parallelism. However, they are not massively-parallel processors for very low-energy applications.

To the best of our knowledge, no previous work has analyzed the impact of aggressive V_{dd} scaling on the memory hierarchy in the context of an ultra-low energy massively-parallel SIMD.

4.6 Summary

In this chapter, we presented our progress in developing the Xetal-Pro processor. Xetal-Pro is the first work to combine wide-range V_{dd} scaling with highly parallel SIMD architectures. While aggressive V_{dd} scaling leads to ultra low energy/op, it also causes severe throughput degradation. Xetal-Pro compensates this degradation by its massively-parallel nature. The predecessors in the Xetal family, such as the Xetal-II, include a large on-chip frame memory (FM), which cannot scale well to an ultra low V_{dd} hence creating a big obstacle to increase energy efficiency. Therefore, we proposed a hybrid memory system which not only exploits the often available data locality, but also enables further V_{dd} scaling. Compared to the reference design, i.e., Xetal-II migrated to 65nm CMOS technology, the new architecture provides up to two times energy efficiency improvement even at the nominal operating voltage when delivering the same amount of throughput. At the ultra low-energy mode, more than 10 times energy reduction is achievable, while still delivering a throughput of 0.69 GOPS. The preliminary result makes Xetal-Pro a very promising building block in Multi-Processor System-on-Chips (MPSoCs) for future low-energy embedded streaming computing.

Currently, we are busy with exploring the proper micro-architecture of the processing engines (PEs) and the communication network among PEs, aiming at improving both performance and energy-efficiency. As another important component of our low-energy SIMD framework, the corresponding compiler is also under construction. In the future work, we would like to pay more attention to the impact of process variation. Fault-tolerance techniques at different design levels will be introduced into our framework.

CHAPTER 5

EFFICIENT COMMUNICATION SUPPORT FOR STREAMING APPLICATIONS

The use of hardwired or weakly programmable accelerator IPs in heterogeneous *Multi-Processor System-on-Chips* (MPSoCs) can greatly improve the performance and energy efficiency of implementations of streaming applications [6]. However, implementing applications on such a system is much more difficult compared to implementing applications on a system that contains only programmable cores. When mapping an application onto an MPSoC that contains accelerators, several problems have to be solved: *i*) how to generate accelerator IPs for the application; *ii*) how to integrate these IPs into an MPSoC; and *iii*) how to predict performance/resource usage at design time. The first problem can be handled through the use of IP libraries or high level synthesis tools [70, 71]. In this chapter, we focus on solving the other two issues, i.e., efficiently integrating IPs into an MPSoC and mapping applications on these MPSoCs using a predictable design flow. Here, *predictable* means that system properties, such as throughput and latency, can be *conservatively* analyzed at design time, and guaranteed at run time.

We propose a predictable hardware module called *communication assist* (CA), which serves as an abstract and unified communication interface between a generic IP and the interconnect in an MPSoC. Communication is separated from computation through the proposed CA. The benefits of introducing such a hardware module are: *i*) Improved system performance because the proposed CA enables overlapped communication and computation; *ii*) Capable of providing IP cores high data-access bandwidth, which is usually a bottleneck in the implementation of streaming applications; *iii*) Supports design-time resource and timing analysis using *Synchronous Data Flow* (SDF) analysis techniques. This is essential when designing systems that require predictable timing behavior; *iv*) Simplified interface design of accelerator IPs, as complex communication functionality is offloaded to the CA, which requires only a one

time design effort; and *v*) Improved IP reusability, as the proposed CA interface acts as a uniform interface between many different types of IP blocks and interconnects.

Based on the proposed CA design, we present a heterogeneous multiprocessor platform template for streaming applications. The template is used in a predictable design flow, *MAMPS+*, which uses SDF graphs for design time analysis. As a case study, we map the complete high-speed vision processing pipeline of an industrial application, *Organic Light Emitting Diode* (OLED) screen printing, onto one instance of the proposed platform. The result demonstrates that system design and analysis effort is greatly reduced with the proposed CA-based design flow.

The remainder of this paper is organized as follows. Section 5.1 explains how inter-core communication can be modeled using SDF. Section 5.2 discusses design requirements for a CA. In this section, we also introduce our communication assist along with its SDF model. Based on the CA design, we introduce a heterogeneous MPSoC template in Section 5.3, as well as a complete design flow for designing heterogeneous MPSoCs with both programmable cores and accelerator IPs. In Section 5.4, an industrial application is mapped to the proposed platform to demonstrate its effectiveness. Section 5.5 discusses related work. Finally, Section 5.6 concludes our findings and discusses future work.

5.1 Inter-Core Communication Modeled with SDF

Synchronous Data Flow (SDF) is a model of computation commonly used to model streaming applications [55]. There are many analysis algorithms for SDF that can be used to analyze at design time the throughput, latency, and buffer size requirements of applications modeled with an SDF graph [56-58]. When an SDF graph is mapped onto an MPSoC, it seems natural to implement an inter-core channel using a hardware FIFO. However, in many kernels, it is easier and more efficient to allow the IPs to have flexible accesses, such as non-destructive reads and out-of-order access patterns, to the acquired data. For example, Figure 5.1 shows a 5-tap FIR filter, with coefficients {*c1, c2, 0, c4, c5*}, which operates on a sliding window of input data samples. To compute one output sample (i.e., one iteration), this filter needs five input samples: *i)* one new data sample from the current iteration; *ii)* four old data samples from previous iterations, one of which can be skipped as its corresponding coefficient is 0. This

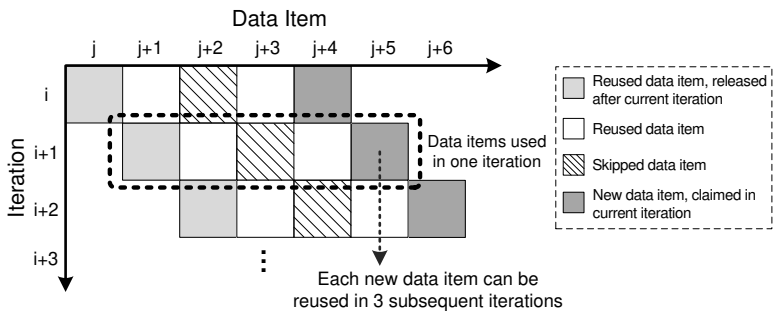


Figure 5.1 Data reuse in a 5-tap FIR filter with coefficients {*c1, c2, 0, c4, c5*}.

filter can be implemented in a very cost-effective way using only one multiplier-accumulator, and processes one input sample each cycle (i.e., 4 cycles /iteration). When communication channels are implemented using hardware FIFOs, explicitly copying data to the IP's internal buffer (i.e., double buffering) is inevitable because data samples need to be reused or skipped. This results in extra buffer management logic, extra memory space, and extra latency. However, if communication channels can be implemented in such a way that non-destructive reads and out-of-order access are supported, this overhead can be removed. Because the same data in the communication channel can be accessed multiple times by different iterations, and unused data as illustrated in Figure 5.1 can be directly skipped.

In order to support these flexible data accesses in communication channels, we provide a high level FIFO interface with low level random access at the communication protocol level. We implement the data buffer in our proposed communication assist (described in Section 5.2.2) with a parameterized dual-ported *circular buffer* (CB). The size of this buffer can be tailored according to the buffer requirement of different applications. As illustrated in Figure 5.2, two sets of pointers are maintained for each circular buffer, namely *Read Start*/*Read End* and *Write Start*/*Write End*. Memory locations between *Read Start* and *Read End* are granted to the consumer as read space. Memory locations between *Write Start* and *Write End* are granted to the producer as write space. IP cores can claim input data or output space in tokens, as long as it does not cause the claimed read and write space to overlap. Within the granted memory space, random access is allowed. Multiple claims before a release are also supported, which is useful in scenarios such as the initial phase of a window operation. In our design, four primitives are used to move the pointers:

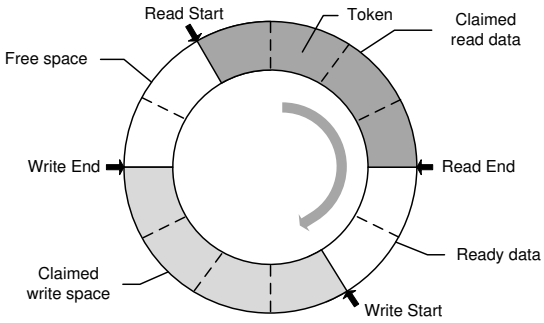


Figure 5.2 Circular buffer management.

1) *Claim Space*: producer claims output space (for writing data) by trying to move the write end pointer;

2) *Release Data*: producer releases output data by moving the write start pointer;

3) *Claim Data*: consumer claims input data by trying to move the read end pointer;

4) *Release Space*: consumer releases the input space by moving the read start pointer.

The aforementioned protocol provides an abstract interface for communication between actors. It can handle data access patterns such as non-destructive reads and out-of-order accesses, which are very suitable for implementing channels in SDF graphs. The producer and consumer of a channel can be synchronized using the same set of primitives, independent of their mapping to IP cores. In other words, from the perspective of the interface and actors it is irrelevant whether the actors are mapped to the same or a different IP core. This simplifies the software design of applications since application designers need to consider only a single FIFO-based interface. Compared to a FIFO implementation, the only drawback of the proposed circular buffer is that extra pointers are used. However, this overhead is very small as shown in our synthesis results, which are presented in Section 5.2.3.

5.2 Communication Assist

A *communicate assist* (CA) is a module that handles the communication between an IP core and other components in the system connected through the communication network. It enables overlapped communication and computation, the-

reby improving the system performance. Apart from performing data transfers like a *Direct Memory Access* (DMA) controller, a CA supports the communication protocol used by the programming model, which decouples an IP's communication from its computation. For a heterogeneous MPSoC, where accelerator IP cores are used as processing components, this decoupling through a CA simplifies the design of accelerator IP cores. This is not only because the interface design of accelerator IP cores is now independent of the data-transfer infrastructure (e.g., P2P, bus, NoC, etc.), but also because the support for various data-access patterns is now offloaded from IP cores to the common CA, which only requires a one-time design effort. The reusability of IP cores is improved as well due to the unified interface between IPs and CAs. Furthermore, with the decoupled communication/computation feature and the predictability feature (described in Section5.2.2 and 5.2.4), a CA-based realization makes it much easier to analyze the complete system and to provide tight timing guarantees at design time, e.g., through SDF analysis tools such as *SDF3* [61].

5.2.1 Design Considerations

In order to deliver the required high performance, different tasks of a computationally intensive application may be executed on different types of processing components which are optimized for execution of particular types of tasks. Therefore, it is crucial that the proposed CA design can handle generic heterogeneous MPSoCs containing accelerator IP cores.

Streaming applications are very common in existing and emerging embedded systems such as smart camera network, unmanned vehicles, and industrial printing. These applications are usually not only computationally intensive, but also very bandwidth demanding. Fortunately, these applications contain a lot of parallelism which makes them very suitable for acceleration using IP cores. These accelerator IPs can greatly increase the performance as long as the bandwidth requirement is fulfilled [6]. Therefore, it is very important that our CA can provide a high bandwidth to a communication channel.

Today's system design is so complex that it is too costly to obtain design information only at the moment a design has been implemented at the *Register Transfer Level* (RTL). Model based approaches, such as SDF, become more and more popular because they can provide design information at a much earlier stage of the design process. In order to facilitate the use of SDF analysis tools, our third design consideration is to provide a predictable CA with an accurate

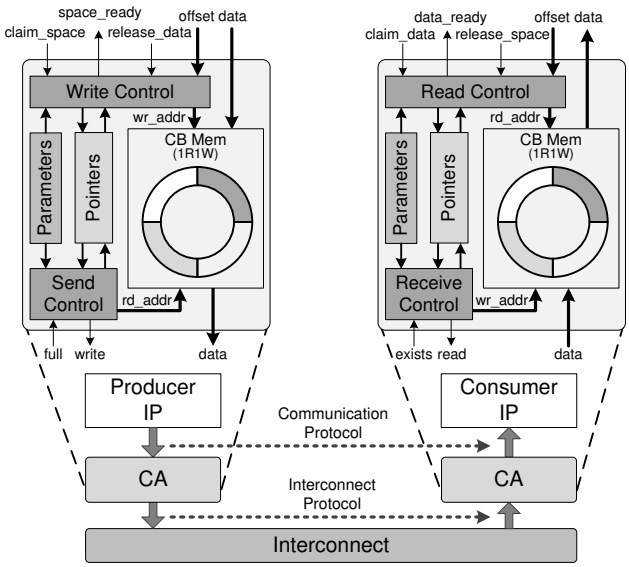


Figure 5.3 Instance of the proposed CA-based platform.

SDF model. This makes it possible to analyze the timing impact of our CA in our design framework.

The proposed CA should also have a low latency and high throughput such that it does not become the performance bottleneck. Moreover, to make it practically adoptable, our CA must have high resource efficiency such that it does not add too much hardware overhead to the system.

In the remainder of this section, we will elaborate the design of our proposed communication assist that meets the aforementioned requirements.

5.2.2 Proposed Communication Assist

As shown in Section 5.1, to support a unified FIFO interface with random access at the communication protocol level, we implement the data buffer of the proposed communication assist using a parameterized dual-port circular buffer. Both the computation side and the communication side can access this buffer at the same time. Figure 5.3 shows an instance of the proposed CA based platform, which contains a producer IP, with a CA in *output mode*, and a consumer IP, with a CA in *input mode*. The *Write Control Unit* and the *Read Control Unit* handle the communication protocol interface between the CA and the IP

cores. The *Send Control Unit* and *Receive Control Unit* are responsible for copying data from/to the circular buffer and to/from the interconnect. The *Pointers Unit* maintains the read/write pointers of the circular buffer. The *Parameters Unit* contains parameters such as *claim-space-size* and *claim-data-size*. These parameters can be reconfigured at run time through a configuration bus connected to a control processor.

The data communication in Figure 5.3 works as follows:

1) The producer task asks the CA for write space through the *claim_space* signal;

2) The producer task starts to write output to the CA memory when the required write space is granted (the *space_ready* signal is high). Within the granted memory space, the producer task can randomly access the memory locations, i.e., data access patterns such as non-destructive reads and out-of-order read/write access are supported;

3) When writing the last data item, the producer task releases the granted memory space through the *release_data* signal, which allows the CA to start pumping data into the interconnect. After the CA has finished sending the data, the memory space of the sent data can be used for output again;

4) The CA at the consumer side automatically copies data into its buffer from the interconnect when there is enough free space;

5) The consumer task tries to claim valid input data by setting the *claim_data* signal. When there are sufficient tokens in the CA buffer for one firing, the CA grants the IP memory space that contains valid data by setting the *data_ready* signal;

6) The consumer task processes the data in the granted read space. As long as the granted input data is not released, it can be accessed randomly and can be accessed multiple times. When the last read access is executed, the consumer task releases the granted space by setting the *release_space* signal. The released space can be used again to buffer the input data from the interconnect. The consumer task can also claim multiple input data spaces before releasing them.

As we can see in Figure 5.3, the signal interface between the proposed CA and the accelerator IP is relatively simple, which eases the effort of the IP inter-

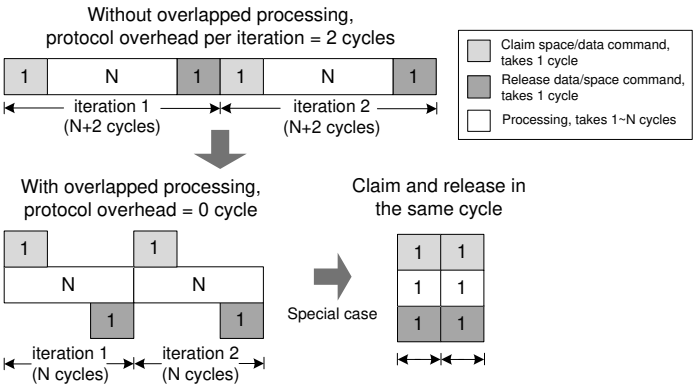


Figure 5.4 Remove protocol overhead with overlapped processing.

face design. When the CA is used with a processor, a bus wrapper is used, which maps the control signals as memory mapped registers on the processor's local bus (see Figure 5.6).

It is worth mentioning that we designed our CA in such a way that the *claim_space*/*claim_data* commands can be handled during the same cycle when the first data is processed. This is done by generating the grant signal (*space_ready*/*data_ready*) in the early phase of each clock cycle, so that corresponding data accesses can be carried out immediately. Likewise, *release_data*/*release_space* commands can be handled during the same cycle when the last data is processed. As a result, protocol overhead is completely removed. This feature is of extreme importance to avoid performance degradation, especially in applications in which small bundles of data are frequently communicated between different IP cores. Figure 5.4 depicts this overlapped processing. We can see that without this feature the protocol overhead per iteration is two cycles. With the proposed overlapped processing, this overhead is removed. In the extreme case (i.e., only one data item sent/received per iteration), the *claim* command, the *release* command, and the data access are handled in only one cycle.

As discussed in the previous sections, our CA behaves like a FIFO at the communication protocol level, but provides random access to the claimed read/write space. This latter feature avoids the need for explicit data copying by the IP core. As a result, IP cores become faster (no copying overhead) and smaller (no internal memory and data reordering logic needed). The proposed

CA needs to adapt to the underlying communication topology. However, the advantage of using a CA is that this adaption is only a one-time effort, and the computation IPs are decoupled and not affected when targeting a different topology (e.g., switch from bus to NoC).

In the instance shown in Figure 5.3, both the producer IP and the consumer IP have one communication channel. Therefore, their corresponding CAs have one channel. When there are multiple communication channels connected to an IP, each communication channel can be assigned a dedicated CA channel, i.e., a dedicated circular buffer and control logic. The design choice of assigning each communication channel a dedicated CA channel is based on the observation that accelerator IPs designed for streaming applications tend to exploit more parallelism than programmable cores, which usually results in much higher bandwidth requirement. A shared CA channel degrades both the throughput and the latency of a high performance accelerator IP. Another reason, which leads to this design choice, is that the resource usage of the proposed CA is very small (see Section 5.2.3). When each communication channel is assigned a dedicated CA channel, the total resource usage of CAs is proportional to the number of communication channels in a design (see Section 5.4).

The detailed hardware implementation of the proposed CA is presented in the next sub section, including information on resource usage, latency, maximum frequency, and memory bandwidth.

5.2.3 Hardware Implementation of the Proposed Communication Assist

The proposed predictable communication assist is realized in Verilog HDL. For the implementation on an FPGA, we use *Xilinx XPS 10.1* and *ISE 10.1* [140] tool flow and *Xilinx xc2vp30* board. Table 5.1 shows the experimental results of the proposed CA. In this table, we also show the comparison with the most related work, which is a CA targeting only programmable cores [3]. We can see that our CA outperforms the CA in [3] in all aspects. Our predictable CA supports both programmable cores and accelerator IPs. As discussed in Section 5.2.2, we designed our CA in such a way that the communication protocol overhead is completely removed, while the reference CA has a considerable overhead (36 cycles). Protocol overhead deteriorates both the throughput and latency especially when the amount of data transferred per iteration is small. For example, in the worst case where only one token needs to be transferred in each iteration, the CA from [3] takes 37 cycles to complete one transaction. Our CA needs only 1 cycle to transfer this data item. The proposed CA can provide a peak BRAM

Table 5.1 Comparison between this work and [3] on Xilinx xc2vp30 FPGA.

		CA in [3]	Proposed CA
Supported System		Homogeneous, with μBlaze cores	Heterogeneous, with generic IPs
Protocol Overhead (in terms of latency)		36 cycles	0 cycles
Maximal Bandwidth per Channel		$BW_{BRAM}/N_{channel}$	BW_{BRAM}
Maximal Frequency		108 MHz	230 MHz
Resource usage (1 input and 1 output channels, 512-word buffer per channel)	Slices	855	57
	LUTs	1626	59
	Registers	430	72
	BRAMs	2	2
Resource usage (2 input and 1 output channels, 512-word buffer per channel)	Slices	1043	75
	LUTs	1994	85
	Registers	562	91
	BRAMs	3	3

bandwidth (BW_{BRAM}) per communication channel (Section 5.2.2), while the bandwidth of the other CA decreases proportionally to the number of communication channels. Note that bandwidth is usually one of the key bottlenecks in the implementation of streaming applications. Our CA also runs twice as fast as the reference CA. To compare the resource usage on the same FPGA, we configure both CAs for two typical cases: *i*) one input channel and one output channel, with a 512-word buffer for each channel; *ii*) two input channels and one output channel, with a 512-word buffer for each channel. We can see that the proposed CA is much smaller compared to the reference CA. This feature of limited resource usage is essential to make the CA practically usable. According to our previous experience, the resource usage of many image processing kernels is less than 1000 slices, so the resource usage of a CA should be far less than this number. If the resource overhead of a CA is too big, introducing CAs into a system becomes less attractive.

It is worth mentioning that when the required buffer size of the proposed CA is relatively small in some applications, or when BRAMs of a target FPGA are the bottleneck resources, the circular buffers inside the proposed CA can be implemented using *Look-Up-Tables* (LUTs). For example, when configured to use LUTs, the proposed CA with a buffer size of 32bit*16 occupies only 49 slices and 0 BRAMs.

Table 5.2 ASIC gate count comparison (8 channels).

	PrimeCell [2] (4 words/channel)	MSAP [4] (8 words/channel)	Proposed CA (4/8 words/channel)
Gate count (NAND2 equivalent)	82k	68k	13.9k/24k

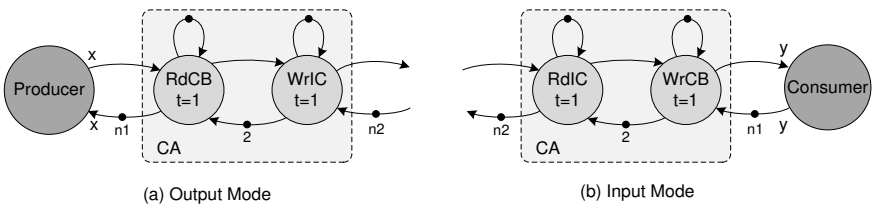


Figure 5.5 Cycle-accurate SDF model of the proposed CA. *RdCB*: read circular buffer; *WrIC*: write interconnect; *RdIC*: read interconnect; *WrCB*: write circular buffer. Unmarked rates and buffer sizes are 1.

When comparing to other related work such as classical DMA controllers and other CA architectures with reported resource usage, our CA again outperforms the design in terms of its hardware resource usage. Table 5.2 shows the ASIC gate counts (*NAND2* equivalent) comparison between our CA and several other architectures. The buffer size of each channel is configured to be either 4 words or 8 words.

5.2.4 Cycle-Accurate SDF Model of the Proposed Communication Assist

In order to facilitate the use of SDF analysis tools to derive design information at an early stage of the design process, an accurate model of the proposed CA is required. In this section, we present a cycle-accurate SDF model that captures the timing behavior of the proposed CA.

Our CA can be modeled as a combination of a write actor and a read actor, as shown in Figure 5.5. Here, x is the output rate of a producer, y is the input rate of a consumer, $n1$ is the buffer size of the CA, and $n2$ is the buffer size of the interconnect link. Unmarked rates or buffer sizes are 1. In output mode, the *RdCB* actor produces in each firing one token to the producer via the channel from *RdCB* to producer, which corresponds to reading a token from the circular buffer and releasing the space. The firing of the *WrIC* actor models the behavior of writing a token to the interconnect. In the input mode, the *RdIC* actor

receives a token from the interconnect. Subsequently, the *WrCB* actor writes the received data (i.e., token) into the circular buffer.

All the CA actors have an execution time of one cycle ($t = 1$), which is required to read/write one token. The self-edge of each actor models the behavior that the next firing cannot start before the previous one has finished. The feedback channel from the write actor to the read actor has two initial tokens. This is to model the independent behavior between write and read ports, and the pipelined behavior of the data transfer. The throughput of each channel in the proposed CA is equal to the maximum throughput of the interconnect. So our proposed CA does not introduce any throughput overhead. Considering latency, when a CA is on the critical path of a design, it introduces two more cycles (one cycle for the *RdCB*/*RdIC* actor and one cycle for the *WrIC*/*WrCB* actor) to the end-to-end latency. This latency overhead is usually negligible compared to the execution time of an application (see Section 5.4).

5.3 Proposed Architecture Template & Tool Flow

Compared to traditional design trajectories, today's system design has become so complex that it is too time-consuming and error-prone to start the design process from the *Register Transfer Level* (RTL). Moreover, the analysis and verification of such designs becomes extremely difficult due to increasing complexity and design requirements. Moving up to a more abstract system level seems to be the only option to address this challenge. The drawback is that this abstraction also opens an implementation gap between the new abstraction level and the RTL level [141]. We believe that the use of an efficient platform-based design methodology is a promising approach to close this gap. To realize such a methodology, a platform template and a hardware-software co-design tool flow are required.

5.3.1 Proposed CA-based Heterogeneous MPSoC Template

A tile-based MPSoC design approach that utilizes the proposed CA brings the opportunity to nicely bridge the aforementioned implementation gap. By decoupling computation from communication and providing unified design interfaces through the proposed CA, complex applications can be partitioned into smaller kernels, which are much easier to design and analyze. The bandwidth requirements of the inter-core communication infrastructure can also be relieved. This is because without a CA, if a computation IP wants to send data to the communication network and the communication network happens to be

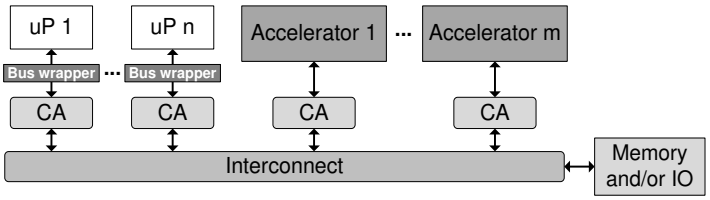


Figure 5.6 Block diagram of the proposed CA-based heterogeneous MPSoC template.

busy, the computation IP is usually stalled, which impacts computation efficiency. However, with a CA or other buffers in between, these kinds of stalls are much less frequent, as the CA can first buffer some data and send them as soon as the communication network is available to access. In other words, with a CA, we can use the bandwidth, which a communication network can provide, more efficiently [142].

Figure 5.6 presents our CA-based heterogeneous MPSoC template for embedded streaming applications. The proposed CA described in Section 5.2 sits between the processing components and the communication infrastructure. When the communication infrastructure cannot accept data, it will initially only stall the CA, and the processing components can continue their execution. New IPs that implement the unified communication protocol/interface described in Section 5.1 can plug-and-play conveniently into this template system.

Besides the fully parameterized CA component, we also provide a *Finite-State-Machine* (FSM) based control module for accelerator IP core design. This FSM-based control module is part of the accelerator IP. It connects the proposed CA through the unified interface at one side, and it connects the IP's computation kernel (in most cases, equivalent to a loop-body at C code level) at the other side. Introducing this module further reduces the design effort of accelerator IPs, as designers now only need to focus on the design and optimization of the computation part. This module is optional to users.

5.3.2 MAMPS+ Design Flow

MAMPS (Multi-Application and Multi-Processor Synthesis) [60] is a design flow for mapping throughput constrained applications on an MPSoC. It integrates several state-of-the-art analysis, mapping, and synthesis tools into an automated tool flow. The inputs of this tool flow are an application modeled as an SDF graph, a C-based implementation for each actor in the graph, and a template based architecture description. The output of the tool is an MPSoC

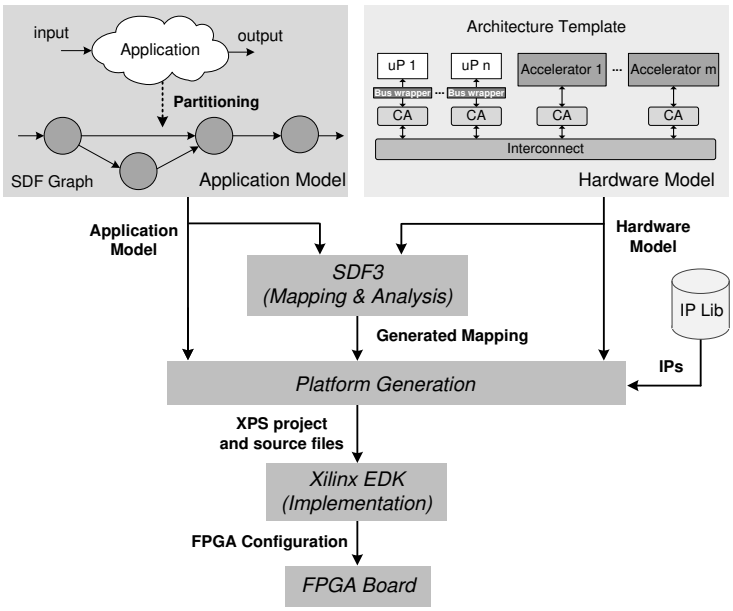


Figure 5.7 MAMPS+ design flow.

platform tailored for the target applications, with C implementation of each actor mapped to general purpose processors (e.g., MicroBlaze) [1]. In the current MAMPS design flow, the application SDF graph is derived manually from its C implementation. The analysis tool that MAMPS uses is called *SDF3* [61]. It can analyze worst-case system properties, such as throughput, latency, and buffer sizes conservatively at design time. Besides these SDF facilities, MAMPS also provides platform generation tools and tightly integrated Xilinx EDA tools, which automate the processes of system instantiation, synthesis, and download.

MAMPS is a very useful design flow for our CA work. However, it only implements support for ISA (Instruction Set Architecture) processors. This work extends the *MAMPS* design flow to support a more generic architecture template, i.e., the proposed CA-based heterogeneous MPSoC template. The new tool flow, *MAMPS+*, is illustrated in Figure 5.7. On the top left, applications are partitioned and modeled with an SDF graph. This model, together with the CA-based architecture template, serves as the input to the *SDF3* tool kit, which analyzes worst-case system properties and generates the mapping to the given platform. It also verifies that such a mapping is deadlock free. It calculates buffer assignments, and predicts the throughput of this mapping [61]. The *Platform Generation* tool instantiates and connects the template components,

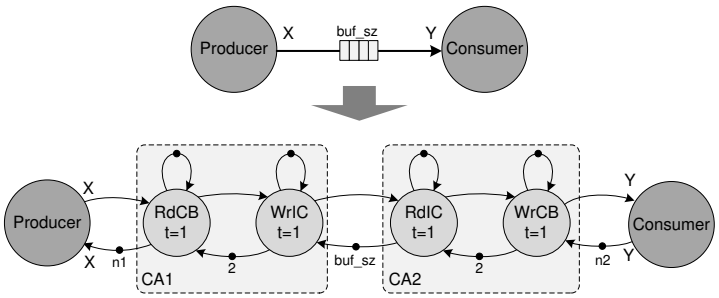


Figure 5.8 SDF graph of two CA-based tiles connected by a FIFO link.

generating an MPSoC platform tailored for the target application. Finally, an FPGA implementation of the whole system is automatically generated by evoking the *Xilinx* EDA tools.

To use the proposed tool flow for real-time applications, it is crucial that the formal analysis of the complete system with CAs can be performed. In the *MAMPS+* tool flow, the SDF model of the proposed CA, which is discussed in Section 5.2.4, is inserted between each pair of producer-consumer actors, resulting in an architecture-aware SDF graph of the complete system. Figure 5.8 shows how this transformation is performed for a producer-consumer pair that communicates via a FIFO link.

5.4 Case Study: Vision Processing in OLED Printing

To demonstrate that the proposed *MAMPS+* design flow enables efficient integration of accelerator IP cores into a heterogeneous MPSoC, we use an industrial high-speed camera application, *Organic-Light-Emitting-Diode* (OLED) printing, as a case study. In OLED manufacturing, organic materials need to be accurately injected into the tiny OLED substrates on the wafer, the size of which are typically in the range of 10 μm to 1000 μm. This fine process has to be done at an extremely high speed due to high yield requirement (1000 frames/s throughput and 1 *ms* latency for the complete system, including sensing, processing, and control). The time budget for the vision processing part is only 350 μs [143]. The top left of Figure 5.9 shows one captured wafer segment from a statically-mounted high-speed camera, on which nine OLED structures are located. The bottom left of Figure 5.9 shows the same image in which detected OLED centers have been marked.

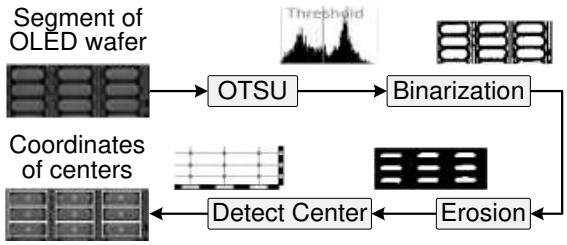


Figure 5.9 Vision pipeline of the OLED center detection.

Table 5.3 Kernel execution time of both μBlaze (software) and accelerator (hardware) implementations for a 120×45 pixel image.

	OTSU	Binarization	Erosion	Detect Center
μBlaze Implementation (cycles)	97552	70201	284819	83080
Accelerator Implementation (cycles)	6227	5401	97	170
Speedup	15.7×	13.0×	2936.3×	488.7×

The complete vision pipeline of OLED center detection consists of four image processing kernels (Figure 5.9) [144]. The input image of the pipeline is typically of 120×45 pixels. The front end of the pipeline applies *OTSU* optimal threshold [145] to segment the OLED structures from the background. The binary image is then eroded to remove the noise in the image. By utilizing the characteristics of the repetitive structures, the segmented OLED structures are reduced into horizontal and vertical vectors. The centers of the OLED structure are found by searching these two vectors.

For each image processing kernel in Figure 5.9 we provide both a MicroBlaze implementation (software) and a dedicated accelerator implementation (hardware) [21]. The execution time of each kernel for a 120×45 pixel image is shown in Table 5.3. This performance and implementation information are used by the *MAMPS+* tool flow: the performance information of each kernel is used by *SDF3* to generate a valid mapping, and the implementation information is used by the *Platform Generation* tool to generate the MPSoC platform.

Figure 5.10 presents the SDF graph of the aforementioned OLED center detection application. Self edges are omitted for the sake of simplicity. The throughput of the *Image Source* actor is one pixel/cycle. According to Table 5.3,

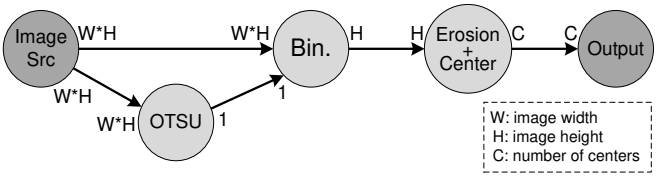


Figure 5.10 SDF graph of the OLED printing application.

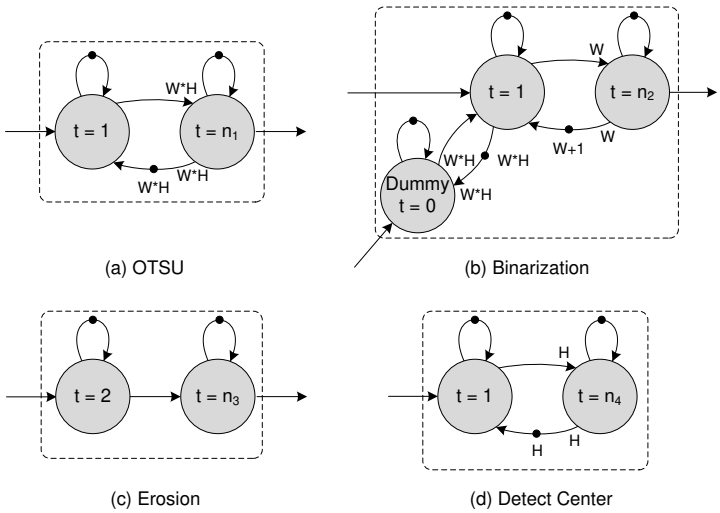


Figure 5.11 Refined SDF models with hardware implementation information for each kernel actor. Unmarked rates are 1.

the MicroBlaze implementation cannot meet the targeted performance, as the processing time of each kernel already exceeds the 350 μs timing budget. A hardware accelerator implementation is then selected for each actor in the mapping stage. To obtain the accurate design information such as throughput, latency, and buffer size using the *SDF3* tool kit, the application SDF graph is first transformed into the architecture-aware SDF graph in two steps:

1) Refine each kernel actor with hardware implementation information;

2) Inserting the proposed CA model between two kernel actors.

Figure 5.11 shows the refined SDF models of each image processing kernel shown in Figure 5.10. For example, the first actor in the *OTSU* SDF graph (Figure 5.11 (a)) models the behavior of receiving and processing one token (i.e., one pixel) per cycle. The second actor models the behavior that the output

Table 5.4 Results of both a non-CA based and CA-based hardware implementations on a Xilinx xc2vp30 FPGA.

	Non-CA Based	CA-Based	Difference
End-to-end latency (cycles)	11764	11780	0.1%
Throughput (frames/cycle)	1.6×10^{-4}	1.6×10^{-4}	0.0%
Frequency (MHz)	160	160	0.0%
No. of slices	4236	4689	10.7%
No. of BRAMs	11	11	0.0%
No. of MULT18×18	9	9	0.0%
Design & analysis effort	4 Weeks	1 Day	> 20×

token (i.e., threshold value for binarization stage) is produced when the complete frame (i.e., $W \times H$ pixels) is processed. Here $n_1 = 1$ according to the hardware implementation. The *Dummy* actor in the refined *Binarization* SDF graph (Figure 5.11 (b)) models the behavior that the binarization kernel only receives the threshold value from the *OTSU* kernel once every frame. This threshold is then applied onto each pixel of the current frame. Since our CA supports non-destructive reads, explicit copying of the threshold value from the communication channel by the binarization IP is avoided.

The way to insert the proposed CA model between two kernel actors is similar to what Figure 5.8 shows. The only difference is that the channels, which model physical buffers, are removed for the sake of buffer size analysis. For visibility reason, we do not show the complete architecture-aware SDF graph here. The *SDF3* tool kit takes this graph as input, and provides timing analysis and buffer size analysis of the complete system. The estimated end-to-end latency of this application is 11780 cycles and the estimated throughput is 1.6×10^{-4} frames/cycle when channels are configured to the optimum buffer sizes suggested by the *SDF3* tool. The measured results from our final FPGA implementation (shown in Table 5.4) confirm the accuracy of the prediction at design time.

When a valid mapping is found, the *Platform Generation* tool instantiates the parameters of each CA according to the mapping and analysis information. With the kernel IP implementations stored in the IP library, an FPGA implementation of the whole system is generated, which can be directly downloaded to and executed on the target FPGA board. As discussed in Section 5.1 and 5.2, the kernel IPs that are stored in the IP library only consist of the computational

part and a simple unified interface connecting with the CAs. When coding with the time-consuming and error-prone hardware description languages (HDLs), it is more convenient to design this kind of IPs compared to the design of IPs which have to take complex communication and different interconnects into consideration.

As a design reference, a non-CA based implementation of the complete vision pipeline is manually designed and optimized [21]. In this manual design, we spent about four weeks to integrate the pre-designed IPs into a working system which meets both the throughput and latency requirements. Most of the time is spent on the performance analysis and debugging the communication between pairs of IPs. In contrast, it took us only one day to accomplish an implementation with the CA-based design flow, which fulfills the same requirements. Table 5.4 shows the comparison between the non-CA based manual implementation and the proposed CA-based implementation. Both implementations start from the same design point, i.e., a pre-designed IP library (only the computational parts). We show that by using the proposed architecture template and tool flow, the design and analysis effort is dramatically reduced. The cost is mainly some resource overhead. However, comparing to the high efficiency and the introduced design-time predictability feature, such a small price is well worthy.

In practice, vision pipelines (i.e., algorithms) often need to adapt to different working environment or scenarios. New kernels could be added into the vision pipeline, or part of the kernels could be replaced with alternatives. For a non-CA based design, this usually means a complete re-design of the system as computation and communication of different kernels are tightly coupled. However, for the CA based approach, the required re-design effort is much less. For example, when the *Detect Center* kernel in the OLED printing vision pipeline is replaced by a *Center-of-Gravity* kernel which provides higher accuracy, we only need to update the SDF graph to re-analyze the performance and re-calculate the required CA buffer sizes with the *SDF3* tool. After reconfiguring the parameters of CAs based on the analysis and plugging in the *Center-of-Gravity* IP, the new system implementation can be quickly achieved.

5.5 Related Work

There are several works addressing the issue of the inter-component communication in MPSoCs. Gangwal et al. presented a synchronization scheme for em-

bedded systems with shared memory, in which channel controllers are used for synchronization between tasks [146]. Compared to our communication assist, it is much slower and consumes more hardware resources. The work in [142, 147] presents a CA-based platform for ISA processors, on which the C-HEAP protocol [26] is used. One disadvantage of this design is the duplication of data in the communication and data memories, which results in large memory space and extra data transfer time. Compared to their design, our CA uses a similar communication protocol but supports both ISA processors and accelerator IP cores. Moreover, our CA uses a unified memory for both communication and data, which saves up to 50% buffer memory space. The communication latency is also decreased accordingly. Shabbir et al. present a CA design which has similar functionality as ours but also only supports tiles with ISA processors in [148]. The detailed comparison in Section 5.2.3 shows that our CA outperforms the CA from Shabbir in all aspects. Our CA has a smaller performance overhead, far less resource costs, and it supports generic IP cores, which make our CA more attractive. CELL B.E. [149] implements communication between processing elements (SPEs) and the external memory through DMA controllers called *Memory Flow Controller* (MFC). The key difference between MFC and our CA is that in MFC the synchronization between the memories has to be performed explicitly by the SPEs, while in our case, it is taken care of by the CA itself and the processor is freed from the synchronization overhead. The MSAP presented in [4] uses a control network for the hand-shake between processors before actual data transfers. Our CA does not require such a control network because it uses *backpressure* as the flow control mechanism. ROCCC is a high level synthesis framework that generates accelerator IPs using C as input [150]. In ROCCC, the *smart buffer*, which is the interface module between the datapath and the memory interface, enables input data reuse for window operations. However, there is no unified protocol for IPs and processors implemented in the smart buffer, which makes the integration in complex MPSoCs difficult.

The design of an MPSoC platform has been a challenging task. Several works tackle this problem by model-based MPSoC design. ESPAM is an MPSoC design framework based on the KPN model [141]. In ESPAM, a communication controller is introduced for a homogeneous multiprocessor platform. An IP wrapper [151], which has similar functionality as our CA, extends ESPAM to a heterogeneous platform with hardware IPs. Different from the CA in this work, the IP wrapper uses hardware FIFO interfaces for input and output. Thus, access patterns such as non-destructive reads and out-of-order

Table 5.5 Summary of our contributions.

Contributions	Features
CA Design	• High resource efficiency & high performance • No protocol overhead • Unified interface • Flexible data accesses • Predictability
CA SDF Model	• Cycle accurate • Easy to be integrated into application models
Architecture Template & Tool Flow	• Ease IP integration effort • Provide design time analysis
Case Study	• Industrial high-speed camera application • FPGA implementation • Demonstrated the efficiency of the proposed CA and tool flow

access are only possible after copying the data to the local storage of the consuming task, resulting in extra memory requirements and design effort. Moreover, the IP wrapper consumes about 50% more resources than the CA of this work. In addition, compared to the SDF model used in this work, it is more difficult, or even impossible to analyze properties like throughput and buffer requirement in the KPN model used in ESPAM, which affects the predictability [152]. The work in [153] also uses the KPN model, in which random access to the data is enabled by introducing a read/write window. However, extra data copies are required from a FIFO to read/write windows. Optimus is a framework for mapping applications modeled in SDF onto FPGAs [154]. In Optimus, the interface module of each channel is orchestrated based on the detailed analysis of the actor behavior, which makes it efficient and fast. However, it requires exact analysis of the details of all IPs and the interconnect to generate such modules, which is not always possible in complex MPSoC designs.

5.6 Summary

In this chapter, we presented a *communication assist* (CA) to efficiently integrate generic IPs into an MPSoC with a predictable design flow. The CA separates inter-core communication from the IP's computation, and provides a unified abstract interface for accelerator IPs and processors. We also presented an accurate SDF model for the proposed CA, which makes it possible to provide timing guarantees for systems using the CA. The experimental results show

that the proposed CA design provides very high performance and resource efficiency compared to the existing works.

Based on the proposed CA design, we introduced a heterogeneous MPSoC template which can be used by the proposed *MAMPS+* tool flow. We used an industrial application, the complete vision processing pipeline in OLED printing, as a case study. We showed that with the proposed CA-based MPSoC template, analysis and mapping of real-time streaming applications onto a heterogeneous MPSoC with dedicated accelerators are easy and efficient. Table 5.5 summarizes the contributions of this work.

To complete the proposed MAMPS+ design flow, there is still some work to be done. An efficient *design space exploration* (DSE) algorithm is missing, which is required to quickly locate the proper hardware instantiations. Integrating shared memory into the *SDF3* tool flow is another interesting but challenging task. We would also like to add more programmable cores, especially the low-power cores we designed in chapter 2, 3, and 4, into our IP library. This requires some engineering work to interfacing the programmable cores with the CA.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In Section 6.1, the most important conclusions which are made in the previous chapters are recapped. Section 6.2 discusses future research directions that we are interested in.

6.1 Conclusions

Streaming applications are an important class of applications which are usually very computationally intensive and have very tight power budget. Therefore, embedded systems designed for these applications not only need to have a large amount of processing power, but also need to provide the high processing power in an energy-efficient way. To achieve such kind of highly efficient systems, significant efforts at different hardware and software design levels are required. Chapter 1 briefly discussed some important challenges we are facing, including *i*) high overhead in supplying of instructions and data to functional units; *ii*) issues of wide-range V_{dd} scaling in both standard-cell based logics and SRAMs; *iii*) increasing variation issue along with the technology scaling; *iv*) energy-aware compilers other than performance-only compilers; and *v*) design issues introduced by heterogeneity. In this thesis, we presented our efforts to overcome part of these design challenges.

In Chapter 2, *MOVE-Pro*, a new TTA based processor architecture is proposed to reduce energy consumption of the register file, and convert this energy saving into the total core energy saving. With optimizations at ISA, architecture, circuit, and compiler levels, the low-power potential of TTAs is fully exploited. Moreover, with a much denser code size, TTAs' performance is also improved. In the head-to-head comparison, we showed that up to 80% of RF accesses can be reduced with the proposed MOVE-Pro framework. The reduction in RF energy is successfully transferred to the total core energy saving. The comparison with RISC counterpart showed that up to 11.6% reduction of the total core

energy is achieved. When compared to VLIWs, the advantage of the proposed MOVE-Pro architecture is even better.

Chapter 3 describes the integration of flexible special instruction support in a generic embedded processor with a compact ISA. Apart from performance improvement, the main focus of this work is on energy efficiency of the processor for different streaming application domains. A partially reconfigurable decoder and a software-controlled explicit bypass network are introduced, allowing the processor to support operation pairs without increasing the instruction width or number of register file ports. The experimental results demonstrated that the proposed architecture is effective: average dynamic cycle count is reduced by over 25%. The total processor energy consumption is reduced by 15.8%. When high performance is required, the proposed architecture is able to achieve a speed-up of 12.6% with 13.1% energy reduction compared to the baseline by introducing multi-cycle SFU operations.

In Chapter 4, Xetal-Pro, a massively parallel SIMD architecture is proposed. We combined massive parallelism and aggressive V_{dd} scaling in the context of a wide SIMD processor. A hybrid memory system was also introduced to reduce the non-local memory traffic and enables further V_{dd} scaling. Preliminary results showed that it is possible to achieve less than 1 pJ/op energy consumption at the ultra low-energy mode, while still delivering a throughput of about 0.7 GOPS. This makes Xetal-Pro a very promising building block in MPSoCs for future low-energy embedded streaming computing.

Finally, we proposed an efficient and predictable *communication assist* (CA) for integrating generic IP cores into heterogeneous MPSoCs in Chapter 5. The CA separates inter-core communication from the IP's computation, and provides a unified abstract interface for accelerator IPs and processors. We also presented an accurate SDF model for the proposed CA, which makes it possible to provide timing guarantees for systems using the CA. The experimental results showed that the proposed CA design provided very high performance and resource efficiency compared to the existing works. Based on the proposed CA design, the existing design flow, MAMPS, is updated with a CA-based hardware template. As a case study, vision processing pipeline of a typical industrial application, *Organic Light Emitting Diode* (OLED) screen printing, is mapped onto the proposed platform. This case study demonstrated that the proposed design flow enables efficient integration of accelerator IPs into a heterogeneous MPSoC which targets streaming applications.

6.2 Future Work

This thesis presented solutions to various problems in designing low-power architectures for streaming applications. As future work, several remaining tasks and interesting issues can be studied further:

- In the MOVE-Pro work (Chapter 2), a two-issue MOVE-Pro instantiation is fully implemented in HDL as the RISC counterpart. However, when comparing to the VLIW architecture, we did not provide the MOVE-Pro instantiations at RTL level. As future work, it would be very interesting if the RTL generation could be automated based on the user configuration. This can be achieved by providing an architecture configuration file and a set of pre-designed IP modules (e.g., ALU, RF, dispatch network). Extending the current compiler to support software pipelining and cross basic block scheduling would also be a promising direction to improve the efficiency.

- In the design of special function unit (Chapter 3), we mainly focused on the support of flexible pair patterns. Though these operation patterns are the most common ones, it would be interesting if more complex patterns in the SFU could also be supported. This should be done after careful exploring the trade-offs between the complexity of the SFU and the energy efficiency of the processor architecture. Another potential issue in the proposed SFU design is frequency loss (14.4% loss even though multiple-cycle execution is applied) compared to the design without SFU. Further optimization should be considered to solve this issue.

- In the ultra low-energy SIMD design (Chapter 4), we analyzed the possibility of achieving 1 pJ/op for typical steaming applications. The result is promising. In this work, we mainly focused on the memory subsystem. A further step could be exploring the proper micro-architecture of the processing engines (PEs) and the communication network among PEs. As another important component, the corresponding compiler also needs to be constructed.

- In the communication assist work (Chapter 5), the MAMPS design flow is updated with the proposed CA-based hardware template. To fully automate the proposed design flow, there is still some work that can be done: *i*) an efficient DSE algorithm which can quickly locates the proper hardware instantiations; *ii*) adding more CA-enabled programmable

cores into our IP library, especially the low-power cores designed in chapter 2, 3, and 4; *iii*) integrating shared memory into the *SDF3* tool flow.

- Variation and fault tolerance are not covered in this work. However, these aspects are becoming increasingly important in the design of modern embedded systems. In the future work, we would like to pay more attention on these directions at different design levels. Interesting techniques in this field, such as Razor latch/flip-flop [122, 123], structural duplication [121], body biasing [124, 125], and cell resizing [126], could be introduced into our designs.

BIBLIOGRAPHY

1. MAMPS. *A design flow to map throughput constrained applications on MPSoC, http://www.es.ele.tue.nl/mamps/*.
2. ARM. *PrimeCell™ DMA controller, http://www.arm.com.*
3. A. Shabbir, et al., *A predictable communication assist*, in Proceedings of the 7th ACM International Conference on Computing Frontiers, 2010, pp. 97-98.
4. S.I. Han, et al., *An efficient scalable and flexible data transfer architecture for multiprocessor SoC with massive distributed memory*, in Proceedings of the 41st annual Design Automation Conference, 2004, pp. 250-255.
5. S. Jalali, *Trends and Implications in Embedded Systems Development*, in White Paper of Tata Consultancy Services, 2009.
6. K. van Berkel, *Multi-core for mobile phones*, in Proceedings of the Conference on Design, Automation and Test in Europe, 2009, pp. 1260-1265.
7. J. Rabaey, *Low Power Design Essentials*, 2009, Springer.
8. G. Mathur, et al., *Ultra-low power data storage for sensor networks,* ACM Transactions on Sensor Networks, 2009, 5(4): pp. 1-33.
9. W. Thies, M. Karczmarek, and S. Amarasinghe. *StreamIt: A language for streaming applications*, in Proceedings of 11th International Symposium on Compiler Construction, 2002, pp. 179-196.
10. E. Dahlman, S. Parkvall, and J. Skold, *4G: LTE/LTE-Advanced for Mobile Broadband*, 2011, Elsiver.
11. E. Dahlman, et al., *3G Evolution: HSPA and LTE for Mobile Broadband*, 2008, Elsevier.
12. ITU-T, *Advanced video coding for generic audiovisual services*, in Technical Report, ITU-T Recommendation H.264, 2005.
13. ITU-T, *Video coding for low bit rate communication*, in Technical Report, ITU-T Recommendation H.263, 1996.
14. MPEG-2, *Generic Coding of Moving Pictures and Associated Audio Systems*, in Technical Report, ISO/IEC 13818-2, 1994.
15. MPEG-4, *Information Technology Coding of Audio-Visual Objects*, in Technical Report, ISO/IEC 14496-2, 2007.
16. K. Brandenburg, *MP3 and AAC Explained*, in Proceedings of 17th International Conference on High Quality Audio Coding, 1999, pp. 1-12

17. *DAB/DAB+/DMB Receivers, www.worlddab.org*.
18. R. Maini, and H. Aggarwal, *A Comprehensive Review of Image Enhancement Techniques,* Journal of Computing, 2010, 2(3): pp. 8-13.
19. Y. HaCohen, et al., *Non-rigid dense correspondence with applications for image enhancement,* ACM Transactions on Graphics, 2011, 30(4): pp. 1-70.
20. M. Gavelli, et al. *Terafly: A THz image-processing-based architecture for semi-automatic industrial inspection and measurement*, in Proceedings of 37th International Conference on Infrared, Millimeter, and Terahertz Waves, 2012, pp. 1-2.
21. Y. He, et al., *Feasibility Analysis of Ultra High Frame Rate Visual Servoing on FPGA and SIMD Processor*, in Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems, 2011, pp. 623-634.
22. S. Izadi, et al. *KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera*, in Proceedings of the 24th ACM symposium on user interface software and technology, 2011, pp. 559-568.
23. A. Geiger, J. Ziegler, and C. Stiller. *Stereoscan: Dense 3d reconstruction in real-time*, in Proceedings of Intelligent Vehicles Symposium, 2011, pp. 963-968.
24. ITRS, *International technology roadmap for semiconductors, System Drivers,* 2007.
25. J. Henkel, *Closing the SoC Design Gap,* Computer, 2003, 36(9): pp. 119-121.
26. A.E. Nieuwland, et al., *C-HEAP: A Heterogeneous Multi-Processor Architecture Template and Scalable and Flexible Protocol for the Design of Embedded Signal Processing Systems,* Design Automation for Embedded Systems, 2002, 7(3): pp. 233-270.
27. H. Andreas, *A composable and predictable on-chip interconnect,* PhD thesis, 2009.
28. J.M. Tarascon, *Key challenges in future Li-battery research,* Philosophical Transactions of the Royal Society A, 2010, 368(1923): pp. 3227-3241.
29. G. Jeong, et al., *Prospective materials and applications for Li secondary batteries,* Energy & Environmental Science, 2011, 4(6): pp. 1968-2002.
30. J. Balfour, et al., *An energy-efficient processor architecture for embedded systems,* Computer Architecture Letters, 2007, 7(1): pp. 29-32.

31. J. Balfour, R.C. Halting, and W.J. Dally, *Operand Registers and Explicit Operand Forwarding,* Computer Architecture Letters, 2009, 8(2): pp. 60-63.
32. J.W. van de Waerdt, et al., *The TM3270 media-processor*, in Proceedings of the 38th IEEE/ACM International Symposium on Microarchitecture, 2005, pp. 331-342.
33. D.R. Gonzales, *Micro-RISC architecture for the wireless market.* IEEE Micro, 1999, 19(4): pp. 30-37.
34. Y. Pu, *On the road towards robust and ultra low energy CMOS digital circuits using sub/near threshold power supply*, PhD thesis, 2009.
35. K. Takeda, et al., *A read-static-noise-margin-free SRAM cell for low-VDD and high-speed applications,* Journal of Solid-State Circuits, 2006, 41(1): pp. 113-121.
36. A. Pavlov and M. Sachdev, *CMOS SRAM Circuit Design and Parametric Test in Nano-Scaled Technologies: Process-Aware SRAM Design and Test*, 2008, Springer.
37. B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk,* 2008, Morgan Kaufmann.
38. S. Borkar, *Designing reliable systems from unreliable components: the challenges of transistor variability and degradation,* IEEE Micro, 2005, 25(6): pp. 10-16.
39. T. Miller, R. Thomas, and R. Teodorescu, *Mitigating the Effects of Process Variation in Ultra-low Voltage Chip Multiprocessors using Dual Supply Voltages and Half-Speed Units,* IEEE Computer Architecture Letters, 2012, 11(2): pp. 45-48.
40. S.K. Springer, et al., *Modeling of variation in submicrometer CMOS ULSI technologies,* IEEE Transactions on Electron Devices, 2006, 53(9): pp. 2168-2178.
41. M. Onabajo and J. Silva-Martinez, *Process Variation Challenges and Solutions Approaches*, in Analog Circuit Design for Process Variation-Resilient Systems-on-a-Chip, 2012, pp. 9-30.
42. X. Tang, V.K. De, and J.D. Meindl, *Intrinsic MOSFET parameter fluctuations due to random dopant placement,* IEEE Transactions on Very Large Scale Integration Systems, 1997, 5(4): pp. 369-376.
43. H. Fukutome, et al., *Direct evaluation of gate line edge roughness impact on extension profiles in sub-50-nm n-MOSFETs,* IEEE Transactions on Electron Devices, 2006, 53(11): pp. 2755-2763.

44. A. Asenov, S. Kaya, and J.H. Davies, *Intrinsic threshold voltage fluctuations in decanano MOSFETs due to local oxide thickness variations,* IEEE Transactions on Electron Devices, 2002, 49(1): pp. 112-119.
45. M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, *Compiler optimizations for low power systems*, in Power Aware Computing textbook, 2002, pp. 191-210.
46. M. Lee, et al., *Power analysis and minimization techniques for embedded DSP software,* IEEE Transactions on Very Large Scale Integration Systems, 1997, 5(1): pp. 123-135.
47. S. Steinke, et al. *Assigning program and data objects to scratchpad for energy reduction*, in Proceedings of Design, Automation and Test in Europe Conference and Exhibition, 2002, pp. 409-415.
48. R. Kumar, et al. *Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance*, in Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004, pp. 64-75.
49. R. Kumar, et al., *Processor power reduction via single-ISA heterogeneous multi-core architectures,* Computer Architecture Letters, 2003, 2(1): pp. 1-4.
50. B. Kienhuis, et al., *An approach for quantitative analysis of application-specific dataflow architectures*, in Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors, 1997, pp. 338-349.
51. F. Balarin, *Hardware-software co-design of embedded systems: the POLIS approach,* 1997, Kluwer Academic Publishers.
52. A. Pimentel, et al. *Towards efficient design space exploration of heterogeneous embedded media systems*, in Proceedings of the International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2001, pp. 57-73.
53. A. Mihal, et al., *Developing architectural platforms: A disciplined approach,* IEEE Design & Test of Computers, 2002, 19(6): pp. 6-16.
54. A. Kumar, *Analysis, Design and Management of Multimedia Multiprocessor Systems*, PhD theis, 2009.
55. E.A. Lee and D.G. Messerschmitt, *Synchronous data flow,* Proceedings of the IEEE, 1987, 75(9): pp. 1235-1245.
56. A.H. Ghamarian, et al., *Throughput Analysis of Synchronous Data Flow Graphs*, in Proceedings of the 6th International Conference on Application of Concurrency to System Design, 2006, pp. 25-36.

57. S. Stuijk, M. Geilen, and T. Basten, *Throughput-Buffering Trade-Off Exploration for Cyclo-Static and Synchronous Dataflow Graphs,* IEEE Transactions on Computers, 2008, 57(10): pp. 1331-1345.
58. Y. Yang, et al., *Automated bottleneck-driven design-space exploration of media processing systems*, in Proceedings of the Conference on Design, Automation and Test in Europe, 2010, pp. 1041-1046.
59. A. Kumar, et al., *Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA,* ACM Transactions on Design Automation of Electronic Systems, 2008, 13(3): pp. 1-27.
60. R. Jordans, et al., *An Automated Flow to Map Throughput Constrained Applications to a MPSoC*, in Bringing Theory to Practice: Predictability and Performance in Embedded Systems, 2011, pp. 47-58.
61. S. Stuijk, M. Geilen, and T. Basten, *SDF³: SDF For Free*, in Proceedings of the 6th International Conference on Application of Concurrency to System Design, 2006, pp. 276-278.
62. Y. He, et al., *MOVE-Pro: a Low Power and High Code Density TTA Architecture*, in Proceedings of the 11th Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2011, pp. 294-301.
63. D. She, et al., *Energy Efficient Code Generation for Processors with Exposed Datapath,* in Proceedings of the 9th Workshop on Optimizations for DSP and Embedded Systemsm, 2011, pp. 55-61.
64. H. Corporaal, *Microprocessor Architectures: From VLIW to TTA,* PhD thesis, 1998.
65. H. Corporaal, *TTAs: Missing the ILP complexity wall,* Journal of Systems Architecture, 1999, 45(12): pp. 949-973.
66. K. Karuri, et al., *A Generic Design Flow for Application Specific Processor Customization through Instruction-Set Extensions (ISEs),* in Proceedings of the 9th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation, 2009, pp. 204-214.
67. N. Clark, H. Zhong, and S. Mahlke, *Processor Acceleration Through Automated Instruction Set Customization*, in Proceedings of the 36th IEEE/ACM International Symposium on Microarchitecture, 2003, pp. 129-140.
68. R. Leupers, et al., *A design flow for configurable embedded processors based on optimized instruction set extension synthesis*, in Proceedings of Design, Automation and Test in Europe, 2006, pp. 581-586.

69. Y. He, et al., *Xetal-Pro: An Ultra-Low Energy and High Throughput SIMD Processor*, in Proceedings of the 47th Annual Design Automation Conference, 2010, pp. 543-548.
70. J. Villarreal and W.A. Najjar, *Compiled hardware acceleration of molecular dynamics code*, in Proceedings of the 18th International Conference on Field Programmable Logic and Applications, 2008, pp. 667-670.
71. AutoESL, *http://www.xilinx.com/tools/autoesl.htm.*
72. J. Kin, M. Gupta, and W.H. Mangione-Smith. *The filter cache: an energy efficient memory structure*, in Proceedings of the 30th ACM/IEEE international symposium on Microarchitecture, 1997, pp. 184-193.
73. G. Semeraro, et al. *Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling*, in Proceedings of 8th Symposium on High-Performance Computer Architecture, 2002, pp. 29-40.
74. M. Woh, et al. *AnySP: anytime anywhere anyway signal processing*, in Proceedings of the 36th International Symposium on Computer Architecture, 2009, pp. 128-139.
75. A.A. Abbo, et al., *Xetal-II: a 107 GOPS, 600 mW massively parallel processor for video scene analysis,* IEEE Journal of Solid-State Circuits, 2008, 43(1): pp. 192-201.
76. L.H. Lee, B. Moyer, and J. Arends. *Instruction fetch energy reduction using loop caches for embedded applications with small tight loops*, in Proceedings of Symposium on Low Power Electronics and Design, 1999, pp. 267-269.
77. M. Sjalander, H. Eriksson, and P. Larsson-Edefors. *An efficient twin-precision multiplier*, in Proceedings of IEEE International Conference on Computer Design, 2004, pp. 30-33.
78. D. She, et al., *Scheduling for Register File Energy Minimization in Explicit Datapath Architectures*, in Proceedings of the Design, Automation and Test in Europe, 2012, pp. 388-393.
79. H. Corporaal and H. Mulder, *MOVE: a framework for high-performance processor design*, in Proceedings of the ACM/IEEE conference on Supercomputing, 1991, pp. 692-701.
80. T. Pitkänen, et al., *Low-power, high-performance TTA processor for 1024-point fast fourier transform*, in Proceedings of the Embedded Computer Systems: Architectures, Modeling, and Simulation, 2006, pp. 227-236.
81. MAXIM-IC. *MaxQ Microcontroller, http://www.maximic.com/.*
82. Delft University of Technology, *MOVE project, http://ce.et.tudelft.nl/MOVE/.*

83. J. Hoogerbrugge and H. Corporaal. *Transport-triggering vs. operation-triggering*, in Proceedings of Compiler Construction, 1994, pp. 435-449.
84. Tampere University of Technology. *TTA-based Codesign Environment (TCE), http://tce.cs.tut.fi/.*
85. OpenCores. *OpenRISC 1200, http://opencores.org/openrisc.*
86. S. Rixner, et al., *Register organization for media processing*, in Proceedings of the 6th International Symposium on High-Performance Computer Architecture, 2000, pp. 375-386.
87. CACTI. *cacti 5.3, rev 174, http://quid.hpl.hp.com:9081/cacti/.*
88. V. Zyuban and P. Kogge, *The energy complexity of register files*, in Proceedings of the international symposium on low power electronics and design, 1998, pp. 305-310.
89. H. Kubosawa, et al., *A 2.5-GFLOPS, 6.5 million polygons per second, four-way VLIW geometry processor with SIMD instructions and a software bypass mechanism,* IEEE Journal of Solid-State Circuits, 1999, 34(11): pp. 1619-1626.
90. L.A. Lozano and G.R. Gao, *Exploiting short-lived variables in superscalar processors*, in Proceedings of the 28th international symposium on Microarchitecture, 1995, pp. 292-302.
91. V. Guzma, et al., *Impact of Software Bypassing on Instruction Level Parallelism and Register File Traffic*, in Proceedings of the Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008, pp. 23-32.
92. V. Guzma, et al., *Reducing processor energy consumption by compiler optimization*, in Proceedings of the IEEE Workshop on Signal Processing Systems, 2009, pp. 63-68.
93. G. Cichon, et al. *Synchronous Transfer Architecture (STA)*, in Proceedings of Computer Systems: Architectures, Modeling, and Simulation, 2004, pp. 343-352.
94. J. Guo, et al., *A phase-coupled compiler backend for a new VLIW processor architecture using two-step register allocation*, in Proceedings fo International Conference on Application-Specific Systems, Architectures, and Processors, 2007, pp. 346-352.
95. ARM. *ARM Thumb Instruction Set, http://www.arm.com*.
96. K. Karuri, et al., *Increasing data-bandwidth to instruction-set extensions through register clustering,* in Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 2007, pp. 166-171.

97. M. Arnold and H. Corporaal, *Automatic detection of recurring operation patterns*, in Proceedings of the 7th International Workshop on Hardware/Software Codesign, 1999, pp. 22-26.
98. P. Yu and T. Mitra, *Characterizing embedded applications for instruction-set extensible processors*, in Proceedings of the 41st Annual Design Automation Conference, 2004, pp. 723-728.
99. C. Lattner and V. Adve, *LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation*, in Proceedings of the International Symposium on Code Generation and Optimization, 2004, pp. 75-86.
100. D. She, Y. He, and H. Corporaal, *Energy Efficient Special Instruction Support in an Embedded Processor with Compact ISA*, in Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2012, pp. 131-140.
101. N. Clark, et al., *Application-Specific Processing on a General-Purpose Core via Transparent Instruction Set Customization*, in Proceedings of the 37th IEEE/ACM International Symposium on Microarchitecture, 2004, pp. 30-40.
102. N. Clark, et al., *An Architecture Framework for Transparent Instruction Set Customization in Embedded Processors*, in Proceedings of the 32nd International Symposium on Computer Architecture, 2005, pp. 272-283.
103. R. Jayaseelan, H. Liu, and T. Mitra, *Exploiting forwarding to improve data bandwidth of instruction-set extensions*, in Proceedings of the 43rd Design Automation Conference, 2006, pp. 43-48.
104. K. Atasu, L. Pozzi, and P. Ienne, *Automatic application-specific instruction-set extensions under microarchitectural constraints*, in Proceedings of the 40th Design Automation Conference, 2003, pp. 256-261.
105. L. Pozzi and P. Ienne, *Exploiting pipelining to relax register-file port constraints of instruction-set extensions*, in Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2005, pp. 2-10.
106. D. She, et al., *Scheduling for Register File Energy Minimization in Explicit Datapath Architectures*, in Proceedings of Design, Automation Test in Europe Conference Exhibition, 2012, pp. 388-393.
107. S. Park, et al., *Bypass aware instruction scheduling for register file power reduction*, in Proceedings of the Conference on Language, Compilers, and Tool Support for Embedded Systems, 2006, pp. 173-181.

108. J. Cong, G. Han, and Z. Zhang, *Architecture and compilation for data bandwidth improvement in configurable embedded processors*, in Proceedings of the International Conference on Computer-Aided Design, 2005, pp. 263-270.
109. P.M. Heysters, G.J.M. Smit, and E. Molenkamp, *Montium - Balancing between Energy-Efficiency, Flexibility and Performance*, in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms, 2003, pp. 235-241.
110. S. Vassiliadis, et al., *The MOLEN polymorphic processor,* IEEE Transactions on Computers, 2004, 53(11): pp. 1363-1375.
111. B. Kastrup, A. Bink, and J. Hoogerbrugge, *ConCISe: A Compiler-Driven CPLD-Based Instruction Set Accelerator*, in Proceedings of the 7th Symposium on Field Programmable Custom Computing Machines, 1999, pp. 92-101.
112. L. Bauer, et al., *RISPP: Rotating Instruction Set Processing Platform*, in Proceedings of the 44th Design Automation Conference, 2007, pp. 791-796.
113. G. Venkatesh, et al., *Conservation cores: reducing the energy of mature computations*, in Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems, 2010, pp. 205-218.
114. H.P. Huynh, J.E. Sim, and T. Mitra, *An efficient framework for dynamic reconfiguration of instruction-set customization*, in Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, 2007, pp. 135-144.
115. G. Venkatesh, et al., *QsCores: trading dark silicon for scalable energy efficiency with quasi-specific cores*, in Proceedings of the 44th International Symposium on Microarchitecture, 2011, pp. 163-174.
116. G. Dasika, et al., *PEPSC: A Power-Efficient Processor for Scientific Computing*, in Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2011, pp. 101-110.
117. R. Kastner, et al., *Instruction generation for hybrid reconfigurable systems,* ACM Transaction on Design Automation of Electronic Systems, 2002, 7(4): pp. 605-627.
118. Y. Guo, et al., *A graph covering algorithm for a coarse grain reconfigurable system*, in Proceedings of the Conference on Language, Compiler, and Tool for Embedded Systems, 2003, pp. 199-208.

119. A.A. Abbo, R.P. Kleihorst, and B. Schueler, *Xetal-II: A Low-Power Massively-Parallel Processor for Video Scene Analysis,* Journal of Signal Processing Systems, 2009, 62(1): pp. 17-27.
120. Y. He, et al. *Real-time implementations of Hough Transform on SIMD architecture*, in Proceedings of the Second ACM/IEEE International Conference on Distributed Smart Cameras, 2008, pp. 1-8.
121. S. Seo, et al. *Process variation in near-threshold wide SIMD architectures*, in Proceedings of the 49th Annual Design Automation Conference, 2012, pp.980-987.
122. S. Das, et al., *RazorII: In situ error detection and correction for PVT and SER tolerance,* IEEE Journal of Solid-State Circuits, 2009, 44(1): pp. 32-48.
123. M. Fojtik, et al. *Bubble Razor: An architecture-independent approach to timing-error detection and correction*, in Proceedings fo IEEE Internatinal Solid-State Circuits Conference, 2012, pp. 488-490.
124. A. Sathanur, et al. *Physically clustered forward body biasing for variability compensation in nanometer CMOS design*, in Proceedings of Design, Automation, and Test in Europe Conference, 2009, pp. 154-159.
125. R. Dreslinski, et al., *Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits,* Proceedings of the IEEE, 2010, 98(2): pp. 253-266.
126. B. Liu, et al. *Standard cell sizing for subthreshold operation,* in Proceedings of the 49th Design Automation Conference, 2012, pp. 962-967.
127. N. Verma and A.P. Chandrakasan, *A 256 kb 65 nm 8T subthreshold SRAM employing Sense-amplifier Redundancy,* IEEE Journal of Solid State Circuits, 2008, 43(1): pp. 141-149.
128. B. Calhoun and A. Chandrakasan. *A 256kb sub-threshold SRAM in 65nm CMOS*, in Proceedings of IEEE International Solid-State Circuits Conference, 2006, 2592-2601.
129. B. Zhai, et al., *A variation-tolerant sub-200 mV 6-T subthreshold SRAM,* IEEE Journal of Solid-State Circuits, 2008, 43(10): pp. 2338-2348.
130. A. Wang and A. Chandrakasan, *A 180-mV subthreshold FFT processor using a minimum energy design methodology,* IEEE Journal of Solid-State Circuits, 2005, 40(1): pp. 310-319.
131. B. Zhai, et al. *A 2.60 pJ/Inst subthreshold sensor processor for optimal energy efficiency*, in Proceedings of the IEEE Symposium on VLSI Circuits, 2006, pp. 154-155.

132. M. Seok, et al., *The Phoenix Processor: A 30pW platform for sensor applications*, in Proceedings of IEEE Symposium on VLSI Circuits, 2008, pp. 188-189.
133. J. Kwong, et al., *A 65 nm Sub-Vt Microcontroller With Integrated SRAM and Switched Capacitor DC-DC Converter,* IEEE Journal of Solid-State Circuits, 2009, 44(1): pp. 115-126.
134. Y. Pu, et al. *An Ultra-Low-Energy/Frame Multi-Standard JPEG CO-Processor in 65nm CMOS with Sub/Near-Threshold Power Supply*, in Proceedings of IEEE International Solid-State Circuits Conference, 2009, pp. 146-147.
135. H. Kaul, et al. *A 300mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45nm CMOS*, in Proceedings of the Solid-State Circuits Conference, 2009, 260-263.
136. P. Francesco, et al. *An integrated hardware/software approach for run-time scratchpad management*, in Proceedings of the 41st annual conference on Design automation, 2004, pp. 238-243.
137. N. Jayasena, et al. *Stream register files with indexed access*, in Proceedings of 10th International Symposium on High Performance Computer Architecture, 2004, pp. 60-72.
138. S. Kyo and S. Okazaki, *IMAPCAR: A 100 GOPS In-Vehicle Vision Processor Based on 128 Ring Connected Four-Way VLIW Processing Elements,* Journal of Signal Processing Systems, 2008, 62(1): pp. 5-16.
139. A. Prengler and K. Adi. *A Reconfigurable SIMD-MIMD Processor Architecture for Embedded Vision Processing Applications*, in Proceedings of SAE World Congress and Exhibition, 2009, pp. 1-9.
140. Xilinx. *http://www.xilinx.com.*
141. H. Nikolov, T. Stefanov, and E. Deprettere, *Multi-processor system design with ESPAM,* in Proceedings of the 4th international conference on Hardware/software codesign and system synthesis, 2006, pp. 211-216.
142. A. Moonen, et al., *Decoupling of computation and communication with a communication assist*, in Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 2007, pp. 63-68.
143. R. Pieters, P. Jonker, and H. Nijmeijer, *Real-Time Center Detection of an OLED Structure*, in Proceedings of the 11th International Conference on Advanced Concepts for Intelligent Vision Systems, 2009, pp. 400-409.

144. Z. Ye, et al., *Feasibility Analysis of Ultra High Frame Rate Visual Servoing on FPGA and SIMD Processor*, in Proceedings of the 12th IAPR Conference on Machine Vision Applications, 2011, pp. 55-58.
145. N. Otsu, *A threshold selection method from gray-level histograms,* Automatica, 1975, 11: pp. 285-296.
146. O.P. Gangwal, A. Nieuwland, and P. Lippens, *A scalable and flexible data synchronization scheme for embedded HW-SW shared-memory systems*, in Proceedings of the 14th International Symposium on Systems Synthesis, 2001, pp. 1-6.
147. A. Moonen, et al., *A multi-core architecture for in-car digital entertainment*, in Proceedings of GSPx Conference, 2005.
148. A. Shabbir, et al., *CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications,* Journal of Systems Architecture, 2010, 56(7): pp. 265-277.
149. M. Gschwind, *The CELL broadband engine: exploiting multiple levels of parallelism in a chip multiprocessor,* International Journal of Parallel Programming, 2007, 35(3): pp. 233-262.
150. Z. Guo, W. Najjar, and B. Buyukkurt, *Efficient hardware code generation for FPGAs,* ACM Transaction on Architecture and Code Optimization, 2008, 5(1): pp. 6:1-6:26.
151. H. Nikolov, T. Stefanov, and E. Deprettere, *Automated Integration of Dedicated Hardwired IP Cores in Heterogeneous MPSoCs Designed with ESPAM,* EURASIP Journal on Embedded Systems, 2008, 2008(1).
152. M. Geilen and T. Basten, *Requirements on the execution of Kahn process networks,* in Proceedings of the 12th European Symposium on Programming, 2003, pp. 319-334.
153. K. Huang, D. Grunert, and L. Thiele, *Windowed FIFOs for FPGA-based multiprocessor systems*, in Proceedings of IEEE International Conference on Application-specific Systems, Architectures and Processors, 2007, pp. 36-41.
154. A. Hormati, et al., *Optimus: efficient realization of streaming applications on FPGAs*, in Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2008, pp. 41-50.

SAMENVATTING

Streaming applicaties vormen een belangrijke klasse binnen opkomende ingebedde systemen zoals slimme camera netwerken, onbemande voertuigen en industrieel printen. Enerzijds zijn deze applicaties meestal erg reken-intensief en aan real-time voorwaarden onderhevig. Anderzijds hebben de ingebedde systemen die deze applicaties uitvoeren vaak een beperkte energiebron, zoals batterijen of zonnepanelen. Daarom wordt energie-bewustheid een steeds belangrijker aspect in het architectuur ontwerp van deze systemen. Om een hoge energie-efficiëntie in dergelijke systemen te bereiken, zijn significante inspanningen op verschillende hardware- en softwareniveau's nodig. Dit proefschrift behandelt een deel van de uitdagingen in het ontwerpen van energie-efficiënte architecturen voor streaming applicaties, en beslaat de volgende vier onderwerpen:

Bij ingebedde processoren wordt een substantieel deel van de energie verbruikt door de register file (RF). Een eerste bijdrage van deze thesis is het voorstellen van MOVE-Pro, een nieuwe op Transport Triggered Architecture (TTA) gebaseerde processor architectuur waarmee het energieverbruik van de register file verminderd wordt. Met fijnkorrelige aansturing van het datapad en het optimaliseren op verschillende hardware/software niveaus, wordt een significante vermindering van het aantal toegangen tot de registerfile bereikt in MOVE-Pro. Gemiddeld wordt ongeveer 70% van de RF-toegangen geëlimineerd, hetgeen een drastische vermindering van het RF energieverbruik tot gevolg heeft. Met de voorgestelde MOVE-Pro architectuur wordt de RF energiebesparing volledig overgedragen aan de totale energiebesparing van de rekenkern. Vergeleken met zijn RISC tegenhanger wordt een totale energiebesparing van de rekenkern tot 11.6% bereikt.

Bij het ontwerpen van Applicatie-Specifieke Instructie-set Processoren (ASIPs) is het gebruikelijk om instructiesets te synthetiseren die ondersteuning bieden voor het uitvoeren van bewerkingspatronen die voorkomen in de applicaties die met deze processoren uitgevoerd dienen te worden. Hiermee worden betere prestaties en energie-efficiëntie bereikt. Echter, in een generieke ingebedde processor met een compacte ISA kunnen dergelijke instructies lei-

den tot een grote overhead. Een tweede bijdrage van dit proefschrift is het voorstellen van een architectuur die flexibele bewerkingsparen in processoren met compacte ISAs ondersteunt. Het ontwerp introduceert een gedeeltelijk herconfigureerbare decoder en een software-aangestuurd omleidingsnetwerk. Dit stelt de processor in staat om bewerkingsparen te ondersteunen zonder de instructiebreedte of het aantal register file poorten te vergroten. Rijkelijke en gedetailleerde experimentele resultaten tonen aan dat de voorgestelde architectuur gemiddeld een 26.0% lagere dynamische cycle count heeft en gemiddeld 15.8% minder energie verbruikt dan vergelijkbare processoren.

Bij veel ingebedde streaming applicaties kan een substantiele hoeveelheid data-level parallellisme uitgebuit worden. De Xetal-Pro, een grootschalig parallelle SIMD architectuur, wordt voorgesteld om met energie-efficiëntie gecombineerd met hoge rekenprestatie-eisen te kunnen omgaan. Het initiële idee om grootschalig parallellisme te combineren met agressieve Vdd schaling wordt gepresenteerd en in detail behandeld. Ook wordt een hybride geheugensysteem voorgesteld dat het niet-lokale geheugen verkeer verminderd en verdere Vdd schaling mogelijk maakt. Dit werk toont aan dat het mogelijk is om een energieverbruik 1 pJ per bewerking voor de rekenkern te realiseren voor typische ingebedde streaming applicatie kernels.

Multi-processor system-on-chips (MPSoCs) zijn een populaire aanpak aan het worden om aan de toenemende vraag naar rekenkracht en efficiëntie in streaming applicaties te voldoen. Als vierde bijdrage van dit proefschrift wordt een efficiënte en voorspelbare communication assist (CA) om generieke IP-kernen in MPSoCs te integreren voorgesteld. Ook wordt het bijbehorende cycle-nauwkeurige synchronous dataflow (SDF) model voor de voorgestelde communication assist gepresenteerd. Door dit SDF model in SDF analyse programma's te integreren kunnen worst-case systeemeigenschappen zoals doorvoer, vertraging en buffer groottes conservatief geanaliseerd worden tijdens het ontwerpen. In een case study wordt een beeldverwerkingspijplijn van een industriële applicatie, het printen van Organic Light Emitting Diodes (OLEDs), afgebeeld op het voorgestelde platform. Deze case study toont ook aan dat de voorgestelde ontwerpmethodologie een efficiënte integratie van versneller-IPs in heterogene MPSoCs voor streaming applicaties mogelijk maakt.

ACKNOWLEDGEMENTS

PhD study is a very special journey in one's life. I wouldn't have enjoyed it so much without the support from so many people. By this opportunity, I would like to express my thanks to all those who either helped me in expertise or shared their life with me during my PhD study.

The foremost thanks go to my supervisor, prof.dr. Henk Corporaal, for his guidance, support, and encouragement over these years. I am very grateful for many inspiring and in-depth discussions with Henk, which is a key to the successful outcome of this research.

I would like to thank prof.dr.ir Pieter Jonker and dr. Bart Mesman for being my second promoter and co-promoter. I really appreciate that. I would also like to thank prof.dr. Koen Bertels, prof.dr. Ben Juurlink, and prof.dr. Jose Pineda de Gyvez for reading the thesis, giving in-depth comments, and participating in my PhD defense.

I want to express my thanks to Dongrui She, Zhenyu Ye, Yu Pu, Sander Stuijk, Sebastian Moreno Londono, Ahsan Shabbir, and Shakith Fernando. It is really great to cooperate with all of you, which also leads to very fruitful outcomes. I also want to thank Richard Kleihorst, Anteneh Abbo, Zoran Zivkovic, Xinting Gao, and all the friends in NXP for their kind support during my stay in the Xetal group.

I am grateful to my officemates Raymond Frijns and Majid Nabi Najafabadi. Whenever I need some help, you are always there. I really enjoyed the time spent with you guys. I also want to thank my students, Corne Kraaij, Tim Vriends, Luc Waeijen, and Rendong He, for giving me the opportunity to supervise them. It was a great time working together with them. I would like to thank all the other members of the Electronic Systems group at Eindhoven University of Technology, especially Marja de Mol-Regels, Rian van Gaalen, and Jan van Dalfsen for your very kind help and support.

My PhD time in the Netherlands would not have been so amazing without the presence of my other friends: Bo Liu, Yang Yang, Hao Hu, Yongjian Tang, Yu-

anjia Du, Ning Xie, Yu Lin, Hao Gao, Feijun Zheng, Lei Xie, Tian Gang, Xin Wan, Weihua Tang, Ying Zhang, Wei Tong, and many many others. And, of course, my dear brothers and sisters in the Xianfeng cell group. I wish you all the best.

Finally, I would like to thank my parents Qingxiang He and Yinfen Ni, my wife Songyue Chen, and my daughter Ruthia Shuxin He, for their everlasting love and support. I am deeply indebted to them.

Thank you all!

Yifan, September 2013

一凡 写于 2013 年 9 月

CURRICULUM VITAE

Yifan He was born in Hangzhou, Zhejiang Province, China, on Sep. 26th, 1981. He received his B.S. and M.S. degrees (cum laude) in electrical engineering from Zhejiang University, Hangzhou, China, in 2004 and 2006, respectively. In 2008, he received his second M.S. degree (cum laude) in electrical engineering from the Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands.

In August 2008, he started working towards a Ph.D. degree within the Electronic Systems group at the department of electrical engineering of the Eindhoven University of Technology. His research was funded by the Ministry of Economic Affairs of The Netherlands within the EVA project. It has led among others to several publications, IPs, and this thesis.

Yifan is currently a researcher at Recore Systems, Enschede, The Netherlands. He is also a part-time researcher at the department of electrical engineering of the Eindhoven University of Technology.

LIST OF PUBLICATIONS

Journal Papers

- **Y. He**, D. She, S. Stuijk, and H. Corporaal, "*Efficient Communication Support in Predictable Heterogeneous MPSoC Designs for Streaming Applications*", in Journal of Systems Architecture (JSA), DOI: 10.1016/ j.sysarc.2013.04.005, 2013

- D. She, **Y. He**, and H. Corporaal, "*An Energy Efficient Method of Supporting Flexible Special Instructions in an Embedded Processor with Compact ISA*", to be appeared in ACM Transactions on Architecture and Code Optimization (TACO), Vol. 10, No. 3, 2013

- Y. Pu, **Y. He**, Z. Ye, S. M. Londono, R. Kleihorst, A. Abbo, and H. Corporaal. "*From Xetal-II to Xetal-Pro: On the Road Towards an Ultra Low-Energy and High Throughput SIMD Processor*", in IEEE Transactions on Circuit and Systems for Video Technology (TCAS-VT), Vol. 21, No. 4, pp. 472-484, 2011

International Conference Papers

- S. Fernando, F. Siyoum, **Y. He**, A. Kumar, and H. Corporaal, "*MAMPSx: A Design Framework for Rapid Synthesis of Predictable Heterogeneous MPSoCs*", accepted by IEEE International Symposium on Rapid System Prototyping (RSP'13), Montreal, Canada, 2013

- L. Waeijen, D. She, H. Corporaal, and **Y. He**, "*SIMD Made Explicit*", accepted by International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS'13), Samos, Greece, 2013

- D. She, **Y. He**, L. Waeijen, and H. Corporaal, "*OpenCL Code Generation for Low Energy Wide SIMD Architectures with Explicit Datapath*", accepted by International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS'13), Samos, Greece, 2013

- D. She, **Y. He**, and H. Corporaal. "*Energy Efficient Special Instruction Support in an Embedded Processor with Compact ISA*", in Proceedings of the International Conference on Compilers Architecture and Synthesis for Embedded Systems (CASES'12), pp. 131-140, Tampere, Finland, 2012 (Best paper nominee)

- D. She, **Y. He**, B. Mesman, and H. Corporaal. "*Scheduling for Register File Energy Minimization in Explicit Datapath Architectures*", in Proceedings of the Conference on Design, Automation and Test in Europe (DATE'12), pp. 388-393, Dresden, Germany, 2012

- **Y. He**, Z. Ye, D. She, B. Mesman, and H. Corporaal. "*Feasibility Analysis of Ultra High Frame Rate Visual Servoing on FPGA and SIMD Processor*", in Proceedings of the 13th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS'11), pp.623-634, Ghent, Belgium, 2011

- **Y. He**, D. She, B. Mesman, and H. Corporaal. "*MOVE-Pro: a Low Power and High Code Density TTA Architecture*", in Proceedings of the 11th International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS'11), pp. 294-301, Samos, Greece, 2011

- Z. Ye, **Y. He**, R. Pieters, B. Mesman, H. Corporaal, and P. Jonker. "*Demo: An embedded vision system for high frame rate visual servoing*", in Proceedings of the 5th ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'11), pp. 1-2, Ghent, Belgium, 2011

- D. She, **Y. He**, B. Mesman, and H. Corporaal. "*Energy Efficient Code Generation for Processors with Exposed Datapath*", in Proceedings of the 9th workshop on Optimizations for DSP and Embedded Systems (ODES'11), pp. 55-61, Chamonix, France, 2011

- Z. Ye, **Y. He**, R. Pieters, B. Mesman, H. Corporaal, and P. Jonker. "*Bottlenecks and Tradeoffs in Ultra High Frame Rate Visual Servoing: A Case Study*", in Proceedings of the 12th IAPR Conference on Machine Vision Applications (MVA'11), pp. 55-58, Nara, Japan, 2011

- **Y. He**, Y. Pu, Z. Ye, S. M. Londono, R. Kleihorst, A. Abbo, and H. Corporaal. "*Xetal-Pro: An Ultra-Low Energy and High Throughput SIMD Processor*", in Proceedings of the 47th ACM/IEEE International Conference on Design Au-

tomation (DAC'10), pp. 543-548, Anaheim, USA, 2010 (Best paper nominee & HiPEAC paper award)

- **Y. He**, Z. Zivkovic, R. Kleihorst, A. Danilin, and H. Corporaal. "*Real-Time Implementations of Hough Transform on SIMD Architecture*", in Proceedings of the 2nd ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC'08), pp. 1-8, Palo Alto, USA, 2008

- **Y. He**, Z. Zivkovic, R. Kleihorst, A. Danilin, H. Corporaal, and Bart Mesman. "*Real-Time Hough Transform on 1-D SIMD Processors: Implementation and Architecture Exploration*", in Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS'08), LNCS, Vol. 5259, pp. 254-265, Juan-les-Pins, France, 2008

Local Conference Papers

- **Y. He**, Dongrui She, and Henk Corporaal. "*A Comparative Study of Energy-Efficient Multiplier Design Using Data-Width-Aware Methodology*", in Proceedings of the 3rd STW.ICT conference on Program for Research in Embedded Systems and Software (PROGRESS'12), Veldhoven, the Netherlands, 2012 (Poster)

- **Y. He**, Z. Ye, D. She, R. Pieters, B. Mesman, and H. Corporaal. "*1000 fps Visual Servoing on the Reconfigurable Wide SIMD Processor*", in Proceedings of the 16th Annual Conference of the Advanced School for Computing and Imaging (ASCI'10), pp. 302-309, the Netherlands, 2010